Embedded Final Report

HEART MONITOR

Habiba Gamal

900151007

Introduction:

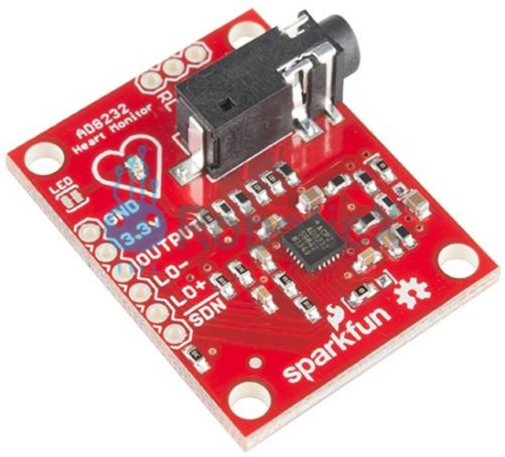
This application is a health embedded application, that uses ECG sensor, that a patient is to wear. This device will collect ECG signals from the user and compute his heart rate, as well as, plot the ECG signals for the user.

Current Progress:

- 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.

Used Components:

- STM32F103

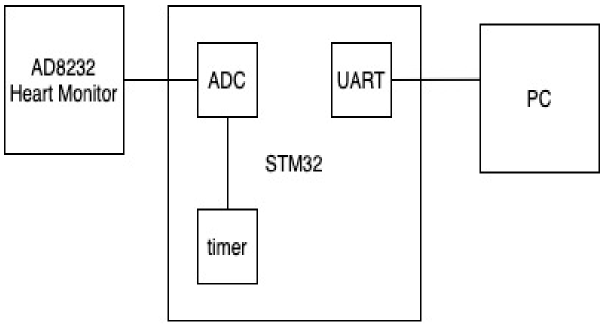


- ECG sensor AD8232

- USB to TTL chip

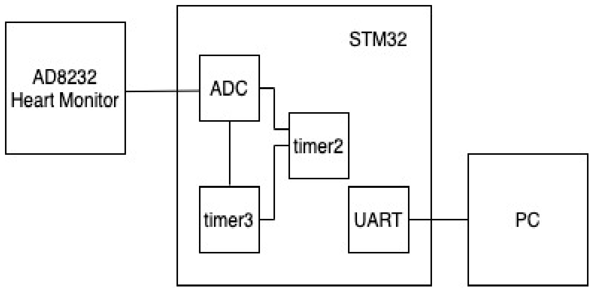
Block Diagram:

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 transmits 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.

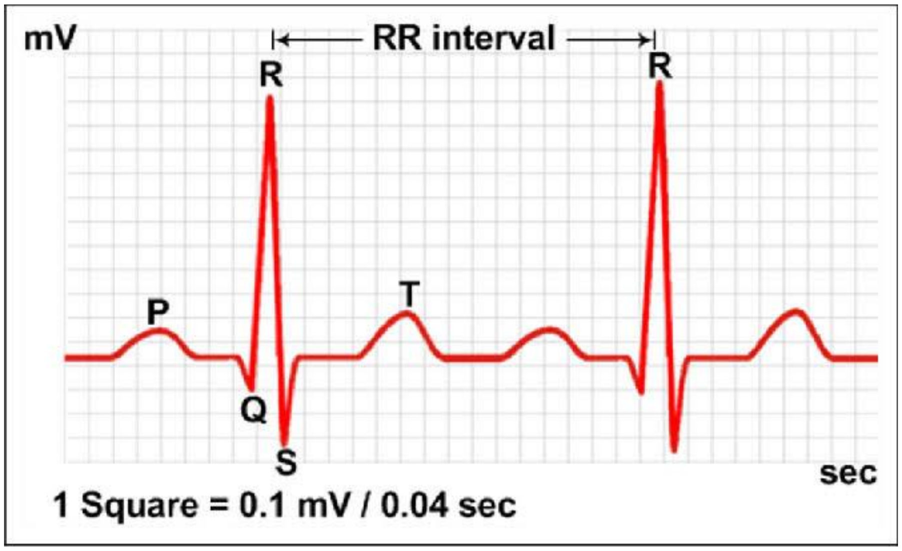
Implementation Details:

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 1 ms
- “start”: if this command is received, the timers are started. Initially, the sampling rate is at 150Hz.
- “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 white spaces if the command is too short, and send 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.

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.