Embedded Final Project

Report by

Hatem Shakweer

900150862

Option 2: Heart Monitor

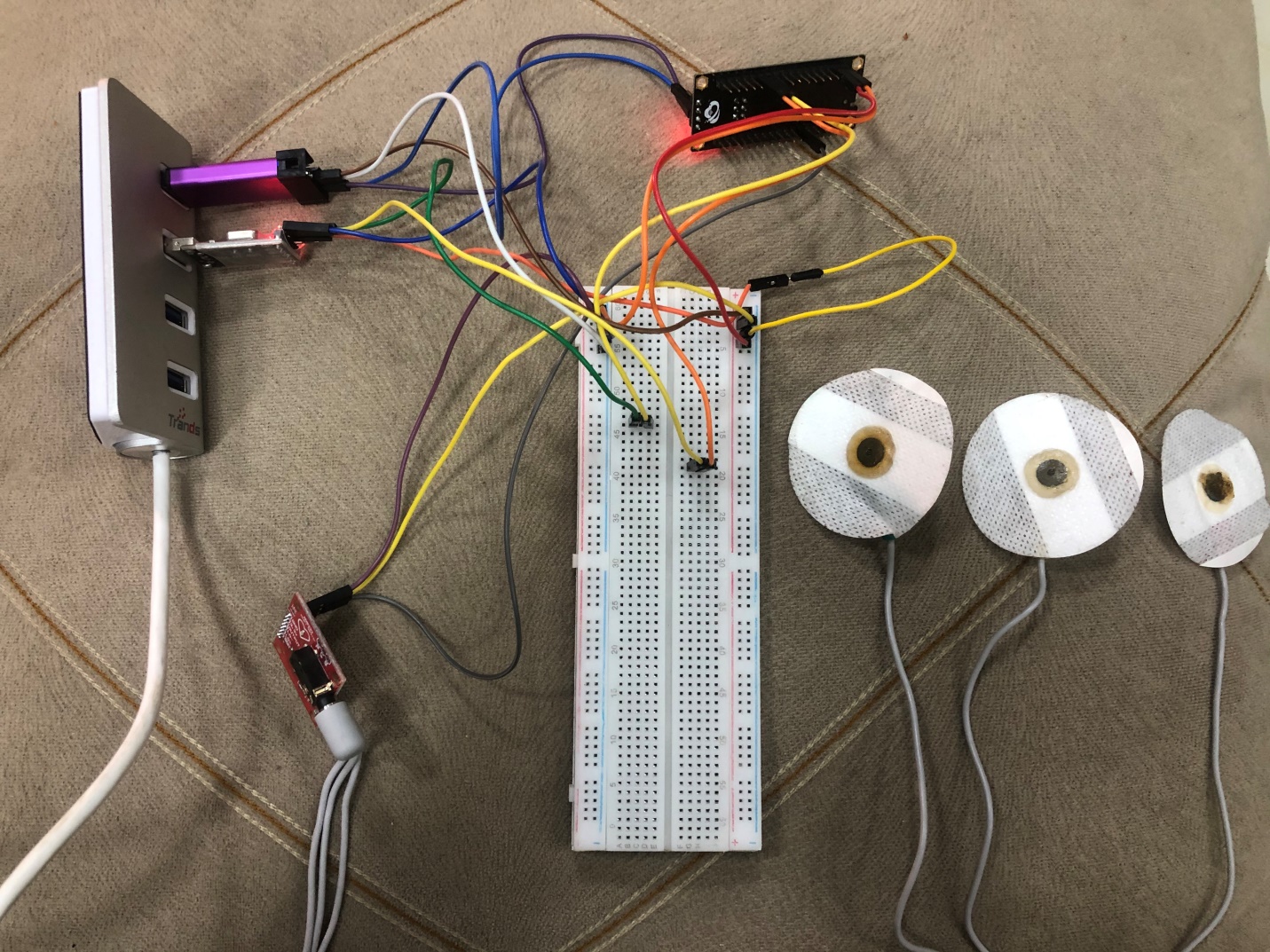
**Overview**

First of all the project chosen is to implement a heart time monitor using the AD8232 and STM32 to report to a PC a 1 minute worth of ECG data to be further analyzed by python. First we are supposed to take the command from the PC over serial wire to the STM32, then sending to the STM32 to set the sampling rate of the ADC. Similarly, another command from the PC will trigger the STM32 to collect 1 minute worth of data and send this data to the PC over UART. This data will then be analyzed in python using the help of “pyseries” and “matplotlib” libraries and plotted as a graph.

**Architecture**

The AD8232 Heart Rate Monitor has nine pins Which are: SDN, LO+, LO-, OUTPUT, 3.3V, GND, which provide the basic needed components to be able to communicate with the STM32. As discussed in lecture, also provided on the AD8232 are RA (Right Arm), LA (Left Arm), and RL (Right Leg) pins to attach and collect the ECG. Additionally, there is an LED indicator light that will pulsate to the rhythm of a heartbeat. The Pulse will be transmitted through the OUTPUT pin as an Analog signal to the ADC of the STM32. After the necessary analysis on the signal and converting it to digital, the values will be sent to a PC Python code over UART to by further plotted as a graph.

**Connection**



**Implementation**

The system is running somehow on client server architecture where the commands taken on the python side (client) are transmitted to the STM32 side (server) and the server responds to the python with the results. Which is then processed.

**Keil**

For the STM32 side, we are using Keil MicroVision and CubeMx to program the chip. We first enable UART1, ADC1, TIM1, and TIM2 on board using CubeMx. We set the TIM1 prescaler to “7” and countperiod to “49999”. Since the clock is 8MHZ, this enables the timmer to count 1 microsecond each clock tick. This timer will be used to take the samples from the ADC every 1/samplingrate seconds. Then TIM2 is similarly configured with prescaler “7999” and countperiod “62999” to count 1 millisecond each clock tick. This timer will be used to time the code to return to receive command state “A” after one minute of data collection. Then we start the timers’ base and set the calibration for better ADC conversion.

We then enter the infinite loop with state “A” ready to accept commands. According to the commands the code is set to a different state where it executes the tasks and returns to initial state “A”. If command received is Sampling command for example code go to state B, where the sampling rate is parsed from the command and our TIM1 delay is calculated as 1/sampling rate. Then it goes back to state A ready to accept commands. If command received is Collect command, code go to state C, where the ADC value is taken, then the ADC is stopped using HAL\_ADC1\_Stop(), then the value is sent over UART to the PC, and after wait time is done, ADC is started again using HAL\_ADC1\_Start() and the next value is taken. The above is repeated until 1 minute passes using TIM2. After 1 minute, it returns to state A ready to accept commands.

**Python**

For the python side, we are using Matplotlib to provide plotting and animation and pyserial to send and receive data over the UART to communicate with the Keil side. First we initialize the data arrays x and y together with plotting labels and figures for the plotting. Then we open a serial communication using serial.serial() setting the baud rate and the com of the USB to TTL. Then we define our “animate” function which will be called several times at intervals of 1/samplingrate to read the values from the UART and update the plotting line.

Then the user is prompted to enter the command. After the input is validated, the command is sent over the serial connection using ser.write(). If the command input is Collect data, we listen to read the values from the UART as FuncAnimate is called to loop over our “animate” function.

Unfortunately the animation does not work on Jupiter and a workaround is to convert the animation to a video and display it which is the case done here.

Also Values are all read in 1 minute and the STM32 stops conversion however the values take time by animation to be read as the frames speed are limited and it takes over 6 minutes to produce the video. However the video shows the animation as it if shown in runtime with same rate.