Embedded Final Report

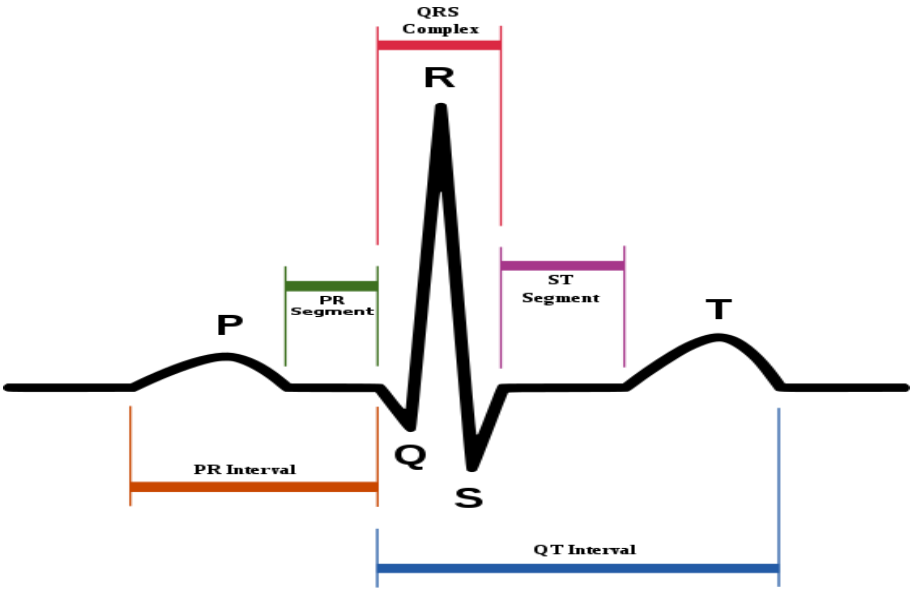
HEART MONITOR

Habiba Gamal

900151007

Introduction

Electrocardiography is the process of generating electrocardiograms (ECG signals). It is plotted as a graph of voltage versus time. This voltage measures the activity of the heart through electrodes that are placed on the skin. A normal ECG pattern is composed of P wave, QRS complex and T wave, as seen in the diagram below.



This project develops an embedded application that reads ECG signals through ECG sensor, plots the ECG signals in real time through an application running on the PC, and reports the heart rate to the user. The embedded application communicates with the PC through serial port. This application will allow the user to measure his beats per minute easily at home, without needing a specialized person.

The PC application can be easily changed into a mobile application that interfaces with the microcontroller through Bluetooth to make the application portable.

In addition, additional capabilities can be further implemented, like having m2m module like Data modules with TCP/IP stack, to allow the application to call for emergency if the user experiences irregular ECG patterns, or abnormal heart rates.

Progress

Current

- Embedded application supports the receipt of 3 commands:
    - start: to collect 1-minute worth of data. The default frequency is 150 HZ.
    - f=xxxx: to set the sampling rate. It supports frequencies from 1 digit to 4 digits. However, practically, the sampling rate should be 150Hz – 1kHz
    - bpm: to report the heart rate
- Python application supports reading the user commands through the command line and printing the program output on the command line, while serially receiving and transmitting the data to the microcontroller. The program also plots the ECG signals in real-time, allows the user to set the baud rate and select the COM port.

Initial Demo

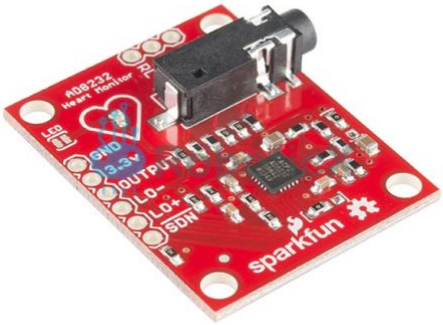
- Fully developed Python application, only missing concatenating dummy characters to short commands to fill the UART buffer and was done manually by user.
- Supporting the receipt of the three commands, but “bpm” used to transmit “report” over UART, and the bpm was not yet calculated.

Progress Since Initial Demo

- Bug fixes:
    - The first “start” command used to be ignored. Fixed by checking in the callback function of period elapsed for the timer that counts 1 minute that it was not in the reset state before stopping the timers.
    - The sampling frequency could not be increased, only decreased. This is because the auto-reload value of the timer is decreased below the current value of the counter. Was fixed by setting the counter value of the timer to 0 before setting the auto-reload value.
- Computing the bpm
- Concatenating dummy characters to the end of short commands to fill the UART buffer.

Used Components

- STM32F103

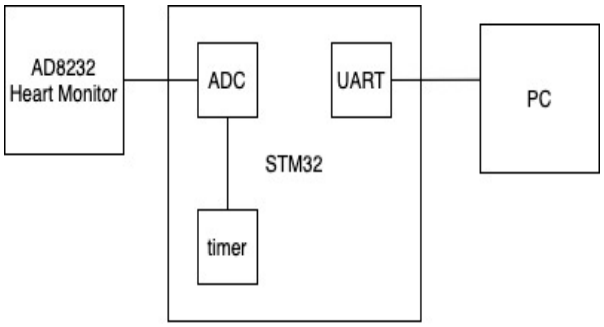


- ECG sensor AD8232 and 3 lead electrodes

- USB to TTL chip

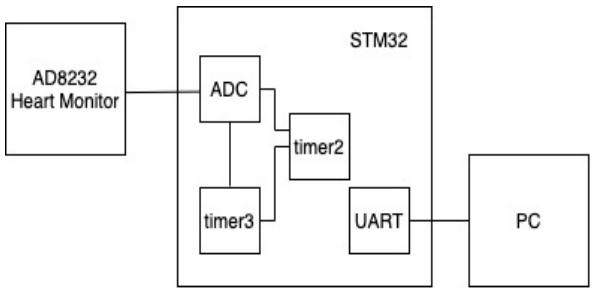
Block Diagram

Initial



This is my initial block diagram in the proposal. When I started implementing, I realized that I will need two timers: timer2 to count 1 minute, and timer3 to trigger the ADC start of conversion and thus control the sampling frequency of the ADC.

Updated



I followed this block diagram. The ECG sensor AD8232 is an external component to the system. Inside the microcontroller, the main used components are ADC to convert from analog to digital, timers to trigger ADC start of conversion and count 1 minute, and UART to serially receive the command and serially transmit the data to a PC application.

The PC application in our case is a developed Python application that is the front-end of the project. This application accepts command-line input and sends these serially to the microcontroller. The application prints the data received by the microcontroller, as well as, plots the ECG signals in real-time, after ADC conversion.

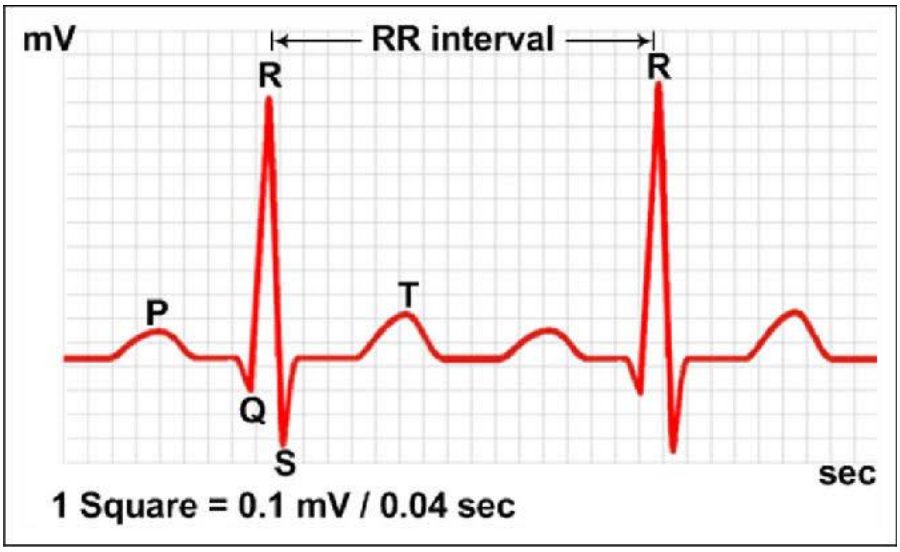
Implementation Details

Embedded Application

- UART receive interrupt is enabled. The call-back function that's called when the buffer is completely filled is where I check for the 3 supported commands.
- ADC is started as soon as the application is started, but the two timers are off
- The prescaler for the timers = clock / 1000. In my case 80MHz/ 1000 = 8000, so the timers count every 1ms
- "start": if this command is received, the timers are started. Initially, the sampling rate is at 150Hz. When the period has elapsed for the timer that counts the 1-minute, the callback function is reached. This is where the timer are stopped.
- "f=XXXX": if this command is received, the auto-reload register of timer 3 (that triggers the start of conversion of the ADC) is set to (1000 / f)
- "bpm": if this command is received, the microcontroller computer and reports the beats per minute
- If any other command is received, the microcontroller UART transmits "no command"

BPM calculation

- The BPM calculation is done through selecting a point in the ECG signal and measuring the time interval until the same point is received again.
- Usually in practice, the selected point is the peak of QRS complex.



- The time interval between the peaks of two QRS complex, or in other words between the two R's of two QRS complex, is called the RR interval.
- The minimum RR interval is 0.6s and the maximum is 1.2s for normal heart rate.
- The R is detected, and the time between two consecutive R's is recorded
- The beats per minute is 60 / RR interval in seconds.
- The storing of the RR interval is done when collecting 1-minute worth of data in the callback function of ADC end of conversion.
- The final computation of the bpm is done in the call-back function of receiving the full UART buffer when "bpm" is received.

Python Application

- The application is multi-threaded, to support concurrency between reading command-line input while reading serial data from the microcontroller and printing to the user.
- The first thread reads command line input from the user, concatenates dummy characters if the command is too short, and sends the data serially to the microcontroller.
- The second thread reads serially the data from the microcontroller and prints it to the user on the command line. This thread also plots the real-time ECG signals.
- The user can configure the COM port and the baud rate at the beginning

Flow (User Scenario)

Assuming that the microcontroller and the ECG sensor are combined together in the same device.

After the user connects the ECG electrodes to his body and connects the USB-to-TTL to the PC, the user will be ready to use the device.

The user sets the sampling frequency as any 1-digit to 3-digit number, ideally between 150 Hz and 1 kHz.

The user can skip this step and use the default sampling frequency of 150 Hz.

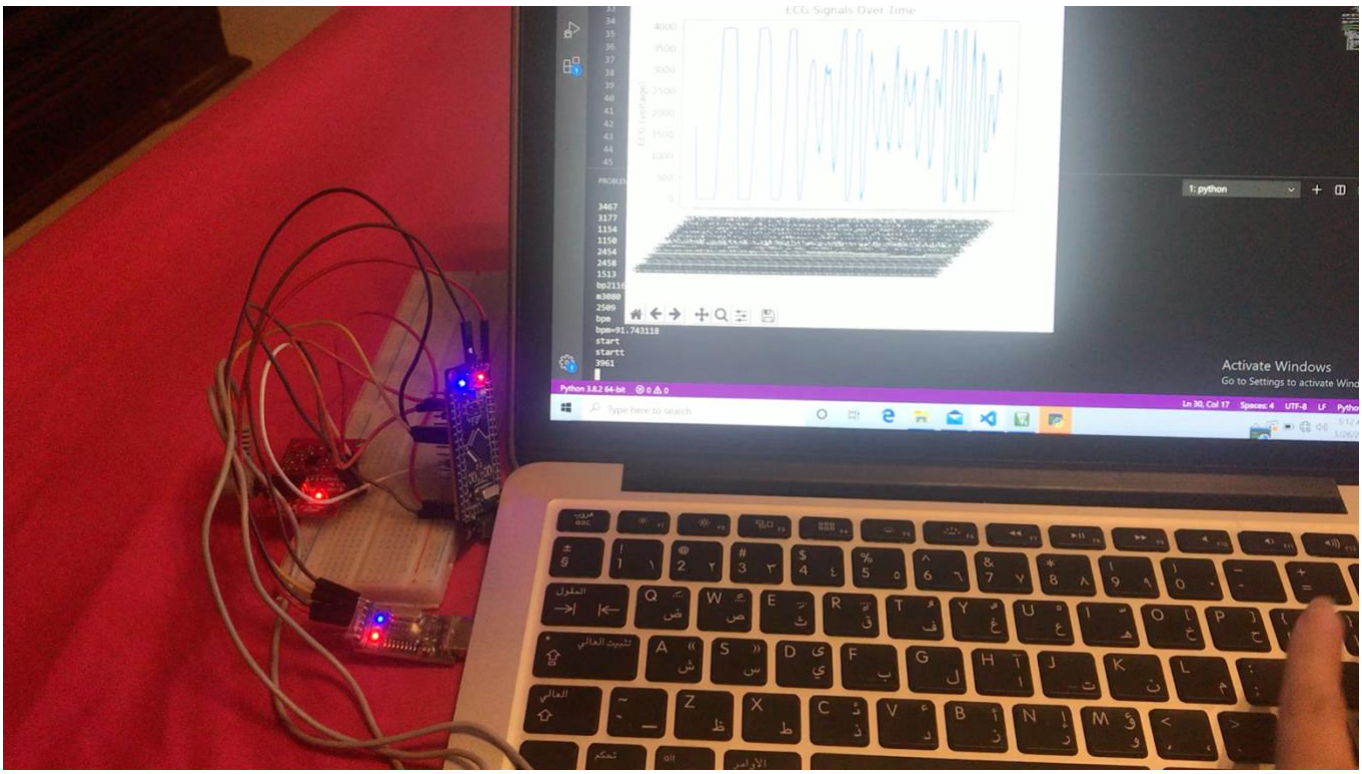
The user can send the command that collects 1-min worth of data, after which the user can send the command that reports the bpm.

Sending the command that reports the bpm before collecting 1-min worth of ECG data does not make sense, and the microcontroller will report back 0.

The user can at any time change the sampling frequency.

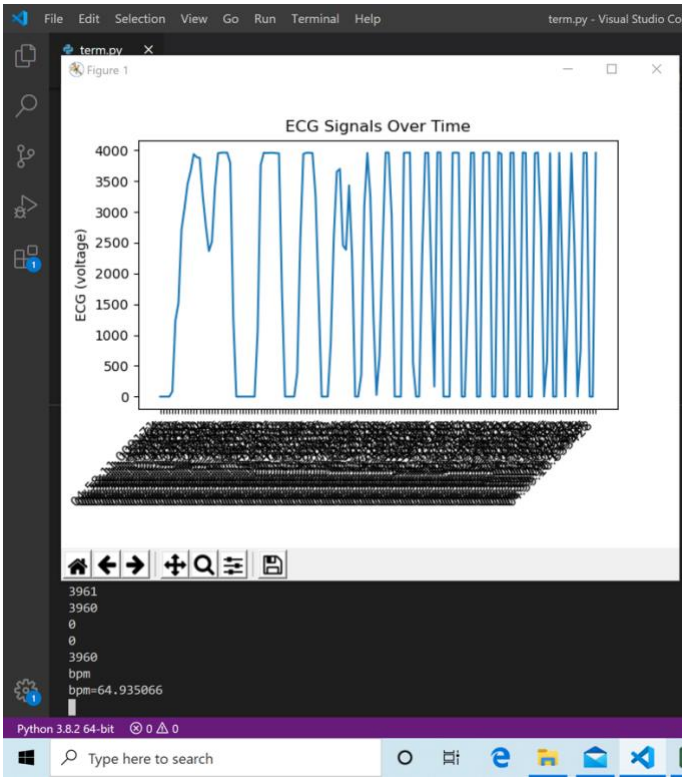
Demo Run

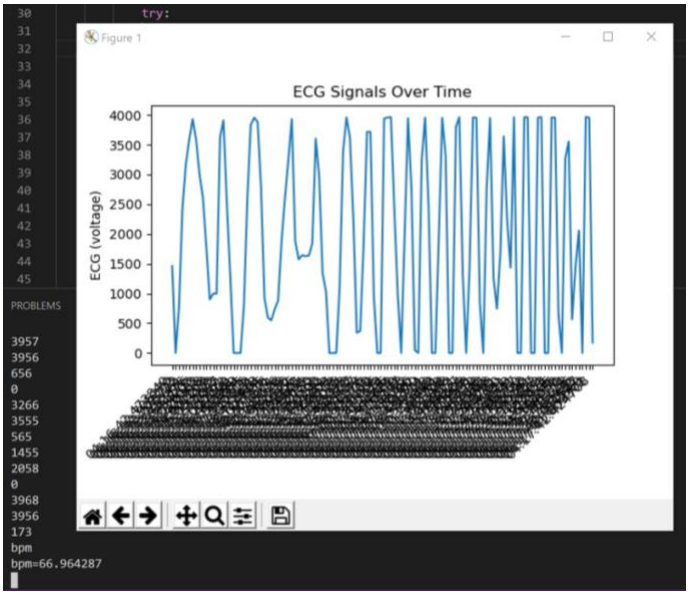
Video for collecting 1-min worth of data and reporting bpm

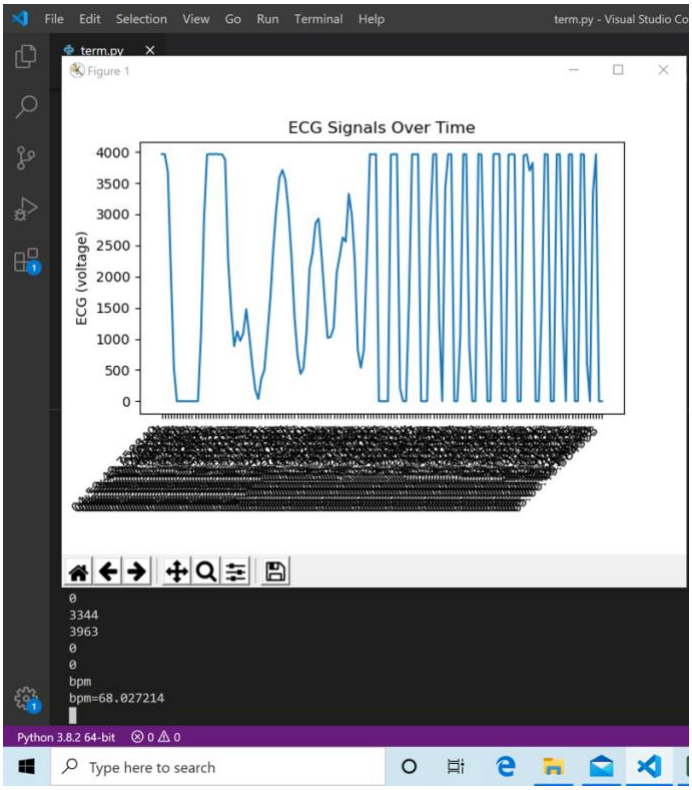


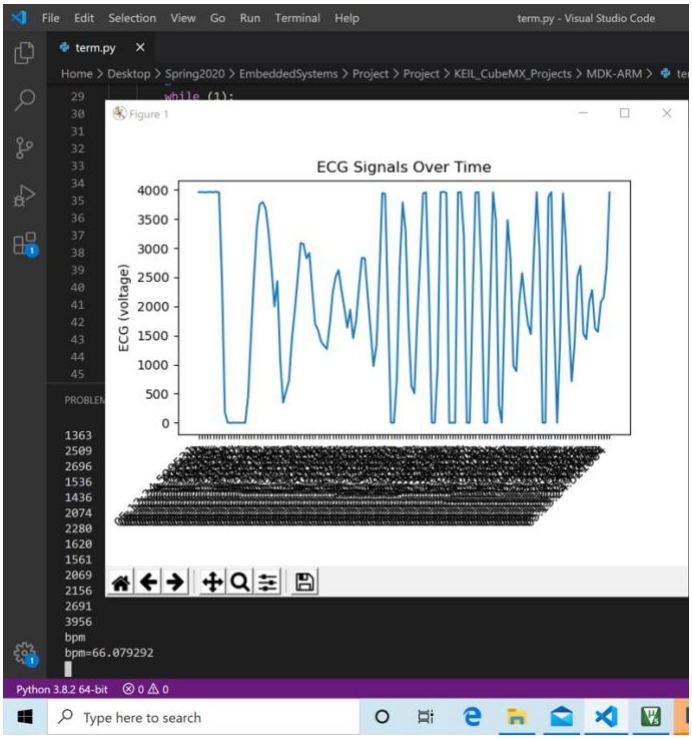
The LED of the microcontroller goes off when it detects the first peak, and goes on again when it detects the second peak (to test that it reads the time of 2 consecutive peaks when computing RR interval)

Screenshots for computed bpm






File Edit Selection View Go Run Terminal Help
term.py - Visual Studio Co
term.py
Figure 1
ECG Signals Over Time
ECG (voltage)
4000
3500
3000
2500
2000
1500
1000
500
0
0
3344
3963
0
0
bpm
bpm=68.027214
Python 3.8.2 64-bit
Type here to search


File Edit Selection View Go Run Terminal Help
term.py - Visual Studio Code
term.py
Home > Desktop > Spring2020 > EmbeddedSystems > Project > Project > KEIL_CubeMX_Projects > MDK-ARM >
while (1);
Figure 1
ECG Signals Over Time
ECG (voltage)
4000
3500
3000
2500
2000
1500
1000
500
0
1363
2509
2696
1536
1436
2074
2280
1620
1561
2069
2156
2691
3956
bpm
bpm=66.079292
Python 3.8.2 64-bit
Type here to search

To-do

- Currently, I calculate 1 RR interval while collecting 1-minute worth of data, and I check that it abides by the bounds of the normal interval. To do: Calculate multiple RR intervals and take their average, then use the average to compute the bpm.

Learning Outcomes

- Should not change auto-reload register value before setting the counter of the timer to zero. Generally, component register values affect each other. Therefore, registers should not be ignored when changing other register values, and the implication of the value held by each register should be examined to avoid erroneous behavior.
- Before setting the UART baud rate, the requirements of the application should be examined so that the baud rate is not a bottleneck to the application.
- Sensor readings are affected by other variables in the environment. In our case, it is affected by movement, posture, how well the electrodes are sticking to the skin, … These are sources of inaccuracies in the sensors' readings. Therefore, computations done using sensor collected data should be computed several times and average is taken.
- Testing hardware without an emulator is way harder, as it does not allow for control over register values or step-by-step execution. Moreover, register values cannot be watched. The need for a debugger/ emulator was revealed when I detected the problem of setting the timer counter to 0 before decreasing the auto-reload register value.

Github Link

https://github.com/habibagamal/EmbeddedProject_HeartMonitor