École polytechnique fédérale de Lausanne
School of Engineering

Master Project - M.S. Electrical Engineering

PATTERN RECOGNITION IN NON-UNIFORMLY SAMPLED ELECTROCARDIOGRAM SIGNAL FOR WEARABLE SENSORS

Author:
Silvio Zanoli

Under the direction of
Dr. Tomás Teijeiro Campo and
Prof. Dr. David Atienza Alonso
In the Embedded Systems Laboratory (ESL)

EPFL

Lausanne, August 28, 2020

Don't try.

Charles Bukowski, 2003,
"So You Want To Be A Writer?"

Acknowledgement

I would like to thanks my family for all the support they gave me during these years. Agata Patané, my girlfriend, sextant of my travel and fixed star for my reference system. Giulio Masinelli for being a formidable friend and always a source of interesting reflections. Thanks to all my friend in my home country for being there for me when I needed them the most.
Thanks to Prof. Dr. David Atienza Alonso and to Dr. Tomas Teijeiro for supporting this work.
Finally, I would like to thank the lab os ESL at EPFL for the friendly climate in which I was welcomed.

Abstract

In this thesis, we explore 3 main topics: non-uniform (in time) sub-sampling, QRS detection in an event-based sub-sampled ECG (electrocardiogram) and implementation on a low-power MCU (Micro Controller Unit). The main idea behind this work is to reduce the energy consumption of a QRS detection algorithm by adapting the sampling frequency using the local frequency of the signal while maintaining the overall performance on the QRS detection without degradation. In particular, we will focus on the compréhension and re-adaptation of 2 popular algorithms for QRS detection: the Pan-Tompkins and the gQRS. This choice was guided by some constraints given by the event-based sub-sampling. The re-adaptation, in particular, was performed in 2 parts: the first step was to change the behavior of the algorithms in order to be able to work in an event-based sampled domain. The results achieved from the first step led to the selection of gQRS as the designed algorithm to undergo step 2: parallelization and optimization for the chosen low-power device. The results achieved are comparable to the results achieved by the original version in the classical uniform-sampled domain. The selected low-power device is the PULP platform, an MCU composed of a single core, low power CPU and a cluster composed of other 8 smaller cores.

Keywords

PULP, Low-power computing, Event-based sampling, QRS detection

Contents

List of Figures

List of Tables

1 Introduction

Continuous bio-signal monitoring through Wireless Body Sensor Nodes (WBSN), in combination with signal processing and machine learning techniques, are guiding a paradigm change in the follow-up of patients with chronic diseases. But as the number of users scales to hundreds of thousands or even to millions, a number of problems arise, mainly related to the management of the large amounts of generated data and the energy efficiency.In the last years, the pursuit of smaller, more efficient wearable devices and with higher battery life has led to different optimizations in the signal acquisition, processing, and communication stages. One of this optimizations is the adoption of a non-uniform sampling scheme, which can reduce the amount of data to be captured, processed, transmitted and stored. The sparse nature of many bio-signals, like the electrocardiogram (ECG), makes it possible to go beyond the classical Nyquist-Shannon sampling theorem and greatly reduce the global sampling rate of the signal without losing significant information, by focusing on the areas with relevant activity, in figure 1 we can see a typical ECG signal sampled uniformly (in blue) and the event-based samples taken (green dots on the orange ECG), as we can notice, the instantaneous sampling frequency increase proportionally with the instantaneous frequency of the signal.

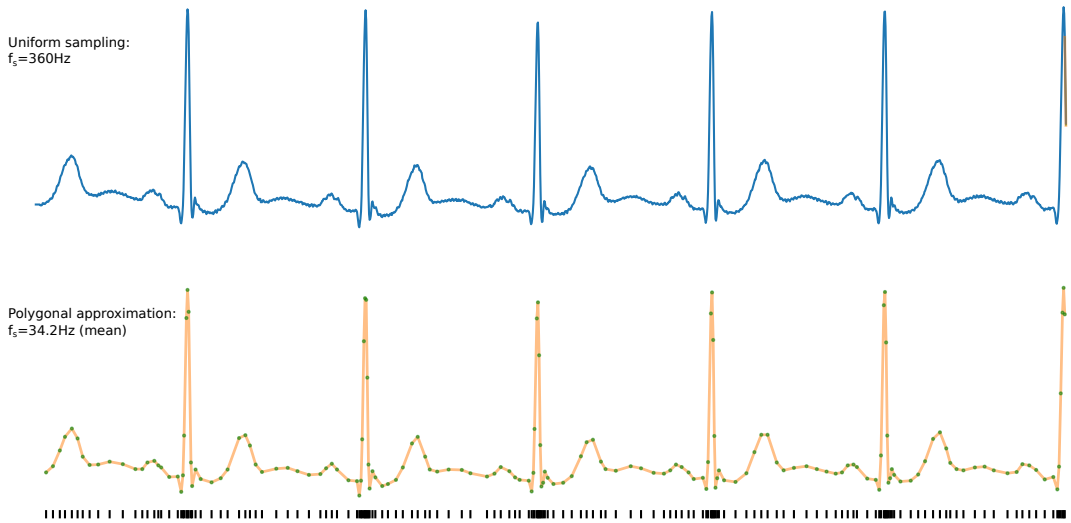

Figure 1: Uniform vs non-uniform sampled ECG segment. Black lines at the bottom represent the sampling density

However, this sampling scheme prevents the use of most of the existing biosignal processing algorithms, which assume a uniform sampling frequency. Therefore, the objective of this project is to start filling this gap by developing a set of algorithms

for QRS detection in ECG signals.This task is of particular interest because it give us the main tool for Heart rate analysis. The algorithms should also be able to be implemented in an embedded architecture, such as the PULP ultra-low power wearable platform (PULP), in order to make it ready for a practical implementation in a real-life embedded wearable device. Finally, the energy efficiency of the new algorithms will be evaluated and compared with state-of-the art methods.

In this first chapter, we're going to discuss the general idea behind the work of this thesis, in particular, we will make a short overview on the ECG signal, with particular attention to one specific standard database. We will then take a short tour around the classical techniques for QRS detection. We will then look at two event-based methods and analyze the selected one. In order to demonstrate the practical feasibility of our proposal we will implement the studied algorithm on a low power MCU and analyze the energy consumption. We will describe it in the sub-paragraph about the used platform

1.1 ECG and dataset

In this section we will try to give a fast overview of the ECG signal, according to [1, 2].

The electrocardiogram is the description of the electrical activity of the heart, recorded by electrodes positioned on the surface of the body. The continuous contraction and extension of the muscular cells that compose the heart are caused by the depolarization and re-polarization of the excitable cardiac cells. This depolarization/re-polarization cycle can be measured as a potential in different parts of the body. Given the expected signal we can use the recorded ECG to be able to diagnose several heart diseases. We can divide the ECG into 3 main sections: P complex, QRS complex and T complex. Each of these complexes holds inside several interesting points. In particular, we will investigate the QRS complex that corresponds with the main ventricular contraction and is composed of 3 points: Q, R, and S. The used database for this thesis is the MIT-BIH Arrhythmia Database[3]. This is a standard database composed of 48 recordings performed on 47 patients of several hospitals, each of them is 30 minutes long and already labeled. The recordings were acquired at a sampling frequency of $F_s = 360Hz$ per channel with 11-bit resolution over a 10 mV range with 2+1(reference) leads placed on the chest with the exception of one recording that was registered using only 1+1 leads. In this work we will use only the signal coming from the lower lead, often referred as V1.

1.2 Classical ECG algorithms for QRS detection

QRS complex detection is an algorithmic task that has been largely investigated in the past 40 years from several points of view. An extensive comparison between several methods can be found in [4, 5]. The main problem with the QRS detection is that normal complexes are detectable with low efforts while the most interesting peaks (relative to arrhythmia and heart diseases) are complex and not always detectable. Moreover, because the ECG is registered with leads on the skin of the patient, muscles contractions can cause artifacts that, along with the electrical noise, lead to misdetections in common algorithms. In order to understand the performance of an algorithm we used the *F-Score* that we can obtain from the *Specificity* and *Positive-predictivity*, using the following formulae:

$$S_p = \frac{Correct\ QRS\ predicted}{Total\ number\ of\ true\ QRS\ peaks} \tag{1}$$

$$P_p = \frac{Correct\ QRS\ predicted}{Total\ number\ of\ QRS\ predicted} \tag{2}$$

$$F_1 = \frac{2 * S_p * P_p}{S_p + P_p} \tag{3}$$

The F-score is the average between specificity and positive-predictivity, in other world, it tell us the average percentage of false alarms. We can separate the typical QRS detection algorithms in two families: statistical methods and deterministic methods. The statistical methods rely, usually, on the description of the QRS complex as a random variable, trying to compute $P(X_i = QRS|X_i, X_{i-1}, ..., X_{i-n})$, several approaches can be taken in order to compute this, some interesting examples can be found in [6, 7, 8]. The deterministic methods rely, indeed, on prior knowledge and a classical algorithmic approach to the problem such as derivative and integral estimation, dominant peak research, etc. , examples of this type of algorithms can be found in [3, 9]. Our main task is not to solve a problem that has already been solved in several different ways. Our objective to change and implement existing algorithms in order to achieve similar or equal results in a event-based sampling setting.

1.3 Sub-sampling techniques

In order to obtain an event-based sampled signal from our departure database, we must apply a non-linear sub-sampling transformation. Note that the non-linear sub-

sampling is needed here because we are using signals from a database acquired with classical ADC(the signals in the database where acquired using a uniform sampling at $F_s = 360Hz$) .
There are several techniques that can be used to sub-sample a signal, each of them relying on different ideas. One of the most common ones is the one proposed by [10]: the main idea behind this method is to use an ADC that output a sample only if the change in the signal exceeds a predefined set of regularly spaced amplitude boundaries. In this work, we decided to use a different one[11] that can be explained as follow:
The sub-sampling used measure the error between the real signal and a linear approximation of it: given a starting point (A) we analyze every successive point (B) and compute the integral in the interval [A-B] of the true signal minus the linear interpolation. If this result in an error less than a given threshold, we proceed to the next point, otherwise we stop and fix that last point (B) as the new sub-sampled point (see figure 2).

The selected threshold tell us what's the maximum cumulative error that we can allow between the real signal and the linear approximation of it that we ignore before taking a new sample, an example of a signal sub-sampled with this techniques can be found in Figure 3(Original) and in 4(Sub-sampled). The understanding of the basic principle of this algorithm is essential for this work: several observations used to adapt the classic algorithms derive directly from the error model used in the sub-sampling.

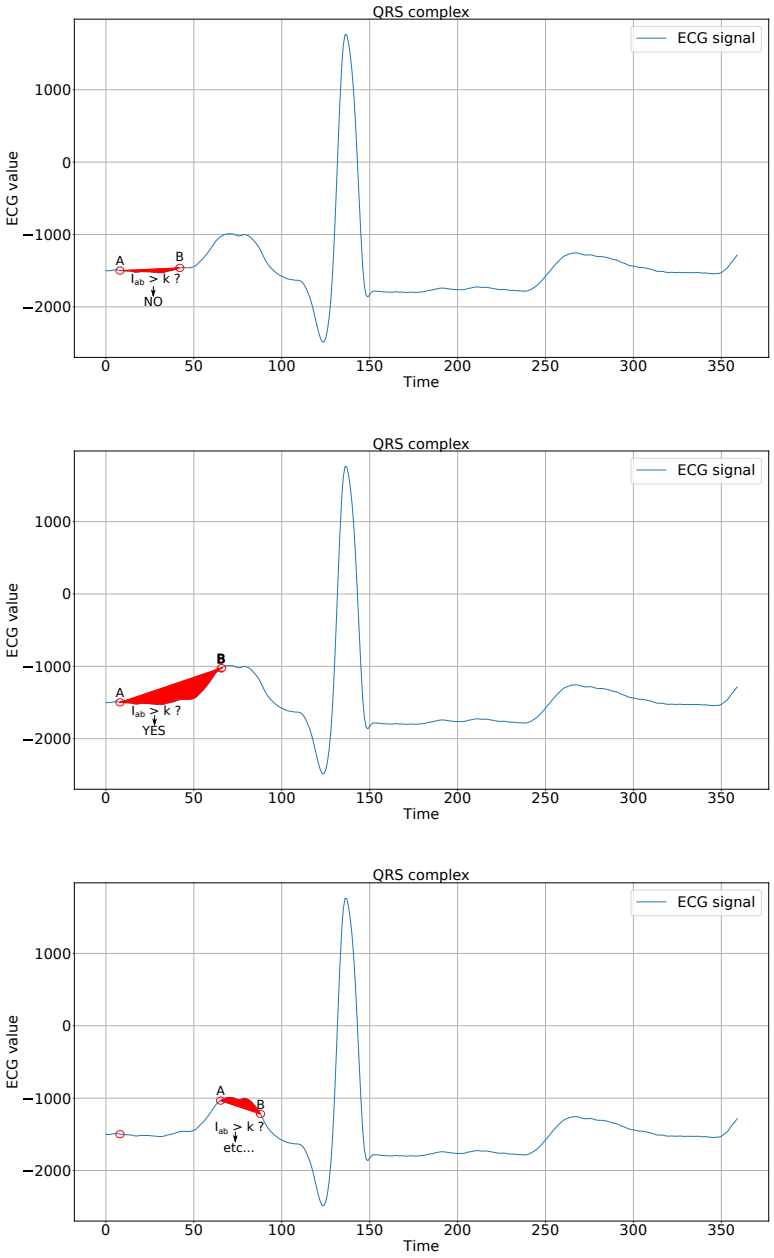

Figure 2: Example of the procedure used to obtain the event-based sampling

1.4 Adopted approach

As we have seen previously, plenty of algorithms exist for QRS detection. However, not all of them can be easily (or at all) adapted in order to work in a sub-sampled domain. Moreover, one of the main objectives of this work is to use a low power plat-

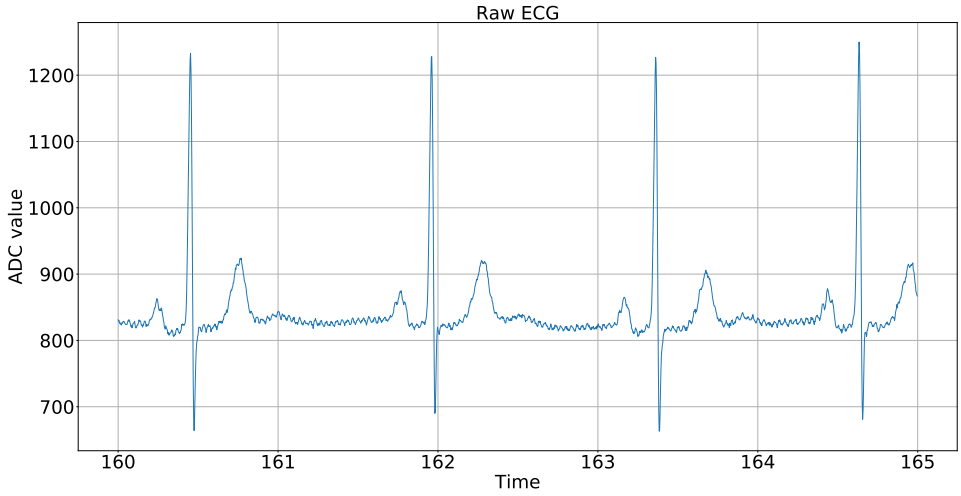

Figure 3: Raw ECG file

form. This leads the choice of the possible candidates toward deterministic methods with light computation and memory footprint. The selected methods need, of course, to be also reliable and achieve a high F-score while keeping the variance among different tests low. The first algorithm tried was a simple adaptive double-threshold with baseline removal. This was only a preliminary test, used to understand the structure of the data and the expected degradation of the results as a function of the used threshold and will not be explained further. The second method tested and adapted was the Pan-Tompkins algorithm [9]. This is one of the most historically famous algorithms for QRS complex detection and even if more well-performing algorithms exist nowadays, it still remains a light and excellent choice for low power (and low performance) devices. During our tests, it reached a maximum F-score of 98.5% (the average among all the 48 recordings). This is considered, in nowadays medical applications, to be a relatively bad score: considering that the human heart beats, in average, around 100,000 times a day, this performance would mean 1,500 false alarms per day. The third algorithm tried is known as gQRS, published in the WFDB software compilation from Physionet [3]. This algorithm, designed by G.Moody, was designed to work on ambulatory records in the MIT format. The base algorithm performs extremely well on all the tested files, achieving F-scores that vary from 99 to 100% (The detailed results will be given in the dedicated chapter). The baseline score of this algorithm lead to the development of a new version adapted to event-based sampled signals. The results obtained where considered satisfying also at high threshold levels (F-score = 99.1% at a high threshold level =

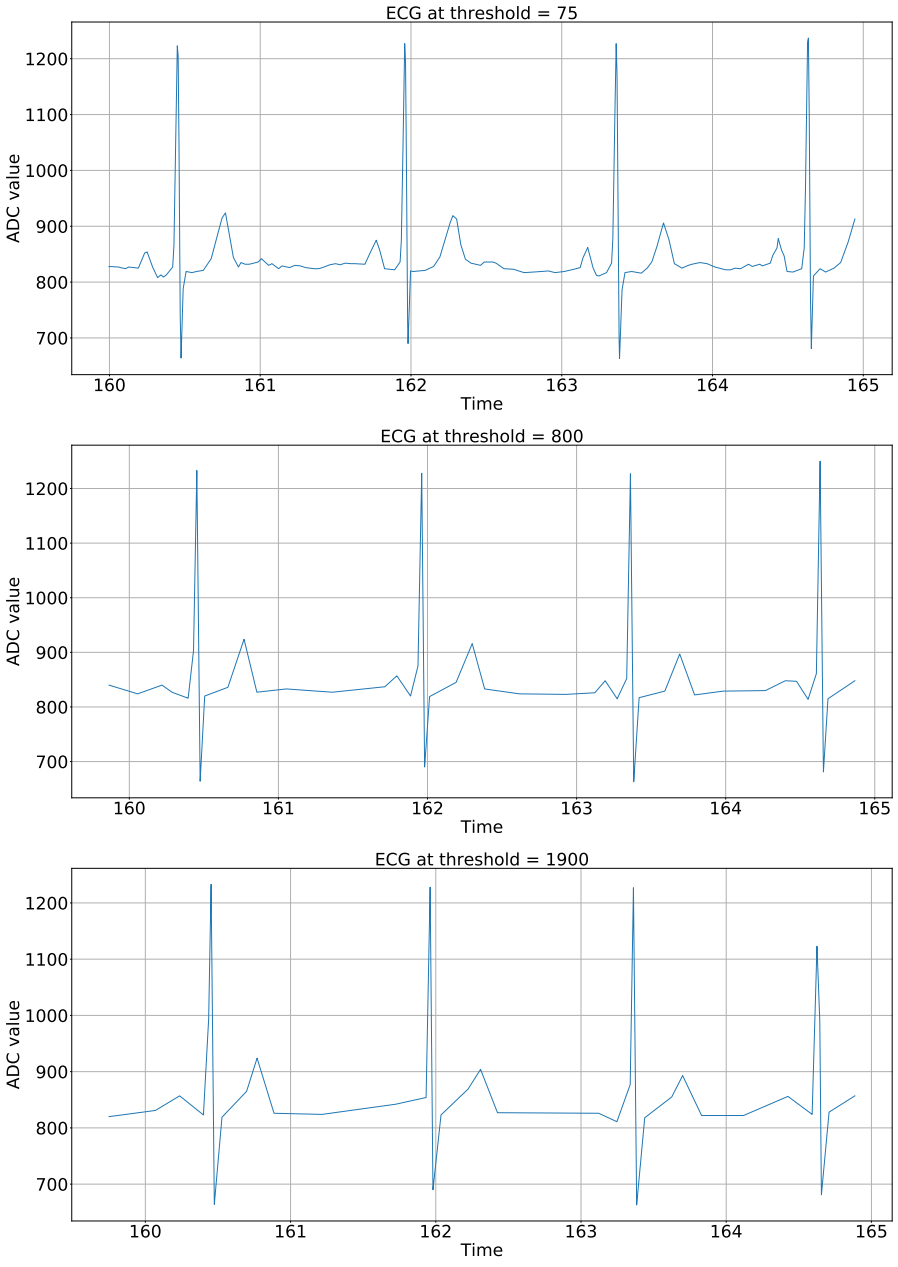

Figure 4: Same signal of Figure 3, sub-sampled as described in 1.3 at three thresholds

1000). Given these results, we decided to focus on the gQRS for the next stage: the implementation on a low power MCU.

1.5 Low energy MCU

As previously said, the idea of a working algorithm in the sub-sampled domain acquire even more strength when applied in a low power MCU. Several of such platforms exist on the market. While most of them are different versions of the low-power versions of the ARM architecture, the platform targeted is based on the "RI5CY" architecture[12]. The platform used is one of the PULP distribution, a low energy MCU designed by ETH and the University of Bologna jointed efforts, in particular, Mr.Wolf[13]. This is a RISC-5 MCU with an extreme computation capability (up to 1 GFlop/s: Mr.Wolf presentation) and low power consumption. It is important to notice that, even if this particular platform has a floating-point unit (FPU), it was decided to not use it and re-write the code in order to use only fixed points data when possible. Three versions of the gQRS algorithm were developed and tested in a simulated environment in order to get a good comparison in terms of energy consumption of different types of implementation. We will see later in this work the exact details of the PULP platform

2 QRS detection

In this chapter we will discuss further about the implementation of several algorithms for QRS detection and then we will put our focus on the one that will go under the optimization process for execution on the PULP platform. It is important, before proceeding any further, to precise what we define as QRS complex: as we can see from figure 5, we call QRS complex the signal that start at point Q and terminate at point S. As seen in 1.1 this signal correspond to main ventricular contraction of the heart muscles.

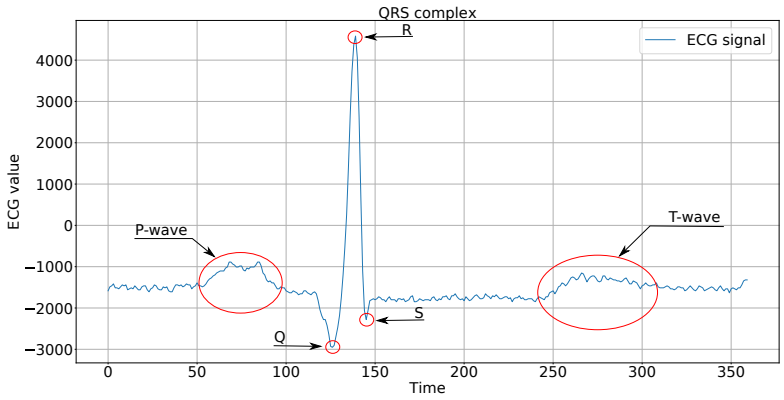

Figure 5: Example of a QRS complex. In this image is possible to observe also the P and T wave

2.1 Methods considered

As seen in the introduction, in 1.4, the two main methods considered were the Pan-Tompkins algorithm[9] and the gQRS algorithm, designed by Dr. G.B. Moody and published in the WFDB software compilation from Physionet[3]. These two algorithms were chosen among the sea of the existing algorithms for QRS complex recognition for two main reasons: The ability to design the full stack, from prototyping to implementation (in an emulator) and the possibility of being re-adapted to a non-uniformly sampled signal. This led to exclude all methods involving wavelet or Fourier transformations: even if a theory of wavelet on event-based sampled domain exist we still opted for more classical and of easier implementation solutions, this because the duration and the scope of this thesis were not suited for the complexity needed to translate "frequency" based methods to non-uniformly sampled signals. Moreover, because one of the main objectives of this work is to obtain a

low energy QRS detection algorithm, we focused on the solutions that result to be computationally lighter.

2.2 Algorithms description

The idea behind the Pan-Tompkins[9] is that the heart rate, once squared and integrated, presents a more predictable structure with periodic peaks denoting where the QRS complex is located. In order to explain this algorithm, we're going to make the assumption that the whole signal (or at least a chunk of it) is available. The first operation executed is filtering with a custom designed band-pass filter that aims to block the muscular noise, the T-wave and the electrical noise. We will call this signal $f(t)$. After this first step, a differentiation is performed to obtain information about the Q-R slope duration. Then the signal gets squared, this operation makes the signal positive and emphasize the high frequency content of the signal (i.e. the QRS complex) because of the non-linear amplification. Lastly, the resulting signal gets integrated with the time windows computed during the differentiation. We will call this signal $i(t)$. In order to obtain the QRS positions we use $f(t)$ and $i(t)$. Eight adaptive parameters (4 that operate on $f(t)$ and 4 that operate on $i(t)$) plus a running time estimator is used to decide if a certain peak is a QRS complex.

The second considered algorithm is the gQRS algorithm, published in the WFDB software compilation from Physionet[3]. This algorithm captured immediately our attention because of its surprisingly good performance. As we will see later, this algorithm reaches, for uniform-sampled signals, an average F-score of 99.8% and was designed with the idea to put the positive-predictivity score over the sensitivity score. The base gQRS algorithm implements a consistently higher amount of adaptive thresholds with respect to the Pan-Tompkins that act on both time and amplitude. The gQRS algorithm also needs to integrate the signal but here it is performed in a total different way, as we are going to see. In order to understand it better, we can divide the gQRS algorithm into 3 phases: filtering, integration and peak detection and thresholding

2.2.1 gQRS Filtering

The first operation applied on the signal is double filtering. Since all the used filters are digital, all of them operate with a delay times δt that is, on average, $\frac{1}{4}T_{QRS}$ where T_{QRS} is the average duration of the QRS complex. This is a settable parameter and is not learned during the execution, so we assume the default value

of $T_{QRS} = 0.07s$. At first, a trapezoidal filter is used to smooth the signal and remove motion artifact and electric noise. A trapezoidal filter is simply a low-pass filter that emulates a hardware R-C low-pass filter, we can observe it's behaviour in figure 6. This filtering is obtained with an IIR (Infinite Impulsive Response) digital

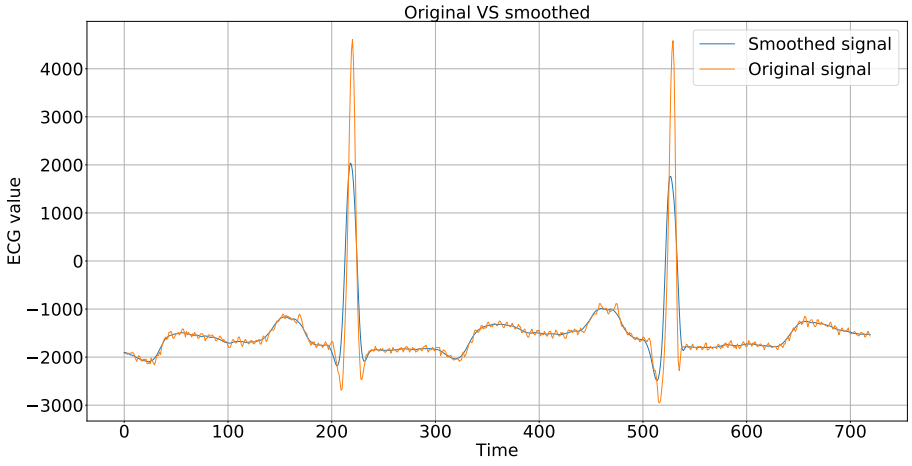

Figure 6: Real VS smoothed signal

filter that takes into account the previous response of the filter and two samples at a distance of $\pm 2\delta t$, hence introducing a delay on the filtered signal of $\frac{T_{QRS}}{2}$. After this first filter, a second one is used in order to enhance the QRS peak in the smoothed signal and make the other parts of the signal lower. This is done using what Dr. G.B. Moody calls an"adapted filter". In figure 7 we can see the impulsive response of the adapted filter. As we can notice, its shape is designed to resemble the shape of a typical QRS complex. Being the filter an odd function with respect to the central sample, this filter is able to give a strong response to both the positive and negative QRS complexes. Moreover, it will be an extremely useful property when we will re-adapt the code to work on a low power MCU. This filtering is obtained with a FIR (Finite Impulsive Response) digital filter that takes into account 9 samples (with the 5^{th} being the currently analyzed sample) at a distance of δt, making this filter $8\delta t$ long in time and introducing a delay on the filtered signal of $T_{QRS}(4\delta t)$. The final latency of the two-stage filtering is, hence, $\frac{3}{2}T_{QRS} = 6\delta t$

2.2.2 gQRS Integration

The obtained results from the filtering step undergo a successive integration operation and a squaring operation. The successive integration only has peaks when the

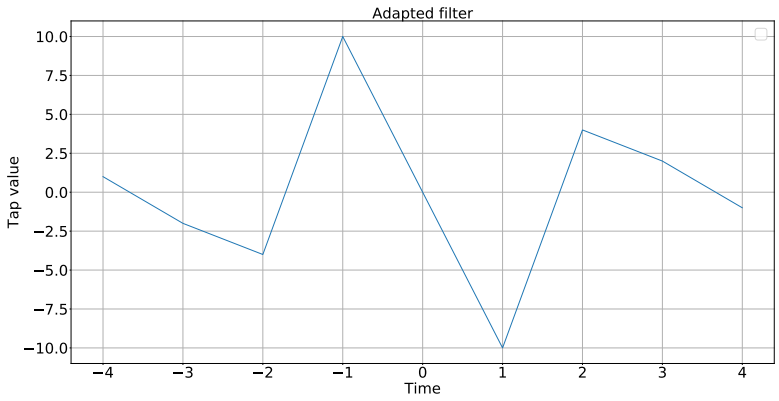

Figure 7: Impulsive response of the adapted filter used on the gQRS algorithm

adapted filter has a high and long enough response and is constant for all the parts of the signal that does not present the required QRS signature. This is because the ECG is a signal with zero average (once the ADC offset is removed). The squaring is needed because, as for the Pan-Tompkins algorithm, this operation makes the signal positive and emphasizes the high frequency content of the signal (i.e. the QRS complex) because of the non-linear amplification.

2.2.3 gQRS Peak detection and thresholding

Finally, in the third phase, we check the presence or absence of a peak and we determine if it is from a QRS complex or not. To do so, we first check if a peak structure is present, checking if $x_{t-1} < x_t > x_{t+1}$, if this condition is true then the detected peak goes through a series of thresholds needed to asses if it comes from a QRS complex or not: firs we check if it is a significant peak using a threshold, and secondly we check if it is possible for it to be a QRS peak using a second threshold and timing information. This double-check is needed because in case a peak results to be significant but not strong enough to come from a QRS complex and no QRS complex is detected before a certain timeout then we can lower (or make higher, if the opposite happens) the two thresholds independently, adapting our algorithm for different scenarios. Another important check that gets performed on the founded peaks is the dominance over a neighbour: this operation is performed recursively and checks if the peak is the strongest in a neighbour of peaks. The interesting part is that, in order to be dominant, a peak does not need to be the highest one in its neighbourhood, it just needs to be higher than the previous non-dominant peak. In other words, if a detected peak certainly comes from a QRS complex and get labeled

as dominant, this creates an interdiction time for the next peak to come in a certain time-span. This strongly helps with discarding false positives detections.

2.3 Algorithms adaptation

The two described algorithm works with the assumption that all the samples in a signal are uniformly sampled in time. That means that the signal is sampled at a constant frequency f_s. While this is true for our database ($F_s = 360Hz$), it is completely wrong after the signal undergoes the process of event-based sub-sampling, hence we need to redesign the chosen algorithms so that they will continue to have similar performances even when we use signals that are non-uniformly sampled. This section will strongly leverage on the specific type of event-based sub-sampling used. As discussed above, the obtained sub-sampled signal can be linearly interpolated while the error is ensured to remaining below a certain threshold ε. The first logical approach would be to just linearly interpolate the signal in order to recover the original sampling frequency and then apply the described algorithms as they are. Even if this is a viable solution it still remains sub-optimal: the main objective of this work is to drastically reduce the used energy on the MCU side while performing these operations would only increase the workload on the target platform. In the following subsections we are going to describe, at first, the techniques used to adapt the Pan-Tompkins algorithm and then reuse the acquired knowledge with the gQRS algorithm.

2.3.1 Pan-Tompkins adaptation

It's important to notice that in our work of adaptation of the Pan-Tompkins algorithm, we did not aimed to re-create the exact behaviour of the original Pan-Tompkins algorithm. This is mainly because of two reasons:

1. The filters of the original Pan-Tompkins algorithm were designed specifically to work with uniformly sampled signals with a certain type of noise. The sub-sampling algorithm used, however, already performs a certain type of low-pass filtering. Moreover, the proposed implementation in [9] would need the modelling of an IIR filter and we will see later on why, in this scenario, the design of this type of filter results to be of high complexity.

2. From a preliminary study, we noticed the overall higher performance of the gQRS algorithm when compared to the Pan-Tompkins. Given this knowledge,

it was already clear that the gQRS algorithm would be the targeted algorithm for the final implementation. The Pan-Tompkins (or even better, the pseudo-Pan-Tompkins developed) was useful for understanding how to handle this type of sub-sampled signal. Moreover, the baseline score achieved by this algorithm was used as a baseline score, useful to evaluate the overall performance of the various versions of the gQRS.

Given these premises, the biggest challenge we faced was how to obtain the integral of a sub-sampled signal. The remaining parts of the algorithm (in which we perform the comparisons with all the thresholds) can be easily implemented by switching from an index-based timescale (each value are equally spaced in time, hence the index of the obtained array of values can be used as a time scale, this is how the original Pan-Tompkins algorithm work) to time based time-scale: each time a sample arrive we save not only its value but also its time. Given ε to be the maximum error allowed of the sub-sampling we know that:

$$\left| \sum_{n=a}^{b} x_n - \tilde{x}_n \right| \leqslant \varepsilon \tag{4}$$

$$\sum_{n=a}^{b} x_n + \varepsilon \geqslant \sum_{n=a}^{b} \tilde{x}_n \geqslant \sum_{n=a}^{b} x_n - \varepsilon \tag{5}$$

where x_n is the n^{th} point of the true signal, $\tilde{x}_n$ is it's corresponding value in the linear approximation from a to b and ε is the wanted threshold. This tells us that as the true error between the true numerical integral and the integral of the linear interpolation never grow higher than ε. Because the expected value of x_n and of $\tilde{x}_n$ is 0, the expected value of the error is also 0 also for any sequence of consecutive numerical integral, given the sequence $[0, q_1, q_2, ..., q_N]$ as the sequence of index that delimit the signal $\widehat{x}_n$ in chunks that respect the Eq. 5:

$$\mathbb{E}\left[\sum_{[a,b]=[0,q_1[}^{[q_{N-1},q_N]} \sum_{n=a}^{b} x_n - \tilde{x}_n \right] = \sum_{[a,b]=[0,q_1[}^{[q_{N-1},q_N]} \mathbb{E}\left[\sum_{n=a}^{b} x_n - \tilde{x}_n \right] = 0 \tag{6}$$

This tells us that, if we continuously integrate the linear interpolated signal chunks, the result is, in expected value, equal to the true numerical integral. Given this result, we can now find the formula for the integration in our sub-sampled domain. We will calculate the integral of the linear interpolation of the signal only

in the obtained sub-sampled points, and this can greatly reduce the computation workload and the memory needed to store the obtained values. Given 2 consecutive sub-sampled points $\{v_1, v_2\}$ and their time $\{t_1, t_2\}$ the easiest method to compute such integral, would be to obtain all the values of the linear interpolation between t_1 and t_2 but then again, as previously said, all the idea of this project is to reduce the total energy consumption while this would instead increase it (when compared to the "vanilla" implementation of the algorithm on an uniform-sampled signal). What we could do is to notice that:

$$I_{t_1-t_2} = \sum_{i=t_1}^{t_2-1} \tilde{x}_n[i] = \sum_{i=t_1}^{t_2-1} (m \cdot i + q) \tag{7}$$

where $m = \frac{v_2-v_1}{t_2-t_1}$ and $q = v_1 - m \cdot t_1$. Notice that the consecutive sum need to arrive only to $t_2 - 1$, otherwise, during the calculations for the next interval $\{t_2, t_3\}$ the point in t_2 would be taken into account 2 times. Let's substitute $t_2 - 1$ with l, using the Gauss sum, we can write:

$$\begin{aligned} I_{t_1-t_2} &= m \cdot \sum_{i=t_1}^{l} m \cdot i + q \cdot (t_1 - l + 1) = \\ \frac{m}{2} &\cdot (l + t_1)(l - t_1 + 1) + q \cdot (l - t_1 + 1) = \\ \frac{m}{2} &\cdot (t_2(t_2 + 1) - t_1(t_1 + 1)) + q \cdot (t_2 - t_1) \end{aligned} \tag{8}$$

This gives us the solution in closed form of the numeric integral using only the points present in the sub-sampled signal. To write m and q in function of $\{v_1, v_2\}$ and $\{t_1, t_2\}$, even if possible, does not bring any advantage. Moreover, even if the Pan-Tompkins algorithm does not need them, we will see that, to store those values (m and q) will be helpful in the gQRS algorithm in order to make the workload slightly lighter at the cost of more memory used.

In order to use the filter proposed in [9], we would need to implement an IIR filter such that it takes as input the previous output (in our specific case, the 2 last outputs), or, mathematically (in the notation without delays):

$$y_n = \sum_{i=n-Q}^{n} a[i-n] \cdot x_i + \sum_{j=n-P}^{n-1} b[j-n] \cdot y_j \tag{9}$$

Where P is the number of previous y_n and Q the length of the direct filter on the

signal x. Given a point x_n, we can linearly interpolate all the previous Q points. In order to obtain the previous output of the filter we would actually need to calculate them in a recursive manner, this can be theoretically equivalent to the implementation of the algorithm with a FIR filter on the signal linearly interpolated up to a certain point. The number of points that we need to interpolate may appear to be $P+Q$ but this would be true only in the case of a filter with all the delay coefficients equal to one. Otherwise, the number of needed interpolated points can grow up to $(P+1)\cdot Q$. Moreover, even if it is true that in order to obtain y[n-j] we need to re-apply the filter to other Q interpolated signal values, it is also true that we would need to consider, again, the previous response of the IIR filter, making this a highly inefficient recursive procedure. The most efficient way to compute such filter is to obtain the fully linear interpolated signal and that would defy the objective of this thesis as said before.

The core part of the algorithm, that relay on the use of adaptive thresholding on the two signals (the original sub-sampled signal and the integrated one) was basically left unchanged. Even if the signal is sampled following the described event-based technique, we can still recover the peak structure: a prominent peak (i.e. anything that is not noise and of enough high frequency to be considered) consists in a big variation of the signal, hence the distance between the ground truth and the linear approximation increase much faster than the part of the ECG where we do not have any event. This causes the instantaneous frequency of the sub-sampled signal to increase around a prominent peak (hence, also in the proximity of a QRS complex). This allows us to use a classic peak detection and to use, on the detected peaks, the classic Pan-Tompkins algorithm.

As we can see from figure 8, the algorithm has 2 regimes:

1. $0 \leqslant \varepsilon \leqslant 800$ The performance increases up to 98.5%, This is due to the fact that we did not implement the custom band-pass filter proposed in [9] so the sub-sampling procedure is acting, as ε increases, as a low pass filter.

2. $800 < \varepsilon \leqslant 2000$ The performance starts to decrease almost quadratically with ε, this is because as the threshold increases, more and more details are lost.

As said before, the achieved performances were far below the one declared in [9]. This is because, as we have seen, some very specific parts of the original algorithm were deliberately not implemented. This is because, given previous studies, we already knew the algorithm to target as our main option: the gQRS while leaving

this as a proof of concept.

2.3.2 gQRS adaptation

As discussed in 2.2, the gQRS algorithm can be divided into three main steps. In this section we are going to describe how we adapted the first two steps. The third step, as for the Pan-Tompkins, remains the same with the sole difference that in the original algorithm, the timescale was given by the index of the sample while in our version we use a custom time vector to identify the timestamp at which a new sample arrives. In the original implementation, in the first step, two filters are applied: an IIR filter and a FIR convolutional filter. As discussed before, the implementation of an IIR filter in closed form, even if mathematically feasible, it defies the purpose of this work. The IIR filter used is a low pass filter that emulates an RC physical filter in order to obtain a smoother signal. Because the sub-sampling algorithm already performs a low pass filtering we decided to try, at first, to avoid this first filter, making the algorithm lighter. One possible option may have been the use of a sufficiently large FIR filter that emulates the impulsive response of the IIR low pass filter. To finish the first step, a second filter (matched filter) was applied and then proceeded to the integration. We can write these two steps as:

$$y[t] = \sum_{i=0}^{t} \sum_{j=-\infty}^{+\infty} h[j] \cdot x[i - \delta j] \tag{10}$$

Where δ is the delay of the filter and $h[j]$ is the j^{th} tap of the filter and is long $\frac{1}{4}$ of the average duration of a QRS complex. The computation of the value of y[t] requires the knowledge of all the previous filter response. This is because even if it's possible to linearly interpolate just the points needed for the filter, given two contiguous filter response we can not say anything about the behaviour in the middle (in this sub-sampled environment) and, hence, we can not compute the integral needed to obtain $y[t]$. One solution, again, could be to linearly sample all the signal and then execute the filtering and integration on it, this, as before, would defy the main idea of this thesis. Another method we could use is the following:
Using Fubini's theorem [14] and numerating the taps from -4 to +4 (where the tap 0 is the one corresponding to the analyzed time instant) we can write:

$$y[t] = \sum_{j=-4}^{4} \sum_{i=0}^{t} h[j] \cdot x[i - \delta j] \tag{11}$$

Now lets assume to know the integral of the signal: $\sum_{i=0}^{t} x[i] = I[t]$, then

$$y[t] = \sum_{j=-4}^{4} h[j] \cdot I[t - \delta j] \tag{12}$$

We have seen in Eq. 8 that is possible to perform the computation of the integral efficiently within an error of $\pm\varepsilon$ and *a priori* and can be performed on-line as the data arrive. Anyway, this forces us to save, aside from the integral signal also, at least, the original signal. In our prototype implementation we saved also the array of m_s and q_s ($y[t] = m_\tau \cdot x[t] + q_\tau$ where m_τ and q_τ are the m and q of the corresponding linear chunk of the signal). We will see, in the low-power implementation, how to reduce drastically the amount of memory required.

As we can observe from Eq. 12, if we want to compute the response for a precise t, we would need to know the value of $I[t - \delta j]$. Because of the event-based sub-sampling, by no mean we can be certain to have the signal $x[t - \delta j]$ and, hence, we can not be sure to have, in the vector of the integrated signal, all the needed points to compute the response of the filter. In order to solve this problem, we can find the nearest two points to the desired one and use the Eq. 8 and the saved past values to obtain the integral only in the desired points.

The third step is basically left untouched aside from some minor change. It has already been explained in section 2.2.3

2.4 Preliminary comparison

In this section, we will take a look at some preliminary results that helped us to decide which algorithm should be optimized and ported to the PULP platform. Further and deeper analysis will be carried on in chapter 4. As we can see from figure 8, the gQRS algorithm reaches an average F-score of 99.8% for low thresholds and decrease almost quadratically with the threshold. It is important to notice that, even with a threshold of $\varepsilon = 1100$ the performance of the gQRS algorithm remains still higher than the performance of the original Pan-Tompkins algorithm described in [9] and this is the main motivation that pushed us to focus on this algorithm and use the pseudo-Pan-Tompkins results as a scale for the evaluation of the achieved performance. More details about variance and deeper behaviour of the algorithm will be discussed in section 4

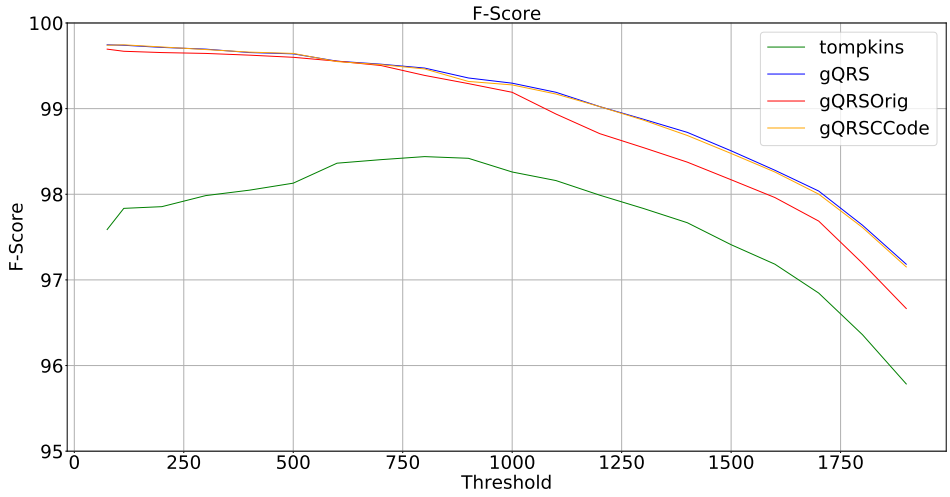

Figure 8: F-score results of gQRS vs. Tompkins. (F-score vs ε)

3 Low power implementation

In this chapter, we're going to briefly describe the ultra-low power MCU targeted (PULP) and then move on to the steps needed to implement the gQRS on such platform. In particular, we're going to analyze the steps needed to be able to use the multi-core cluster present on the MCU. The code for the implementation of the "event-based sampling gQRS" was originally prototyped in *Python* and then ported to *C*99.

3.1 PULP platform

The targeted MCU used is one of the various versions of the PULP MCU called Mr.Wolf[13]: an advanced microcontroller based on an ultra-low-power 32-bit RISC-V processor. We can see the chip block design in Figure 9. This platform has several interesting tools that can be used both for performance enhancement and power management. In order to describe it, is helpful to split the chip in two main domains: the Fabric Controller (FC) and the Cluster (CL). These domains are defined by zones under the control of the main core or of the the cluster and by separated power management. We will call SoC the portion of the chip under the direct control of the FC

3.1.1 Fabric Controller

The FC is the main core (and domain) of Mr.Wolf. It is able to perform fast access on 512 kB of L2 memory, divided into eight blocks. As we will see in the next section, Mr.Wolf also has a smaller bank of L1 memory. This memory is intended to be used as the main memory for the cluster cores. By default, the cluster and hence the L1 memory are turned off, making the L1 memory not accessible and not retentive. If the cluster is turned on, it is possible to access also the L1 memory directly from the fabric controller but this would require substantially more time (measured in numbers of clock cycles), making this operation inefficient to be performed like a classical memory access. It is possible to switch the SoC between three power profiles:

1. Active: while the FC is in active mode the core is executing all the code at the maximum clock frequency possible for that operating voltage,and all the L2 memory is powered on and retentive.

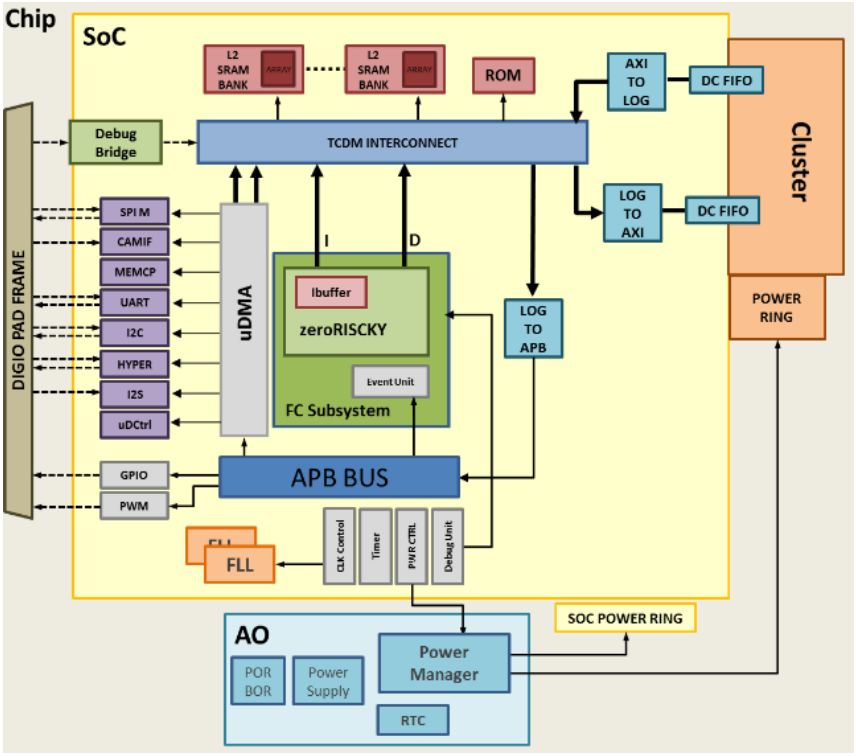

Figure 9: PULP chip block design, from Mr.Wolf presentation

2. Sleep (Power gated): while in sleep, the FC is power gated. This means that the core is not working and no energy is consumed on it. It's important to notice that this state is different than to be shut down: from this state it is possible to resume the interrupted operations with a relatively low overhead when compared to re-start from being shut down. Moreover, even if in this state the memory is powered off and not retentive, is possible to select a number of L2 blocks to continue to flag as retentive. This essential feature comes at a cost: any selected bank will consume energy in order to keep them powered on. We will see in chapter 4, how this consumption strongly impacts on the energy profile of our application.

3. Clock gated: This power profile is managed automatically by the PULP-platform and is entered any time the FC does not need to execute any instruction. In this profile, all the SoC is powered on but the clock is blocked before entering in it. This makes the memory retentive and accessible but also produces a big amount of leakage. This profile is useful, for example, if we need to access data in the L2 memory from the CL while the FC is waiting

Setting the working voltage will give the maximum frequency at which the FC (and

the CL) will be able to work: starting with the minimum voltage $V = 0.75V$ the FC will be able to reach a working clock frequency up to $f_{clk} = 125MHz$. The maximum speed is achieved with $V = 1.1V$ and reaches $f_{clk} = 478MHz$.
One of the most interesting features is the $\mu - DMA$: inside the SoC there is a DMA (direct memory access) module that can work even when the FC is in sleep mode (although the needed area of memory needs to be retentive). This makes it possible to transfer data from all the peripherals present on the SoC (I2C, UART, etc.). This can be used to perform data acquisition while the core is doing different things or to obtain better power performance, putting the SoC in sleep mode, retain only the needed bank of L2 memory and wake up the FC only when the desired amount of data has been received.

3.1.2 Cluster

The Cluster is a separated (i.e. not on the SoC as we can see from figure 9) system composed of 8 DSP(Digital Signal Processor) enhanced RISC-V processors and 64kB of L1 shared memory with an access time of 1 cycle. In the whole CL system it is also present a high performance DMA that results to be extremely useful, especially if a data transfer is needed from L2 to L1 while executing other instructions: the two memories (L2 and L1) enable multiple access allowing data transfer and execution to be parallelized. Setting the working voltage will give the maximum frequency at which the CL will be able to work: starting with the minimum voltage $V = 0.75V$ the CL will be able to reach a working clock frequency up to $f_{clk} = 40MHz$. The maximum speed is achieved with $V = 1.1V$ and reaches $f_{clk} = 350MHz$. Because of the difference in the working frequency between CL and FC, the operations needed to turn on the cluster (and adjust the clock frequency) require a big amount of time. From our experiments, this time is around 53 kCycles on average. This can be significantly reduced if we allow the FC to go slower and match the same frequency of the CL.

In this work, we decided to use an operating voltage of $V = 0.75V$ and a shared frequency between CL and FC of $f_{clk} = 40MHz$. As we will see in chapter 4 these operating conditions achieve speed performance too high for this type of application.

3.2 gQRS on PULP

As previously discussed, the only algorithm that we decided to port on PULP is the gQRS. This because, as we will see, it uses several computational structures

that can be easily parallelized and it can achieve good results while maintaining the algorithm light in terms of computational power.
We divided the work into three main redesigns of the original algorithm. The first is a porting of the original (vanilla) gQRS algorithm to the PULP platform, making it working in an on-line mode. The second is the modification of the ported gQRS algorithm to work on signals that are event-based sampled as we have seen in 2.3.2. The third was a large redesign redesign of the structure of the algorithm so it can execute, efficiently, in a multi-core environment.

3.2.1 Vanilla gQRS

The adaptation of the vanilla gQRS algorithm (that works only with uniformly sampled signals) did not required an intensive effort: once the code was translated from the prototyping language to plain $C-99$ only few more steps were needed: in particular we redesigned all the data-structures used in order to satisfy the memory constraints: while in the plain $C-99$ we used specific structures to emulate the behaviour of a classic object in order to be similar to the python version, here we redesigned the data-structures in order to work with a more classic C approach. Moreover, we changed all the dynamic allocated data to be, instead, positioned in the stack: even if the PULP platform gives access to specific functions to work with dynamic allocation on an MCU environment we preferred to keep our code compliant with the classic MCU programming techniques. The original gQRS algorithm was using data and results in floating-point representation. Even if Mr.Wolf has a FPU unit, we decided to avoid to use it for two reasons:

1. The FPU requires anyway both more time and power to operate. If possible, is better to execute the computations in fixed-point.

2. Even if Mr.Wolf is a low energy MCU, it is still extremely powerful. We ere aiming to develop a generic implementation for low low-power MCUs. These type of MCUs not always have a FPU

Further smaller optimizations were performed on the code in order to reduce the number of cycles needed.

3.2.2 Custom gQRS

As discussed above, the "event-based sampling gQRS" was developed, at first, using a fast-delivery prototyping language such as *Python*. Once we had the Python code

the porting in standard *C-99* took a relatively small effort. Then we needed to adapt it in order to be able to run on the FC. In order to do so, we first applied the same optimization techniques used for the vanilla version. After that, we needed to re-code the integral re-sampling (see 2.3.2) and the filtering. Minor corrections were needed in order to implement an efficient sliding buffer and multiple peaks detection.

3.2.3 Parallel gQRS

In order to be able to parallelize the gQRS algorithm a re-design of the central part of it was needed. As discussed in the previous chapter, the "event-based sampling gQRS" can be divided into 3 main sections: integration, filtering, and thresholding. As we will see, it was possible to parallelize integration and filtering. Unfortunately, the thresholding procedure adapts the thresholds based on the previous results and it was not possible to use the same techniques used to parallelize the integration. Not only the structure of the parallelized code was changed, but we also needed an efficient method to transfer data between the FC and the CL. We will describe, now, these three main steps, a schematic of the code can be found in Figure 11 while a scheme of the vector handling can be found in figure 10.

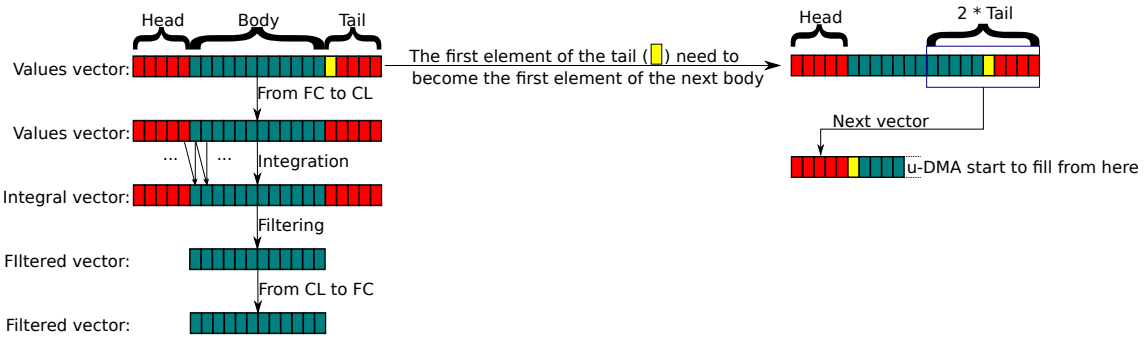

Figure 10: Block design for the described vector handling techniques: as we can see on the right, only the last $2 \cdot Tail$ elements needs to be stored for the next execution

FC-CL Bridge: in order to send data to the CL, the FC acquires two vectors: time ($\underline{t}_k$) and value($\underline{v}_k$) where k is the k^{th} chunk of data. These two vectors are composed of a tail, a body and a head with fixed dimensions. The tail and the head are needed in order to be able to apply the matched filter to the first and last element of the body without the need to add padding and delete a certain amount of value later. Once these two vectors are full, the CL starts a DMA transfer in order to bring them from the L2 to the L1 memory and start to

operate on them (on the body). When the CL has finished, it puts the results of the integration and filtering in a return vector $\underline{r}_k$ and starts a second DMA transfer that copy the result vector from the L1 to the L2 memory. Once this copy is finished, the FC start the peak detection and thresholding procedure. Once $\underline{r}_k$ has been completely analyzed, the FC copies the last elements of $\underline{t}_k$ and $\underline{v}_k$ at the beginning of $\underline{t}_{k+1}$ and $\underline{v}_{k+1}$ so that the next first element of the body will be the previous first element of the tail. In other words, let T_{len} be the length of the tail, we copy the last $2 \cdot T_{len}$ from the end of the two vectors to the beginning of them and we start to fill them with new values starting from $2 \cdot T_{len} + 1$.

Integration: As $\underline{t}_k$ and $\underline{v}_k$ arrive, we divide the body, the tail and the head in eight parts and send them to the eight cores of the cluster. Each core applies the integration in Eq. 8 to the chunk received and puts the results in three vectors (The vectors of the I, m and q) shared between the cores.

Filtering: Each chunk of the body is divided again between the eight cores that will implement the filtering to each of them in parallel. As discussed in 2.3.2, in order to implement the filtering around a central point, we need other eight equally spaced points at a distance of δ. To obtain them, we must use the re-sampling technique already discussed but with some minor changes. The integral vector used will be discontinuous from chunk to chunk. To deal with this, whenever the adapted filter is applied to a point that is in the i^{th} window, we calculate the required timing of each interpolated point, knowing that every needed point will fall between 2 points already present. For every needed point we can have three conditions:

1. The point is in the $i^{th} - 1$ window. In this case, we simply implement the discussed re-sampling algorithm
2. The point is in the i^{th} window or between the i^{th} window and the $i^{th} - 1$ window. In this case, we sum the last value of the $i^{th} - 1$ window to the two points that we will use for the interpolation
3. The point is in the $i^{th} + 1$ window or between the $i^{th} + 1$ window and the i^{th} window. In this case we sum the last value of the $i^{th} - 1$ window and the last value of the i^{th} window to the two points that we will use for the interpolation

After this check, we can apply the interpolation formula.

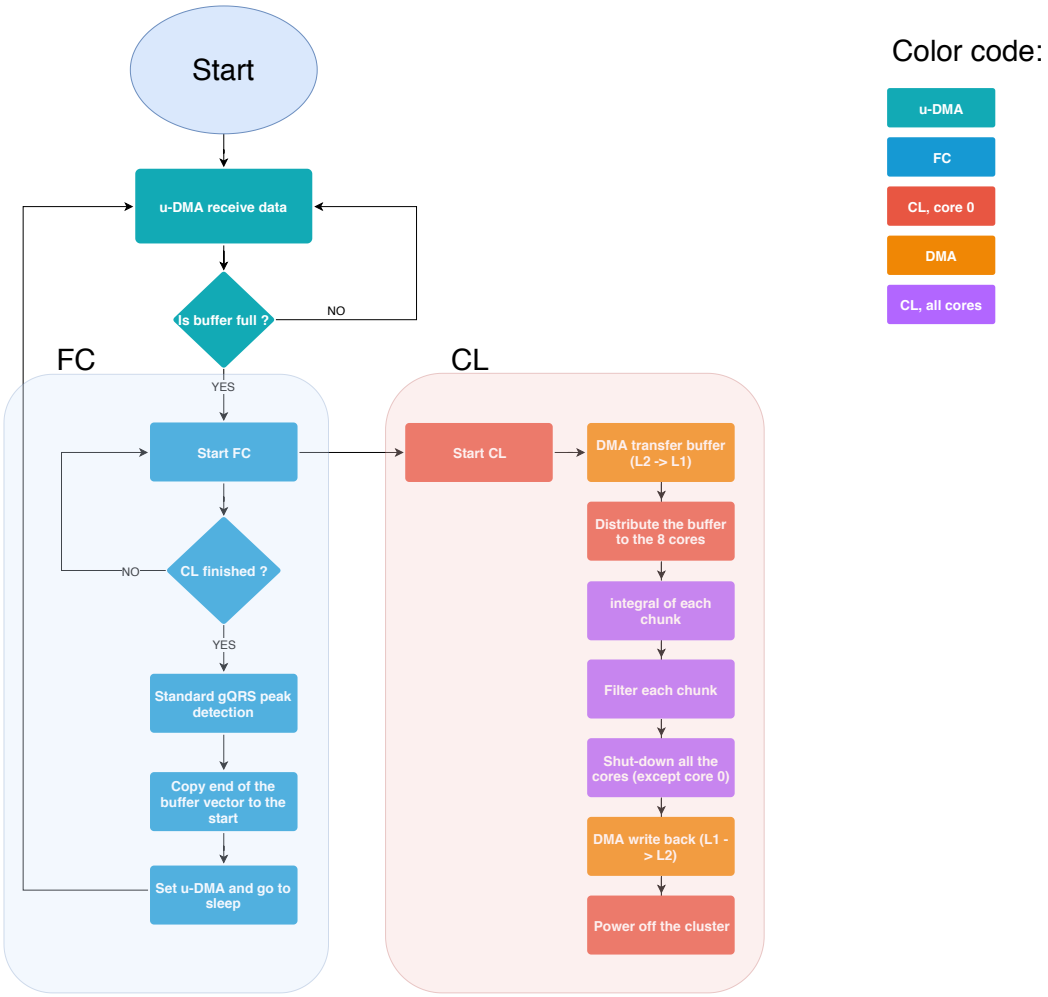

Figure 11: Block design for the multi-core version of the gQRS algorithm

It is important to notice that we did not mention anything about the used power-state of the MCU. This because the used SDK does not allow to simulate the sleep state of the FC and the functioning of the $\mu - DMA$. Because in the real scenario both the FC and CL are in sleep mode until the $\mu - DMA$ had transferred enough data and then they light up together. We decided, for profiling, to keep them light up and measure the performance only after the buffer has been filled and the "true" code start to works.

Another important detail is that we only save $2 * \cdot Tails$ points from one Run phase and the other as we can see from Figure 10 and, hence, the memory footprint of such program results to be really small. This is because of how we implemented the integration and because of the odd symmetry of the matched filter.

4 Results

In this chapter, we're going to discuss the performance of the event-based version we developed when compared to the original gQRS algorithm. Then, we are going to apply the original gQRS algorithm to a linearly-interpolated signal and compare the results with our version executed on the sub-sampled signal at different ε. Finally, we will see some interesting results that we obtained from the simulations of the execution on the PULP platform for different versions of the code.

4.1 Heartbeat detection comparison

The full-stack development of the event-based sampled gQRS algorithm was performed in three steps:

1. Python prototyping
2. C-99 development
3. PULP implementation

Each of these steps served a different purpose: The Python prototyping was used to obtain results in easy-to-test conditions, essential in the understanding of the mathematical principles of this algorithm. The C-99 version was used to obtain an algorithm able to run on a PC but ready to be implemented on any MCU in a relatively short time. In this section, we are going to see the results of these first two steps while in the next one we're going to see the results obtained from the implementation on the PULP platform. In order to obtain interesting metrics (i.e. useful to compare the results of different algorithms) we performed the following measurements:

1. Pseudo Pan-Tompkins algorithm on the sub-sampled signals: as discussed before (see 2.3.1), the results of this algorithm are not comparable to the one obtained by the gQRS algorithm nor to an accurate adaptation of the Pan-Tompkins. These results remain, anyway, helpful in order to establish a baseline used to read the results obtained from the gQRS algorithm.
2. Original gQRS algorithm on the linearly re-sampled signal: doing so we were able to apply the original gQRS algorithm to the sub-sampled signal at a variety of ε.

3. Python and C-99 implementation on the sub-sampled signals.

As seen in eqs. (1) to (3) the metrics used are Positive Predictivity (PP), Sensitivity (S) and F-Score. As we will see in the next figures, for every threshold ε used and for every algorithm described we compute the average and the dispersion of the described metrics over all the files of the MIT-BIH arrhythmia database. Because the distribution of the results is highly non-normal, we have decided to use the Median Absolute Deviation (MAD, [15]) instead of the classic standard deviation (we can observe the difference between them in Figure 12 and 14).

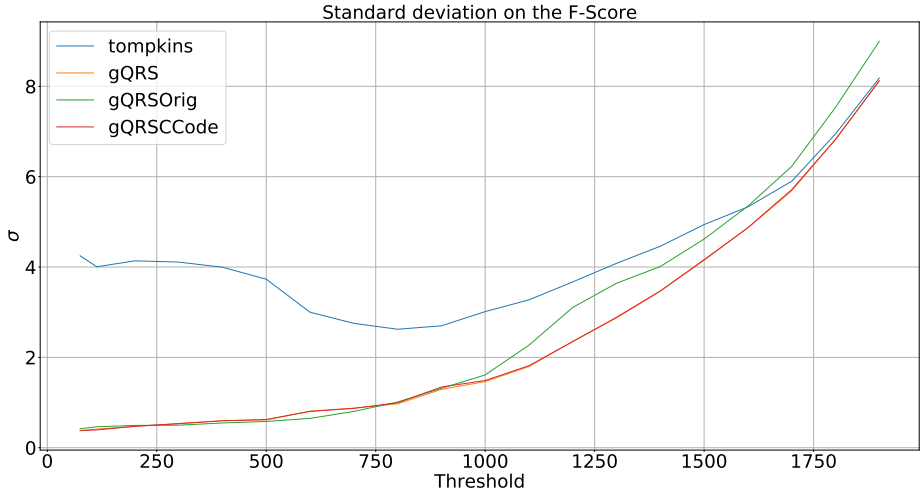

Figure 12: $\sqrt{Var}$ for different thresholds

As we can see from Figures 13 and 14 the Python (gQRS) and C-99 (gQRSCCode) the implementations are almost indistinguishable. Because of this, from here on we will just show the results of the Python version. Not considering the pseudo-Pan-Tompkins, we can observe from Figures 12 and 13 that, even if the original gQRS algorithm has basically the same variance of the one we developed, its overall performance degrades faster than the one designed explicitly to work in the event-based domain, We will explain the reasons at the end of this section.

In Figure 14 It is possible to observe the behaviour at different thresholds of the MAD of the F-Score. As we were expecting, the pseudo-Pan-Tompkins has higher MAD due to its lower capacity to adapt to the signal.
Figure 15 should be considered as the final and most interesting result of this section: even if sensitivity and positive predictivity remain of fundamental interest, from an utilitarian point of view, the F-Score and the MAD confidence interval give us the

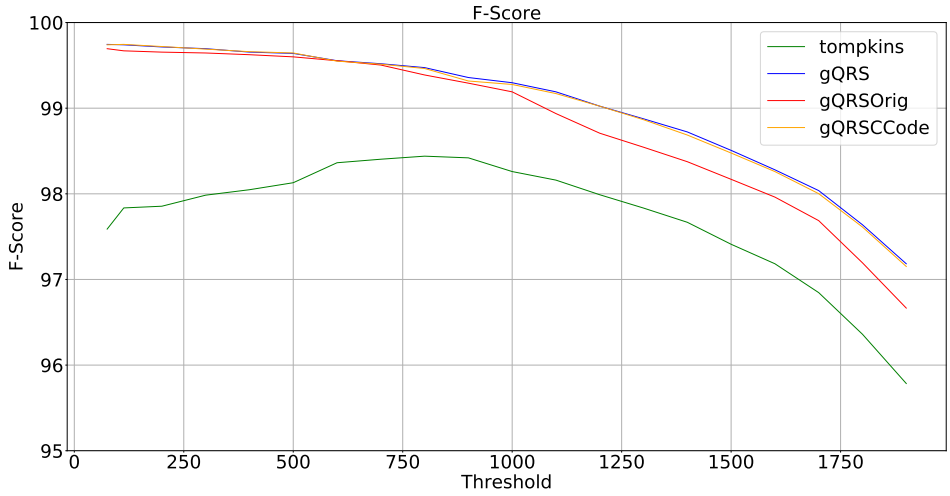

Figure 13: Average F-score results of various versions of gQRS vs. Tompkins. (F-score vs ε)

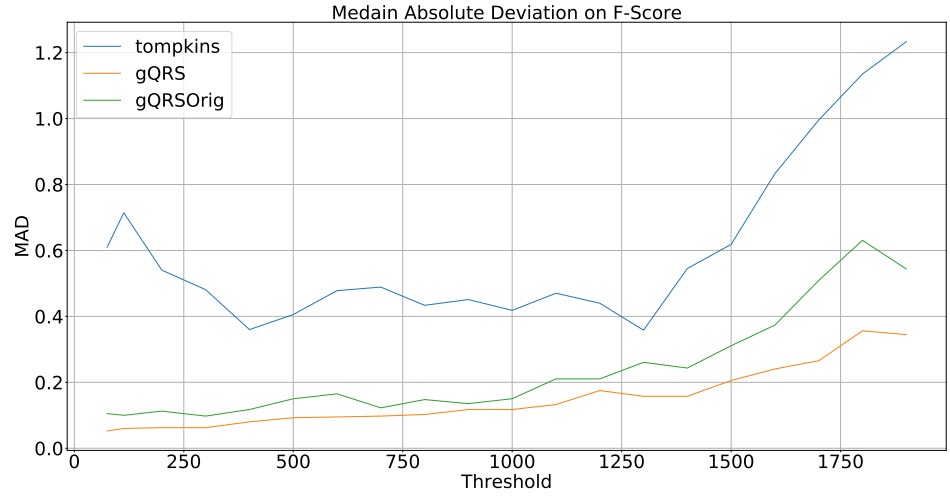

Figure 14: Median Absolute Deviation of the F-Score for different thresholds

metric to use in order to discern the best implementation.

The results shown in Figures 16 and 17 were expected: The gQRS algorithm was designed to prefer the Positive Predictivity over the Sensitivity while the Pan-Tompkins the exact opposite. Moreover, in the adapted gQRS algorithm the section responsible to discern if a peak is from a QRS complex or not (discriminator) was not changed. The small differences that we can observe are caused by the difference in sensitivity: if one of the algorithms detect less (or different) peaks, the adaptive thresholds get modified differently, making the behaviour of the discriminator

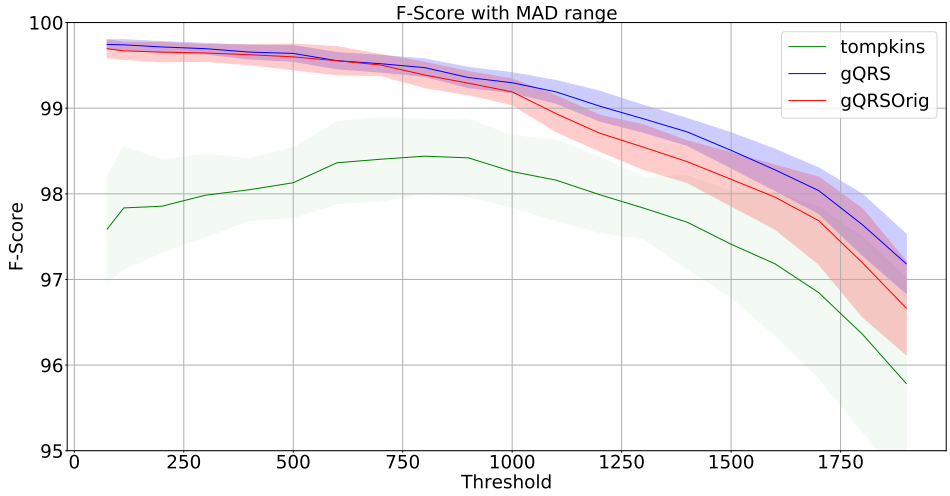

Figure 15: F-Score with confidence bands computed using the MAD

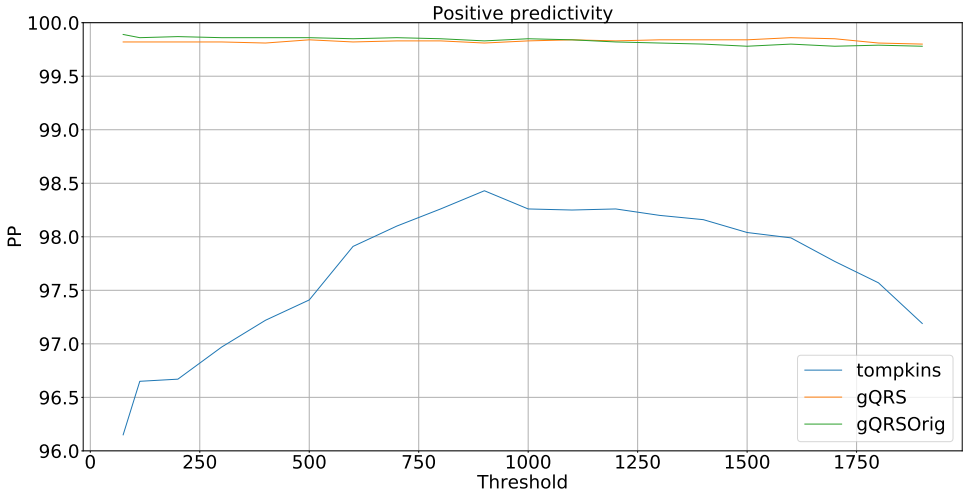

Figure 16: Average Positive Predictivity of the three algorithms

slightly different.

In Figures 18 and 19 we can observe that the original gQRS behaves worse than the one we developed due to the lower sensitivity of it as the threshold increases: the sensitivity of the original gQRS decreases faster than the custom one and even becoming lower than the pseudo-Pan-Tompkins. This is because of the continuous integration on the linearly re-sampled and filtered signal (The original gQRS still has the low-pass IIR filter). While in our version we are sure that the absolute error between the integral on the event-based signal and the true one remains lower

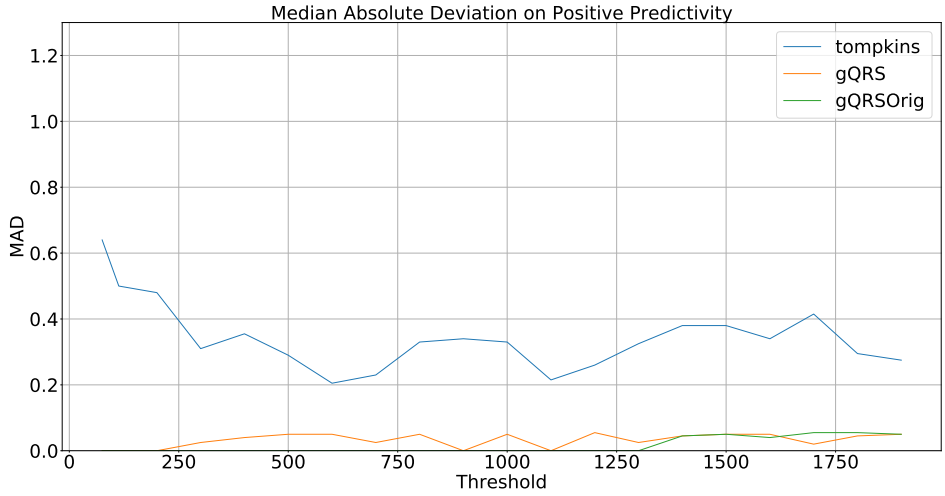

Figure 17: Median Absolute Deviation of the Positive Predictivity for different thresholds

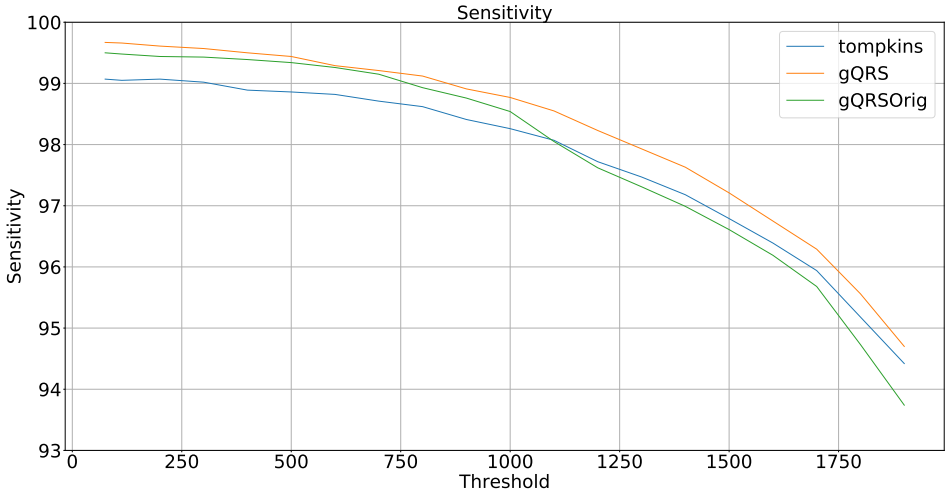

Figure 18: Average Sensitivity of the three algorithm

than a certain ε, this is not true for the original algorithm applied to the linear interpolation.

4.2 Energy consumption results

The PULP implementations of the gQRS algorithm seen in 3.2 behave in different ways, achieving really similar results (in terms of F-score) but with extremely different energy performance. The difference in terms of F-Score ranges from -0.4%

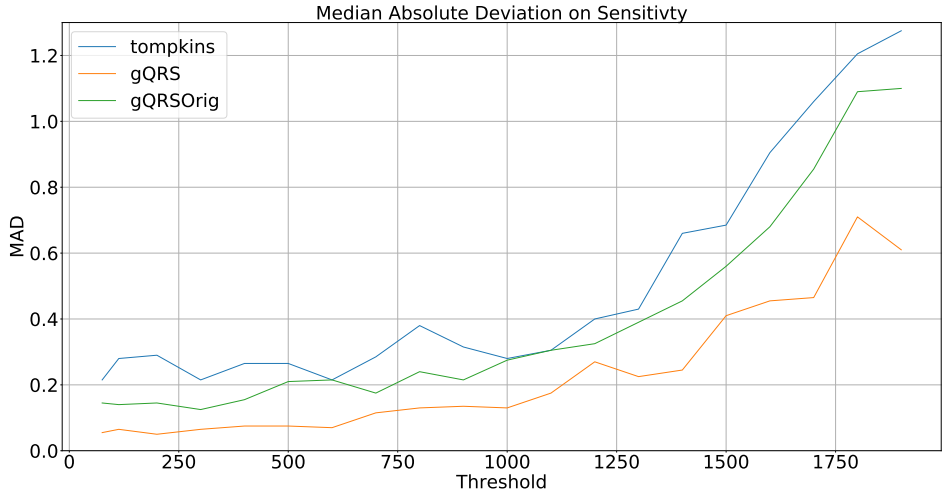

Figure 19: Median Absolute Deviation of the Specificity for different thresholds

in the worst case and +1.2% in the best case. This problem comes from avoiding the use floating-point types on PULP. The overall results remain consistent with the "classic" C and/or Python version. The main performance metric in which we were interested in this phase of our work was the energy consumption. The energy consumption results were obtained using the profiling tools present in the PULP SDK. Such tools allow us to obtain a series of performance measurements, for example, the total number of active cycles, the number of memory access for both L1 and L2, etc. In order to obtain the energy consumption we were interested in the total number of active cycles (the total number of cycles in which the core under exam was executing instructions). This measure can be converted into the used energy with the help of tabulated values that tell us the energy consumption per cycles, the leakage energy, and the energy consumed in sleep mode. Table 1 shows the energy consumption results for the different algorithms, that were obtained considering a sampling frequency $F_s = 360Hz$ for the original gQRS version and $F_s = 17Hz$(average) for the event-based version. As we can see the energy in run is greatly improved with each successive version. However, because even the "vanilla" version of this algorithm is already highly efficient in terms of computation the total time in sleep remains almost constant among all the different implementations. In particular, the measurements obtained in Table 1 come from the processing of 20 seconds worth of data and the sleep time ranges from 19.935 seconds to 19.998 seconds. This makes the energy consumption to be dominated by the leakage current that happens while the platform is in sleep and the energy used for the retention of one of the eight blocks

of L2 memory. It is important to notice that the energy consumption measured is defined by the used technology. In this work we assumed to use the Mr. Wolf distribution developed in 40 nm technology described in [16]. As the used technology is scaled down, the energy consumption for memory retention decrease with it. Moreover, we have seen that the energy consumption is dominated by the fact that we must make retentive, during sleep, at least one of the eight blocks of L2 memory. Because every bank is 64 kBytes, this lead to a big waste of energy for unused memory: the total amount of memory that is strictly needed is dependent on the length of the window we decide to acquire and the used threshold. For the twenty seconds window discussed the total amount of memory needed is less than 1 kByte (considering both the memory for data storage and past results storage). This means that with a more advanced technology and with smaller custom memories we can expect a significant reduction of the energy in the sleep phase. For example, we can achieve 240 times less energy in sleep (for memory retention) with 20 nm techonlogy and a custom 1 kByte L2 memory bank.To achieve the same energy in sleep and in run, the complexity of the Fabric Controller implementation would need to increase around 47 times while the complexity of the Cluster version would need to increase about 120 times. This increase would balance the energy in run and in sleep (but not the time). In this case, the total used energy would be 3.02 mJ. These data tell us that the algorithm we designed achieves high efficiency when compared to the original one (the energy and time passed in Run) while still having computational resources that can be used for the implementation of other algorithms of higher complexity.

	Energy in Run [mJ]	Energy in Sleep [mJ]	Time in run [ms]	Total Energy [mJ]
Original gQRS	0.2534	1.53	44.6	1.78
gQRS FC Implementation	0.0320	1.53	5.64	1.56
gQRS CL Implementation	0.0124	1.53	1.85	1.54

Table 1: Energy usage of the 3 different gQRS algorithm on PULP for the processing of 20 seconds worth data.

From the results seen we can also compute the average energy per processed point (in Run):

1. Original version: $E_{point} = 35nJ$
2. FC Implementation: $E_{point} = 94nJ$
3. CL Implementation: $E_{point} = 36nJ$

5 Conclusions

The general idea behind this work of thesis was to show how it is possible to extract interesting features form an event-based sampled signal and how to use the developed techniques in a low power MCU such as the PULP. In particular, we focused on the detection of the QRS complex in ECG signals. We used the ECG signals contained in the MIT-BIH arrhythmia database. In order to achieve this task we adapted two famous algorithms to work in this environment: the Pan-Tompkins [9] and the gQRS algorithm [3]. The studies on the Pan-Tompkins algorithm were used to understand some principles on how to adapt an algorithm in order to make it work with an event-based signal while the gQRS algorithm was chosen as the target algorithm to implement on the designed MCU due to its higher performance.

The work was divided into 3 main steps: prototyping, implementation, and optimization. In the implementation, we saw how re-writing the mathematical idea behind an algorithm could lead to interesting results and allows us to use that algorithm in a different environment. During the implementation, we were able to use the powerful SDK that the PULP-project distribute as open-source and understood how an algorithm should work on this platform. Finally, the optimization gave us the most interesting results and allowed us to be able to parallelize the code in a multi-core environment.

5.1 Future development

The main problem we found in this project is the amount of energy wasted while in sleep: the gap between the energy required by the algorithm and the energy consumption for memory retention and leakage while in sleep makes the optimized cluster algorithm and the original version essentially identical in terms of wasted power even if the relative improvement (i.e. the energy strictly used to execute the algorithm) is of 20 times when we compare the original gQRS to the multi-core version on the sub-sampled file. This problem can not be solved within the PULP architecture but it allows us to be able to greatly increase the complexity of the algorithm while still being energy efficient. In the foreseeable future, we plan to implement also the QRS complex delineation[17] and then move to T and P wave delineation, heartbeat, and arrhythmia classification.

References

[1] L. Sörnmo and P. Laguna, "Chapter 6 - The Electrocardiogram—A Brief Background," in *Bioelectrical Signal Processing in Cardiac and Neurological Applications* (L. Sörnmo and P. Laguna, eds.), Biomedical Engineering, pp. 411–452, Burlington: Academic Press, 2005.

[2] L. Sörnmo and P. Laguna, "Chapter 7 - ECG Signal Processing," in *Bioelectrical Signal Processing in Cardiac and Neurological Applications* (L. Sörnmo and P. Laguna, eds.), Biomedical Engineering, pp. 453–566, Burlington: Academic Press, 2005.

[3] A. L. Goldberger *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals," *Circulation*, vol. 101, pp. 215–220, June 2000.

[4] S. Raj, K. C. Ray, and O. Shankar, "Development of robust, fast and efficient QRS complex detector: a methodological review," *Australasian Physical & Engineering Sciences in Medicine*, vol. 41, pp. 581–600, sep 2018.

[5] B.-U. Kohler, C. Hennig, and R. Orglmeister, "The principles of software QRS detection," *IEEE Engineering in Medicine and Biology Magazine*, vol. 21, no. 1, pp. 42–57, 2002.

[6] Y. Suzuki, "Suzuki, y.: Self-organizing qrs-wave recognition in ecg using neural networks. ieee trans. neural netw. 6(6), 1469-1477," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 6, pp. 1469–77, 02 1995.

[7] B. Abibullaev and H. D. Seo, "A new qrs detection method using wavelets and artificial neural networks," *J. Med. Syst.*, vol. 35, pp. 683–691, Aug. 2011.

[8] A. Cost and G. Cano, "Qrs detection based on hidden markov modeling," pp. 34 – 35 vol.1, 12 1989.

[9] J. Pan and W. J. Tompkins, "A Real-Time QRS Detection Algorithm," *IEEE Transactions on Biomedical Engineering*, vol. BME-32, pp. 230–236, mar 1985.

[10] Z. Tian, R. Ying, P. Liu, G. Wang, and Y. Lian, "Event-driven analog-to-digital converter for ultra low power wearable wireless biomedical sensors," in

2015 IEEE 11th International Conference on ASIC (ASICON), pp. 1–4, IEEE, nov 2015.

[11] K. Wall and P. E. Danielsson, "A fast sequential method for polygonal approximation of digitized curves.," *Computer Vision, Graphics, & Image Processing*, vol. 28, pp. 220–227, nov 1984.

[12] P. D. S. Andreas Traber, Michael Gautschi, "RI5CY: User Manual." https://www.pulp-platform.org/docs/ri5cy_user_manual.pdf, 2019.

[13] E. Zürich and U. of Bologna, "PULP-Platform." https://www.pulp-platform.org/, 2017. [Online; accessed 20/06/2019].

[14] G. Barozzi, G. Dore, and E. Obrecht, *Elementi di analisi matematica*. No. v. 2 in Elementi di analisi matematica, Zanichelli, 2015.

[15] T. Pham-Gia and T. Hung, "The mean and median absolute deviations," *Mathematical and Computer Modelling*, vol. 34, no. 7, pp. 921 – 936, 2001.

[16] E. Zürich and U. of Bologna, "Mr Wolf details." http://asic.ethz.ch/2017/Mr.Wolf.html, 2017. [Online; accessed 20/06/2019].

[17] T. Teijeiro, P. Felix, and J. Presedo, "A noise robust QRS delineation method based on path simplification," in *2015 Computing in Cardiology Conference (CinC)*, pp. 209–212, IEEE, sep 2015.