Ahmed Fahmy

900160127

Embedded project

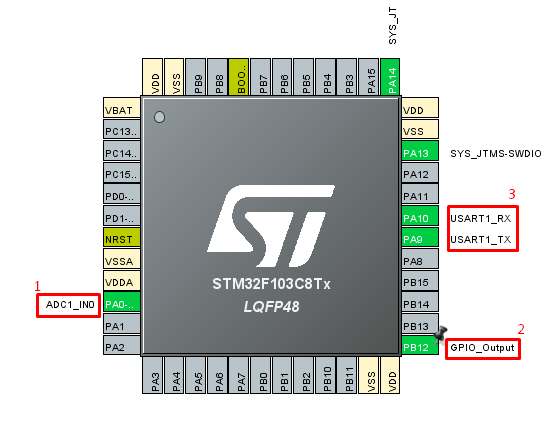
REPORT

**Embedded ECG**

This project is an embedded application using the STM32 module and the AD8232 integrated signal conditioning block to collect an ECG signal and report it to a PC over USB link. The associated python program proceeds with displaying the ECG live, as well as the heart rate in beats per minute.

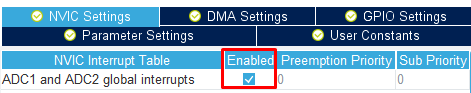
**CubeMX settings**

Below is a description of the settings used within CubeMX to configure the c code for the STM32 microcontroller.

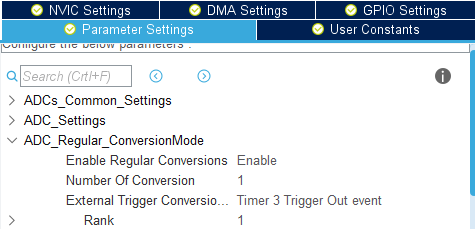


**1: ADC settings**

NVIC global interrupts were enabled, as we will need to use said interrupts at regular intervals with the timer (mentioned below). This is because such an application cannot use polling due to its high number of samples taken per second.



Regular conversion was selected as we will be taking samples at timer generated interrupts. This is why the External trigger is set to TIM3, which is in turn tuned to interrupt at the rate specified by the user via the UI.

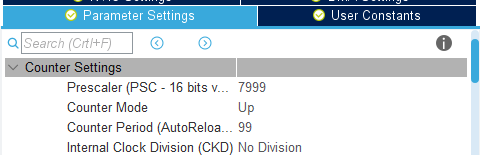


**2: GPIO ,Timer and UART**

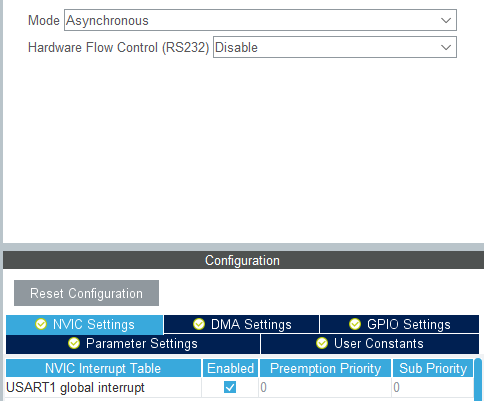
GPIO was activated for validation, blinks at every sample.Pin B12 was used for this.



As for the timer, a prescaler of 8000-1 (from zero to 7999 are 8000 digits) was used to scale down the 8MHz main system clock to 1000Hz which is more appropriate for this application

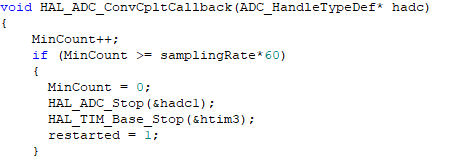


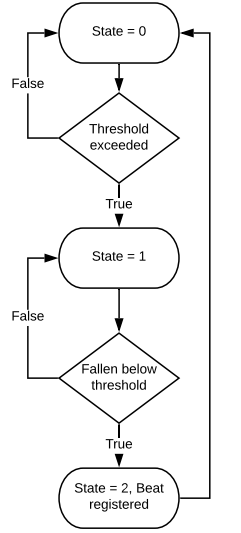
UART1 was used to asynchronously send and receive the data. UART global interrupts are used to interrupt the microcontroller whenever the user sends the signal to start sending.

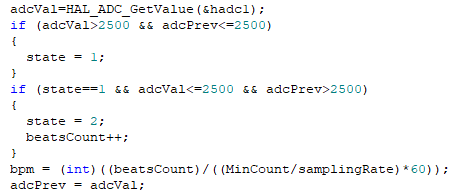


**Keil MDK code (ADC Callback)**

Starts by incrementing a counter that is used with the user defined sampling rate to determine remaining time and calculate BPM



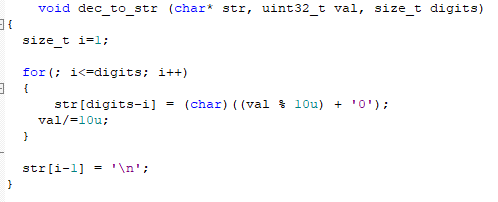


To calculate the bpm, a threshold of 2500/4096 was chosen after testing the output range of the ECG. Transitioning above and below the threshold (not necessarily consecutively) registers a beat. This is then divided by the time to calculate BPM.

**Keil MDK code (decimal to string):**

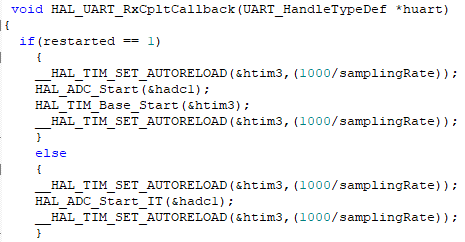
Used to convert from uint32\_t to string, which is required for transmission.

The function is utilized in the ADC callback above.

****

**Keil MDK code (UART callback):**

Sets the ARR according to the received sampling rate from the user. According to the prescaler, this corresponds to ARR = 1000/User input Starts the ADC as it might be disabled (not first usage). The received value is discretized in steps of 5 to enable samples per second from 0 to 1280 samples per second.



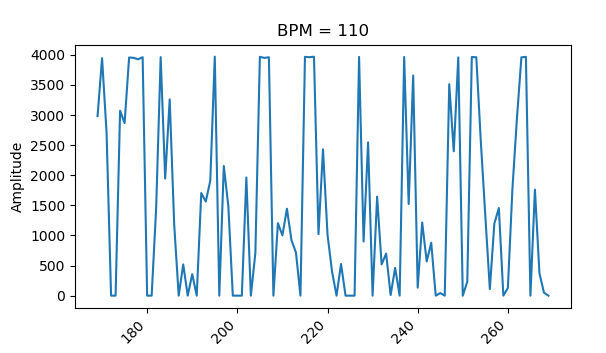
**Python code:**

The user runs the script with the following arguments:

* Sampling rate
* Close/hold
  + Hold freezes the chart after the minute has passed, close does not.
* COM port
* Baud rate

Example:





Pyserial is used to communicate as follows:

ser = serial.Serial(third, baudrate=int(fourth), timeout=None)

to open the user defined COM port.

ser.write(struct.pack('>B', int(first)))

to send the user defined sampling rate to the microcontroller via UART

**Animate function (FuncAnimate):**

This Function takes the current amplitude as well as the BPM from the serial link and continuously draws the graph seen above live (both BPM and ECG). The function lasts 60 seconds before terminating the program.

def animate(i, xs, ys):  
 # Aquire and parse data from serial port  
 line = ser.readline().decode('utf-8')  
 x = line.split(",")  
 i = i + 1  
 xs.append(i)  
 ys.append(int(x[0]))  
 if (i > 100): # window size  
 del xs[0];  
 del ys[0];  
  
 # Draw x and y lists  
 ax.clear()  
 ax.plot(xs, ys)  
  
 # Format plot  
 plt.xticks(rotation=45, ha='right')  
 plt.subplots\_adjust(bottom=0.30)  
 plt.title('BPM = '+str(round(int(x[1][:-1])/((time.time()-tic)/60))))  
 plt.ylabel('Amplitude')  
 if time.time()-tic > 58:  
 ani.event\_source.stop()  
 if argv[2] != 'hold':  
 plt.close()

This function is called through the matplotlib’s FuncAnimation method, which continuously plots the graph as follows.

ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys), interval=0.01, save\_count=100, blit=False)