Impact of Threshold Computation Methods in Hardware Wavelet Denoising Implementations for Neural Signal Processing

Nicola Carta(✉), Danilo Pani, and Luigi Raffo

DIEE - Deptartment of Electrical and Electronic Engineering, University of Cagliari, Via Marengo 3, 09123 Cagliari, Italy
{nicola.carta,danilo.pani,luigi}@diee.unica.it

Abstract. Wavelet denoising effectiveness has been proven in neural signal processing applications characterized by a low SNR. This non-linear approach is implemented through the application of some thresholds on the detail signals coming from a sub-band decomposition. The computation of the thresholds could exhibit a high latency when involving some estimators such as the Median Absolute Deviation (MAD), which is critical for real-time applications. When a VLSI implementation is pursued for low-power purposes, such as in the neuroprosthetic field, these aspects cannot be overlooked. This paper presents an analysis of the main VLSI hardware implementation figures related to this specific aspect of the signal denoising by wavelet processing. Xilinx System Generator has been exploited as a design and co-simulation tool to ease the hardware development on off-the-shelf FPGA platforms. The MAD estimator has been both combinatorially and sequentially implemented, and compared against the sample standard deviation. The study reveals similar performance on the neural signals but dramatically worse implementation figures for the MAD. The combinatorial version of the MAD actually prevents an efficient implementation on medium-small devices. This result is important to perform a correct implementation choice for implantable real-time systems, where the device size is relevant for an usable realization.

Keywords: Wavelet denoising · Neural signal processing · FPGA · Design tools

1 Introduction

Wavelet denoising (WD) is a non-linear filtering technique usually adopted to remove the background noise added to the signal of interest, especially in presence of a Gaussian source whose spectrum overlaps the useful signal bandwidth. Its effectiveness has been proven in several biomedical signal processing applications [1,19], including neural signals denoising [5]. When a poor signal to noise ratio (SNR) is present, the adoption of WD can help in revealing even hidden

G. Plantier et al. (Eds.): BIOSTEC 2014, CCIS 511, pp. 66–81, 2015.
DOI: 10.1007/978-3-319-26129-4_5

events in the time domain [3]. Algorithmically, it consists of a sub-band decomposition of the signal, thresholding (introducing the non-linearity) and recomposition. Several methods for calculating the thresholds have been presented in the scientific literature, with different performance both in terms of quality (effectiveness) and efficiency. This issue is normally overlooked even though it is actually important when real-time performance is required.

Some specific applications such as neural signal processing for motor/sensory neuroprostheses could benefit from the adoption of WD [3]. In this case, in general, both real-time performance and low power dissipation are required, all the more so when the device is aimed to be implanted. From this perspective, Application Specific Integrated Circuits (ASICs) represent the most powerful implementation platform, even though their design requires highly specific skills and a longer development time [13]. Since the outcome of the design process is quite inflexible, taking into account the quickly mutable environment generated by the advancements of the research in biomedical signal processing, the adoption of tools for automatic creation of hardware description language (HDL) designs can speed up the prototyping phase on Field Programmable Gate Array (FPGA) or ASIC. Examples of such tools are ORCC[1] or Xilinx System Generator[2]. Without specific add-ons, these tools are generally rather ineffective for low-power design [15]. The integration with tools such as Simulink enables a faster development thanks to the possibility of considering the implementation at an higher level.

This paper presents an analysis of the main VLSI hardware implementation figures related to the specific aspect of the threshold estimation in wavelet denoising for neural signal processing. The threshold estimation stage, which must iteratively evaluate the average level of the noise affecting the signal of interest, is marginally considered in the largest part of the applications. Except when a fixed threshold is used [7], the estimation of the standard deviation of the noise is usually required. The Median Absolute Deviation (MAD), known to be a robust estimator of the dispersion in presence of outliers, is compared here to the sample standard deviation in terms of effectiveness and efficiency when the algorithm is implemented in hardware on an FPGA platform, in the light of a perspective development of an implantable neural signal processing ASIC. Both combinatorial and sequential versions of the MAD have been implemented, along with sample standard deviation and a pure software implementation on a MicroBlaze processor. The results in terms of area and latency reveal the poor scalability of the MAD implementations and the comparable effectiveness with simpler approaches, resulting from the evaluation onto an open neural signals database [17].

2 Wavelet Denoising of Neural Signals

WD has been used in neural signal processing since a long time to cancel the background noise that can be approximated to a Gaussian distributed

[1] orcc.sourceforge.net.
[2] www.xilinx.com/tools/sysgen.htm.

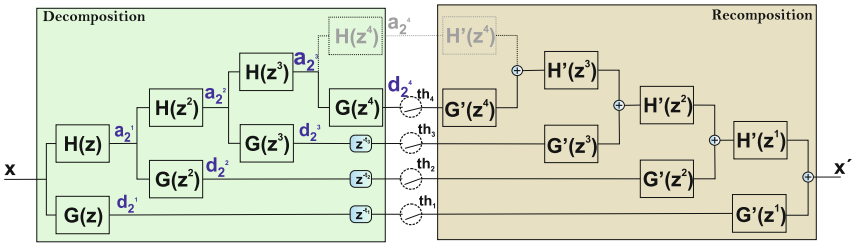


Fig. 1. Block Diagram of the Wavelet Denoising scheme using the à-trous approach.

random source [14]. For some wavelets, WD can be implemented using a system of quadrature mirror Finite Impulse Response (FIR) filters for each stage. The input signal is decomposed in its low-frequency and high-frequency bands, respectively called "approximation" and "detail". The approximation is split again in the same way repeatedly until the $N-$th level of decomposition has been reached and the detail signals are thresholded, before the recomposition is pursued, as can be seen in Fig. 1. The basic analysis elements are a low-pass $H(z)$ and an high pass $G(z)$ filters. Being the Nyquist frequency of the approximation one half of that of the incoming signal, the sample rate can be reduced (decimated approach) so that the same filters can be used in every level without any information loss [9]. Alternatively, the sample rate can be preserved upsampling in each level the filter coefficients of the previous one, in the so called *algorithme à trous* scheme [6]. In practice, the filters at the stage i are simply $H(z^i)$, $G(z^i)$ for the analysis and the mirrored versions for the synthesis. Such a redundant approach leads to the same time resolution in every level [11] and to shift invariance [4]. When the N-th level has been computed, all the details are thresholded either *hard* or *soft*, respectively whether the samples of the detail signals are simply cleared to zero if below the threshold or also the samples above threshold are modified by subtracting the value of the threshold itself. During recomposition, the sample-wise averaging between an approximation and the related detail can be implemented as a simple sum provided that the coefficients of the synthesis filters are multiplied by 0.5.

2.1 The Thresholding Aspects

The choice of the threshold influences the quality of the denoising so much that even data-specific approaches have been presented so far [12]. The threshold can be fixed [7] or adaptive [18], the same or different for all the details. In particular, adaptive thresholds are typically computed estimating the rms or the standard deviation σ of the signal at the different levels of the decomposition and then correcting them through a multiplicative factor. Different scaling factors have been derived and are preferred by different authors, as for the Minimax [3], Stein's Unbiased Risk [8] or Universal [2] methods. The last one, chosen in this work regardless the method used to estimate the standard deviation of the noise, is defined as:

$$\theta = \sigma\sqrt{2\log M} \tag{1}$$

where M is the length of the signal frame in terms of number of samples.

For the estimation of the standard deviation of the noise, due to the robustness to the presence of outliers[3], usually the preferred method is the MAD, defined as:

$$MAD = median_i\left(|X_i - median_j(X_j)|\right) \tag{2}$$

Furthermore, because of the high-pass nature of the detail signals, it is common to implement the MAD as simply the median of the absolute value of the details:

$$\overline{MAD} = median_j\left(|X_j|\right) \tag{3}$$

It has been proved that $MAD \approx 0.6745\sigma$. Block-on-line threshold adaptiveness can be guaranteed exploiting a sliding window approach with variable window length and overlap. For the sake of the comparison between the different threshold estimation techniques in terms of performance and hardware figures, the overlapping parameter is without effects. For the computation of the sample standard deviation exploiting this sliding window approach, starting from the technique used in [16] and thanks to the zero-mean nature of the high-pass detail signals, for each N new input samples in the window, the related sum of squares for the *j-th* decomposition level can be computed as:

$$s_j = \sum_{n=1}^{N} d_j^2[n] \tag{4}$$

and then used to determine σ for the 4-time larger windows as:

$$\sigma = \sqrt{\frac{1}{4N-1}\sum_{k=1}^{4} s_j} \tag{5}$$

Thanks to the sliding window approach, the threshold value is updated every N sampling periods ($M = 4 \times N$). The longer the observation window, the better the estimation accuracy, provided that instantaneous variations (neural spikes) do not influence the threshold computation.

3 Hardware System Design

From an hardware perspective, when the final goal is a low-power embedded architecture, the MAD estimation is effective for multiplier-free systems. Nevertheless, it pays the absence of actual mathematical computations with an algorithmic complexity associated to the required sorting of one half of the block of data. It is also onerous from the memory perspective when the block size

[3] In neural signal processing, the spikes corresponding to the action potentials of the active neurons can be considered as partly composed of outlier samples in recordings with an average SNR.

is huge. For such reasons, the MAD is more suited for off-line processing than for on-line systems [16,20] such as those required in brain-machine interfaces exploiting adaptive thresholds. Threshold adaptation is useful when the noise process in non-stationary, for instance in real-world scenarios when the subject moves in a real environment characterized by different noise sources spread over the space.

For the analysis of the hardware features associated to the different design choices, the "*algorithme à-trous*" has been selected using the simplest Haar wavelet to reduce the memory footprint. The approximation signal at the 4^{th} level has been cleared, so that at 12 kHz the overall processing without the non-linearity introduced by denoising would restrict the bandwidth between 375 Hz and the Nyquist frequency [16]. The FIR filters have been implemented in the Transposed Direct-Form I, not implementing specific optimizations since the filters banks are the same regardless the chosen threshold estimation strategies. The Universal scaling has been selected, as already said. The overlap between adjacent windows has been fixed to three quarters of the window length.

Hereafter, we consider the two alternative solutions implemented for the estimation of the standard deviation of the noise:

- the $\overline{MAD}$ of the signal, using either a combinatorial or an iterative approach;
- the sample standard deviation σ of the signal.

In order to evaluate the different solutions from a hardware perspective, we adopted Xilinx System Generator to speed-up the hardware design and perform accurate co-simulations. The final design can be straightforwardly mapped onto an FPGA board for performance evaluation. In our tests, a Xilinx Virtex-5 LX330 has been chosen as target device for its considerable amount of available resources.

The estimation of the standard deviation of the noise has been performed as described in the previous section. For the $\overline{MAD}$ implementation, on System Generator the absolute value of each input sample is extracted, then the median is computed on the whole incoming window of M input rectified samples and the multiplication by a constant is performed to estimate the standard deviation.

As already said, the median value calculation requires the hardware implementation of a sorting algorithm which represents a costly operation from a hardware point of view. A first possible solution can be the unfolded sorter presented in [2], for which the Simulink model considering windows of $M = 8$ input samples is presented in Fig. 2. The basic sorting cell makes the comparison between two inputs A and B and swaps them if $A < B$. It is possible to demonstrate that, if the comparators work in parallel, $M - 1$ steps are sufficient to properly perform the sorting of M elements. The output is updated in a combinatorial way every time a sample arrives in input at the sampling frequency f_s, after the proper shift of the values saved into the registers needed to prepare the input samples for the processing. The $\overline{MAD}$ is computed as the arithmetic mean of the two central elements of the sorted array for an even number of samples.

This solution clearly presents scalability issues with the enlargement of the observation window. In this case, beyond the penalty associated to the huge

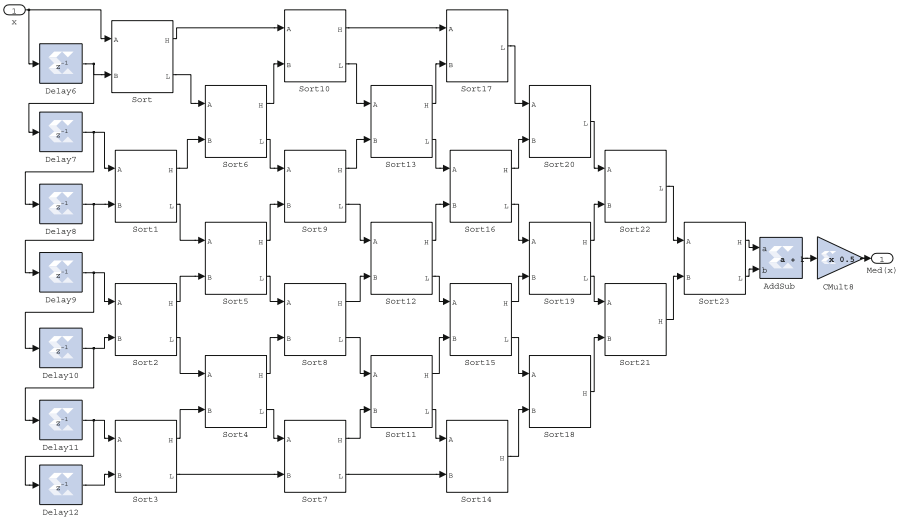


Fig. 2. The Simulink Model of the sorting algorithm for the median estimation using an unfolded combinatorial approach for windows with M=8 samples.

amount of hardware resources, the increasing internal critical path determined by the cascade of comparators largely limits the maximum operating frequency.

To overcome such problems, an iterative (folded) approach to the implementation of the sorting algorithm, able to reuse the same resources at each step, can be used. It is similar to the one proposed in [10] about the Burrows-Wheeler transform but in this work it has been adapted to the wavelet denoising case. In this case, the sorting strategy uses only two levels of comparators. At the beginning, the swaps are performed only for the registers related to odd adjacencies, activating only the first level of comparators. If the vector is not yet sorted, at the next iteration only the comparators of the second level are active, and so on until the sorting process is completed. It is possible to demonstrate that the number of necessary steps is $M/2$ if M is the number of samples to sort. Figure 3 shows the iterative scheme.

The *swp* signal coming out from the comparator block is used to specify that the two inputs have been swapped. The samples in input to the parallel sorter, belonging to each observation window, are temporarily saved into a single-port memory. Immediately after the last sample of the window has been saved in this memory, its content is copied into the registers and the sorting process can start. When all the *swp* signals are equal to 0 during the last necessary iteration, the input vector is correctly sorted. A finite state machine, one for each *Threshold Estimator* block (i.e. one for each decomposition level), is used to control the various phases of the process.

In order to compare the hardware characteristics of these models of threshold estimation based on the $\overline{MAD}$ against those of a traditional sample standard

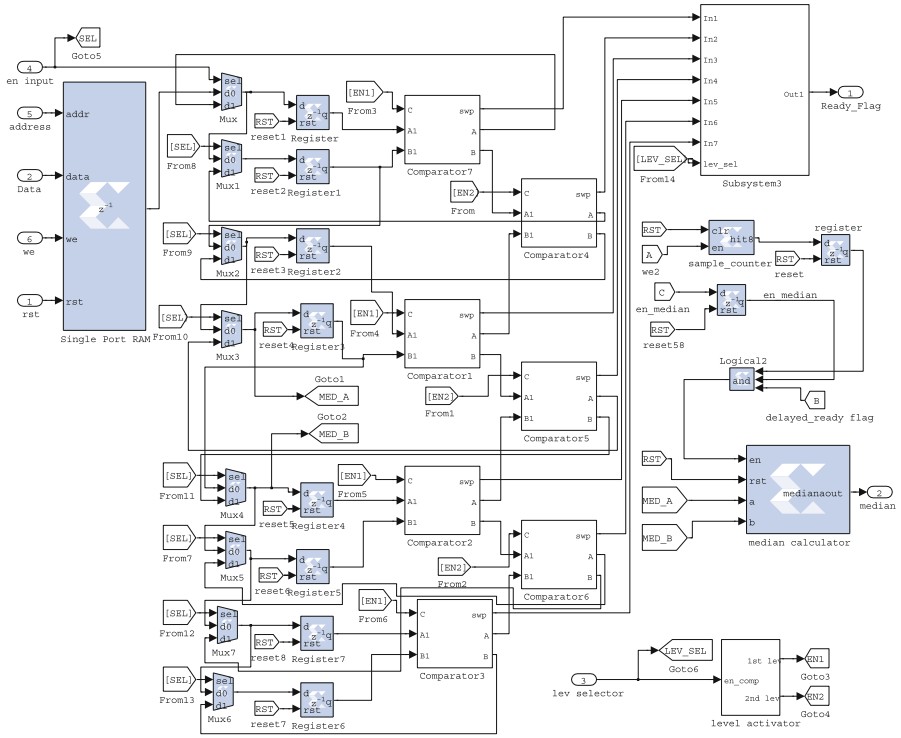


Fig. 3. The Simulink Model of the sorting algorithm for the median estimation using an iterative approach for windows with M=8 samples.

deviation as described above, an hardware model has been designed also for such an approach.

Figure 5 shows the Simulink model for this implementation. Every time an input sample arrives, it is squared and added to the current value of s_j. After N samples, the final value of s_j is saved in one of the 4 locations of the single-port RAM used as circular buffer in order to determine the correct value of σ over the sliding window. It involves the hardware implementation of the squared root as the final processing stage for the threshold estimation able to work with the proper internal fixed point representation. To this aim, we chose to exploit an iterative approach that is possible to demonstrate is able to converge in less than 64 iterations in case of input samples with a 16-bit fixed-point representation using 15 bits of fractional part, achieving a good level of accuracy. Its pseudocode is presented hereafter.

```
max = sqrt_in;
min = 0;
for(int i=0; i<ITERS; i++)
     avg = (max+min)/2;
     avg_2 = avg*avg;
```

```
    if(avg_2 > sqrt_in) max = avg;
    else min=avg;
sqrt_out = min;
```

It operates as a bisection research method, dividing by 2 at each iteration the range of possible values in which the result is searched until the algorithm converges to a specific output value. This algorithm determines the hardware implementation shown in Fig. 4.

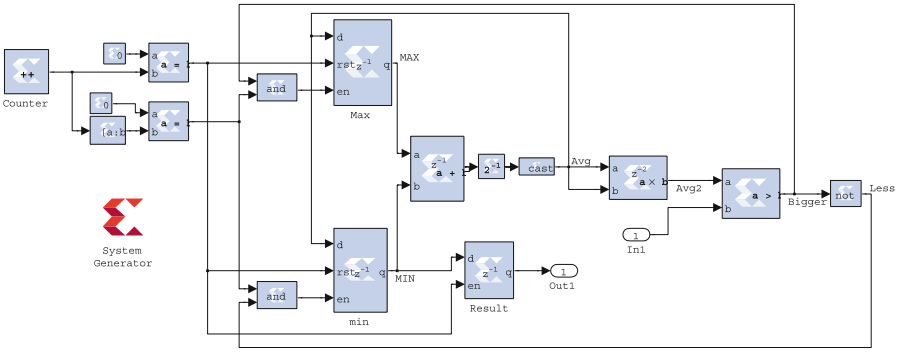


Fig. 4. The Simulink Model of the Squared Root block.

Regardless the chosen approach, the value of the threshold θ obtained through the Universal scaling approach is sent in input to the *Thresholder* block, able to apply the hard thresholding on the detail samples. It should be also considered that the value of θ is different for the various levels.

4 Results

Before analysing the results in terms of hardware resources which are necessary for the different solutions presented above, such solutions have been evaluated from a functional perspective. A publicly available dataset of simulated neural signals obtained from real physiological action potentials recorded from animals at the central nervous system level has been used [17]. The synthetic signals are obtained by linearly mixing an artificial sequence of real spikes from three neurons to other spikes at random times and amplitudes, representative of the background activity (of tunable intensity) of the neurons at a greater distance from the recording electrodes. The sampling frequency has been scaled to 12 kHz and the useful bandwidth is declared to be in the range 300 Hz - 3 kHz.

4.1 Performance Analysis

Figure 6 shows in the first row the neural signal with a low level of background noise used as input for the two hardware implementations based on the calculation of the $\overline{MAD}$ and of the sample standard deviation (the two versions of the

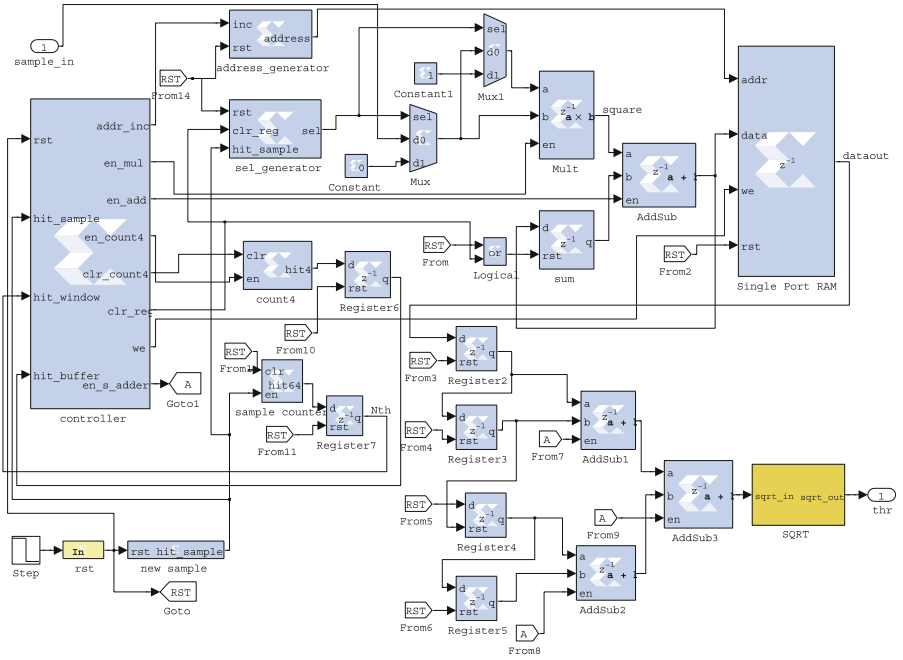


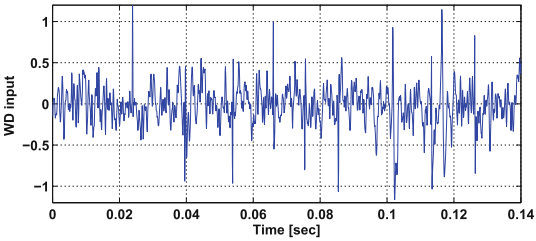
Fig. 5. The Simulink Model of the Threshold Estimator block based on the calculation of the sample standard deviation.

one implementing the $\overline{MAD}$ produce the same results). The next rows present the related outputs considering observation windows of $M = 4 \times 64$ samples.

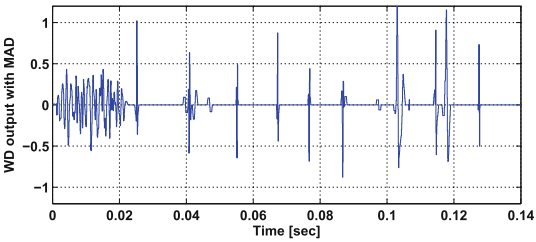
It is possible to see that for both solutions, the wavelet denoising is able to remove, after an initial transient, the background noise added to the neural signal without cutting significant spikes. The same performance can be achieved using the same input signal but with a stronger background noise for which it is difficult to identify the various spikes on the raw signal, as can be shown in the first row of the Fig. 7. Even using neural signals with very low SNR, the two implementations behave similarly preserving the relevant spikes.

Then, we considered the possibility of enlarging the observation window in order to provide a more significant frame for computing the statistics on the signal. For example, Fig. 8 shows the outputs of the two solutions using $\overline{MAD}$ and sample standard deviation in the case of windows of $N = 128$ samples and a low level of background noise. The initial transient is obviously longer in comparison to the previous cases.

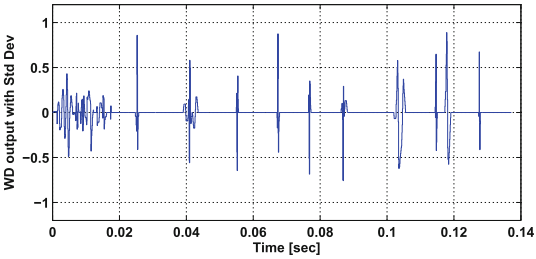
We also analysed the trend of the thresholds in output from the same decomposition level for different observation window lengths, for the two hardware solutions. The aim is to verify which is the minimum value of N, considering a sliding window length of $4 \times N$, that allows obtaining a good denoising.



(a) Input Signal



(b) WD output using the $\overline{MAD}$-based threshold



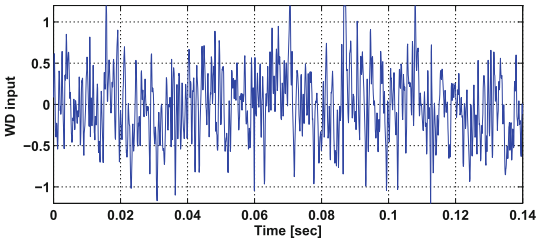
(c) WD output using the sample standard deviation based threshold

Fig. 6. Wavelet Denoising input and outputs: low level noise, N=64 samples.

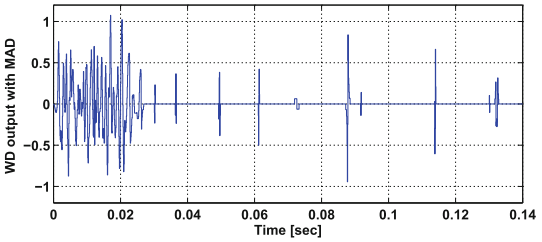
As can be noticed from Fig. 9, after a variable transient period according to the chosen value of N, the longer the observation window the better the stability of the threshold, not influenced by the presence of the neural spikes of interest. In fact, in the case of $N = 128$, the threshold estimation assumes an almost constant trend; the goal should be that of selecting the solution which provides the best compromise in terms of threshold estimation and required hardware resources.

4.2 Hardware Implementation Results

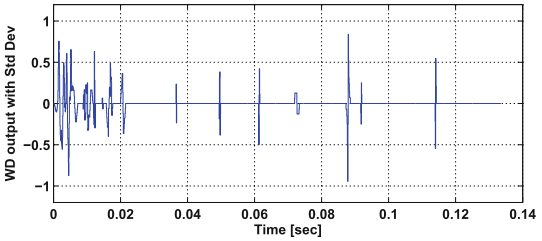
Synthesis results on a Xilinx FPGA Virtex-5 LX330 are presented in Table 1, which shows the percentage of available slices and Look-up Tables (LUTs) needed



(a) Input Signal



(b) WD output using the $\overline{MAD}$-based threshold

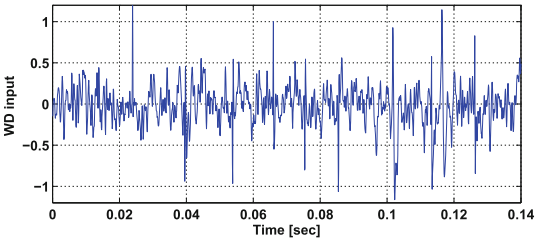


(c) WD output using the sample standard deviation based threshold

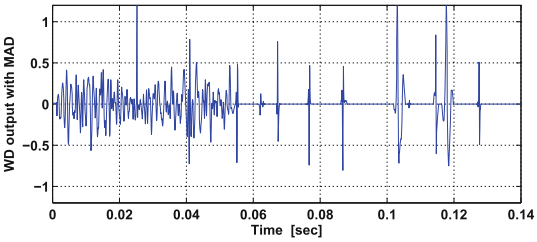
Fig. 7. Wavelet Denoising input and outputs: high level noise, N=64 samples.

for the three considered threshold estimation blocks only, since the remainder of the wavelet denoising implementation is the same regardless of this stage.

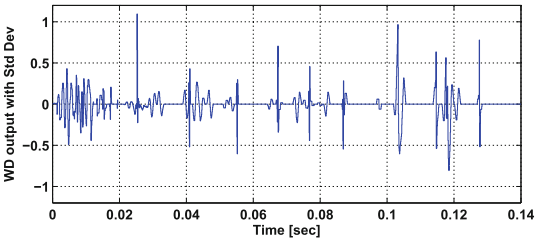
The solution based on the combinatorial (unfolded) $\overline{MAD}$ implementation, as highlighted in Table 1, is absolutely inefficient, taking into account it has been presented only for $N = 8$, with the usual 4-times larger observation window. A rough estimation of the hardware resources required in case of $N = 32$ would lead to more than 330kLUT over the 207360 available ones, thus exceeding the considerable amount of physical resources on the target FPGA. The huge amount of LUTs, compared to the folded version, is incompatible with a real implementation in the context of this application, taking into account that the observation window length should be large enough to properly estimate the statistics of a signal sampled at 12 kHz.



(a) Input Signal



(b) WD output using the $\overline{MAD}$-based threshold



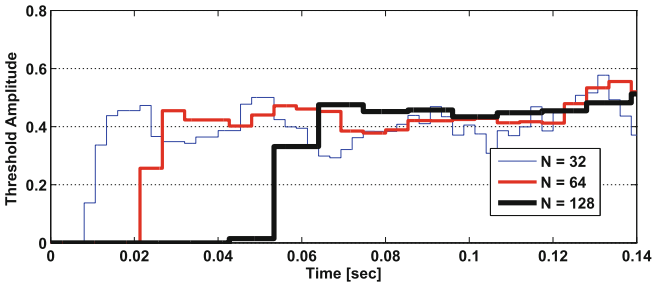
(c) WD output using the sample standard deviation based threshold

Fig. 8. Wavelet Denoising input and outputs: low level noise, N=128 samples.

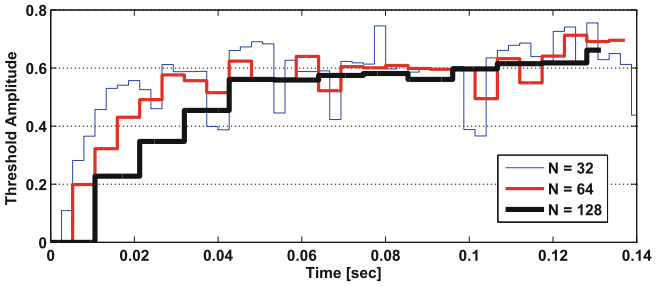
FPGA synthesis results demonstrate that the wavelet denoising solution based on the threshold estimation by the sample standard deviation allows minimizing the necessary hardware resources regardless the length of the observation window. Furthermore, the usage of slices and LUTs of the $\overline{MAD}$-based solution, even using a folded approach, is clearly incompatible with an efficient implementation of this processing stage, especially compared to the same data related to the sample standard deviation based implementations.

4.3 Latency Execution Evaluation

To evaluate the efficiency of the various approaches at the enlargement of the observation window, a cycle-accurate profiling has been performed. It has been carried out on the first-level detail obtained when processing the low-noise signal used for the performance evaluation. These results have been compared to the



(a) $\overline{MAD}$-based solution



(b) sample standard deviation based solution

Fig. 9. Threshold variation over time using different lengths of the observation window.

corresponding values of execution latency of a totally-software solution. To this aim, an equivalent C code for each solution has been implemented on a MicroBlaze processor and mapped on the same FPGA, analysing the latency by means of a counter which is enabled when the last detail sample of the current data block is received and disabled when the threshold has been computed.

Obviously, when implementing the folded $\overline{MAD}$ approach, the latency is data-dependent and related to the number of swaps performed during the sorting of the absolute values passed in input to the processing stage, as can be seen in Table 2. This is not true for the sample standard deviation, which with the proposed approach always requires 68 cycles in hardware. For the $\overline{MAD}$, the number of swaps on a real signals follows a Gaussian distribution, as evaluated through the Lilliefors tests. For this reason, for different values of M we reported in Table 2 the minimum and maximum observed number of swaps, the mean and standard deviation. The range of variation, mean value and standard deviation of the execution latency is also provided for both the hardware and the software solutions, evaluating them on the whole detail signal and verifying even in this case a Gaussian distribution by the same statistic test. As can be seen, the number of execution cycles is always higher for the software implementation of the $\overline{MAD}$, and grows with the size of the observation window.

Table 1. FPGA synthesis results for the *Threshold Estimator* varying the length of the observation window.

	N	$f_{max}[MHz]$	Slice Registers	LUTs
Sample std	32	417.34	205 / 207360 (0.07 %)	142 / 207360 (0.04 %)
	64	416.61	206 / 207360 (0.07 %)	144 / 207360 (0.04 %)
	128	416.02	207 / 207360 (0.07 %)	147 / 207360 (0.04 %)
unfolded $\overline{MAD}$	8	417.08	558 / 207360 (0.27 %)	20270 / 207360 (9.77 %)
folded $\overline{MAD}$	32	242.78	2493 / 207360 (1.20 %)	7164 / 207360 (3.45 %)
	64	246.70	4932 / 207360 (2.38 %)	14377 / 207360 (6.93 %)
	128	221.42	9806 / 207360 (4.73 %)	28861 / 207360 (13.91 %)

Table 2. Comparison of execution latency for the various approaches with respect to a totally-software solution.

		$\overline{\mathbf{MAD}}$			
		max	min	avg	std
M=32	Swaps	412	88	247	45
	lat. HW sol.[cycles]	139	59	114	19
	lat. SW sol.[cycles]	21070	16210	18595	679
M=64	Swaps	1498	531	997	134
	lat. HW sol.[cycles]	294	209	239	26
	lat. SW sol.[cycles]	82960	68455	75455	2004
M=128	Swaps	5426	2637	3987	381
	lat. HW sol.[cycles]	600	360	518	69
	lat. SW sol.[cycles]	325240	283405	303755	5712

5 Conclusions

Despite the ASIC implementation of digital signal processing algorithms is usually the preferred choice when low-power requirements and high processing speed are needed, the straightforward implementation of the best algorithmic solution exploited in the field could lead to unsatisfactory results. This paper analysed the particular case of the threshold computing for wavelet denoising algorithm. This part of the algorithm is usually marginally considered, but could be in the critical path when real-time update of the threshold is required.

In this paper, comparisons between a sample standard deviation and the widespread $\overline{MAD}$ reveals similar functional performance with dramatically better characteristic of the former in terms of hardware implementation, regardless the $\overline{MAD}$ is implemented as a combinatorial trellis as suggested by some authors or in a more efficient folded version. A latency analysis also reveals the superior performance of the sample standard deviation, jointly to its data-independence, which is an important aspect for real-time implementations. The paper also stresses the

benefit of using hardware-software co-simulations tools such as Xilinx System Generator for rapid prototyping and verification on FPGA, which represents a value added for the research in rapidly evolving fields such as neural engineering.

Acknowledgements. The research leading to these results has received funding from the Region of Sardinia, Fundamental Research Programme, L.R. 7/2007 "Promotion of the scientific research and technological innovation in Sardinia", CRP-60544, ELoRA Project.

References

1. Anand, C.S., Sahambi, J.S.: Wavelet domain non-linear filtering for MRI denoising. Magn. Reson. Imaging **28**(6), 842–861 (2010)
2. Bahoura, M., Ezzaidi, H.: FPGA-implementation of discrete wavelet transform with application to signal denoising. Circuits Syst. Sig. Process. **31**(3), 987–1015 (2012)
3. Citi, L., Carpaneto, J., Yoshida, K., Hoffmann, K.P., Koch, K.P., Dario, P., Micera, S.: On the use of wavelet denoising and spike sorting techniques to process electroneurographic signals recorded using intraneural electrodes. J. Neurosci. Methods **172**(2), 294–302 (2008)
4. Cohen, A., Kovacevic, J.: Wavelets: the mathematical background. Proc. IEEE **84**(4), 514–522 (1996)
5. Diedrich, A., Charoensuk, W., Brychta, R., Ertl, A., Shiavi, R.: Analysis of raw microneurographic recordings based on wavelet de-noising technique and classification algorithm: wavelet analysis in microneurography. IEEE Trans. Biomed. Eng. **50**(1), 41–50 (2003)
6. Holschneider, M., Kronland-Martinet, R., Morlet, J., Tchamitchian, P.: A real-time algorithm for signal analysis with the help of the wavelet transform. In: Combes, P.J., Grossmann, P.A., Tchamitchian, P.P. (eds.) Wavelets, pp. 286–297. Springer, Heidelberg (1990)
7. Kuzume, K., Niijima, K., Takano, S.: FPGA-based lifting wavelet processor for real-time signal detection. Sig. Process. **84**(10), 1931–1940 (2004)
8. Mahmoud, M.I., Dessouky, M.I.M., Deyab, S., Elfouly, F.H.: Signal denoising by wavelet packet transform on FPGA technology. special issue of ubiquitous computing and communication. J. Bioinform. image **3**, 54–58 (2008)
9. Mallat, S.: Multifrequency channel decompositions of images and wavelet models. IEEE Trans. Acoust. Sign. Process. **37**(7), 2091–2110 (1989)
10. Martinez, J., Cumplido, R., Feregrino, C.: An FPGA-based parallel sorting architecture for the burrows wheeler transform. In: International Conference on Reconfigurable Computing and FPGAs, ReConFig 2005 (2005)
11. Martínez, J., Almeida, R., Olmos, S., Rocha, A., Laguna, P.: A wavelet-based ECG delineator: evaluation on standard databases. IEEE Trans. Biomed. Eng. **51**(4), 570–581 (2004)
12. Medina, C., Alcaim, A., Apolinario Jr., J.A.: Wavelet denoising of speech using neural networks for threshold selection. Electron. Lett. **39**, 1869–1871 (2003)
13. Montani, M., Marchi, L.D., Marcianesi, A., Speciale, N.: Comparison of a programmable DSP and FPGA implementation for a wavelet-based denoising algorithm. In: Proceeding of IEEE 46th Midwest Symposium on Circuits and Systems. vol. 2, pp. 602–605 (2003)

14. Oweiss, K.G., Anderson, D.J.: Noise reduction in multichannel neural recordings using a new array wavelet denoising algorithm. Neurocomputing **38–40**, 1687–1693 (2001)
15. Palumbo, F., Carta, N., Pani, D., Meloni, P., Raffo, L.: The multi-dataflow composer tool: generation of on-the-fly reconfigurable platforms. Journal of Real-Time Image Processing pp. 1–17 (2012)
16. Pani, D., Usai, F., Citi, L., Raffo, L.: Impact of the approximated on-line centering and whitening in OL-JADE on the quality of the estimated fetal ECG. In: Proceedings of the 5th International IEEE/EMBS Conference on Neural Engineering (NER), pp. 44–47 (2011)
17. Quiroga, R.Q., Nadasdy, Z., Ben-Shaul, Y.: Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. Neural Comput. **16**(8), 1661–1687 (2004)
18. Radovan, S., Saša, K., Dejan, K., Goran, D.: Optimization and implementation of the wavelet based algorithms for embedded biomedical signal processing. Comput. Sci. Inf. Syst. **10**, 502–523 (2013)
19. Singh, B.N., Tiwari, A.K.: Optimal selection of wavelet basis function applied to ECG signal denoising. Digit. Sign. Proc. **16**(3), 275–287 (2006)
20. Zhang, M., Deng, R., Ma, Z., Zhang, M.: A FPGA-based low-cost real-time wavelet packet denoising system. In: Proceedings of 2011 International Conference on Electronics and Optoelectronics (ICEOE). vol. 2, pp. V2–350-V2-353 (2011)