Data processing system for denoising of signals in real-time using the wavelet transform

Hilton de O. Mota[1] and Flávio H. Vasconcelos[2]

[1]Electrical Engineering Post-graduate Program – PPGEE – UFMG,
UNI-BH – Centro Universitário de Belo Horizonte, Belo Horizonte, Brazil
hilton@cpdee.ufmg.br

[2]Electrical Engineering Department,
UFMG – Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
fvasc@ufmg.br

Abstract *— This paper describes a system able to acquire, process and eliminate noise in continuous streams of data in real-time. The signal processing algorithms were based on the discrete wavelet transform and employ a new approach to deal with border problems, allowing to process the data continuously. The system was implemented using a DSP coupled to a digitizer through its external memory bus to guarantee deterministic behavior while maintaining some degree of flexibility in its configuration. The achieved performance and potential applications are discussed at the end of the text.*

1 Introduction

In the last two decades denoising techniques based on the wavelet transform have come out as one of the most useful alternatives to those based on the Fourier transform. The great advantage wavelets do offer is the ability to extract frequency information from the signal while maintaining, to some degree, its time information. Moreover, the transform allows acting locally in the signal with minimal interference on its vicinity, so creating processing alternatives never achievable by other filtering mechanisms. When wavelets are used in real-time applications some particularities have to be considered. Due to the complexities involved in the transform, there are some tendencies to associate it to dedicated hardware architectures, like FPGAs or ASICs [2][3]. These approaches can bring some benefits, mainly related to the processing speed, but frequently lack some flexibility on parameters configuration and are harder to modify in the field.

This paper presents a description of a data acquisition system aimed at processing digital data in real-time using wavelet techniques. The text begins with a brief description of the wavelet transform, followed by the algorithms that were developed to allow its application over continuous streams of data. The objective of this study is to provide a system capable of processing the data as fast as possible while offering some degree of flexibility in the choice of the wavelet, the number of decomposition levels, the type of denoising technique and threshold level. These two criteria, speed and flexibility, lead to the several hardware and software decisions explained in the following. The structures used to implement the system are presented in sections 3 and 4, followed by an analysis of performance and some obtained results.

2 The wavelet transform and the discrete wavelet transform

2.1 The wavelet algorithms

The wavelet transform (WT), as all time-frequency related transforms, was an attempt to overcome the limitations imposed by the Fourier transform inability to decompose a signal while still maintaining its time information. The discrete counterpart of the WT, named discrete wavelet transform (DWT) [4], arose in the context of the multiresolution analysis theory and is implemented by a filter bank that decomposes the signal in successively coarser approximations and details, as shown in figure 1. H_1 and H_0 are, respectively, high-pass and low-pass complementary filters, $d_{j+k,n}$ are detail coefficients at level $(j+k)$ and $a_{j+k,n}$ are approximation coefficients. The circles containing down-arrows sided by 2's represent the decimation by two operation.

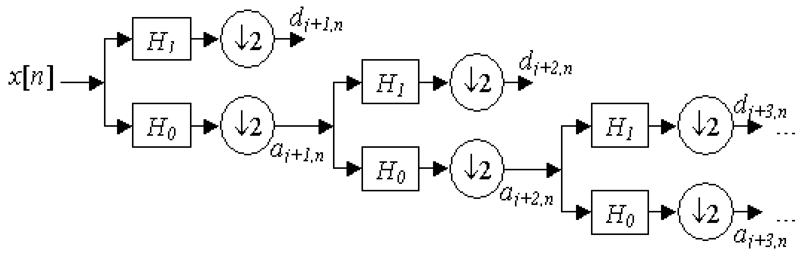


Figure 1: filter bank for a 3 level DWT decomposition.

The inverse discrete wavelet transform (IDWT) is obtained by a quadrature filter bank that is the mirror of the one shown in figure 1. This filter bank begins from the lowest level, which represents the coarser approximation, and progressively adds more and more details, until the original signal is recovered. The filter structure is shown in figure 2, where G_1 and G_0 are, respectively, the high and low-pass mirror filters of H_1 and H_0.

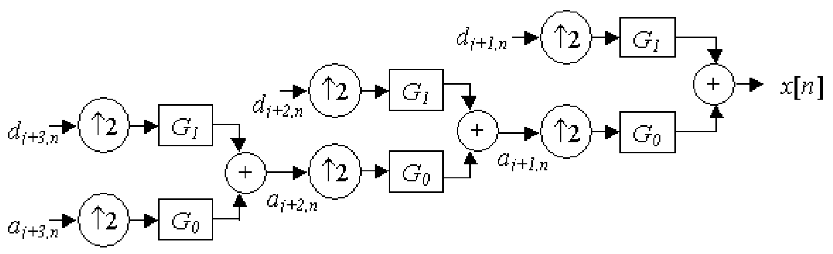


Figure 2: filter bank for the 3 level reconstruction.

The decimations are replaced by interpolations, shown as up-arrows followed by 2. For a specific wavelet filter of size *M*, the algorithms perform at most *2.N.M* multiplications and *2.N.(M-1)* additions, so the complexity bound is independent of the maximum number of decomposition levels.

The way to implement the filter banks depends mainly on two factors: the intended use of the transform and the hardware characteristics. When the hardware is composed by only one processor, it is necessary to develop algorithms that mimic their cascade structure. The first algorithm developed for this task was named Pyramid Algorithm (PA) [4], but there are a number of others, usually fitted to specific data.

In this work the choice was to use a single DSP to perform both the DWT, IDWT and filtering algorithms, what has allowed a good balance between performance and flexibility. The system is under a prototype stage, so the purpose is to gather information about processing speed and efficacy of various denoising algorithms. The results obtained from these studies can be used in later stages to specify an improved version involving other approaches like multiple processors or an hybrid system containing both ASICs and programmable devices.

2.2 The RunningDWT and RunningIDWT algorithms

The PA is an algorithm fitted to be performed by just one processor, so, when the data come in a continuous stream, it is usually advantageous to do a block processing to minimize the overhead caused by the data transfer. This imposes the need to deal with the borders of the sections, what can be made by several well-known techniques [1][5]. However, these techniques can incur in some overcomputation or false information, so it is necessary to use them with some care [2][7].

The algorithms RunningDWT and RunningIDWT were developed to deal with the borders in a way as transparent as possible [8]. They were inspired in the overlap-save convolution method [9] and the Recursive Pyramid Algorithm (RPA) [10]. Figure 3 shows how a 3 level decomposition is performed by these algorithms, using a wavelet filter of size $M=4$. The input is automatically segmented in sections of size 2^J, where J denotes the number of desired levels (in this example $J = 3$, so the sections have $2^3 = 8$ data). x_n denotes the n-th input data, d^j_n denotes the n-th detail of level j and a^j_n the n-th approximation.

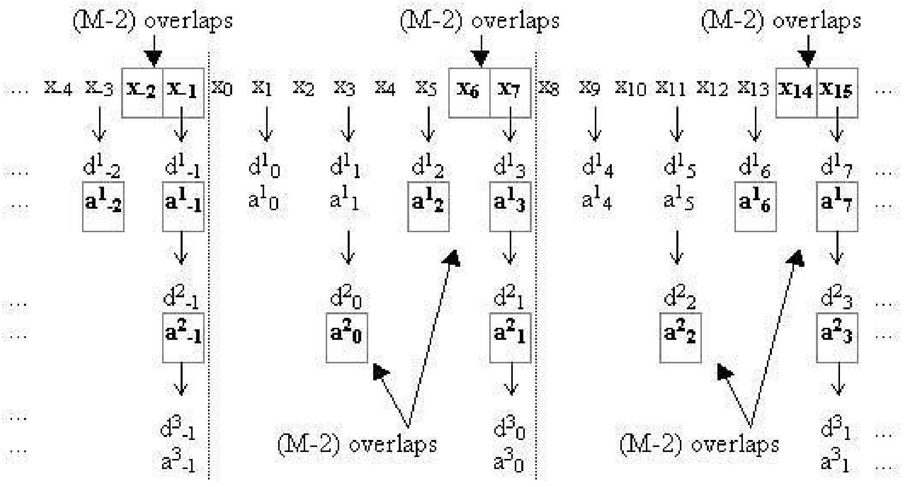

(M-2) overlaps
(M-2) overlaps
(M-2) overlaps
... x-4 x-3 x-2 x-1 x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ...
... d1-2 d1-1 d1 0 d1 1 d1 2 d1 3 d1 4 d1 5 d1 6 d1 7 ...
... a1-2 a1-1 a1 0 a1 1 a1 2 a1 3 a1 4 a1 5 a1 6 a1 7 ...
... d2-1 d2 0 d2 1 d2 2 d2 3 ...
... a2-1 a2 0 a2 1 a2 2 a2 3 ...
... d3-1 d3 0 d3 1 ...
... a3-1 a3 0 a3 1 ...
(M-2) overlaps
(M-2) overlaps

Figure 3: three level decomposition using the RunningDWT algorithm.

The RunningDWT uses a matrix that keeps in its lines overlapping segments of each input section and the succeeding decomposed levels. The matrix has (*J+1*) lines, each with *M* cells. The first line stores the input samples while the others store the approximations calculated at each level while the algorithm evolves. The details are delivered as the output of the algorithm. Figure 4 shows the sequence of operations performed for the decomposition of the first section (x_0 through x_7) of the above example, supposing the matrix is initially zeroed.

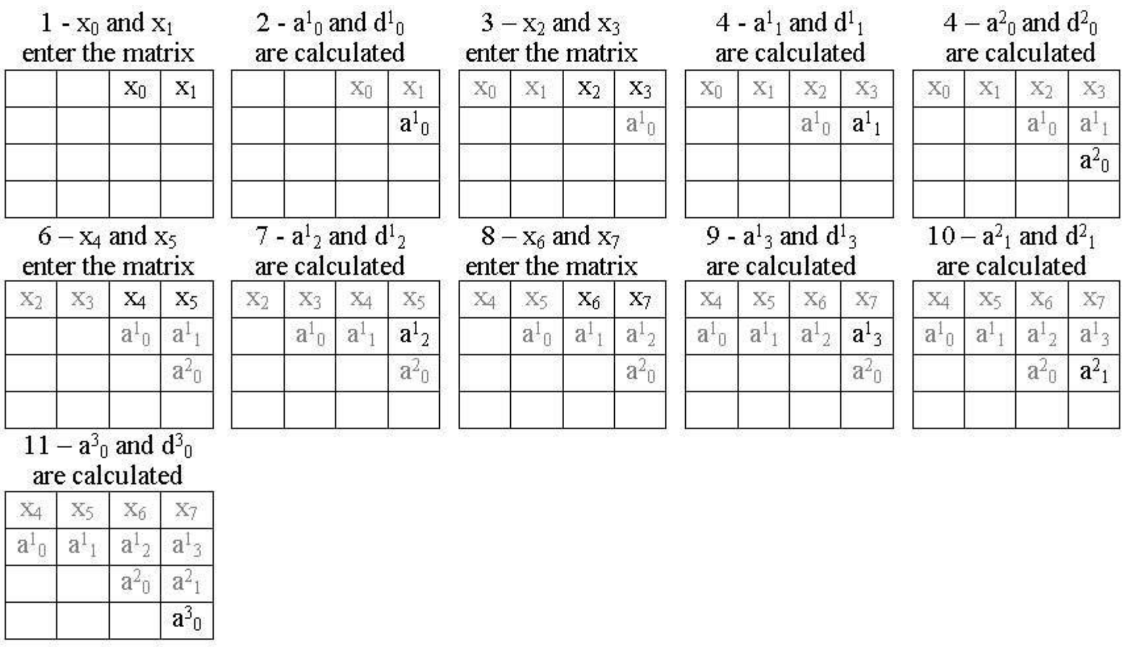

1 - x0 and x1 enter the matrix
2 - a10 and d10 are calculated
3 – x2 and x3 enter the matrix
4 - a11 and d11 are calculated
4 – a20 and d20 are calculated
6 – x4 and x5 enter the matrix
7 - a12 and d12 are calculated
8 – x6 and x7 enter the matrix
9 - a13 and d13 are calculated
10 – a21 and d21 are calculated
11 – a30 and d30 are calculated
x0
x1
x2
x3
x4
x5
x6
x7
a10
a11
a12
a13
a20
a21
a30

Figure 4: sequence of operations performed by the RunningDWT algorithm.

When the decomposition of the second section begins (x_8 through x_{15}) the matrix already holds the overlapping segments that resulted from the processing of the first section, so the border between them becomes transparent.

After the decomposition of each section, the algorithm delivers the details of each level and the approximation of the last level in the order shown in (1).

$$\begin{array}{l} \vdots \\ output\ of\ \sec tion\ 0 = d_0^1, d_1^1, d_0^2, d_2^1, d_3^1, d_1^2, d_0^3, a_0^3 \\ output\ of\ \sec tion\ 1 = d_4^1, d_5^1, d_2^2, d_6^1, d_7^1, d_3^2, d_1^3, a_1^3 \\ output\ of\ \sec tion\ 2 = d_8^1, d_9^1, d_4^2, d_{10}^1, d_{11}^1, d_5^2, d_2^3, a_2^3 \\ \vdots \end{array} \qquad (1)$$

All approximations computed in intermediary levels are thrown away, except those retained by the matrix.

To do the reconstruction without relying on "padding" solutions it is necessary to take into consideration the delay introduced by every filter that composes the filter bank [8]. The only way to do that is reconstructing in the order shown in figure 5. The coefficients obtained from the decomposition of one section are used to reconstruct the data of previous sections. The actual section will be reconstructed only after incoming sections be decomposed, thus causing an overall delay on the reconstructed signal.

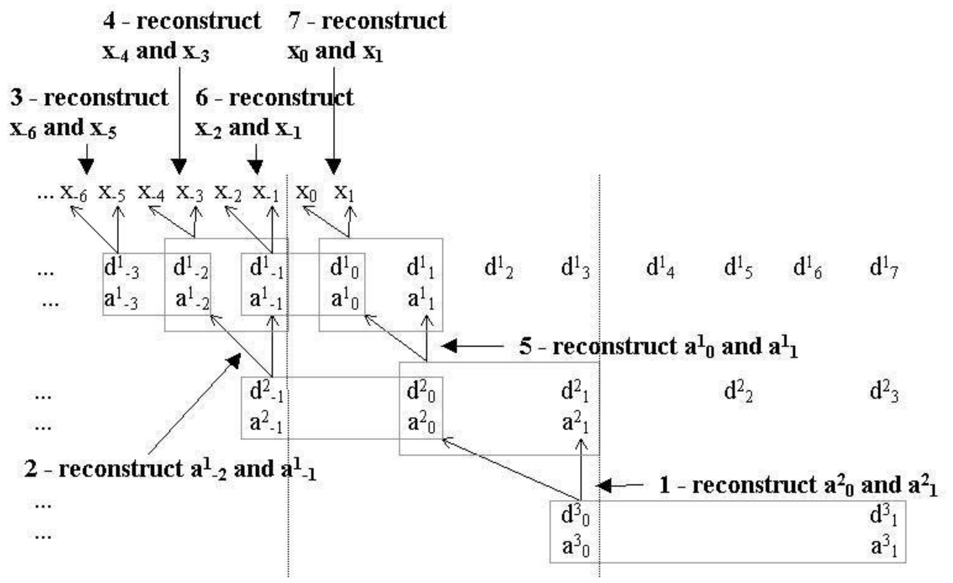

4 - reconstruct x-4 and x-3
7 - reconstruct x0 and x1
3 - reconstruct x-6 and x-5
6 - reconstruct x-2 and x-1
5 - reconstruct a¹0 and a¹1
2 - reconstruct a¹-2 and a¹-1
1 - reconstruct a²0 and a²1

Figure 5: reconstruction after a 3 level decomposition using the RunningIDWT.

The RunningIDWT uses 2 matrixes, one to store the details furnished by the RunningDWT (named *D*) and the other to store the approximations that are progressively reconstructed (named *A2*). Matrix *A2* has *J* lines, each with *(M+2)/2* cells. *D* also has *J* lines, but each has its own size since it is necessary to store a variable number of details at each level. All lines have a structure composed by (*M-2*)/*2* + *additional cells*, where the first term is responsible for the overlaps and the additional cells are computed as follow:

$$\begin{aligned} &\textit{additional cells}(J) = 1; \\ &\textit{additional cells}(j) = M + [\textit{additional cells}(j+1) - 1]\cdot 2 = \\ &\qquad = \left(2^{J-j} - 1\right)\cdot (M-1) + 1, \qquad J > j > 0. \end{aligned} \tag{2}$$

The reconstruction process follow a structure similar to the one performed by the RunningDWT, but in reverse order. For every *M/2* details and *M/2* approximations of the last level the algorithm reconstruct two approximations of the last but one level, two approximations of the last but two level and so on, until the original data are recovered.

Besides allowing the decomposition and reconstruction transparently over the borders, the RunningDWT and RunningIDWT have the same computational load of the original PA. One disadvantage they have is that the matrixes used by the RunningIDWT can exceed the storage of the PA if the decomposition is carried to deeper levels. At first sight this could be a problem, but since in most applications a decomposition down to the 6th or 7th level is fairly sufficient, it is possible to overcome this problem through the judicious sectioning of the input, maintaining the storage in acceptable levels. In the processing of continuous streams of data, the size of the sections will be determined by a trade-off between the available memory and the desired number of levels. At this moment the system works with vectors of size 1024, it is possible to decompose the input to a maximum of 10 levels.

The wavelet filters chosen for this application were all members of the Daubechies' families of orthogonal wavelets [1][4]. This choice was made because they have the most compact support for a given number of vanishing moments, so allowing the lowest computational loads. The family members DB4 to DB20 were implemented, but it is easy to introduce new members as necessary (the numbers following the DB abbreviature denotes the support of the wavelet and, so, the number of taps its filter has).

The methods for denoising with wavelets follow a generic structure in which the original data are transformed by the DWT into a time-scale coefficient space and each coefficient is compared to a threshold level. The coefficients are modified if they are greater or smaller than this level and, afterwards, are used to reconstruct the signal. Up to now the system realizes soft and hard-thresholding [1], but one of the proposals in this work is to evaluate the efficiency of other thresholding techniques.

3 Hardware configuration

The hardware used in this development is shown as a block diagram in figure 6.

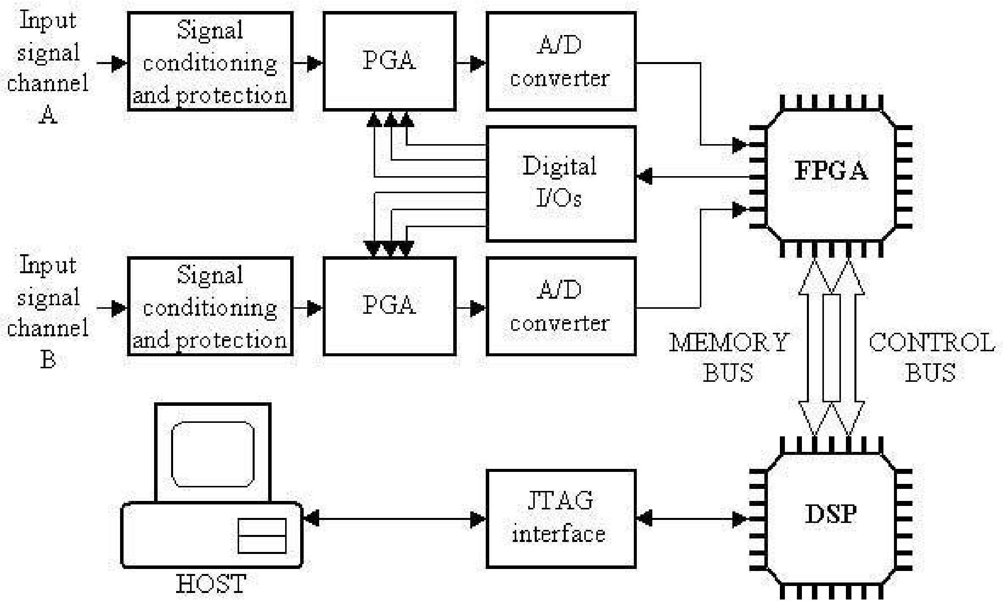


Figure 6: schematic representation of the hardware used in the system.

The input signals are fed into the system through signal conditioning and protection blocks, which are dependent on the application. The signals are then passed through Programmable Gain Amplifiers (PGA) that allow the adjustment of amplification or attenuation via software.

The system offers two channels simultaneously sampled by independent A/D converters, allowing a maximum sample rate of 70 MS/s with 12 bits of resolution. The A/Ds are controlled by a FPGA device, which interfaces to the DSP through its external memory and control buses. The samples of channel A and B are grouped into one single 32 bit word, stored in the FPGA FIFO and transferred to the DSP in one single read operation. Though the maximum sample rate of the A/Ds is 70 MS/s, the DSP bus limits the real-time data transfer at approximately 17 MS/s.

The DSP is responsible for the configuration, data transfer control and processing. A fixed-point device was chosen because they usually offer more processing speed than floating-point units (although they impose their own limitations [3]). The processor is a 720 MHz superscalar device with 2 independent register banks, 2 independent data-paths with 8 parallel functional units, 2 levels of cache with up to 1 MB of internal memory, four 64-bit internal busses and packed data processing capability (SIMD architecture). The chip interfaces to a host computer through a JTAG interface both for programming and data presentation.

4 Software configuration

4.1 System overview

Figure 7 shows an overview of the control and processing software.

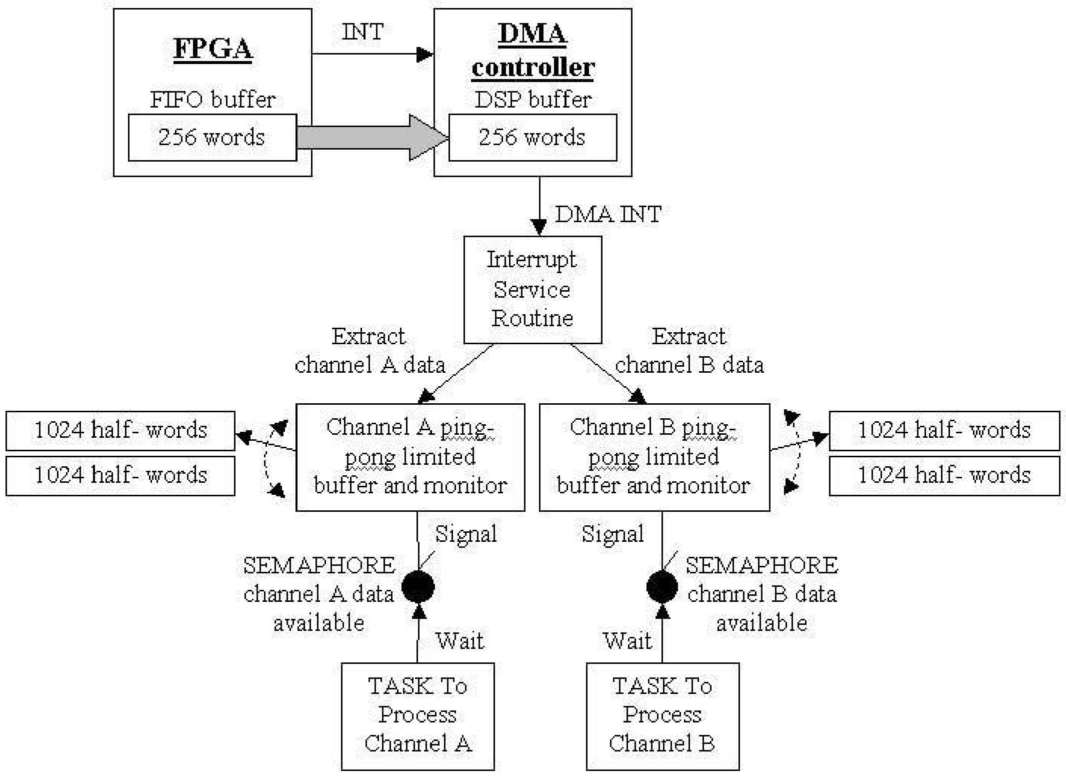

FPGA
FIFO buffer
256 words
INT
DMA controller
DSP buffer
256 words
DMA INT
Interrupt Service Routine
Extract channel A data
Extract channel B data
1024 half- words
1024 half- words
Channel A ping-pong limited buffer and monitor
Channel B ping-pong limited buffer and monitor
1024 half- words
1024 half- words
Signal
SEMAPHORE channel A data available
Wait
Signal
SEMAPHORE channel B data available
Wait
TASK To Process Channel A
TASK To Process Channel B

Figure 7: overview of the control and processing software.

Data transfer is controlled by DMA, which generates an interrupt at every 256 words reading. The DMA interrupt service routine (ISR) performs the separation of data of channel A and B and store them in two 1024 cells ping-pong buffers. These become full only after 4 FPGA interrupts, what allows the reduction of the overhead imposed by the control structures once the processing starts only when a larger amount of data is available. The ping-pong buffers are also monitor buffers, so the operations of writing and reading are mutually exclusive.

Every time the ISR tries to write to the buffer the later automatically tests if there is available room to store the data. If so, the ISR proceeds with the writing, otherwise it is advised and can take the necessary steps to correct the problem. When one task tries to read data that are not ready, it is blocked until one of the 1024 point vectors becomes available. Finally, the system continuously monitors the state of the ping-pong buffers and the FPGA FIFO using tasks that run in the processor's idle loop. This avoids overflowing and guarantee that the processing is being made in real-time.

Data processing is performed by 2 concurrent tasks, one for each channel. The data from channel B is used to synchronise the acquisition to the mains power system, so only channel A data goes through the DWT.

5 Results

The development process of a real-time system usually goes through a number of phases where it is gradually converted from a generic and architecture independent implementation to a high performance and architecture dedicated one. As one of the objectives of this work was to provide a system capable of processing the data as fast as possible, the control and processing algorithms were progressively refined using the phases shown in figure 8. As the system was restricted to a single processor, the refinement went on just to the phase of assembly coding.

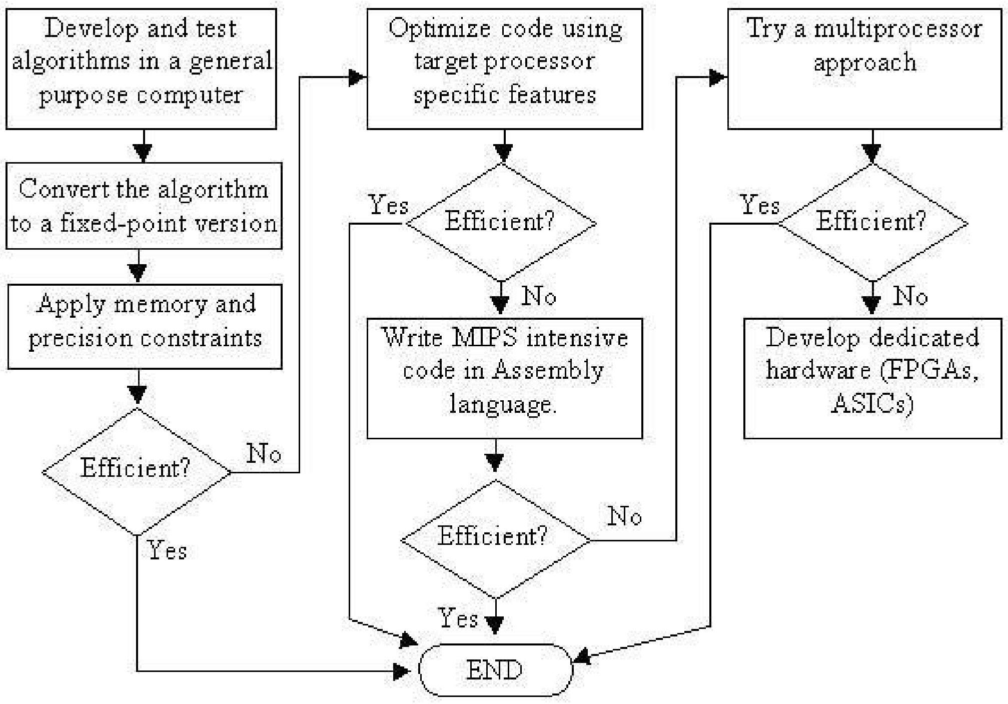

Develop and test algorithms in a general purpose computer
Convert the algorithm to a fixed-point version
Apply memory and precision constraints
Efficient?
No
Yes
Optimize code using target processor specific features
Efficient?
Yes
No
Write MIPS intensive code in Assembly language.
Efficient?
No
Yes
Try a multiprocessor approach
Efficient?
Yes
No
Develop dedicated hardware (FPGAs, ASICs)
END

Figure 8: phases for algorithm optimisation.

The first stages allow the highest performance gains, reaching up 300 to 400 % of speed-up. They involve optimisation through the following procedures:

- optimisation of memory access by transferring multiple data in one single operation. This is possible because the processor has a wide memory bus and specialised instructions to perform wide transfers;
- explicit use of the SIMD instructions the processor offers, which work with up to four operands in one single operation;
- provision of additional information to allow optimisation of processor pipeline. This information allows the compiler to use a special set of instructions dedicated to pipelined execution.

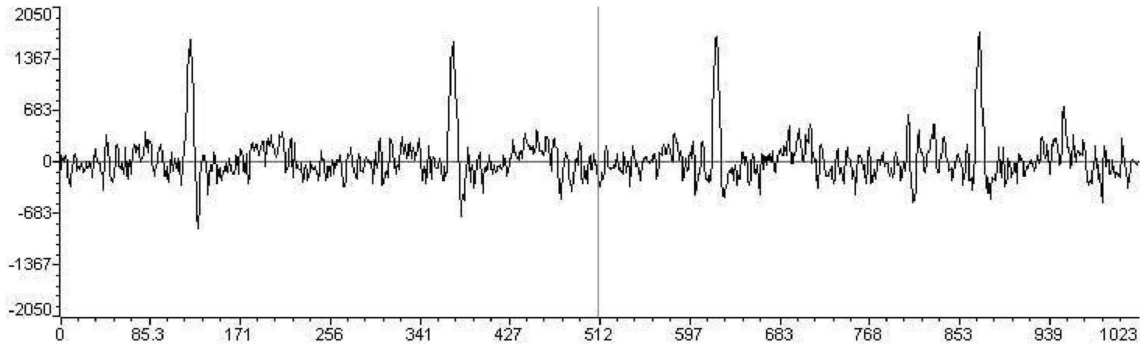
Table 1 gives the worst case execution times (WCET) for the main algorithms as measured by the DSP's high-resolution internal clock.

Algorithm name	Execution time
RunningDWT	144 μs
RunningIDWT	156 μs
DMA ISR (processing and storage)	1.70 μs

Table 1:WCET of the main algorithms.

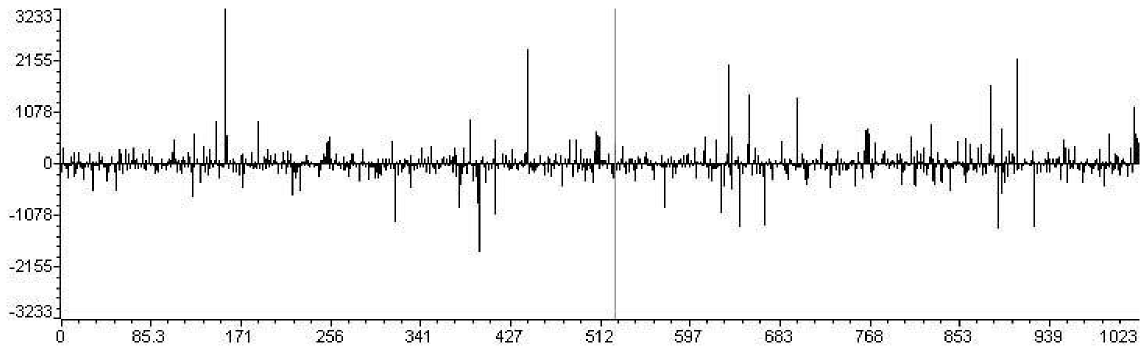
The attained performance allowed real-time processing with sample rates up to 3 MS/s and 94 % of processor usage. Daubechies 4 wavelet and 8 levels of decomposition were used as comparison parameters. When using Daubechies 20 it is possible to go up to 1.7 MS/s in real-time. As expected, the performance is highly dependent on the chosen wavelet since this implies in changing the number of taps of the filters.

Below there is an example of the denoising capabilities of the system. The original signal is a cardiogram like plot added to random noise. The Daubechies 6 wavelet was used and the decomposition was carried down to the 8th level. Hard-thresholding was used as the denoising method.

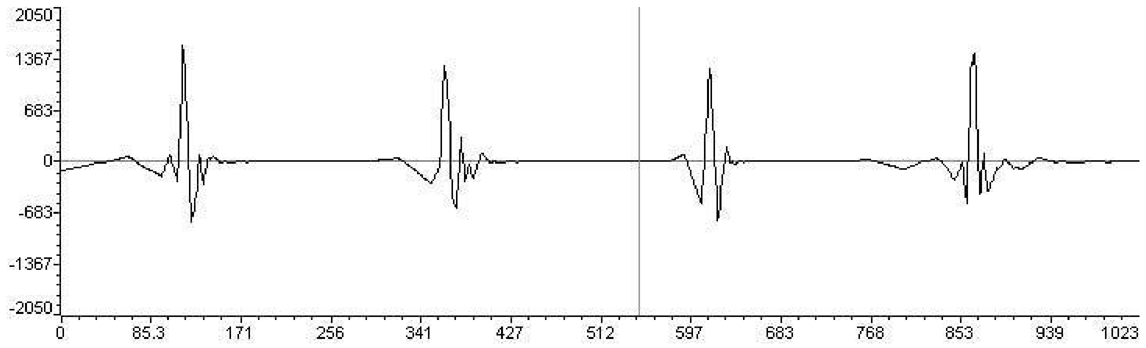


a) Input data.

Figure 8: example of denoising of a signal (vertical axis: A/D output value; horizontal axis: sample number).



b) Wavelet coefficients.



c) Reconstructed signal after denoising.

Figure 8: continued.

6 Conclusion

This paper described the development of a real-time data acquisition and processing system aimed at the denoising of signals using the wavelet transform. The criteria that guided the development were both processing speed and flexibility, so a DSP-based architecture was chosen to optimize both. Data digitalization was accomplished by two independent A/D converters connected to the DSP bus through a FPGA device. This architecture, associated to software optimizations, allowed the achievement of sample rates as high as 3 MS/s while still processing in real-time. As can be seen in table 1, the system's bottleneck are the processing algorithms, while the data transfer and storage are responsible by less than 1% of the computational load. Therefore, it is safe to suppose that it is possible to increase the performance by the use of a multi-processor approach.

To enable the processing of the continuous stream of data two modified DWT and IDWT algorithms were implemented. They have as main feature the ability to decompose and reconstruct the signal without relying on extension techniques. This allowed the transparent processing along the vector borders and eliminates the overcomputation and false information that usually appears when extension techniques are used.

Denoising was made through the use of hard and soft-thresholding. These techniques showed to be very hard to apply due to the fixed threshold level and empirical adjustments, so it is necessary to use them with some caution. As the system is under development the next phases will involve the study of more refined methods of thresholding. In despite of this, the wavelet techniques showed to be very efficient as a complementary tool to the ones based on the Fourier theory.

Potential applications of this system include on-line monitoring of high voltage equipment, transmission line surge detection and location mechanisms, automation and control systems, image processing systems, pattern recognition and so on.

Acknowledgments

The authors thank CAPES - Coordenação de Pessoal de Nível Superior - and FAPEMIG - Fundação de Amparo à Pesquisa do Estado de Minas Gerais - for sponsoring this work.

References

[1] P. S. Addison, *The illustrated wavelet transform handbook*. Philadelphia, PA: Institute of Physics Publishing, 2002.

[2] M. Ferretty and D. Rizzo, “Handling borders in systolic architectures for the 1-D discrete wavelet transform for perfect reconstruction,” *IEEE Transactions on Signal Processing*, vol. 48, n. 5, May 1995.

[3] Ackenhusen, J. G. *Real-time signal processing*. Upper Saddle River: Prentice Hall Inc. 1999

[4] S. Mallat, *A wavelet tour of signal processing*. 2^{nd} ed., San Diego, CA: Academic Press, 1998.

[5] O. Rioul and P. Duhamel, “Fast algorithms for discrete and continuous wavelet transforms,” *IEEE Transactions on Information Theory*, vol. 38, n. 2, March 1992.

[6] Hubbard, B., The world according to wavelets: the story of the mathematical technique in the making. 2nd ed., Natick, MA: A K Peters, Ltd., 1998.

[7] C. Taswell and K. McGill, “Algorithm 735: wavelet transform algorithms for finite-duration discrete-time signals,” ACM Transactions on Mathematical Software, vol. 20, n. 3, September 1994.

[8] Mota, H. and Vasconcelos, F. “Discrete Wavelet Transform decomposition and reconstruction algorithms for continuous streams of data,” *IEEE Transactions on Signal Processing*, submitted for publication in December 2004.

[9] Mitra, S. *Digital signal processing: a computer based approach*. New York, NY: The McGraw-Hill Companies, Inc. 1998.

[10] M. Vishwanath, “The recursive pyramid algorithm,” IEEE Transactions on Signal Processing, vol. 42, n. 3, March 1994.