μBio

Margarida Reis*, Hugo Silva*†

*Instituto de Telecomunicações, Instituto Superior Técnico, Lisboa, 1049-001, Portugal
†Escola Superior de Tecnologia, Instituto Politécnico de Setúbal, Setúbal, 2914-761, Portugal

***Abstract*—The abstract goes here.**

I. Introduction

BITalino (Figure 1) is a novel development platform [1].

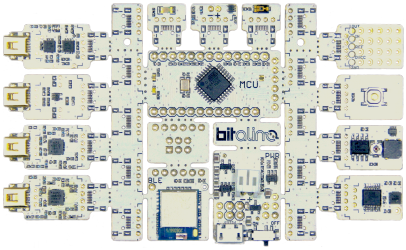


Fig. 1: BITalino (r)evolution board.

BITalino is currently limited to sampling its channels with a 10-bit maximum resolution and at a maximum sampling rate of 1 kHz which becomes inadequate when one intends to sample acoustic signals, for instance to obtain a phonocardiogram.

Furthermore, in what concerns its connectivity interfaces, in its native form, BITalino can only stream data to Bluetooth-enabled devices via classic Bluetooth or Bluetooth Low Energy (BLE). Although BITalino now provides a solution based on the R-IoT [2], an accessory to provide WiFi connectivity based on the CC3200 chipset with Inertial Measurement Unit (IMU), this accessory limits the functionality of BITalino to 200 Hz sampling rate and operates over User Datagram Protocol (UDP), hence not guaranteeing reliability in the acquisition process, i.e., data can be lost.

In this paper we present the μBio, a platform that seeks to expand the current functionalities of BITalino, by allowing for more channels to be sampled, at a higher sampling rate and with more resolution, with collected data streamed in real-time over WiFi, thus adding a cloud connectivity feature.

II. Background

III. Implementation

we devised μBio, a biosignal acquisition system
which is capable of wireless real-time streaming via WiFi

A. Architecture

μBio was designed
Figure 2
a status Light-Emitting Diode (LED), a low battery LED

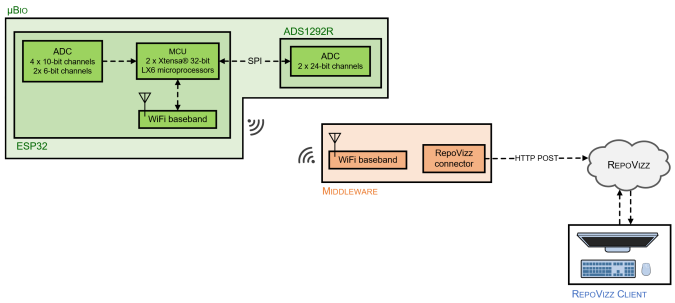


Fig. 2: Block diagram of the proposed μBio architecture.

[1]

Hardware-wise it is supported by the ESP32 and an external

ESP32 can be setup to function in Software-Enabled Access Point (SoftAP) mode, i.e., by becoming an Access Point (AP) in its own, to which other devices can connect. PONTE PARA VANTAGEM DESTE MODO. As such, once ESP32 becomes a SoftAP, a Transmission Control Protocol (TCP) server socket is created, flagged as ready to listen for new connections. The client devices (e.g., a computer) then use this socket to establish a connection with μBio in order to control the acquisition, its configuration settings and receive the acquired data in real-time. It should be noted that the server socket is binded to a specific address, which by default it corresponds with the Internet Protocol (IP) address of the ESP32 while operating in SoftAP mode (`192.168.4.1`), and port (`8001`).

we built our implementation upon the ESP32 chipset

we used 6 internal analog channels to mimic those of BITalino (r)evolution

The ADS1292R is a 2-channel 24-bit Analog-to-Digital Converter (ADC) Serial Peripheral Interface (SPI)

external voltage reference, set to 3.3 V

binary two's complement format, meaning that the left-most bit is the sign bit. As such, the positive full-scale output is $2^{24-1} - 1 = 8388607$ and the negative full-scale output is $-2^{24-1} = -8388608$, with the least significant bit having a weight of $V_{REF}/2^{24-1}$ [3].

B. Programming Interfaces

BITalino has a wide range of Application Programming Interfaces (APIs) that easily allow its incorporation into custom

[1] At the moment, analog channel *A0* is hard-codded as also being the *ABAT* analog input pin, as currently the ESP32 only supports the use of 6 analog channels if the Wi-Fi driver has been started.

made end-user software applications by providing the required tools to interact with the actual hardware, e.g., creating and maintaining a Bluetooth connection, through which the acquired data can be received. In fact, BITalino (r)evolution is capable of streaming the acquired data in real-time over any Universal Asynchronous Receiver/Transmitter (UART)-compatible interface (e.g., Bluetooth), through which it is also prepared to receive a set of configuration commands [4].

To be as user-friendly as possible, these APIs have high-level functions that wrap the low-level operations responsible for having the base station communicating with BITalino by sending the commands, of one or more bytes, that are then recognized by the firmware running on the device [1]. In this work, we focused on expanding the current support of the BITalino (r)evolution Python API[2] so that it is also compatible with μBio.

Initially, when developing a computer application for BITalino (e.g., a data logging one) with its Python API it is necessary to connect to the Bluetooth device with its Media Access Control (MAC) address by creating a client socket. It is through this socket that the connection is established and data is sent to the Bluetooth module, which then forwards it to the Microcontroller Unit (MCU) unit of BITalino via UART. To adapt this flow of execution to μBio, the Python API must create a TCP client socket that connects to the server socket, i.e., to the IP address to which the latter one was binded to.

1) State Commands: Once the connection has been successfully established, BITalino is ready to receive a set of commands to control its state - idle, live or simulated. While in idle mode, the default one once the device is turned on, it is in standby. In live mode the device acquires and streams data in real-time, whereas in simulated mode, mostly aimed at developers, it streams synthetic data generated by the firmware (sine, saw tooth, square waves and a pre-recorded electrocardiography time series).

Figure 3 presents the state commands for BITalino (r)evolution. To encode the three available modes it is necessary to use 2 bits ($\lceil \log_2 3 \rceil$) that are, as can be seen, defined as being the least significant ones of the byte that is sent for each mode. Moreover, for the live and simulated modes, the user is also required to choose which of the 6 channels are active, i.e., those whose samples are to be received. This is done by setting (activating) or clearing (deactivating) the 6 channel mask bits, labeled `A1` to `A6` in correspondence with the respective channel of the device. This mask is then sent along with the bits for mode selection, thus generating a complete byte.

However, these commands are not compatible with μBio because, as previously mentioned, it expands on BITalino (r)evolution by increasing the number of available analog inputs from 6 to 8, meaning that the channel mask alone requires its own byte. In light of this, new μBio-compatible state commands were defined, depicted in Figure 4. The encoding of the mode selection remained the same, but instead

[2]https://github.com/BITalinoWorld/revolution-python-api

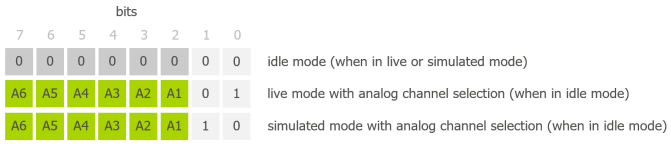


Fig. 3: State commands for the BITalino (r)evolution firmware.

of aggregating such bits and the channel mask into a single byte, this information is now spread across two bytes. The first byte is similar to the original command previously explained, but the 6 most significant bits are now set to 0. The second byte is filled with the 8-bit channel mask, labeled `A1` to `A6` to represent the corresponding internal analog channel from the ESP32 and `A7` to `A8` to represent the corresponding external analog channel from the ADS1292R chip, as pictured in Figure 2.

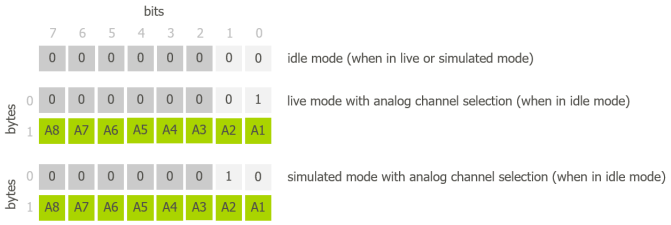


Fig. 4: State commands for the μBio firmware.

2) Data Packets: While data is being acquired, BITalino streams it as a structured sequence of bits, according to the scheme depicted in Figure 5a. The last byte includes a 4-bit Cyclic Redundancy Check (CRC) code and a 4-bit sequence number, which ranges from 0 to 15 in a loop, used to evaluate the integrity of the packet and to detect and recover from data packet inconsistency and loss, respectively. The following byte, in its 4 most significant bits, describes the state of the digital output pins *O1* and *O2* and the state of the digital input pins *I1* and *I2* using 1-bit for each pin state - `0` for *LOW* and `1` for *HIGH*. The 4 least significant bits of such byte and subsequent ones describe the ADC value for the corresponding analog input ports *A1* to *A6*. The analog channels can be sampled with a 6 or 10-bit resolution, depending on the number of channels selected for acquisition: if all 6 channels are sampled then the first 4 arrive with 10-bit resolution (ranging from 0 to 1023) while the last 2 arrive with 6-bit (ranging from 0 to 63); however, if 5 channels are acquired then only the last one will have 6-bit and when 4 or less channels are selected then all are sampled at 10-bit resolution.

With the incorporation of the external ADC in μBio, this packet structure was altered to include the ADC value of the external analog channels *A7* and *A8*, should the case be that they were selected for acquisition. For each chosen external analog channel, 3 bytes are added to the packet structure, as those channels are always acquired at 24-bit resolution (ranging from 0 to $2^{23} - 1 = 8388607$). As such, if the user opts for sampling the external ADC only, then the overall packet size remains the same (8 bytes) in comparison with

sampling all internal analog channels only, with the first 6 bytes now containing the ADC values of *A7* and *A8*, as seen in Figure 5b.

The maximum possible packet size is achieved

3) Configuration Commands: While in idle mode, the BITalino (r)evolution firmware is also prepared to receive a set of four device configuration commands, as shown in Figure 6.

The first configuration command allows the user to set the sampling rate and with BITalino there are four choices available - 1 Hz, 10 Hz, 100 Hz and 1000 Hz - meaning that 2 bits ($\log_2 4$) are needed to encode the four possible options. It is by setting the command's two most significant bits, labeled `F`, that the sampling rate is selected. However, with μBio there are seven sampling rates available - 1 Hz, 10 Hz, 100 Hz, 1000 Hz, 2000 Hz, 4000 Hz and 8000 Hz - which requires an extra bit ($\lceil \log_2 7 \rceil$) making the command to set the sampling rate of the BITalino's interface incompatible with μBio. In light of this, we adapted this command so that the third most significant bit is also part of those that are used to encode the sampling rate, as seen in Figure 7. The corresponding encoding of the seven usable sampling rates is shown in Figure 8.

The second configuration command is for controlling the low battery LED according to the voltage at the *ABAT* analog input pin (e.g., the user can connect this pin to the power line that contains the voltage of the battery and thus monitor its level). The battery level threshold can be set to any 6-bit value, hence from 0 to 63, in which values closer to 0 trigger the low battery indication for battery voltages bellow 3.4 V, and values closer to 63 trigger the low battery indication for battery voltages bellow 3.8 V, this for generic rechargeable Lithium Polymer (LiPo) batteries with a nominal voltage of 3.7 V[3].

The last two commands from these configuration ones allow the user to query the device on two different parameters: its firmware version and its status. For compatibility and tracking purposes, it can be useful to know the firmware version running on the device, to which the latter responds to this command with the string "`microbio_v0.1\n`". As for the status command, it allows the user the retrieve the state of the device without having it in live or simulated mode, by retrieving a status packet (Figure 9) which is similar in structure to the packet that is sent while the data is acquired, thus including the state of the digital inputs and output pins, the battery threshold, the ADC value at the *ABAT* analog channel and the ADC value for the remaining analog input ports[4].

4) Action Commands: At last, by default BITalino also includes a set of action commands.

set the Pulse Width Modulation (PWM) output duty cycle

[3]At the moment, in what concerns portable power supplies, μBio can not operate with LiPo batteries but only from microUSB-powered power-banks.

[4]At the moment, the status packet sent by the μBio firmware does not return the ADC value for the external analog channels *A7* and *A8* from the ADS1292R chip.

IV. Experimental Results

To evaluate the performance of μBio we performed two different tests under distinct scenarios: assess the loss of samples for different configurations of the number of channels to acquire and sampling rate, as this makes the packet size and the rate at which data is sent vary, respectively; and do a signal distortion test against a normal setup (streaming over Bluetooth using BITalino).

A. Sample Loss

For this test, whose results are presented in Table I, the evaluated metrics were the total number of CRC fails and the total estimated number of lost packets associated with those fail (based on analyzing the sample sequence numbers), for continuous acquisitions with a total duration of 8 h.

B. Transmission Delay

C. Signal Distortion

At last, we tested μBio in a common usage scenario, that is, record physiological signals. To do an experimental comparison, we did a simultaneous recording, with optical synchronization, of an Electrocardiography (ECG) signal sampled at 1 kHz while it was being acquired by μBio and also by a BITalino with the Bluetooth module streaming all data to a computer. In addition to evaluating the performance of μBio while acquiring physiological data, this test also allows us to compare the quality of the acquired signals when they are sampled at different resolutions. With μBio the ECG signal was acquired at a 24-bit resolution (by using the external ADC) and with BITalino it was sampled at a 10-bit resolution.

We based our analysis on the study conducted in [5], in which the suggested comparison metrics to quantify the similarity between acquisitions are the Root Mean Squared Error (RMSE) (Equation 1) and the coefficient of determination R^2 (Equation 2, where μ corresponds to the data mean).

$$RMSE(x,y) = \sqrt{\frac{\sum_{k=1}^{n} \left(x\left[k\right] - y\left[k\right]\right)^2}{n}} \tag{1}$$

$$SS_{xx} = \sum_{k=1}^{n} x\left[k\right]^2 - n\mu_x^2 \tag{2a}$$

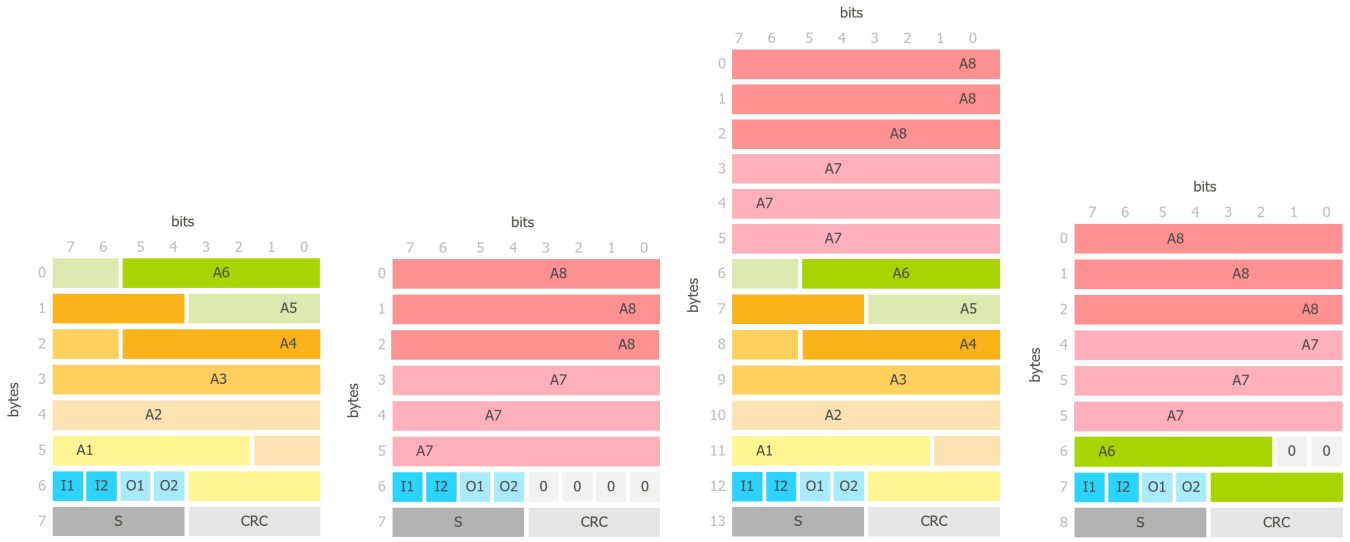
$$SS_{yy} = \sum_{k=1}^{n} y\left[k\right]^2 - n\mu_y^2 \tag{2b}$$

$$SS_{xy} = \sum_{k=1}^{n} x\left[k\right] y\left[k\right] - n\mu_x\mu_y \tag{2c}$$

$$R^2 = \frac{SS_{xy}^2}{SS_{xx}SS_{yy}} \tag{2d}$$

V. Conclusions

The conclusion goes here.



(a) All internal analog channels (b) All external analog channels (c) All internal and external analog channels (d) One internal and all external analog channels

Fig. 5: Data packet structure sent while the device is in live or simulated modes, according to the number of channels selected for acquisition.

TABLE I: Performance of the μBio while logging data for 8 h in live mode.

		Analog Channels / # of Channels / # of Bytes			
		Internal	External	Both (Internal + External)	
	Sampling rate [Hz]	1 CH 3 B	1 CH 5 B	6 CH (4+2) 13 B	8 CH (6+2) 14 B
# of CRC fails	1000	-	-	-	0
	2000	-	-	0	3.5 min
	4000	0	0		
# of lost packets	1000	-	-	-	0
	2000	-	-	0	3.5 min
	4000	0	0		

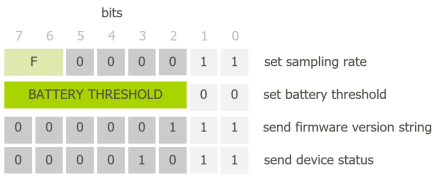


Fig. 6: Configuration commands for the BITalino (r)evolution firmware.

Fig. 7: Command for setting the sampling rate on the μBio firmware.

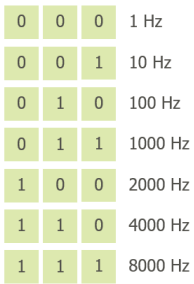


Fig. 8: Encoding of the sampling rate for the μBio firmware.

ACKNOWLEDGMENT

This work is funded by FCT/MEC through national funds and when applicable co-funded by FEDER PT2020 partnership agreement under the project UID/EEA/50008/2013 “SmartHeart”.

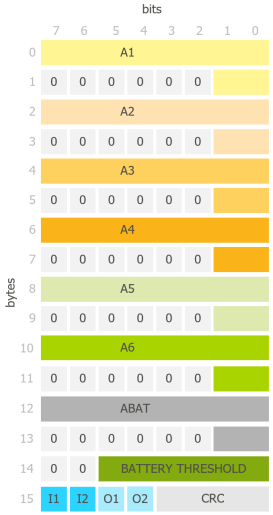


Fig. 9: Status packet structure sent in response to a device status inquiry command on the BITalino (r)evolution and μBio firmwares.

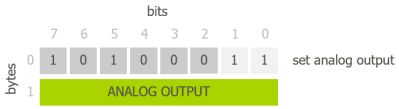


Fig. 10: Action command to set the state of the analog output for the BITalino (r)evolution and μBio firmwares.

Fig. 11: Action command to trigger the digital outputs for the BITalino (r)evolution and μBio firmwares.

REFERENCES

[1] H. Silva, A. Lourenço, A. Fred, and R. Martins, "BIT: Biosignal Igniter Toolkit," *Computer Methods and Programs in Biomedicine*, vol. 115, pp. 20–32, 2014.

[2] Emmanuel Flety, "A comprehensive guide to using, programming & flashing the BITtalino R-IoT WiFi sensor module," http://bitalino.com/docs/R-IoT_User_Guide.pdf, 2017.

[3] Texas Instruments, "Low-Power, 2-Channel, 24-Bit Analog Front-End for Biopotential Measurements," http://www.ti.com/lit/ds/symlink/ads1292.pdf, 2012.

[4] PLUX, "Microcontroller Unit (MCU) Block Data Sheet," http://bitalino.com/datasheets/REVOLUTION_MCU_Block_Datasheet.pdf, 2016.

[5] D. Batista, H. Silva, and A. Fred, "Experimental Characterization and Analysis of the BITalino Platforms Against a Reference Device," in *International Conf. of the IEEE Engineering in Medicine and Biology Society - EMBC*, 2017.