FPGA Implementation of an Adaptive Filter Robust to Impulsive Noise: Two Approaches

Alfredo Rosado-Muñoz, Manuel Bataller-Mompeán, Emilio Soria-Olivas, Claudio Scarante, and Juan F. Guerrero-Martínez, *Member, IEEE*

***Abstract*—Adaptive filters are used in a wide range of applications such as echo cancellation, noise cancellation, system identification, and prediction. Its hardware implementation becomes essential in many cases where real-time execution is needed. However, impulsive noise affects the proper operation of the filter and the adaptation process. This noise is one of the most damaging types of signal distortion, not always considered when implementing algorithms, particularly in specific hardware platforms. Field-programmable gate arrays (FPGAs) are used widely for real-time applications where timing requirements are strict. Nowadays, two main design processes can be followed for embedded system design, namely, a hardware description language (e.g., VHDL) and a high-level synthesis design tool. This paper proposes the FPGA implementation of an adaptive algorithm that is robust to impulsive noise using these two approaches. Final comparison results are provided in order to test accuracy, performance, and logic occupation.**

***Index Terms*—Adaptive filters, field-programmable gate array (FPGA), hardware design languages, high-level synthesis (HLS), impulsive noise.**

I. Introduction

ADAPTIVE systems are a fundamental tool in digital signal processing (DSP). Adaptive filters are appropriate in temporal variant environment applications where statistical parameters of signals change over time and, additionally, nonlinear characteristics are foreseen in the analyzed or predicted process. A typical application in communication systems is the multipath distortion channel resulting in intersymbol interference (ISI); ISI also brings on higher bit error rate [1]. In this case, it is necessary to design a filter to compensate channel distortion; adaptive equalization techniques have been widely studied and tested. This example belongs to a group of applications known as prediction, where two types exist, namely, future value (forward prediction) or past value prediction (backward prediction). Other prediction applications of adaptive filtering are speech processing or spectral signal estimation. Active noise cancellation is also an important field where adaptive filtering is used [2]. System identification is also an important area in control systems; in this case, adaptive filtering is used for transfer function identification of an unknown system by means of its inputs and outputs. Acoustic environment modeling for sonar applications is an example [3].

The structure of an adaptive algorithm consists of a temporal variant system where the transfer function directly depends on the statistical characteristics of the input signal, an algorithm connecting the inputs, and a signal informing about the error obtained by the adaptive system. Multiple adaptive algorithms exist [4], [5], but most of them assume that the adaptive system obtains an error following a Gaussian distribution. However, in many real problems, the noise encountered is more impulsive than the one predicted by a Gaussian distribution. Examples for this problem are underwater acoustic noise, low-frequency atmospheric noise, and many types of man-made noise [6]. In addition, the presence of outliers, either at the input signal or at the desired signal, may cause the adaptive algorithm to become unstable. For this reason, different adaptive algorithms that are robust to impulsive noise have been proposed [7]–[9] in order to provide proper operation.

Usually, impulsive noise cancellation is achieved by means of independent filter techniques. Impulsive noise suppression in speech is very important for clear voice communications, e.g., using a statistical estimator [10] or in room acoustics using a digital filter using fuzzy reasoning [11]. In digital motor control systems, thyristor power converters need accurate synchronization, and impulsive noise might cause malfunction. Here, a multistage signal processing using a finite-impulse response (FIR) predictive filter is used [12]. For electrocardiogram signal processing, impulsive noise suppression is carried out using nonlinear M-filters, namely, median and myriad filters [13]. In image processing, impulsive noise removal is an important research area, e.g., in video broadcasting [14] and image filtering [15], seldom applying an adaptive filtering technique.

Due to computational requirements, hardware implementation of adaptive systems is not always straightforward, restricting the range of applications where they can be applied. However, a high-performance low-cost system on chip implementations of DSP algorithms is receiving increased attention. Implementation platforms of DSP algorithms range from an application-specific integrated-circuit (ASIC) custom chip to general-purpose processor or DSP microprocessors. While DSP microprocessors provide flexibility and programmability for implementation of a large set of algorithms, their design is suboptimal for specific applications due to the sequential execution that limits possibilities for parallel operation. Moreover,

Manuscript received May 30, 2008; revised January 29, 2009; accepted April 6, 2009. Date of publication June 5, 2009; date of current version February 11, 2011.

A. Rosado-Muñoz, M. Bataller-Mompeán, E. Soria-Olivas, and J. F. Guerrero-Martínez are with the Digital Signal Processing Group, Department of Electronic Engineering, University of Valencia, 46100 Valencia, Spain (e-mail: alfredo.rosado@uv.es; manuel.bataller@uv.es; emilio.soria@uv.es; juan.guerrero@uv.es).

C. Scarante was with the University of Padova, 35131 Padova, Italy (e-mail: scarante@dei.unipd.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2009.2023641

power consumption is high for portable application usage, and an internal hardware structure is not optimized. On the other hand, ASIC could provide the solution that meets all the constraints. However, they lack the flexibility existing in a DSP processor or field-programmable gate array (FPGA) and leave no room for reconfigurability of their circuitry. Hence, the design and development process for an ASIC can be both time consuming and expensive, being considered only when high-volume manufacturing is foreseen or very strict applications require such a design.

Nowadays, the use of FPGAs is increasing. They are powerful hardware devices, combining the main advantages of ASIC and DSP processors, since they provide both a programmable and a dedicated hardware solution [16], [17], which makes them very attractive devices for rapid prototyping. Even if the final design will be implemented in an ASIC, it is usual to use an FPGA as the prototyping device due to factors such as time and cost [18]. Moreover, an FPGA is more efficient in power consumption, an advantage for battery-operated systems, and, for the same application, requires less clock system speed compared to a DSP or a general-purpose processor, offering better electromagnetic compatibility properties. Industrial applications are taking advantage of this hardware; FPGA implementations are used, e.g., to control an induction motor drive [19], implement active control filters [20], perform automatic online diagnosis algorithm for broken-bar detection on induction motors using discrete wavelet transform [21], and mitigate the distortion in RF power amplifiers [22].

Thus, for a wide range of applications, FPGA implementation might be the best option. However, in the case of low sampling frequency requirements or no low power consumption needs, some other devices could be more suitable. In this paper, FPGAs are the target hardware.

Apart from usual reprogrammable resources (lookup table (LUT), slices formed from LUT, registers, etc.), modern FPGAs contain many resources that support DSP applications such as embedded multipliers, multiply accumulate units, intellectual property cores for advanced specific functions, and processor cores for simpler programming. Some of the resources are implemented in the FPGA fabric and optimized for high-performance concurrent operation and low power consumption.

Usually, the task of describing a complex system for a hardware implementation is still very hard. FPGAs are normally used for applications with tremendous time to market constraints; thus, there is a need for design tools that are able to achieve this. In a traditional approach, a design tool takes the VHDL description of the hardware and generates a bitstream (configuration file for the FPGA device). Nowadays, the use of high-level synthesis (HLS) tools is increasing, enabling designers to enter designs at a much higher level of abstraction. Such an HLS tool would accelerate the design process by taking the algorithmic description of the required hardware, introducing certain area and performance requirements according to the design needs, and automatically generating the bitstream. This paper develops VHDL and HLS implementation for a novel algorithm (adaptive impulsive noise filtering), providing some hints and comparisons for each design process. Xilinx devices and software environment have been chosen. Other manufacturers offer similar devices and tools (e.g., Altera with DSP Builder, Accel DSP from Xilinx, and SynplifyDSP); in this case, Xilinx devices are only considered.

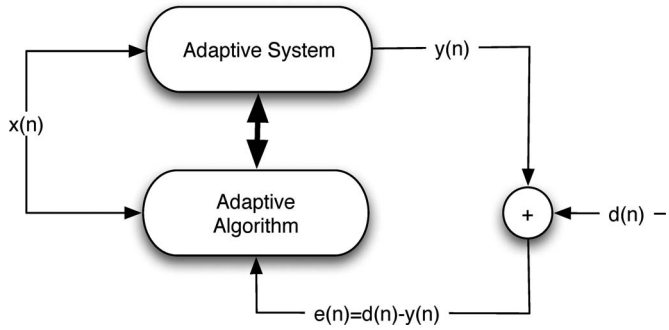

Fig. 1. Scheme of an adaptive filter.

A combination of adaptive filtering, impulsive noise immunity, and hardware implementation is an area where few references can be found, mainly focused in the digital image processing field [23]–[25]. This paper deals with the FPGA hardware implementation of an adaptive algorithm that is robust to impulsive noise developed in [26]. This algorithm offers a low computational cost by employing a cost function widely used in independent component analysis (ICA). Some refinements to the original algorithm have been introduced to reduce computational cost without affecting functionality.

The remainder of this paper is outlined as follows. Section II shows the proposed cost function and the algorithm description. Section III describes the specifications of the system, the proposed stages carried out to perform the calculation, and the optimal architecture to balance speed and area in the two hardware implementations proposed. Section IV shows the simulation results and design verification and provides a performance study in terms of the area, time, and accuracy. Finally, this paper concludes with a summary.

II. Algorithm Description

An adaptive system is composed of two modules (Fig. 1).

1) *A temporal variant system (adaptive system).* Most of the applications propose a digital FIR filter defined in a time instant n by a coefficient vector $\mathbf{w}_n = [w_n(0), w_n(1), \ldots, w_n(L-1)]$, using an input vector $\mathbf{x}_n = [x(n), x(n-1), \ldots, x(n-L+1)]$ to calculate the output signal $y(n)$ as the $\mathbf{w}_n$ and $\mathbf{x}_n$ convolution.
2) *An adaptive algorithm*. This module adjusts the coefficients so that the adaptive system output $y(n)$ and the desired signal $d(n)$ are equal. The similarity between both signals is given by the error signal $e(n)$ defined as $d(n) - y(n)$. The error signal $e(n)$ is used to adjust the coefficients of the adaptive system for the next time instant. Different algorithms exist for the coefficient adjustment.

Adaptive algorithms are based on the minimization of a certain cost function (optimization problem). The most widely used cost function is the quadratic error. This cost function is optimum when errors follow a Gaussian distribution [4]. However, it is not a good choice in the presence of outliers or

impulsive noise. In [26], the use of a cost function commonly used in ICA (1) provides a good result in the presence of outliers as in the case of impulsive noise, where $\beta \in \Re^+$ controls the sensitivity to large outliers in the value of signal error $e(n)$

$$J(n) = \frac{\log\left[\cosh\left(\beta \cdot e(n)\right)\right]}{\beta}. \tag{1}$$

By using the Delta rule [4], [5] for the update of the coefficients, we obtain the following equation where $\mu \in \Re^+$ is the learning rate:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \cdot \tanh\left[\beta \cdot e(n)\right] \cdot \mathbf{x}_n. \tag{2}$$

Equation (2) shows two attractive features.

1) *The proposed cost function is noise robust*: since the hyperbolic tangent saturates to ± 1 for extreme values.
2) *Easy hardware implementation*: An efficient hardware implementation for the hyperbolic tangent based on fuzzy logic is used, giving

$$\tanh\left(\beta \cdot e(n)\right) = \begin{cases} \operatorname{sign}\left(e(n)\right), & \text{if } |e(n)| > 1/\beta \\ -e(n) \cdot |e(n)| \cdot \beta^2 + 2 \cdot \beta \cdot e(n), & \text{if } |e(n)| \leq 1/\beta. \end{cases} \tag{3}$$

Therefore, the update of the algorithm coefficients in (2) is given by

$$\mathbf{w}_{n+1} = \begin{cases} \mathbf{w}_n + \mu \cdot \operatorname{sign}\left[e(n)\right] \cdot \mathbf{x}_n, & \text{if } |e(n)| > 1/\beta \\ \mathbf{w}_n + \mu \cdot \left[2\beta - \beta^2 \cdot |e(n)|\right] \cdot e(n) \cdot \mathbf{x}_n, & \text{if } |e(n)| \leq 1/\beta. \end{cases} \tag{4}$$

The value of beta is very relevant since it controls which outliers are taken into account by the adaptive system. This parameter is related to the variance of the error produced by the adaptive system. The value of β can be modified iteratively according to the evolution of the error in the learning process. We have taken the classical threshold of three standard deviations to consider a pattern as an outlier. Moreover, since $\tanh(|3|) = |0.96|$, i.e., a value close to unity, the parameter β can be obtained

$$\beta = \frac{3}{(m + 3 \cdot \sigma)} \tag{5}$$

where m is the mean value of the error signal and σ is its standard deviation estimated in an N-length window containing the last errors [26].

This threshold for the outliers can be modified, providing different levels of immunity to impulsive noise.

III. Hardware Implementation

From an external hardware view, the interface is defined with two 12-b data inputs, namely, $x(n)$ and $d(n)$, as the reset and clock signals, and DATAIN_VALID, a synchronization signal from a hypothetical acquisition device, to inform the FPGA that new data are ready for processing. Regarding the outputs, the system provides 12-b filtered data $y(n)$ and a synchronization signal DATAOUT_VALID to inform a potential subsequent system that data have been calculated and a filter result sample is available.

As indicated in (4), the calculation process for coefficient update requires the values of $e(n)$, β, and μ. However, for hardware implementation, some modifications are done. The update algorithm can be divided into four logic steps.

Step 1) Calculation of $y(n)$ according to (6). In this case, $\mathbf{t}$ is the transpose and $L = 9$

$$y(n) = \sum_{k=0}^{L-1} x(n-k) \cdot w_n(k) = \mathbf{x}_n^{\mathbf{t}} \cdot \mathbf{w}_n. \tag{6}$$

Step 2) Calculation of $e(n)$. The *estimation error* $e(n)$ is the difference between the desired response $d(n)$ and the adaptive system output $y(n)$, i.e., $e(n) = d(n) - y(n)$.

Step 3) Calculation of β. Typically, most of the applications in adaptive filtering have zero mean signals; then, when the number of samples is high, the value of m can be rounded to zero [4]. Moreover, as β controls the outlier immunity, and thus, a fixed value is not necessary, we have experimentally proved that, using both the standard deviation or its square (i.e., the variance), the resultant β does not affect the system behavior. Finally, (5) becomes $\beta \approx 1/\sigma^2$. This is very useful because an iterative calculation procedure for the variance [28] can be used (7) and β calculation is simplified

$$\sigma^2(n) = \frac{n-1}{n} \cdot \sigma^2(n-1) + \frac{1}{n-1}\left(e(n) - m\right)^2. \tag{7}$$

For a high number of samples ($n \gg$), (7) can be approximated as

$$\sigma^2(n) = 0.95 \cdot \sigma^2(n-1) + \frac{e^2(n)}{n} \qquad \text{if } n = 0,\ \sigma^2(n) = 1. \tag{8}$$

Step 4) Calculation of next sample coefficient values $w_{n+1}(k)$. The filter coefficients are updated based on (4), which can be rewritten as indicated in the following equation where $w_n(k)$ and $w_{n+1}(k)$ are the current and updated coefficients, respectively ($k \in \{0, \ldots, 8\}$), μ is the learning rate, and $x(n-k)$ is the delay-line samples provided by the input of the filter:

$$w_{n+1}(k) = \begin{cases} w_n(k) + \mu \cdot \operatorname{sign}\left[e(n)\right] \cdot x(n-k), & \text{if } |e(n)| > \sigma^2 \\ w_n(k) + \mu \cdot \left[\frac{2}{\sigma^2} - \frac{|e(n)|}{(\sigma^2)^2}\right] \cdot e(n) \cdot x(n-k), & \text{if } |e(n)| \leq \sigma^2. \end{cases} \tag{9}$$

As appreciated, this step is the most costly in terms of required computations.

A. Implementation Considerations

Computing floating-point arithmetic in an FPGA is possible, but it is not the most efficient method. In fact, it is very intensive in terms of logic resource usage and does not take full advantage of the parallelization possibilities offered by an FPGA. All

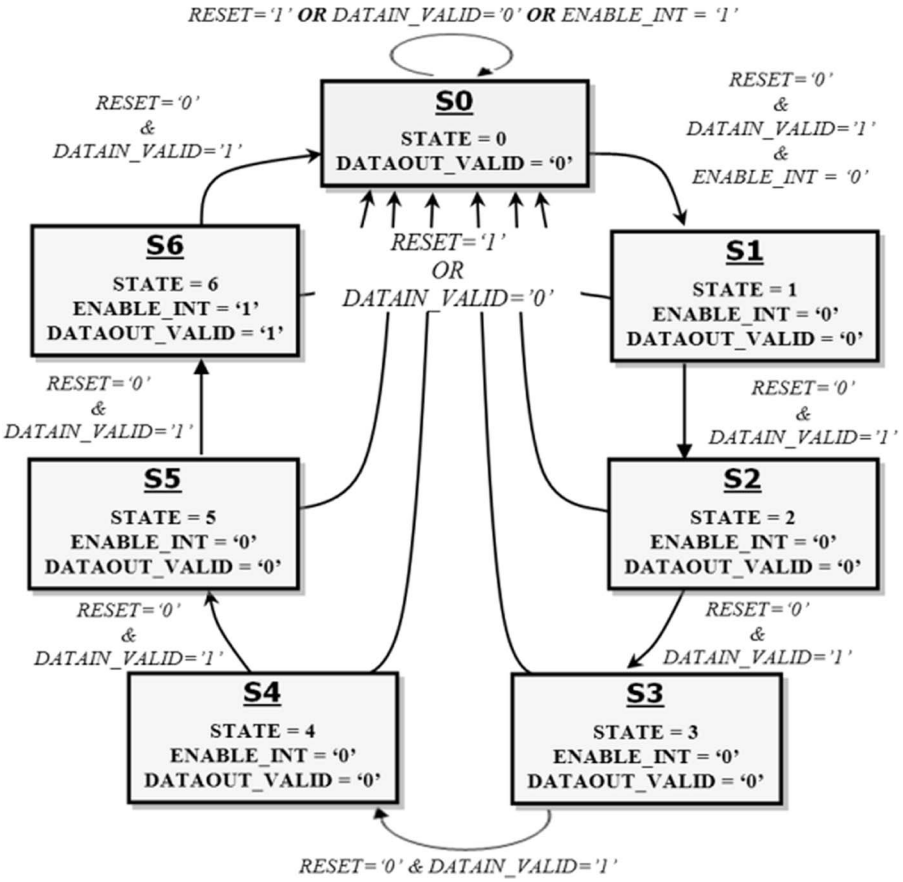

Fig. 2. State machine used for VHDL filter implementation.

calculations are therefore mapped to a fixed point, but this can introduce analog-to-digital conversion quantization error, coefficient quantization error, overflow error, and round-off error. All of these issues must be addressed and handled properly to reduce the errors committed by fixed-point arithmetic.

The first decision taken in fixed-point arithmetic is to keep a limited number of decimal digits. The number of decimal positions required for a given algorithm to converge depends on the algorithm. For instance, if two decimal places are sufficient for accurate data processing, this can easily be obtained by multiplying the filter's coefficients by 100 and truncating to an integer value. Dividing the output by 100 recovers the value. Since multiplying and dividing by powers of two can be done easily in hardware by shifting bits, a power of two can be used to simplify the process. In this case, we multiply by 128, which would require seven extra bits in hardware, but precision is improved for internal calculations. In addition, divisions must be corrected with a subsequent multiplication. Addition and subtraction require no adjustment.

In this paper, we use 12-b normalization. This means that the scale constant is $s = 2047$, being the size of all data inputs and outputs.

B. FPGA Implementation Based in VHDL

In order to perform all the required calculations for each new data sample (iteration), we defined a finite state machine (FSM) shown in Fig. 2. The FSM is in State0 waiting for a new data sample; once it is received, the FSM increases the state every new clock cycle in order to perform different calculations at each state. It is necessary to obtain several results needed for further calculations; thus, most of these calculations are intermediate values ($internal\_^*$ signals). After six clock cycles, all the calculation process is completed, including the output value and coefficients' update, being ready for a new data sample. For the FSM to work properly, the signal DATAIN_VALID must be high during all the states. There also exists an ENABLE_INT signal preventing the FSM to start a new calculation for the same input data, only allowing a new calculation when DATAIN_VALID goes low and then high, i.e., new input data are received.

All the calculations are distributed among the states, obtaining partial results to optimize logic resources. In this case, nine 18-b multipliers, one divider 25 by 14-b length, and some additional adders and subtracters are used. We have paid special attention to multiplications and divisions, being optimized to share the same arithmetic unit in different states. The data flow for the algorithm is described step by step in Fig. 3. The result from the division takes one additional clock cycle to be calculated; it is marked as a dotted line in the data flow of Fig. 3.

In *State S1*, the filter *output* signal $y(n)$ is calculated. This state makes use of nine multiplications, eight additions, and one FIFO register to store the new sampled input and the $L-1$ previous inputs (*inputvector*). In *State S2*, calculation

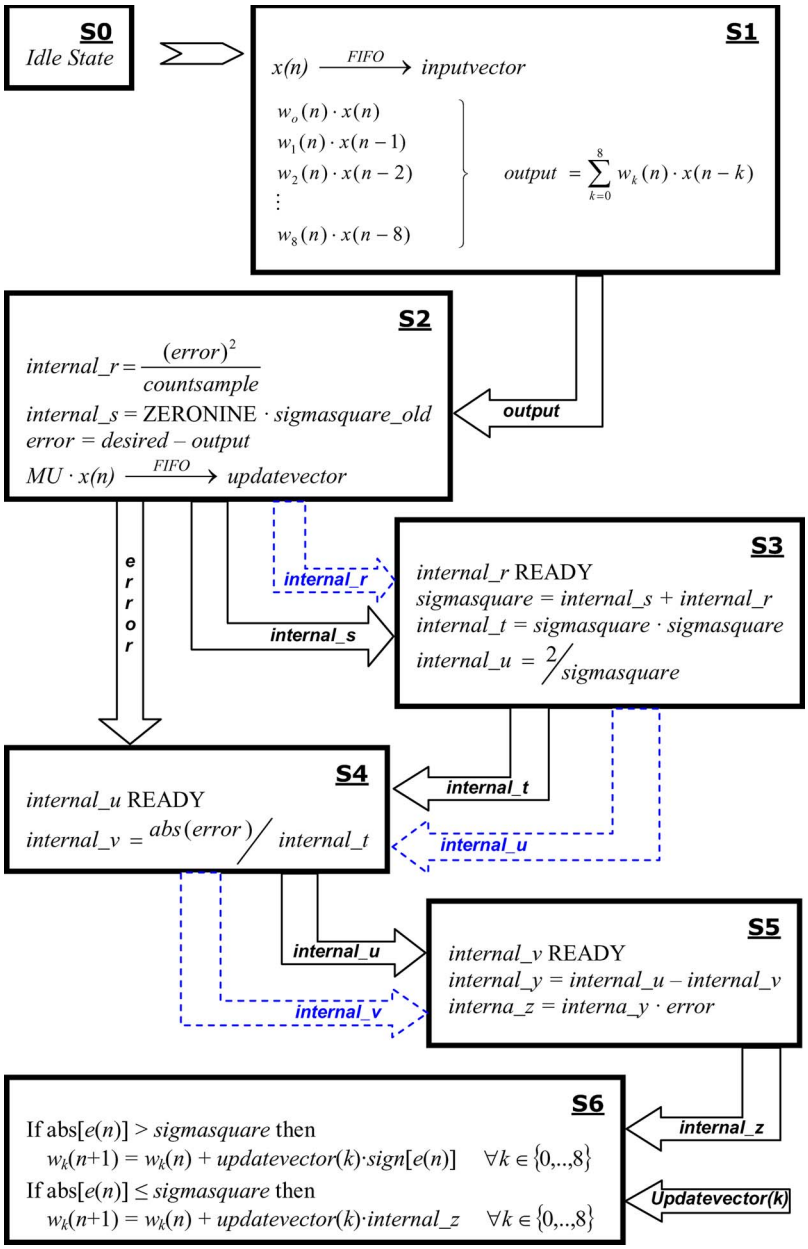

Fig. 3. Data flow for the state machine in VHDL filter implementation.

of $internal\_r$, $internal\_s$, *error*, and the product $\mu \cdot x(n)$ is done. $Internal\_r$ is the result of a division; then, it will be ready for the next state (S3), and the product $\mu \cdot x(n)$ is stored in a second FIFO register called *updatevector*. Here, two multiplications, one division, and one addition are required.

In the *State S3*, the variance (*sigmasquare*), $internal\_t$, and $internal\_u$ values are obtained, requiring a single multiplication, an addition, and a division. Concerning *State S4*, it computes one division for the calculation of $internal\_v$. In *State S5*, $internal\_y$ and $internal\_z$ are calculated using a multiplication and an addition. Finally, in *State S6,* the coefficients' update algorithm is performed for all the kth coefficients, using all the intermediate values obtained from previous states by $internal\_^*$ signals.

A balance between logic resource usage and performance has been considered, obtaining all the results in the fewer states as possible. Special attention is paid to the division operation as it is costly in terms of resources; thus, only one division module is used to perform all the divisions needed in the algorithm. By using this approach, after six clock cycles, a result is obtained, which means that a sampling data rate of one-sixth of the clock signal could be achieved for the input signal, thus being the output update rate.

C. FPGA Implementation Based on Simulink System Generator HLS Tool

Different implementation strategies can be followed when using an HLS tool, some closer to the software algorithm

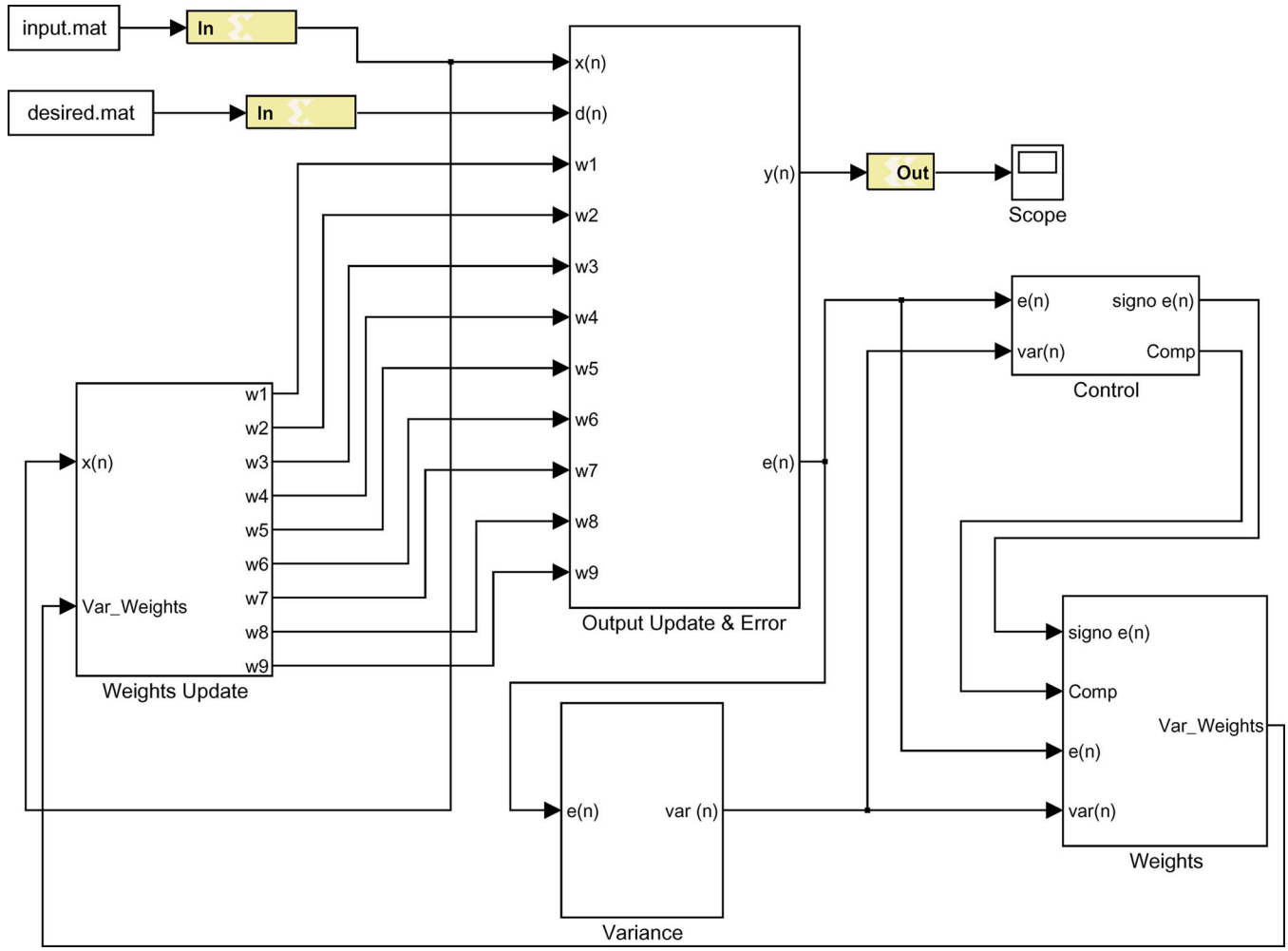

Fig. 4. Top-level block diagram for the HLS tool.

and some others to the hardware level. However, the main advantage compared to hardware description languages lies in the reduced knowledge required in the field of hardware design. For the Simulink–Xilinx System Generator (XSG) tool, once the software algorithm is obtained in Matlab, mathematical operations must be converted to predefined blocks given in libraries (blockset from XSG library), interconnecting them according to the computations required. Finally, the designer sets up some options for data bit width and the type of arithmetic to implement. This blockset provides a range of high-level functionality, from arithmetic operations to complex DSP functions. Simulink, together with XSG and Xilinx ISE design software packages, is used for the filter implementation. In this case, the target was the simplicity design entry, without further refinement in the modules, as a software designer would do. A block diagram of the proposed system is shown in Fig. 4.

All the calculations are carried out using signed fixed-point arithmetic (2's complement). In the case of overflow, the chosen option has been to saturate to the largest positive or smallest negative value, and for quantization, rounding is done. Internal calculations are carried out using double precision. The main parts of the proposed system are as follows.

1) Block "*Output Update & Error*," responsible for the output calculation $y(n)$ and error calculation $e(n)$ and containing a set of registers to store previous input data samples, nine multipliers, eight adders, and one subtracter. Equation (6) and β value are calculated using this module. Fig. 5 shows its block diagram.
2) Block "*Variance*," obtaining the variance value according to (8). This block stores the previous variance value for iterative calculation. It is composed of three multipliers, one adder, a ROM for the inverse value, and a counter to address the ROM.
3) Block "*Control*," calculating the absolute error value and the sign of the error signal according to (9). In addition, the error signal and the variance value are compared, sending the result to the "*Weights*" block using the *Comp* signal. Inside, a *negate* block computes the arithmetic negation (2's complement) of $e(n)$, with a *Xilinx Bit Slice Extractor* extracting the sign of $e(n)$ and a comparator evaluating whether $|e(n)| > \sigma^2(n)$.
4) Block "*Weights*," obtaining an internal value named "*Var_Weights*" (10) to be used further in the coefficients' update. A main part of (9) is carried out in (10), using four adders, one subtracter, one multiplier, one divider,

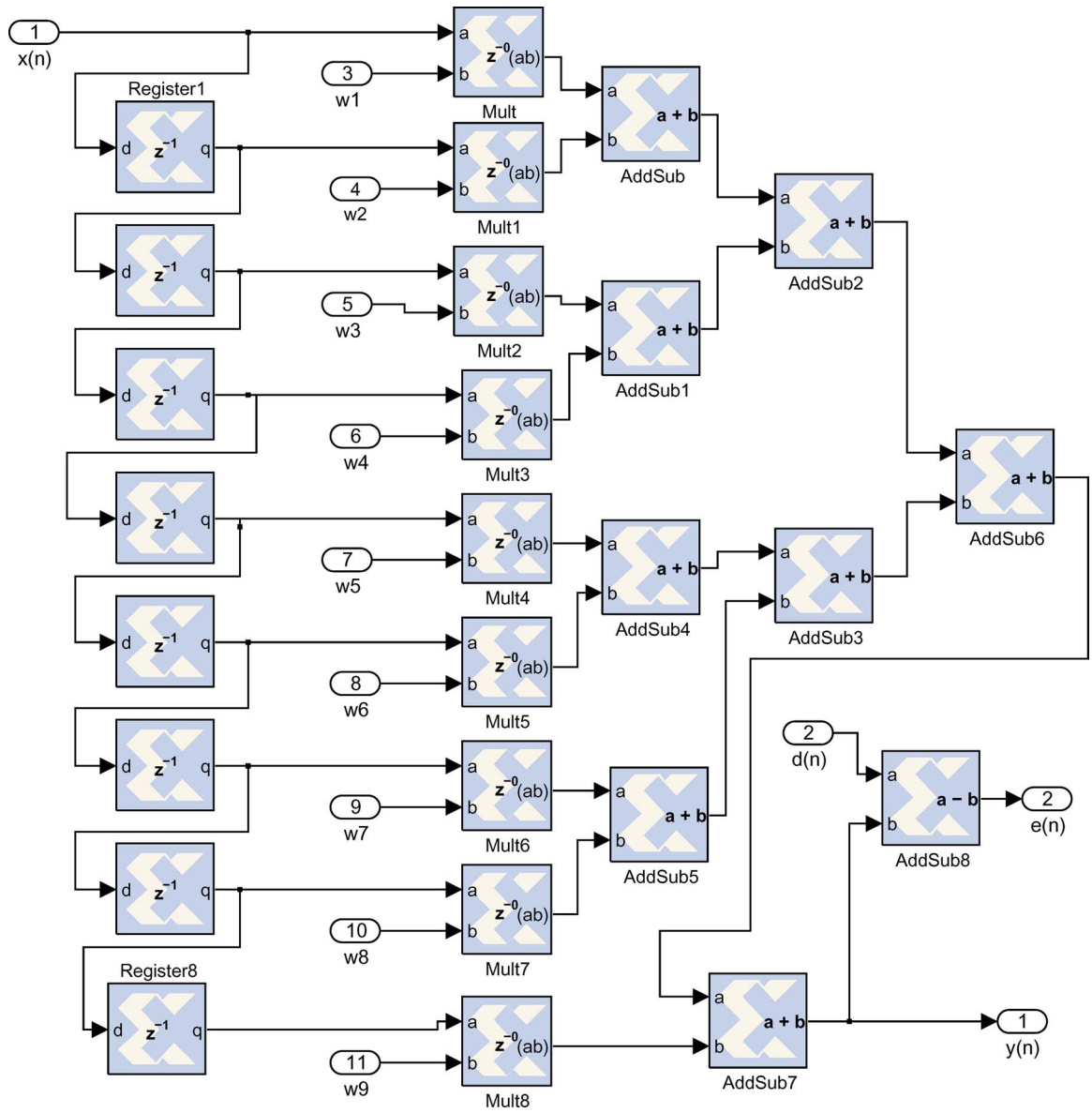

Fig. 5. "*Output Update & Error*" block diagram.

and five shift blocks to perform a right shift in the input signal. By using these resources, products by constants 0.05 and 0.025 are reduced to shift and add operations

$$Var\_Weights = \begin{cases} 0.025 \cdot \text{sign}(n), & \text{if } |e(n)| > \sigma^2(n) \\ 0.05 \cdot \left[\frac{e(n)}{\sigma^2}\right] & \\ \quad -0.025 \cdot \left[\frac{e(n)}{\sigma^2}\right] \cdot \left[\frac{|e(n)|}{\sigma^2}\right], & \text{if } |e(n)| \leq \sigma^2(n). \end{cases} \tag{10}$$

5) Block "*Weights Update*," responsible for updating the weight values of the adaptive filter using the value obtained in block "*Weights*". This block completes the calculation of (9) and contains registers to store previous values of data input samples, 97 multipliers and 9 accumulators.

In this implementation, simplicity and design time to describe the design in the HLS tool has been the goal. Thus, the previously designed Matlab algorithm code has been easily migrated without special attention to optimization in resources and performance. As a result, the tool implemented arithmetic units concurrently, increasing the performance at the cost of using 21 18-b multipliers, one divider, one ROM, and additional adders and subtracters.

IV. Results

By using the HLS approach, the system is described as a data flow, simpler, and faster than in VHDL. Some functional modules exist in a library, and others can be done manually, but in general, the description of complex systems typically used in an embedded system design becomes easier. In contrast, some specific refinements cannot be done, and optimization in terms of performance is not simple. System Generator for HLS and ModelSim for VHDL descriptions have been used to simulate the two approaches for hardware implementation.

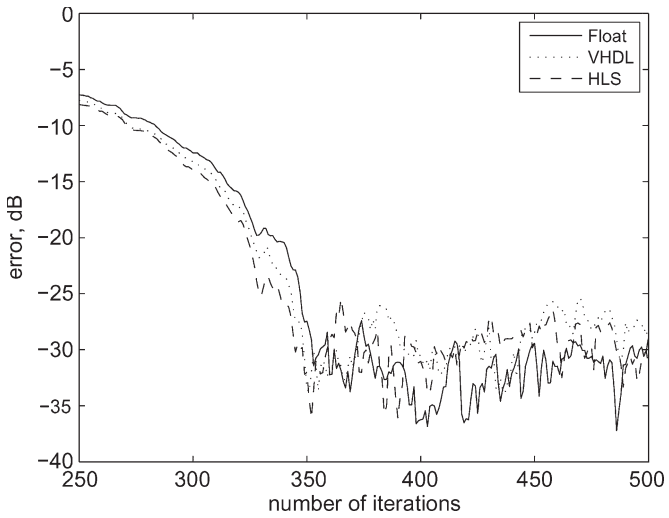

Fig. 6. Error versus iterations for different approaches ($\mu = 0.025$).

The proposed adaptive algorithm implementation was tested on a system identification application with impulsive noise contribution. In this case, $x(n)$ is the stimulus signal, which feeds directly to the unknown system and the adaptive filter. The output of the unknown system, $d(n)$ or desired signal, is corrupted by impulsive noise and contaminated with Gaussian noise, not correlated to the adaptive system input. The input signal $x(n)$ and desired signal $d(n)$ are wide sense stationary and zero mean processes. The unknown system is an FIR filter with coefficients $w_{\text{unknown}} = [-0.1, -0.2, -0.3, -0.4, 0.5, 0.4, 0.3, 0.2, 0.1]$. This simulation has been widely used in a great number of publications, thus considered a good test for this system and comparison to others [7]–[9].

In order to evaluate the goodness of the proposed system, the difference between the unknown and the filter-obtained coefficients is calculated according to

$$error_n(\text{dB}) = 10 \cdot \log_{10}\left[\frac{\|\mathbf{w}_n - \mathbf{w}_{\text{unknown}}\|^2}{\|\mathbf{w}_{\text{unknown}}\|^2}\right]. \qquad (11)$$

Fig. 6 shows the error achieved. As can be seen, the speed of convergence is fast, in less than 400 iterations (samples), and the filter gets an excellent identification value of coefficients for the unknown system (the error is below -25 dB). As an example, in iteration 417, the coefficient values obtained by the adaptive system are $w_n = [-0.092, -0.206, -0.290, -0.411, 0.515, 0.395, 0.288, 0.205, 0.113]$, which are close to the unknown system values w_{unknown}. Both VHDL and HLS hardware results are very close to the results obtained by floating-point software, although, due to precision errors, the floating-point result gives a lower error.

Regarding noise immunity, for simulation and hardware results, Fig 7 shows the filter output result $y(n)$ compared to the unknown system output $d(n)$ for simulation [Fig. 7(a)] and hardware implementation [Fig. 7(b)]. Despite the noise, the algorithm identifies the system and adjusts coefficients accordingly; it can be observed that $d(n)$ and $y(n)$ signals in Fig. 7(a) and (b) are the same except for the noise, as expected. Some minor differences are appreciated between simulation and hardware implementation.

A. Device Implementation

Different device implementations have been done; Tables I and II show the results for VHDL and HLS descriptions, respectively. For occupation and performance comparison purposes, three Xilinx devices have been used, namely, a low-cost Spartan3 and two leading edge devices from Virtex4 and Virtex5 device families.

Logic occupation is very similar for VHDL and HLS except for the usage of multiplier units (9 for VHDL and 21 for HLS); apart from that, logic resources used by an HLS are lower. However, for VHDL implementation, the target device could be simpler than in HLS due to the low number of multiplier units. Regarding performance, the HLS implementation calculates the result in a single clock cycle, being six clock cycles necessary for the VHDL. Thus, despite the fact that the maximum clock frequency is lower in HLS, the maximum achievable sampling rate is higher than in VHDL. In any case, a sampling data rate from 1.58 to 12.32 MHz could be achieved.

In Virtex5 family, clock frequency increases dramatically compared to the other devices, mainly due to silicon fabric and new reconfigurable architecture from the manufacturer.

One of the most restricting design modules for performance in both designs is the division as it is calculated in a single clock cycle. Several division approaches have been explored, considering that single clock division could be optimum. Pipelined division does not match the requirements in this design because a division result is further divided before the final result, not allowing the pipeline scheme.

Final hardware verification has been carried out in a Xilinx Spartan3E device family (XC3S500E-4) included in the Digilent Spartan3E starter board; the data acquisition is done using an LTC1407A-1 serial SPI dual 14-b analog-to-digital converter from linear technology, reduced to 12-bit to accommodate filter input data length. The filter output result is converted using an LTC2624 12-b serial SPI digital-to-analog converter from linear technology. National Instruments NI USB-6008 data acquisition board is used to generate filter inputs and read FPGA filter output from a PC computer and evaluate FPGA real-time processing, performance, and accuracy. Fig. 8 shows a photograph of the test system and the results obtained from the FPGA device.

V. Conclusion

The implementation of a useful adaptive filter to remove impulsive noise for system identification or prediction is done using FPGA devices. Furthermore, the FPGA implementation allows the usage in hardware platforms where low power consumption is required. In contrast to other devices such as DSP processors, a high data sample rate is achieved at a relatively slow system clock frequency operation, thus reducing electromagnetic interferences.

In addition, two different implementation procedures have been followed. The first method used VHDL, where hardware implementation details are optimized and the filter structure can

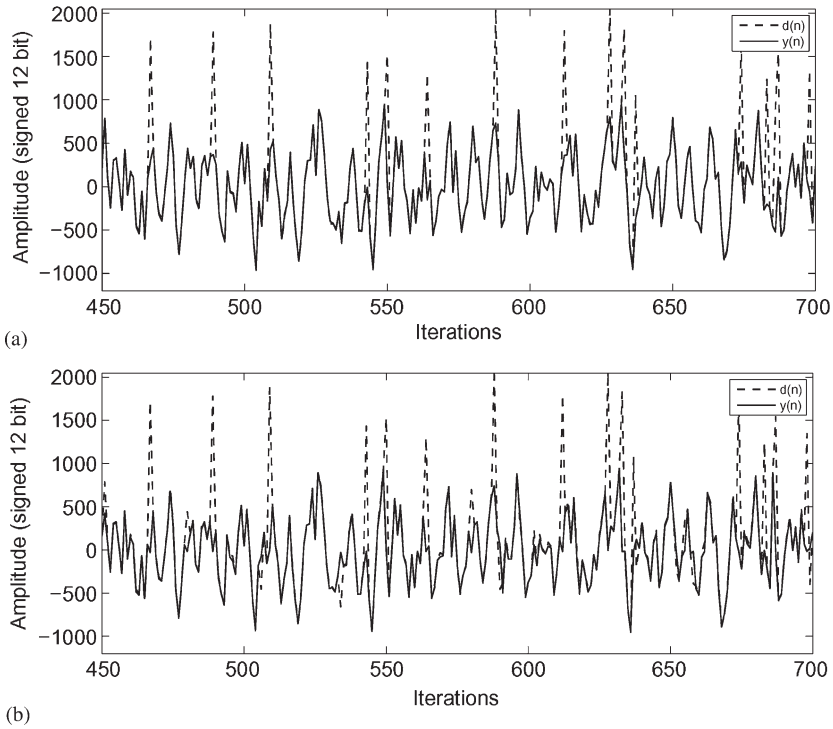

Fig. 7. Output of the unknown system $d(n)$ and the filter $y(n)$ for (a) simulation and (b) real-time hardware implementation.

TABLE I
VHDL IMPLEMENTATION RESULTS IN XILINX FPGA DEVICES

FPGA Chip	XC3S1000 fg320-5	XC4VFX12 ff668-12	XC5VLX30 ff324-3
Slices used	2429 (31%)	2586 (47%)	3906 (20%)
MULT used	9 (37%)	9 (28%)	9 (28%)
4 Input LUTs	4525 (29%)	4777 (43%)	3513 (18%)
Max. clock Freq.	9.48MHz	18.97MHz	25.27MHz
Max. sampl. Freq.	1.58MHz	3.16MHz	4.21MHz

TABLE II
HLS IMPLEMENTATION RESULTS IN XILINX FPGA DEVICES

FPGA Chip	XC3S1000 fg320-5	XC4VFX12 ff668-12	XC5VLX30 ff324-3
Slices used	2268 (29%)	2316 (42%)	2568 (13%)
MULT used	21 (87%)	21 (65%)	21 (65%)
4 Input LUTs	4264 (27%)	4331 (39%)	2536 (13%)
Max. clock Freq.	5.51MHz	9.65MHz	12.32MHz
Max. sampl. Freq.	5.51MHz	9.65MHz	12.32MHz

be defined more precisely. The second implementation method used an HLS environment; in this case, a fast design entry can be done without the need of extensive hardware implementation knowledge. The proposed structure using the HLS was an easy filter design as an inexperienced hardware designer would do.

Simulations and real-time hardware implementation results show that both implementations are very accurate, being very similar in accuracy when compared to a floating-point result and also between VHDL and HLS. In the case of resource consumption and speed of operation, different device implementations have been done, showing that for the same device, similar resources are used for both approaches. Regarding speed of operation, higher speed of system clock can be achieved using

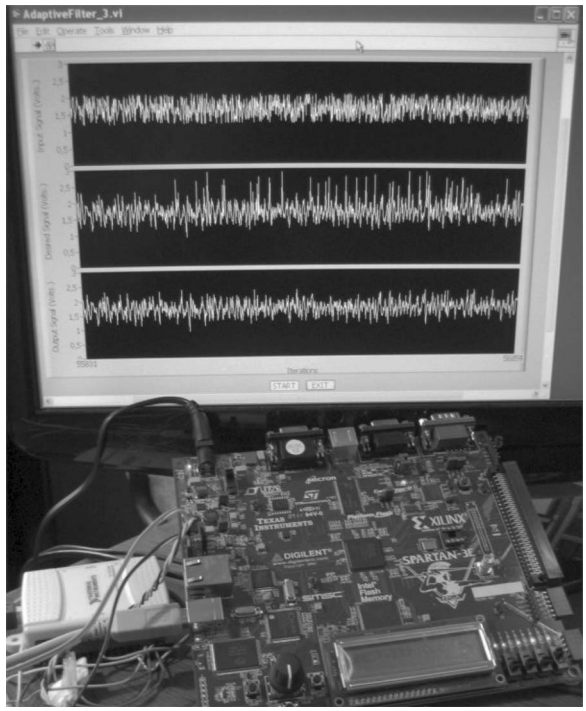

Fig. 8. Hardware testing system and results for the proposed adaptive filter.

VHDL description, but due to the need of six clock cycles to perform the calculation of filter output for every data sample, the HLS description offers greater data sample frequency.

We can state that an HLS design environment is very useful, particularly for those very complex designs or inexperienced hardware designers. An HLS is closer to the software and algorithm level design, making hardware migration from simulation

software a relatively easy task. The evaluated HLS (Simulink System Generator), and any HLS in general, does not allow specific control for low-level design aspects. However, due to the specific libraries of optimized components, good performance results can be obtained. On the other hand, a high level of optimization can be achieved using a low-level hardware design, such as VHDL description, which requires more time and effort and good expertise in the field.

REFERENCES

[1] J. G. Proakis, *Digital Communications*. New York: McGraw-Hill, 2008.

[2] J. Benesty, T. Gänsler, D. R. Morgan, M. M. Sondhi, and S. L. Gay, *Advances in Network and Acoustic Echo Cancellation*. Berlin, Germany: Springer-Verlag, 2001.

[3] E. S. Nejevenko and A. A. Sotnikov, "Adaptive modeling for hydroacoustic signal processing," *Pattern Recognit. Image Anal.*, vol. 16, no. 1, pp. 5–8, Jan. 2006.

[4] A. Sayed, *Fundamentals of Adaptive Filtering*. New York: Wiley, 2003.

[5] B. Widrow and S. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[6] C. L. Nikias and M. Shao, *Signal Processing With Alpha-Stable Distributions and Applications*. New York: Wiley, 1995.

[7] C. Rusu and C. F. N. Cowan, "Adaptive data echo cancellation using cost function adaptation," *Signal Process.*, vol. 80, no. 11, pp. 2457–2473, Nov. 2000.

[8] D. P. Mandic, E. V. Papoulis, and C. G. Boukis, "A normalised mixed norm adaptive filtering algorithm robust under impulsive noise interference," in *Proc. Int. Conf. Acoust., Speech, Signal Process. VI*, 2003, pp. 333–336.

[9] J. Chambers and A. Avlonitis, "A robust mixed-norm adaptive filter algorithm," *IEEE Signal Process. Lett.*, vol. 4, no. 2, pp. 46–48, Feb. 1997.

[10] M. A. Gandhi, C. Ledoux, and L. Mili, "Robust estimation methods for impulsive noise suppression in speech," in *IEEE Int. Symp. Signal Process. Inf. Technol.*, 2005, pp. 755–760.

[11] I. Akira, O. Mitsuo, and O. Hitoshi, "A digital filter for estimation of impulsive noise and vibration by introducing fuzzy reasoning and its application to room acoustics," *J. Acoust. Soc. Jpn.*, vol. 55, no. 1, pp. 32–36, 1999.

[12] O. Vainio and S. J. Ovaska, "Noise reduction in zero crossing detection by predictive digital filtering," *IEEE Trans. Ind. Electron.*, vol. 42, no. 1, pp. 58–62, Feb. 1995.

[13] T. P. Pander, "A suppression of an impulsive noise in ECG signal processing," in *Proc. Conf. IEEE Eng. Med. Biol. Soc.*, 2004, vol. 1, pp. 596–599.

[14] J. Armstrong, H. A. Suraweera, C. Chai, and M. Feramez, "Impulse noise mitigation techniques for OFDM receivers and their application in digital video broadcasting," *Mediterr. J. Electron. Commun.*, vol. 1, no. 1, pp. 1–10, Oct. 2005.

[15] I. Aizenberg, T. Bregin, and D. Paliy, "Method for the impulsive noise detection and its application for the improvement of the impulsive noise filtering algorithms," in *Image Process. Conf.*, San Jose, CA, 2002, vol. 4667, pp. 204–214.

[16] K. Benkrid, D. Crookes, and A. Benkrid, "Design and implementation of a novel algorithm for general purpose median filtering on FPGAs," in *Proc. ISCAS*, 2002, vol. 4, pp. 425–428.

[17] U. M. Baese, *Digital Signal Processing With Field Programmable Gate Arrays*, 3rd ed. Berlin, Germany:Springer-Verlag, 2007.

[18] G. Spivey, S. S. Bhattacharyya, and K. Nakajima, "Logic foundry: Rapid prototyping of FPGA-based DSP systems," in *Proc. ASP-DAC*, 2003, pp. 374–381.

[19] D. Zhang and H. Li, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms," *IEEE Trans. Ind. Electron.*, vol. 55, no. 2, pp. 551–561, Feb. 2008.

[20] Z. Shu, Y. Guo, and J. Lian, "Steady-state and dynamic study of active power filter with efficient FPGA-based control algorithm," *IEEE Trans. Ind. Electron.*, vol. 55, no. 4, pp. 1527–1536, Apr. 2008.

[21] A. Ordaz-Moreno, R. de Jesus Romero-Troncoso, J. A. Vite-Frias, J. R. Rivera-Gillen, and A. Garcia-Perez, "Automatic online diagnosis algorithm for broken-bar detection on induction motors based on discrete wavelet transform for FPGA implementation," *IEEE Trans. Ind. Electron.*, vol. 55, no. 5, pp. 2193–2202, May 2008.

[22] J. L. Mato, M. Pereira, J. J. Rodríguez-Andina, J. Farina, E. Soto, and R. Perez, "Distortion mitigation in RF power amplifiers through FPGA-based amplitude and phase predistortion," *IEEE Trans. Ind. Electron.*, vol. 55, no. 11, pp. 4085–4093, Nov. 2008.

[23] F. J. Gallegos-Funes and V. I. Volodymyr, "Real-time image filtering scheme based on robust estimators in presence of impulsive noise," *Real-Time Imaging*, vol. 10, no. 2, pp. 69–80, Apr. 2004.

[24] G. Louverdis, I. Andreadis, and N. Papamarkos, "An intelligent hardware structure for impulse noise suppression," in *Proc. 3rd Int. Symp. Image Signal Process. Anal.*, 2003, pp. 438–443.

[25] M. A. Vega-Rodríguez, J. M. Sánchez-Pérez, and J. A. Gómez-Pulido, "An FPGA-based implementation for median filter meeting the real-time requirements of automated visual inspection systems," in *Proc. 10th Mediterr. Conf. Control Autom.—MED*, Lisbon, Portugal, 2002.

[26] E. Soria, J. D. Martín, A. J. Serrano, J. Calpe, and J. Chambers, "Steady-state and tracking analysis of a robust adaptive filter with low computational cost," *Signal Process.*, vol. 87, no. 1, pp. 210–215, Jan. 2007.

[27] E. Soria, J. D. Martín, G. Camps, A. J. Serrano, J. Calpe, and L. Gómez, "A low-complexity fuzzy activation function for artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1576–1579, Nov. 2003.

[28] M. Melvin-Bruce, "Estimation of variance by a recursive equation," Langley Res. Center, Hampton, VA, NASA Tech. Note D-5465, 1969.

Alfredo Rosado-Muñoz received the B.S. and Ph.D. degrees in physics from the University of Valencia, Valencia, Spain, in 1993 and 2000, respectively.

He is currently a Lecturer and Researcher with the Department of Electronic Engineering, University of Valencia. His work is related to automation systems and digital hardware design in several fields.

Manuel Bataller-Mompeán received the B.Sc. degree in physics and the Ph.D. degree in electronic engineering from the University of Valencia, Valencia, Spain, in 1984 and 1989, respectively.

Since 1984, he has been with the Department of Electronic Engineering, University of Valencia, within the Digital Signal Processing Group, working as an Associate Professor. His research interests include digital signal processing and its hardware implementation, focusing on programmable logic systems.

Emilio Soria-Olivas was born in Albacete, Spain, in 1969. He received the B.Sc. degree in physics and the Ph.D. degree in electronic engineering from the University of Valencia, Valencia, Spain, in 1992 and 1997, respectively.

Since 1993, he has been with the Digital Signal Processing Group, Department of Electronic Engineering, University of Valencia, where he is currently an Associate Professor. His current research interests include advanced signal processing using neural networks and fuzzy systems.

Claudio Scarante was born on October 4, 1983, in Padua, Italy. He received the M.Sc. degree in electronic engineering from the University of Padua, Padua, in 2008.

His main research interests are in programmable systems using FPGAs, DSPs, and microcontrollers. During 2008, he was with the University of Valencia, Spain, involved in research on adaptive DSP algorithm development, high-speed FPGA implementations, and VLSI architectures. He is currently the Product Manager for a company involved in the design, project engineering, production, and sales of environmental monitoring systems.

Juan F. Guerrero-Martínez (M'90) received the B.Sc. degree in physics and the Ph.D. degree in electronic engineering from the University of Valencia, Valencia, Spain, in 1985 and 1988, respectively.

Since 1985, he has been with the Digital Signal Processing Group, Department of Electronic Engineering, University of Valencia, where he is currently an Associate Professor. His research interests include biomedical digital signal processing and biosignal.