Improving the performance of analog acquisition in low-power low-range microcontrollers

Diego Belzarena, Giannina Marrero, María Sofía Rijo and Julián Oreggioni
Universidad de la República. Montevideo, Uruguay
Email: {diego.belzarena, giannina.marrero, maria.sofia.rijo, juliano}@fing.edu.uy

***Abstract*—Low-power low-range microcontrollers provide the means to acquire analog signals. This work focuses on the development of a real-time embedded system that acquires and processes analog signals using the MSP430G2553 microcontroller, which could be valuable for the recording of biopotentials. We aim to push this system to its limits in terms of throughput, identifying the bottlenecks that restrict the maximum sample frequency.**

In short, our system is capable of acquiring, processing, and transmitting using UART communication eight analog channels at an effective throughput of 729 kbps. This throughput is achieved by improving the performance of each of the system's building blocks and implementing a simple yet effective data packing scheme. In addition, we propose another packing scheme that offers the means to detect the loss of information and synchronize the data acquisition while slightly reducing the effective throughput (625 kbps).

I. Introduction

Low-power low-range microcontrollers (CPU of 8 and 16 bits and up to tens of MHz, 10-bit or 12-bit analog-to-digital converters, etc.), such as Microchip's ATmega328PB or Texas Instruments' MSP430, specify the availability of multiple analog channels (up to eight in the mentioned microcontrollers) and high sampling frequencies for their analog-to-digital converters (ADCs), for example, 80 ksps (kilo samples per second) in the case of the ATmega328PB, or 200 ksps in the MSP430 family.

Operating analog acquisition in low-power low-range microcontrollers at its maximum speed is fundamental in a broad range of applications in the manufacturing industry [1] (4 channels, 10 ksps, 16 bit, 640 kbps), physiological monitoring [2] (2 channels, 1 ksps, 10 bits, 20 kbps), ultrasound [3] (8 channels, 6 Hz, 12 bits, 0,6 kbps), among others.

This work is a continuation of [4] (1 channel, 35,6 ksps, 10 bits, 356 kbps). [4] performed a first approach to identify bottlenecks and explore the feasibility of harnessing the aforementioned theoretical maximum sampling rates. In this work, we introduce the prototype of an embedded system based on the MSP430G2553 microcontroller (see Fig. 1). Our system acquires up to eight analog signals, processes them (data packing), and transmits them to a personal computer (PC) via serial communication, aiming to operate at its maximum capacity. Additionally, we implemented a graphical user interface (GUI) in Python on the PC, allowing data reception and visualization.

This work was partially funded by CSIC (Comisión Sectorial de Investigación Científica, Udelar, Uruguay), and Erasmus+ Project NEON, 618942-EPP-1-2020-1-AT-EPPKA2-CBHE-JP.

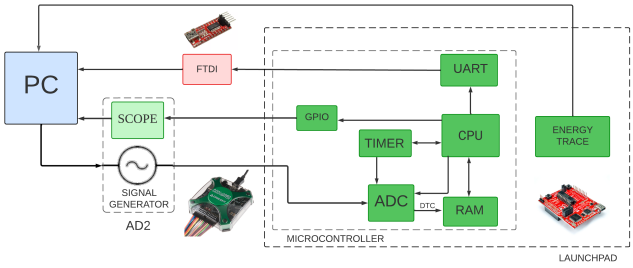


Fig. 1: Schematic diagram of the system.

II. System description

In Fig. 1 a schematic diagram of the system is presented. The analog signals acquired by the microcontroller are generated using an Analog Discovery 2 (AD2) and subsequently sent to a PC using the UART protocol. Furthermore, the scope of the AD2 is connected to the UART transmission pin (TXD) to analyze the processed signals and measure response times.

A. Hardware

We use the MSP430G2553 microcontroller from Texas Instruments, which is included in the MSP-EXP430G2ET Launchpad, and it is powered by 3,3 V. The microcontroller has a 16-bit RISC architecture with a CPU speed of up to 16 MHz. It features 512 bytes of RAM and 16 kB of flash memory. It offers three basic clock configurations, two 16-bit timers, a 10-bit A/D converter with 8 channels (ADC10), and a Universal Serial Communication Interface (USCI) with a maximum UART communication rate of 2 Mbps. Additionally, it includes the Data Transfer Controller (DTC) peripheral, which allows the transfer of data acquired by the ADC to RAM without involving the CPU.

The Launchpad includes a programmer, a debugger, a UART terminal, and a system for measuring energy consumption (Energy Trace). The Launchpad's UART data transfer rate is limited to 115,2 kbps (see Table 4 on page 19 of [5]) but the use of an external UART/USB adapter (FTDI) allows a maximum frequency of 3 Mbps.

The ADC10 peripheral has four operating modes: single-channel (single conversion), sequence of channels, single-channel repetition, and sequence of channels repetition. To acquire data from all 8 channels (A_7-A_0), we have chosen to operate the ADC10 in the sequence of channels mode. The sequence of channels mode facilitates the acquisition

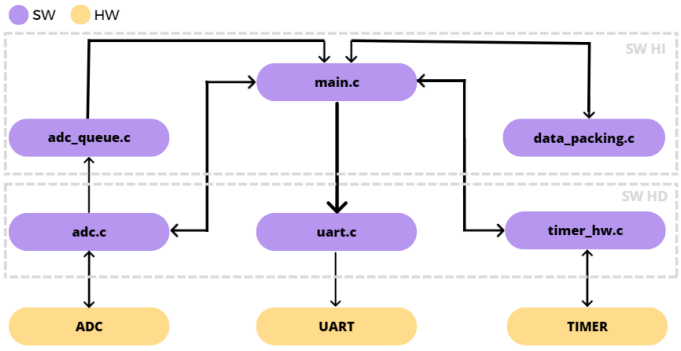


Fig. 2: Embedded software module diagram.

of multiple channels without the need for manual channel switching, as is required in the single-channel mode (single conversion). In this way, the TIMER triggers a new series of 8 acquisitions each time it interrupts.

B. Embedded software architecture

The development of this system uses C as the programming language, and Texas Instruments Code Composer Studio (CCS) as the development, programming, and debugging tool. It employs a Round Robin embedded software architecture with interrupts [6]. The use of interrupts allows us to "put the CPU to sleep" when there are no pending tasks, effectively reducing power consumption while ensuring a fast response to critical events, such as the completion of data acquisition.

C. System modules

The system implementation is divided into the modules shown in Fig. 2. Firstly, we have a hardware-independent software layer (SW HI). On the other hand, we identify a software layer that controls the hardware being used, meaning it facilitates the management of peripherals such as TIMER, ADC, and UART (SW HD). The operation of each module is detailed below.

timer_hw.c contains the function config_Timer, in which the timer is initialized along with its corresponding interrupt service routine. During initialization, the SMCLK clock source is selected, and it's configured in up-mode. This means that the timer counts from zero up to the value specified in the TACCR0 register. The interrupt service routine (ISR), isr_timer, is responsible for enabling the ADC10 conversion, automatically triggering the acquisition. These interrupts occur periodically and determine the sampling frequency.

In **adc_queue.c**, functions for managing the queue where data acquired by the ADC will be stored are defined. A queue with the capacity to store 48 samples (6 acquisitions of 8 channels) is utilized.

The module **uart.c** has an initialization function where the port for transmission is configured, and it includes functions for managing the UART transmission queue.

The module **data_packing.c** contains functions that allow data to be added to the UART transmission queue in a packed format (see Section IV).

The **adc.c** module contains the initialization function for ADC10 and its interrupt service routine. The ADC references are set to VR_+ = 1,5 V and VR_- = 0 V. Additionally, the sample-and-hold input is set to be stable for 4 ADC10CLKs (the faster option). The MSC (multiple sample and convert) bit is set to 1 to perform successive conversions automatically and as fast as possible (no SAMPCON trigger is used to initiate acquisition). We configure the ADC clock with the fastest clock source available (see Section III-A). Furthermore, the ADC is configured to operate in the sequence of channels mode (see Section II-A). Whenever a conversion is completed and the result is loaded into the ADC10MEM register, the DTC is enabled and carries out the data transfer to RAM. In this case, the DTC has been configured in continuous mode, and the block size has been set to 48 samples. This means that 48 consecutive conversions will be transferred before an interrupt is generated. When this occurs, the ADC ISR (isr_adc) is responsible for setting the ADC_flag, which is checked in the main routine.

The main module of the system, **main.c**, is the one that implements the Round Robin architecture with interrupts. First, the TIMER, ADC, and UART peripherals are initialized. Then, it enters the main loop where it checks the ADC_flag. If the flag is set, data is retrieved, packed and added to the UART transmission queue. Once the data is acquired, the ADC flag is cleared. The low power mode LPM3 is activated when the ADC_flag is off and will end when the ADC ISR is executed.

III. Testing and Measurements

A. DCO Oscillator

To achieve the highest possible sampling frequency, it is necessary for both the CPU clock and the ADC10 clock to operate at their maximum speed. The CPU clock (MCLK) is limited to a maximum speed of 16 MHz.

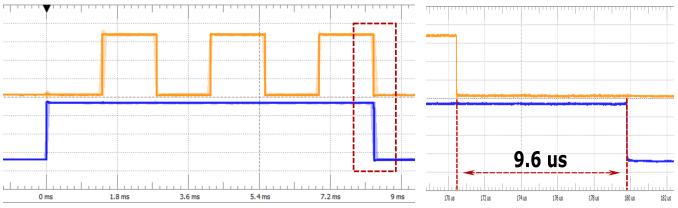
These clocks are sourced by the Digitally-Controlled Oscillator (DCO) as it is capable of providing the highest frequencies. It is observed in MCU datasheet (page 29 in [7]) that for the highest frequency ranges, there isn't a typical value, and the range of variation is very large. This is why the DCO was configured to the maximum possible value below 16 MHz, and its value was experimentally verified to be 15,6 MHz in our test-bench.

B. Maximum UART Transmission Speed

To determine the limit of the UART transmission speed, we conducted the following test: i) the microcontroller transmits 10.000 characters to the PC via UART; ii) we evaluate how many of them are correctly received in a terminal on a PC. It was possible to receive all 10.000 characters at a speed of 1,54 Mbps. Additionally, it was observed that when attempting to transmit at a speed of 1,56 Mbps, virtually none of the transmitted characters were received.

C. Time Analysis

1) Acquisition time: to measure the acquisition time, two of the microcontroller's pins are set as output. One of them



(a) (b)

Fig. 3: Acquistion time with ADC10CLK = DCO = 15,6 MHz. The orange signal switches every time the TIMER interrupts while the blue one does so every time the ADC interrupts. (b) shows the same signals as (a) but zoomed-in on an ADC interruption.

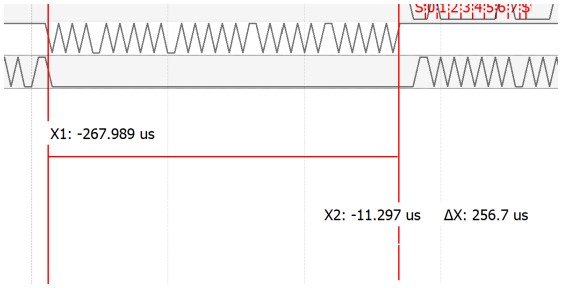


Fig. 4: Processing time with MCLK = DCO = 15,6 MHz.

switches every time the TIMER interrupts (orange signal in Fig. 3, signaling that it is time to acquire a new set of 8 channels), while the other one switches every time the ADC interrupts (blue signal in Fig. 3, signaling that the digitized block of 48 samples has already been stored in memory). Afterwards, the distance between the falling edges of the signals is measured with an oscilloscope, which resulted in an acquisition time of 9,6 μs for the 48 samples, as seen in Fig. 3b. The time it takes for the outputs to switch, once the TIMER interruption occurs, is negligible.

2) Processing time: as a baseline reference for the processing time, we measured the time required to copy the data from where it was stored in RAM by the DTC to the UART transmission queue. This baseline time, with a DCO frequency of 15,6 MHz, is 256,7 μs, equivalent to approximately 4.000 MCLK clock cycles (see Fig. 4). Any additional processing performed on the data will contribute to increase the actual processing time.

3) UART transmission: an experimental setup was designed to measure the time it takes for our system to transmit the 48 samples (6 acquisitions for each channel). Depending on the configured baud rate, different transmission times are obtained (see Table I). Fig. 5 shows the acquired signal for a baud rate of 115.200 bps.

TABLE I: Transmission time for different UART speeds.

Baud rate (kbps)	T_{TX} (ms)
115,2	8,28
256	3,68
921	1,02
1.540	0,62

Table I shows that for the lower baud-rates, the UART

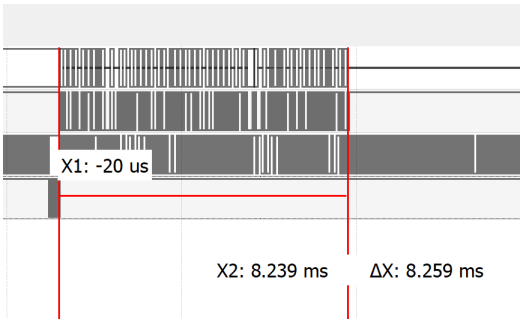


Fig. 5: Data transmission time at 115.200 bps.

transmission times are of a higher order than the times it takes to acquire and copy the data to the transmission queue (256,7 μs). On the other hand, as the speed increases, even though UART transmission times are still higher, they become comparable. This shows that the system's bottleneck is the UART transmission.

For the case with the highest transmission rate (1,54 MHz), using the DCO at 15,6 MHz, our system takes a total time of T_{total} = 9,6 μs + 256,7 μs + 624 μs = 890,3 μs to acquire and transmit a total of 48 samples. This equates to a per-sample time of 18,5 μs. Similarly, this calculation can be performed for the other baud-rates (see Table II) to arrive at the minimum time values required for our system to function correctly (or equivalently, the maximum sampling frequencies for each baud-rate). Then, the maximum sampling frequency is 54,1 kHz and it is achieved at the highest transmission rate (1,54 MHz).

TABLE II: Comparison of the maximum sampling frequency (f_S) of the system for different transmission rates.

Baud rate (kbps)	T_{48} (ms)	T_1 (μs)	f_S (kHz)
115,2	8,54	178,0	5,6
256	3,94	82,1	12,2
921	1,29	26,8	37,3
1.540	0,89	18,5	54,1

Note: T_x is the time to acquire and transmit x samples

IV. Data packing

In the application presented so far, digitized signals are acquired and transmitted in 10 bits, occupying 16 bits in memory. This results in a waste of 6 bits per sample. With the aim of improving the efficiency in the information transmission, a data packing scheme is introduced, where multiple samples are combined into a single byte.

A. 7&3 Data packing

Without using data packing, we have an efficiency in memory usage for transmission of 62,5 % (10 bits of data for every 16 bits transmitted). The proposed solution involves packing the data in such a way that 3 samples are transmitted per 2 bytes. This solution improves the data-to-bit transmission ratio up to 83 % (20 bits of data for every 24 bits transmitted).

The proposed packing method is referred to as "7&3 packing" because the samples are divided into sets of 7 and 3 bits

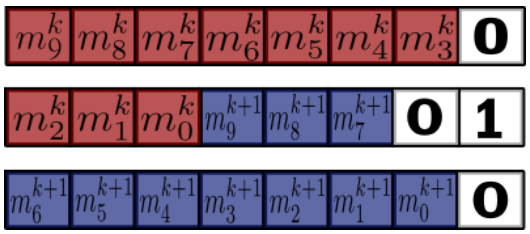


Fig. 6: Distribution of samples with 7&3 packing.

(see Fig. 6). We begin by splitting the first sample into its 7 most significant bits on one side and the remaining 3 bits on the other. Then, the 7 most significant bits are placed in the first transmission byte, and the remaining 3 bits are added in the next transmission byte. This second transmission byte will be shared by the two samples to be transmitted. In the case of the second sample, it is also divided into 7 and 3 bits. However, unlike the previous case, the 3 most significant bits are separated from the 7 remaining bits (it could be said that the division is "3&7" instead of "7&3" in this case). These 3 most significant bits are added to the second transmission byte following the bits from the first sample. Then, the remaining 7 bits are placed in the third transmission byte.

The four unused bits (one in the first transmission byte, two in the second, and one in the third, see Fig. 6) are used for packet loss detection and synchronization. In the case of packet loss, it was chosen to add a specific value in the last bit of each byte. As seen in Fig. 6, a "0" is placed in the first byte, a "1" in the second, and a "0" in the third. In this way, during reception, we can check that this sequence of last bits is always as expected, allowing the detection of both single and double packet losses.

For synchronization, a "0" is placed in the second transmission byte's penultimate bit. This ensures that the byte 0xFF is never transmitted. Therefore, at the beginning of the transmission, a 0xFF can be sent, which when detected at the receiver indicates the start of the sequence. This also allows for periodically sending (every m packets) the 0xFF "flag", and with a counter (counting up to m), detecting if any packets were lost.

B. Data packing time analysis

Data packing affects two of the three previously studied times: processing time and transmission time (see Section III-C). It is expected that the former will increase, while the latter will decrease.

An initial implementation (referred to as *Imp1*) consists of a function that packs a sample according to a flag that is set to 0 if it is the first sample of the block, or 1 if it is the second. Then, it checks which case it is (with an *if statement*) and performs the corresponding actions to pack it. Algorithm 1 displays the corresponding pseudocode. With this implementation, the time it took to process (pack and copy to the transmission queue) 48 samples (T_{proc}) and transmit them (T_{TX}) was measured (see Table III). The processing time is independent of the transmission speed being used and depends solely on the CPU clock speed (MCLK).

Algorithm 1: IMP1 7&3 DATA PACKING SCHEME.

Input: $\mathbf{m} = [m_9, m_8, ...m_1, m_0]$ is a 10-bit ADC sample; and c_flag is a flag that indicates whether $\mathbf{m}$ is the first or second sample of the 7&3 data block.

Result: The algorithm packs and places the input sample in the UART transmission queue in concordance with the 7&3 data packing scheme.

Data: txq_head is the transmission queue head.

if $c\_flag == 0$ **then**
- $txq\_head \leftarrow [m_9^k m_8^k m_7^k m_6^k m_5^k m_4^k m_3^k \ 0]$;
- txq_head ++;
- $txq\_head \leftarrow [m_2^k m_1^k m_0^k \ ----]$

else
- $txq\_head \leftarrow [m_2^k m_1^k m_0^k m_9^{k+1} m_8^{k+1} m_7^{k+1} \ 0 \ 1]$;
- txq_head ++;
- $txq\_head \leftarrow [m_6^{k+1} m_5^{k+1} m_4^{k+1} m_3^{k+1} m_2^{k+1} m_1^{k+1} m_0^{k+1} \ 0]$;
- txq_head ++

end

Without packing, 2 samples are transmitted for every 4 bytes, while with packing, 2 samples are transmitted for every 3 bytes. Therefore, it is expected that the transmission time for the packed data will be 75% of the case without packing. This is confirmed when observing Table III.

The Table III also shows that as the transmission speed increases, the transmission time begins to lose importance in determining the total time (T_{proc} + T_{Tx}), and the processing time, which is fixed, becomes more relevant. Additionally, when packing is applied, the processing time increases, which confirms that at very high transmission speeds, packing adversely affects the total time. For example, in the case of 1,54 MHz, the total time without packing is 0,88 ms, while with packing using *Imp1*, it is 0,94 ms.

Therefore, for data packing to be profitable, it must satisfy the condition: $T_{proc} < (0,25).T_{TX}$, where in this case T_{proc} is time used to pack the data.

In Fig. 7, you can see the frequency at which packing starts to become detrimental ($\approx$ 1,12 MHz in *Imp1*). From the above analysis, it can be concluded that *Imp1* is not suitable for using data packing, as its execution time is very high.

In order to fully leverage the benefits of data packing at the highest possible transmission speed, the implementation *Imp2* is proposed. It serves the same packing function but does not significantly increase the processing time (it increases by approximately 35 µs). This implementation, in broad strokes, involves iterating over 2 data samples at a time and constructing the entire 3-byte block in each iteration (without *if statements* and flags to evaluate the current case). Algorithm 2 displays the corresponding pseudocode.

Furthermore, in Fig. 7, it can be observed that with *Imp2*, packing proves to be beneficial for all transmission speeds

TABLE III: Results of processing and transmission times.

Baud rate (kbps)	Method	Implementation	T_{proc} (μs)	T_{Tx} (μs)	($T_{proc} + T_{Tx}$) (μs)
115,2	Without Packing	-	257	8280	8540
	7&3 Packing	*Imp1*	471	6210	6680
		Imp2	292	6210	6500
	Alternative Packing	-	266	5080	5350
1.540	Without Packing	-	257	624	880
	7&3 Packing	*Imp1*	471	470	940
		Imp2	292	470	760
	Alternative Packing	-	266	380	650

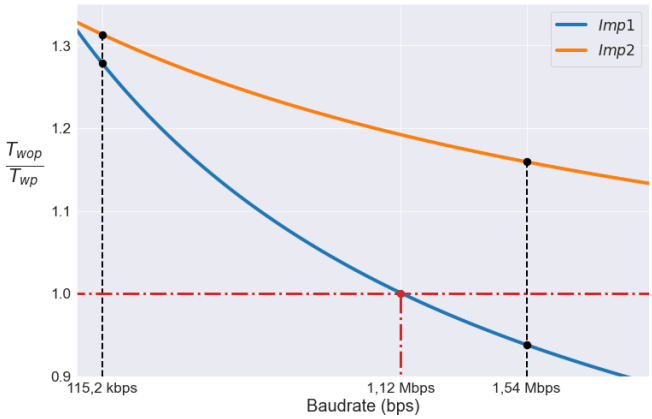


Fig. 7: Impact of 7&3 packing on processing and transmission times. The ratio between the sum of processing and transmission times without packing (T_{wop}) and with packing (T_{wp}) is plotted. Additionally, a comparison is made between the two implemented approaches. The dashed red line indicates where the ratio of times is equal to 1, meaning that in terms of time, packing is equivalent to not doing it.

Algorithm 2: IMP2 7&3 DATA PACKING SCHEME.

Input: Two 10-bit ADC samples,
$\mathbf{m^k} = [m_9^k, m_8^k, ...m_1^k, m_0^k]$ and
$\mathbf{m^{k+1}} = [m_9^{k+1}, m_8^{k+1}, ...m_1^{k+1}, m_0^{k+1}]$

Result: The algorithm builds a 7&3 data block from two input samples and places it in the UART transmission queue.

Data: txq_head is the transmission queue head.

$txq\_head \leftarrow [m_9^k m_8^k m_7^k m_6^k m_5^k m_4^k m_3^k \ 0]$;
txq_head ++;
$txq\_head \leftarrow [m_2^k m_1^k m_0^k m_9^{k+1} m_8^{k+1} m_7^{k+1} \ 0 \ 1]$;
txq_head ++;
$txq\_head \leftarrow [m_6^{k+1} m_5^{k+1} m_4^{k+1} m_3^{k+1} m_2^{k+1} m_1^{k+1} m_0^{k+1} \ 0]$;
txq_head ++

within a reasonable range (the dashed red line would intersect with the orange one at a baud rate of approximately 6,6 MHz).

C. Alternative data packing methods

Other data packing strategies were evaluated and implemented. In particular, one of them involved not leaving any bits unused when packing, which means filling the entire transmission block with data. This allowed sending 4 samples every 5 bytes and, by design, resulted in 100 % data bits per transmitted bit (40 bits of data for every 40 bits transmitted). This packing method has the disadvantage that there are no bits available for controlling and detecting packet losses (as it is the case with the 7&3 packing). Additionally, the packets can take any value (including 0xFF), so the same synchronization strategy as in 7&3 cannot be followed.

Tests were conducted with this packing strategy, similar to those for 7&3, and the results in Table III were obtained. It can be observed that this packing strategy presents a significant reduction in processing time, reaching values close to those of not doing packing, and reduces the transmission time by approximately 20 % with respect to the 7&3 packing scheme. However, despite the significant improvement, it should be kept in mind that there may be cases where the error detection and synchronization capabilities of 7&3 are essential.

V. MAXIMUM SAMPLING FREQUENCY

To determine the maximum sampling frequency, the sum of all the involved times is considered. The maximum sampling frequency with 7&3 packing is f_s = 62,5 kHz (see Table IV). This implies an improvement of $\approx$ 15% compared to not packing, and $\approx$ 76 % compared to [4]. On the other hand, if the alternative packing scheme is used, the maximum sampling frequency is f_s = 72,9 kHz. This implies an improvement of $\approx$ 17 % compared to 7&3 packing, $\approx$ 35% compared to not packing, and $\approx$ 105% compared to [4]. f_s = 72,9 kHz corresponds to the maximum sampling frequency if only one channel is used. If we acquire 8 channels, the maximum sampling frequency of each channel would be f_s/8 = 9,1 kHz.

While the transmission time is greatly reduced, the transmission stage is the one that takes most of the time. Given this, we conclude that the bottleneck lies in the UART transmission:

$$T_{adq} = 9,6\mu s \ll T_{proc} = 266\mu s < T_{Tx} = 380\mu s$$

In order to validate the above presented results, an experiment was carried out where the end to end system was tested (see Fig. 8). This experiment consisted in sampling 8 analog signals, packing the data (with the alternative packing strategy), transmitting the data via UART to a PC, and finally unpacking and visualizing the data in the GUI. All this at a throughput of 729 kbps.

Table V shows the consumption measurements obtained using Energy Trace for the system while acquiring and transmitting data.

TABLE IV: Comparison of the maximum sampling frequency of the system with [4] and with the addition of packing.

	Baud rate (kbps)	T_{48} (ms)	T_1 (μs)	f_s (kHz)	Throughput (kbps)
[4]	921	1,35	28,1	35,6	356
Without packing	1.540	0,89	18,5	54,1	541
7&3 Packing	1.540	0,77	16,0	62,5	625
Alternative Packing	1.540	0,66	13,7	72,9	729

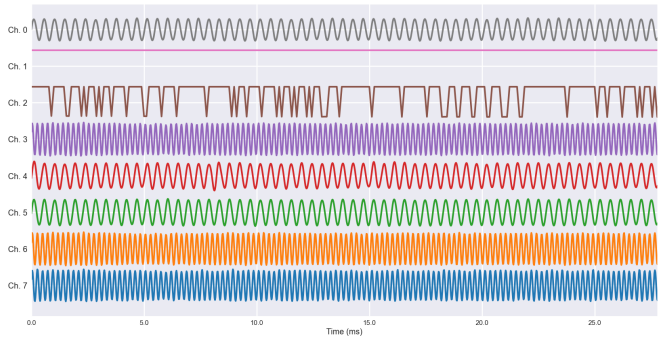


Fig. 8: GUI with an 8 channel end-to-end test. We use f_s = 9,1 kHz and the alternative packing strategy. A 2,2 kHz sine wave was introduced in channels 0, 4 and 5; and a 4,4 kHz ($\approx \frac{f_s}{2}$) in channels 3, 6 and 7. Channels 1 and 2 are multiplexed with the UART ports.

TABLE V: Power consumption measurements.

	Power (mW)	Voltage (V)	Current (mA)
Average	13,84	3,29	4,2
Minimum	4,48		1,36
Maximum	18,58		5,64

VI. Conclusions

We presented a real-time embedded system capable of acquiring, processing, and transmitting eight analog signals at its maximum capacity. We improved the acquisition and processing times by setting the MCLK frequency to the maximum possible value. Additionally, we managed to reach a data transmission speed of up to 1,54 Mbps.

We determined that the bottleneck of the acquisition chain is in the UART transmission, and we estimate that we need a 3,75 Mbps UART transmission speed to shift the bottleneck to the processing stage. In addition, we showed that this bottleneck can occur at 2,47 Mbps if the 7&3 data packing is performed (which includes packet loss detection and synchronization) and at 2,31 Mbps if the alternative packing is used.

The system increased the maximum sampling frequency from the 35,6 kHz reported by [4] to 54,1 kHz by improving the performance of the blocks we were already working with. Furthermore, with the implementation of data packing, we further increased this frequency to 72,9 kHz. Regarding memory, our implementation involves a similar use of RAM as in [4], and no significant differences in consumption were observed.

Finally, Table VI provides a comparative summary of our work with other works. This comparison shows that our data packing scheme generates a very competitive throughput of 729 kbps, the highest amongst those compared.

As a next step, we propose to replace the AD2 with an analog front-end developed in our laboratory based on [8], and carry out tests recording biopotential signals.

TABLE VI: Comparison with other works.

	[4]	[1]	[2]	[3]	TW
MCU: MSP430*	G2553	-	FR2355	FR5043	G2553
ADC bit resolution	10	16	10	12	10
Number of channels	2	4	2	8	8
Max fs (kHz)	35,6	40	2	0,05	72,9
Max throughput (kbps)	356	640	20	0,6	729
Baud rate (kbps)	921,6	-	-	-	1540
Acquisition time (μs)	29,8	-	-	-	9,6
Processing time (μs)	295,1	-	-	-	266

Note: TW = This Work

Acknowledgments

We are thankful to Rosina D' Eboli for her participation in the early stages of this work, and Leonardo Steinfeld for his advice and suggestions.

References

[1] A. Barrancos, R. L. Batalha, and L. S. Rosado, "Towards enhanced eddy current testing array probes scalability for powder bed fusion layer-wise imaging," *Sensors*, vol. 23, no. 5, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/5/2711

[2] S. Wood, D. Chakraborty, and J. Schmalzel, "Low power sensor fusion targeted for ai applications at the edge," in *2023 IEEE Sensors Applications Symposium (SAS)*, 2023, pp. 1–6.

[3] S. Frey, S. Vostrikov, L. Benini, and A. Cossettini, "Wulpus: a wearable ultra low-power ultrasound probe for multi-day monitoring of carotid artery and muscle activity," in *2022 IEEE International Ultrasonics Symposium (IUS)*, 2022, pp. 1–4.

[4] L. Gómez, R. González, M. J. Millán, and J. Oreggioni, "Cuellos de botella en sistemas embebidos en la adquisición y transmision de señales analógicas," in *Congreso Argentino de Sistemas Embebidos (CASE), Bahía Blanca, Argentina*, August 2023.

[5] Texas Instruments, "MSP debuggers user's guide," https://www.ti.com/lit/ug/slau647o/slau647o.pdf, 2015.

[6] D. E. Simon, *An embedded software primer*. Addison-Wesley Professional, 1999, vol. 1.

[7] Texas Instruments, "MSP430G2x53, MSP430G2x13 mixed signal microcontroller datasheet (rev. j)," https://www.ti.com/lit/ds/slas735j/slas735j.pdf, 2011.

[8] J. Oreggioni, A. Caputi, and F. Silveira, "Current-efficient preamplifier architecture for cmrr sensitive neural recording applications," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 3, pp. 689–699, jun 2018.