Contents

1 Introduction

The measurement of electrical impulses originating in

Electrophysiology is TODO: scrie coaie mai mult

Biopotential measurements are TODO: scrie coaie mai mult

Relevance: TODO: [1] chapter 7 + novel human computer interaction methods: [2], Maris maybe

- medical relevance: - diagnosis - rehab - monitoring - ??? - Novel human-computer interaction methods

TODO: FIND CITATIONS FOR THESE: Various projects tackle the problem of biopotential data acquisition: - openBCI: expensive, low data rate - that one from Mark: TODO: find eet - quanser board: depends on expensive ELVIS, one channel, not open - EXG pills: one channel, not a stand-alone product, but they're quite good for prototyping otherwise - TODO: MORE

1.1 Objectives

MUST: - [ ] Open Hardware and So20ftware - [ ] 8 channels - [ ] Per-channel configuration - [ ] 1000Hz data rate (500Hz bandwidth) - [ ] High CMRR, PSRR (<1LSB after 50Hz filtering) - [ ] High SNR (>40dB EMG; >20dB for ECG) - [ ] Open and extensible interfacing: - [ ] ROS2 - [ ] IIO

SHOULD (unchecked go to future work): - [ ] 16 channels - [ ] Differential / single-ended configurability - [ ] Right Leg Drive - [ ] 1024Hz/2048Hz /channel data rate (1 Hz per FFT bin) - [ ] <1uV noise after filtering - [ ] 1uV precision

Guiding principles: - [ ] Cost - [ ] Usability as an educational tool - [ ] ADI parts - [ ] Modularity

TODO: After finishing the prototype, make a comparison table between the already existing products and mine, especially on these objectives

2 Bibliographic research

(current state of the art + problem justification + possible approaches)

Although the electrical design of biopotential measurement devices has been studied for hundreds of years, tracing back to Luigi Galvani's work in the 18th century, and there exist a plethora of current-day commercial and DIY solutions, very few systems aspire to the ideals of Open Source Hardware and Open Source Software (EMG pill, openBCI, TODO: the one from Mark), and there exist no systems that integrate with ROS2 (ros-neuro is ROS1 and they don't define hardware, just ros infrastructure) or the Linux IIO (Industrial I/O) subsystem.

Of the Open Source projects, none check all our requirements, and as such this project fills in this gap and provides an end-to-end hardware and software solution for measuring multichannel biopotential signals and exposing them to ROS2 and IIO. Adapting this system to work with other software suites should be made as simple as possible thanks to the versatility of IIO, and adapting it to other hardware platforms should also be facilitated by the decoupled nature of the hardware, firmware and software parts.

2.1 Anatomical basis

As part of normal function, all biological cells have a difference of electrical potential between their interior and their exterior, called the *membrane potential*. In the steady state, the *resting membrane potential* of both skeletal muscle cells and that of cardiomyocytes (cardiac muscle cells) is in the $-70 \cdots -90\,\mathrm{mV}$ range [3], [4]

TODO: EMG, skeletal muscles: Sarcomere, sliding filament model

TODO: ECG: ???

2.2 Noise sources

TODO: Thermal noise model for baseline noise TODO: ECG noise models: baseline, mains, EMG TODO: do EMG noise models exist?

2.3 Signal chain gain management

Stage N noise should be >= stage N+1 noise, just barely passing it, to get the most out of all stages. I should really find a citation for this, because I just know it from Mark. We can use this relationship, some noise density models, and the guaranteed datasheet figures of the components used in the signal chain to compute the ideal amplifier stage gain.

3 Analysis, design and implementation

TODO: Signal chain figure: 1. Patient, anatomy 2. Electrodes 3. Buffer and amplification 4. ADC 5. Microcontroller, using the no-OS framework. Contains customizable digital filters 6. IIO protocol 7. Linux IIO subsystem 8. libiio, pyadi-iio variant A, with a network-enabled microcontroller: from 5, ROS2 protocol find the name! to ROS2 network running on the microcontroller variant B, with a microcontrollwe without networking capabilities: from 8, ROS2 protocol to ROS2 network running on the carrier board / host

3.1 Electical part

3.1.1 Interfacing with the patient

The first step we control in the signal chain is the point of contact with the patient. The electrode type and quality have a great influence on the overall design, performance, and subjective user experience.

Leading research: PEDOT:PSS cite, cite, cite!. Flexible, self-healing polymer, conforms to skin ensuring very good contact even when in motion,

DIY cheap alternative cite that paper with DIY AgCl electrodes

This project settles for clip-on adhesive ECG gel electrodes because: - Cheap, readily avaialble, standard - They can be interfaced with inexpensive aligator clips, besides special (but more expensive) snap clips Downsides: - Shelf life TODO: compare expired electrode signal quality with fresh electrodes - Large-ish contact area - Adhesive gives for a bad subjective experience author's anecdote, or citeable? you zetetic asshole...

3.1.2 ADC

Initially chose the AD7124-8 thanks to its very diverse feature set, including: programmable gain amplifiers (which had the potential to remove the need for the external amplification stage), high precision, high number of channels, and high sample rate. Unfortunately, it is not a single cycle ADC. This means that although it has very good throughput on a single channel, after switching between two channels there is a dead time for the integrated filter to settle, which effectively reduces the per-channel sample rate to less than 100Hz when sampling all 16, which is basically useless for ECG and EMG cite: most of the signal's useful information is in the 300Hz ballpark

The next choice is the AD4114. It doesn't include PGAs, but otherwise has very good specs and has a mugh higher sample rate, and a single cycle conversion mode. stai, sigur?

TODO spec comparison between AD7124-8 and AD4114.

3.1.3 Amplification and buffer stage

Ask whether they're needed. To figure this out, we'll check if we get better measurement noise with an amplification stage than without, because in preliminary tests, we got decent signals without any amplifier. Is "decent" as well as we can get it, or can we have *great* signal quality with an amp stage? Should be able to do these computations quite easily.

3.1.4 Microcontroller

The chosen microcontroller platform is the: Inca nu am ales...... - ADARPblabla max arduino - MAX78000FTHR Both are overkill for the needs of this project.

Main requirements for the microcontroller: - SPI rate high enough to read the ADC at the wanted sample rate - Enough CPU freq and memory to run the IIO Daemon. TODO: iiod specs? if they don't exist, I could do a small experiment for this paper

Nice-to-haves: - Networking capabilities so it can run a ROS2 node as a standalone device, not needing another "carrier" to forward the data to the ROS network.

Suitable very cheap microcontrollers: - ATMEGA328p series TEST? - ESP8266, ESP32 - RP Pico wifi and clones (groundstudio pico) I sure bloody hope no-OS has support for them......... TODO: check

3.2 Firmware part

TODO: Short 100-200 word intro about firmware

TODO: What is No-OS and why do weuse it

3.2.1 Serial communication with host

In order to send data to the attached computer or carrier board, the de facto solution is using UART-over-USB, which exposes an USB interface to the attached host and UART to the microcontroller. Most commonly, microcontrollers have dedicated circuitry for UART transmit and receive, which allows the rest of the program to run concurrently to serial communication. This is implemented using two FIFO buffers, one for data pending to be sent from the microcontroller (TX FIFO) and one for received data ready to be processed by the program (RX FIFO).

De facto, most microcontroller development boards, including the MAX78000FTHR, MAX-Arduino, and the more common ATMEGA368P "Arduino-like" boards provide UART-USB translation via a dedicated IC, abstracting away the complexities of the USB communication stack.

UART modules have varying degrees of configurability of the baud rate, flow control, and framing. For most practical uses, though, including this project, one may just specify the baud rate, defaulting to 8N1 (8 data bits, no parity bit, 1 start bit, 1 stop bit) framing and no flow control. Typical baud rates include 9600 baud, a legacy of 20th century POTS modems (introduced by the V.32 ITU-T standard) and 115200 baud, common both in last-century modem systems, but also recently popularized by its default use in the Arduino development environment.

To achieve the required bandwidth for this project, a suitable baud rate must be chosen. Given the sample rate f_s, number of channels N, and number of bytes per channel sample d, and 8N1 framing (giving 10 baud/byte), the baud rate f_{baud} must be:

$$f_{baud} \geq f_s \cdot N \cdot d \cdot 10 = 1024 \cdot 16 \cdot 3 \cdot 10 = 491520 \text{ baud} \tag{3.1}$$

Additionally, choosing a baud rate larger than this minimum will allow for less timing concerns, especially when it comes to not overflowing the receive and transmit buffers.

Baud rate selection is not universal across devices and the firmware for each hardware platform must choose a rate that is both compatible with the hardware and that satisfies relation 3.1. In the case of the MAX78000FTHR, a rate of exactly 491520 baud can be achievend by using the IBRO (Internal Baud Rate Oscillator) of 7.3728MHz with a clock divider of 15, but [we] will use a divider of 9 for a "round" value of 819200 baud to have a good margin of error and not need such strict synchronization between the UART module and the rest of the program.

3.2.2 SPI Communication basic proof of concept (2024-06 single channel demo)

For initial validation of the concept on the AD4114, I wrote a basic TODO: write about the PoC

3.2.3 No-OS Driver

For easy portability to different microcontroller platforms and even other ADCs, I'm using the No-OS framework, which facilitates separating the general logic from product-specific code and settings.

3.2.4 IIOd integration

Defining the structure of the IIO context is a very important step as it solidifies a mapping between the device-specific functionality on the hardware side and the application-specific functionality on the software side. Choosing a versatile collection of attributes to configure the device and channels to structure the data is no trivial task with no unique solution. The exposed attributes may encode very low level details, as in find an AD part with a very bare-metal attribute mapping, or more high level ones. Choosing which configuration is to be specified on the software side and which configuration is to be derived from that on the firmware side of pula mea ce scriu aici... vreau sa zic ca nu o sa expun direct registri, ci o sa expun chestii mai high level: frecventa de mains hum (50/60), parametri de filtrare, poate sample rate daca o sa imi dau variante (posibil tho sa fie fix la 1k), si pe baza lor se ocupa firmware sa puna registrii cum trebuie.

For lower-level functionality, an IIO debug interface is also specified, which allows for direct manipulation of the ADC's registers. This shouldn't be normally used, but allows for total control of the device in case the user needs it.

1. Defining the desired iio context structure: what do we expose, in which formats, etc 2. Handler functions 3. Boom done, connect to it from software

3.2.5 Continuous and asynchronous data acquisition using IIO triggers

By default, the expected behaviour of IIO devices is to start acquisition on selected channels when requested (via an `OPEN` command), and stop and disable the channels once the requested number of samples is reached. In low sample rate scenarios, the host may request the next acquisition window very soon after the current one finishes, and assuming the time delay between windows can be minimized with respect to the overall window size, this could prevent missing any samples and only introduce a small amount of phase drift. Additionally, there are high-speed applications in which continuity between acquisition windows is not strictly required. On the other hand, with ourroughly 16 kHz sample rate (cca. $60 - 62$ ⊠s/sample) and requirement to not miss any samples, this application has very little room for error in timing. As per [5], modern real-time linux setups have scheduler latencies in the tens and hundreds of microseconds, which realistically leaves no time to actually receive the current window's buffer and send a new acquisition request without missing any samples, even if the acquisition channels were left enabled.

TODO: figure, vertical stack of sequence diagrams of: normal iio polling with phase drift in slow ADC, fast ADC - loss of samples, trigger and buffer

With this general issue in mind, IIO defines triggers as software or hardware events that initiate data acquisition without requiring regular polling from the host. Data captured as a result of triggers is stored in circular buffers and provided with minimal latency whenever the host polls next.

With the AD4114 in continuous conversion mode, two trigger sources are apparent: using the $\mathtt{DOUT}/\overline{\mathtt{RDY}}$ behaviour in continuous conversion mode (as described in ??) or using a timer to poll the AD4114 at least as fast as the conversion rate. While the first option is more elegant and wouldn't do any unnecessary polling, it requires nonstandard operation of the SPI protocol by keeping CS active and monitoring the MISO ($\mathtt{DOUT}/\overline{\mathtt{RDY}}$) line, which isn't generally possible unless webitbang the SPI protocol or have access to more configurable IO options like on FPGA platforms or the Programmable Input and Output subsystem of RP2040 fame ([6]). The second option of setting up a timer interrupt which polls the AD4114 status register to check if a new conversion result is available

TODO: scrie coaie mai mult

3.2.6 ROS2 on network-capable microcontrollers

In recent years, there has been a wave of developments in fast, network-capable, cheap microcontroller platforms, leading to products such as the Espressif ESP8266 and ESP32, or the Raspberry Pi Pico-W, which are capable of running ROS2 nodes all by themselves.

TODO: scrie coaie mai mult

3.2.7 Digital filtering of the signal

Although the digital signal processing part of such biopotential measurement systems is very important for a full product, it is not the main topic of this thesis, and yet the chosen system architecture offloads most of the signal conditioning to the digital domain. The main source of noise is mains hum, with a fundamental frequency of 50 Hz and various harmonics, which can all be removed using a notching filter, resulting in minimal loss of information in other portions of the frequency spectrum. The Qaulity factor Q of the filter controls how narrow the attenuated frequency bands are, and it can be adjusted. It poses a tradeoff between good noise rejection and signal quality, and is designed to be adjustable. Experimentally, we found values of around 5-10 to give the best waveforms for visual inspection. Figure X illustrates the tradeoff between small Q doing ??? and high Q doing ???.

Taking from scipy's implementation of an IIR Notching filter ('scipy.signal.iircomb'), I also use the structure described in cite: Sophocles J Orfanidis, "Introduction to Signal Processing", Prentice-Hall, 1996, ch. 11, "Digital Filter Design" thanks to its adjustable Q factor, numerical stability even at higher orders, and default rejection of the DC component of the signal, which are all very useful for the required signal conditioning. Parametrized by the the sampling frequency f_s, the cutoff freqiency f_0 (which must be an integer divisor of f_s), quality factor Q, the IIR filter $H(z)$ is computed as:

$$
\begin{aligned}
N &= f_s/\omega_0 \overset{!}{\in} \mathbb{N} \\
\omega_0' &= \frac{2\pi\omega_0}{f_s} \\
\omega_\Delta &= \frac{\omega_0'}{Q} \\
\beta &= \tan(N * \omega_\Delta/4) \\
a &= \frac{1-\beta}{1+\beta}
\end{aligned}
$$

$$
H(z) = \frac{1}{1+\beta} \cdot \frac{1-z^{-N}}{1-az^{-N}} \tag{3.2}
$$

TODO: check I transcribed these correctly and actually implement them myself! I only blindly copied these for now

3.3 Software part

3.3.1 ROS2 agent for non network-capable microcontrollers

In the case of microcontroller platforms that aren't network capable, such as the ATMEGA328P family popularized by Arduino, the MAX78000FTHR, the device itself won't have a means of directly connecting to the ROS2 network as a node. As such, it needs to pass along the data to another device that can. Two implementation variants are immediately evident: - Using the micro-ROS framework, letting ros logic take over the serial device instead of having it communicate using the IIO protocol, and running a ROS-micro agent on the host - Using the ros-control framework and an IIO-ROS2 adapter on the host / carrier board.

Of which I chose the second because it doesn't require reconfiguring the serial port and allows for simultaneous use of ROS and IIO.

TODO: implement :)

3.3.2 libiio, pyadi-iio

ROS is for robotics, what about just doing some basic signal processing in python without setting it up as a ros node? Libiio has got you covered! Pyadi-iio adds a cherry on top, exposing the entire system as a very simple to initialize and query python object.

TODO: implement :)

4 Results and validation

To test the performance of the system:

- ECG is more difficult and as such a greater achievement. It has smaller amplitudes and very well-defined features, requiring more care all across the signal chain. - EMG is the main goal of the project, but it is more forgiving, because: - The signal has greater amplitude (about 10x? check both sources and experimental data!!!) - The signal features are not as well defined and are usually modeled like noise for signal processing.

4.1 Electrical validation

Using a waveform generator, we can compare this device's measurements with both: - Precision oscilloscope readings - Another biopotential DAQ system (Tassos')

TODO: scrie coaie mai mult :)

4.2 Usage for EMG

TODO: scrie coaie mai mult :)

4.3 Usage for ECG

TODO: scrie coaie mai mult :)

4.4 Usage for EEG

TODO: maybe try out a quick and dirty N100 demo? orthogonal PRBS activation of multiple visible targets, correlate EEG (single ended frontal, occipital) and determine which target the user is focusing on

TODO: scrie coaie mai mult :)

5 Conclusion

5.1 Future work

6 Acknowledgements

- Mark Thoren

7 Bibliography

[1] M. Barbero, R. Merletti, and A. Rainoldi, *Atlas of Muscle Innervation Zones.* Springer Milan, 2012, ISBN: 9788847024632. DOI: 10.1007/978-88-470-2463-2. [Online]. Available: http://dx.doi.org/10.1007/978-88-470-2463-2.

[2] P. N. Müller, P. Achenbach, A. M. Kleebe, *et al.*, "Flex your muscles: Emg-based serious game controls," in *Lecture Notes in Computer Science.* Springer International Publishing, 2020, pp. 230–242, ISBN: 9783030618148. DOI: 10.1007/978-3-030-61814-8_18. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-61814-8_18.

[3] P. M. Hopkins, "Skeletal muscle physiology," *Continuing Education in Anaesthesia Critical Care & Pain*, vol. 6, no. 1, pp. 1–6, Feb. 2006. DOI: 10.1093/bjaceaccp/mki062. [Online]. Available: https://doi.org/10.1093/bjaceaccp/mki062.

[4] R. E. Klabunde, *Cardiovascular Physiology Concepts*, 2nd ed. Philadelphia, PA: Lippincott Williams and Wilkins, Sep. 2011.

[5] D. B. de Oliveira, D. Casini, R. S. de Oliveira, and T. Cucinotta, "Demystifying the real-time linux scheduling latency," en, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. DOI: 10.4230/LIPICS.ECRTS.2020.9. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2020.9.

[6] S. Smith, "How to initialize and interact with programmable i/o," in *RP2040 Assembly Language Programming.* Apress, Oct. 2021, pp. 177–200, ISBN: 9781484277539. DOI: 10.1007/978-1-4842-7753-9_10. [Online]. Available: http://dx.doi.org/10.1007/978-1-4842-7753-9_10.