Volume 57, Number 4, 2016

ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

EFFICIENT FPGA IMPLEMENTATION OF AN ADAPTIVE FILTER FOR NOISE CANCELLATION

Ervin SZOPOS[1], Ioana SARACUT[1], Horia HEDESIU[2]
[1]Bases of Electronics Department, [2]Electrical Machines Marketing & Management Department, Technical University of Cluj-Napoca
Str. G.Barițiu nr.26-28, Cluj-Napoca, Email: Erwin.Szopos@bel.utcluj.ro

Abstract: This paper presents an efficient architecture of the Least Mean Square (LMS) adaptive algorithm implemented as a FIR filter on a reconfigurable platform. The architecture of the adaptive filter was developed with a general structure so it can be used in many applications with minimum limitations. Besides general use, the architecture has the advantage of using optimal hardware resources on the chip. The effectiveness of the proposed architecture is demonstrated on the noise cancelling application, and the performances are evaluated in terms of signal-to-noise ratio (SNR), convergence speed, mean square error (MSE) and hardware usage.

Keywords: *FPGA, LMS-FIR, architecture, convergence, SNR, MSE.*

I. INTRODUCTION

Adaptive filters are useful in estimating a signal which has been sent through an unknown channel with variable characteristics. The aim is to minimize the mean square error (MSE), which is the difference between the desired response and the filter estimate. The minimum MSE criterion is the basis of Wiener filters theory in a stationary environment. The Least Mean Square (LMS) algorithm is an adaptive algorithm introduced by Widrow and Hoff in 1960 and is a widely used technique for adaptive filtering [1].

By using FPGAs (Field Programmable Gate Array), designers can implement special-purpose signal processing architectures using specialized data paths, optimized sequencing, and pipelining while still providing some flexibility.

Several solutions for implementing the LMS algorithm in FPGA platforms have been formerly proposed: an improved hardware architecture of LMS is presented in [2]; ref. [3] investigates the implementation of an adaptive decorrelator based on two cross-coupled LMS algorithms; in ref. [4] the adaptive filter is implemented on Xilinx Virtex-4 FPGA.

In this work, the LMS algorithm is implemented in LabVIEW FPGA in a more general form using RAM memories of the module NI cRIO-9104 as delay cells. This leads to the following features of the implementation: more efficient hardware usage, and faster time convergence. Our previous work [5] deals with architecture for the LMS algorithm that uses shift registers as delay cells. In this manner the algorithm's convergence is relatively slow and the hardware implementation needs more resources.

The paper is organized as follows: in Section II the LMS algorithm is presented; Section III provides a detailed description regarding the FPGA implementation of the proposed adaptive algorithm's architecture; Section IV shows the results of the experiments outlining the efficiency of the adaptive algorithm and Section V is dedicated to the concluding remarks.

II. LMS-FIR ADAPTIVE ALGORITHM

The LMS algorithm is described by three simple equations (the filtering, the error estimation and the update of filter coefficients) [1], as follows:

$$\begin{aligned} y(n) &= \mathbf{w}^T(n)\cdot\mathbf{x}(n) \\ e(n) &= d(n) - y(n) \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + 2\cdot\mu\cdot e(n)\cdot\mathbf{x}(n) \end{aligned} \tag{1}$$

where $\mathbf{w}(n) = \left[w_0(n) \;\; w_1(n) \;\; \ldots \;\; w_{N-1}(n)\right]^T$ is the Nth order filter coefficient vector (also called the weight vector), $\mathbf{x}(n) = \left[x(n)\, x(n-1) \ldots x(n-N+1)\right]^T$ is the input vector, $d(n)$ is the desired response, $e(n)$ is the error signal and μ is the step-size.

Figure 1 illustrates the block diagram of the LMS adaptive algorithm on which the implementation presented in *Figure 2* relies.

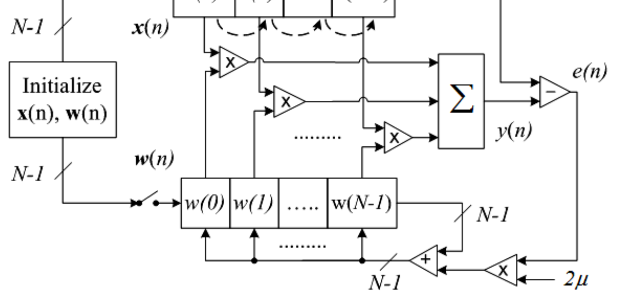


Figure 1. The block diagram of the LMS-FIR adaptive algorithm

Manuscript received October 28, 2016; revised December 8, 2016

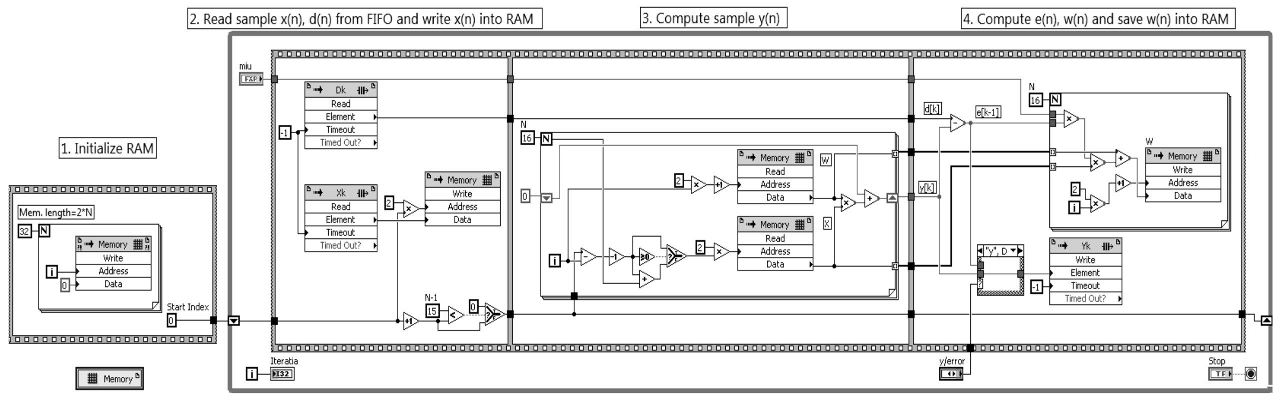


Figure 2. The proposed architecture of the adaptive LMS algorithm implemented in LabVIEW FPGA

Figure 2 presents the Virtual Instrument (VI) of the LMS algorithm implemented on a FPGA platform. In the first iteration, the vectors ***x(n)*** and ***w(n)*** are initialized with a null vector of size equal to *N* (set by the user). Then the input samples are one by one included in the vector ***x(n)*** which acts as a circular buffer. In each iteration, the samples are shifted to the right by one position and the sample of index *0* is replaced by the new received sample, denoted by *Ref_in* (x_{new}) in *Figure 1*. The rest of the arithmetic operations from the VI are intended to compute the samples of *y(n)* and *e(n)*, which contribute to updating ***w(n)*** on each iteration according to (1). Except for the first iteration, the current values of ***x(n)*** and ***w(n)*** are the values calculated in the previous iteration. The process continues until the stop condition is reached.

III. LABVIEW FPGA IMPLEMENTATION OF THE ADAPTIVE ALGORITHM

In our implementation, the signals are acquired using the real-time module NI cRIO-9014. The role of this module is to control the whole adaptive process, which involves: initializing the filter parameters, setting the stop condition, accessing the samples of the processed signals, controlling the input/output modules connected to the FPGA module, controlling the FIFO memories, analyzing the data and saving the samples in a file.

Generally, there are four main signals which are processed in an adaptive system: the input sequence *x(n),* the desired signal *d(n)*, the error signal *e(n)* and the output signal *y(n)*. To send/receive these signals to/from the reconfigurable module, we assigned a memory of FIFO type in the structure of the FPGA module. The module NI cRIO-9104 has an area allocated to a FIFO memory, which can be divided into only 3 individual memories with no need for special appeals. Consequently, for the 4th signal we used a case structure before applying it to the FIFO memory. The case selector is operated by the user and connects either *y(n)* or *e(n)* at the input. This memory can be accessed with graphical instruction from the FPGA Interface library, and the type of the reading / writing operation is performed by simple change. Finally, the output of the selector is connected to the memory and the stored signal will be available for further processing.

The adaptive algorithm also produces the coefficient vector ***w(n).*** For this vector we used the RAM memory of the module NI cRIO-9104, which is accessed using a statement similar to the FIFO memory (Memory read, Memory write), from the same library, except that this memory requires the calculation of the addresses to which we want to store data during the process. Given that ***w(n)*** is calculated in terms of ***x(n),*** for simplicity we took the samples of ***x(n)*** from the FIFO memory and store them into the RAM memory. For this we split the RAM as follows: starting with the *0* address, we stored the samples of ***x(n)*** at even positions and the samples of ***w(n)*** at odd positions.

Figure 3 illustrates the management of the RAM and the processing of data in order to calculate a sample of *y(n).* In each iteration, the sample ***x(n)*** from the first address is multiplied by the last nonzero coefficient *w*, then the second sample *x* is multiplied by the last but one coefficient *w*, and so on. The process repeats until the stop condition is reached; the results of multiplications are added iteratively.

For the representation of the values of ***x(n)***, *y(n)* and ***w(n)*** we used a fixed-point (FXP) data type. This allows to represent each value of ***x(n)***, *y(n)* and ***w(n)*** on 34 bits, from which 9 bits for the integer part and 1 bit for the sign. It results that the range for one value is of *±512* with an accuracy of *5.9605E-8*, meaning that the quantization errors are not significant.

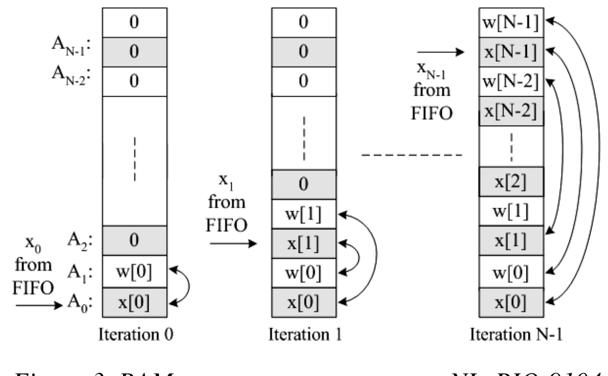


Figure 3. RAM memory management on NI cRIO-9104

IV. EXPERIMENTAL RESULTS

This section first presents the results using the adaptive architecture from *Figure 2*, on the reconfigurable module NI cRIO-9104. The efficiency of the proposed architecture was evaluated using the following metrics: the signal-to-noise ratio at the output (SNR_{out}), the convergence speed (number of iterations), *MSE* and the use of the chip surface for the filter implementation.

In all the experiments, the input and output *SNR* are calculated respectively with:

$$SNR_{in} = \frac{RMS\left[\mathbf{s}(n)\right]}{RMS\left[\mathbf{z}(n)\right]} \qquad (2)$$

$$SNR_{out} = \frac{RMS\left[\mathbf{s}(n)\right]}{RMS\left[\mathbf{e}(n)-\mathbf{s}_1(n)\right]} \qquad (3)$$

where *RMS* is the root mean square, ***s****(n)* is the original signal (without noise), ***z****(n)* is the noise, ***e****(n)* is the filtered signal and ***s****$_1$(n)* is the original signal (without noise) passed through the filter and measured at the output. The difference ***e****(n)*-***s****$_1$(n)* represents the output noise.

The architecture was analyzed using a wide range of input signals (determinist, non-determinist) with different types of noises. In the following, we present the case when the main input of the filter is a speech signal with the sampling frequency of *16 kHz* - signal *a)* from *Figure 4*. A Gaussian noise gets added to the speech signal, so that SNR_{in} becomes *-11.54 dB*. The noisy signal is illustrated in *Figure 4* – signal *b)*. The adaptive filter is of type LMS-FIR, and filter parameters were set to $N = 4$ and $\mu = 0.002$.

With these settings, the results are presented in *Figures 5-7*. The quality of the filtered signal (*Figure 5*) has significantly improved; the measured SNR_{out} is *12.87 dB*, showing at least *20 dB* improvement, compared to the noisy signal from *Figure 4*. The transient state ends at the instant when the peak-to-peak value of the noise does not exceed the range of *±50 mV*. This reference range represents only *5%* of the peak-to-peak value of the initial noise (*2V*). The convergence time was measured and included in *Table 2*. Although subjectively the input speech signal is indistinguishable because of the noise, it becomes perceptible at filters output. This is proven both subjectively and by the large difference between SNR_{in} and SNR_{out}. The length of the transient state is best illustrated in *Figure 6*, which shows the variations of the filter coefficients. One can also notice that after the transient state, the variation of the coefficients is carried out in a smooth manner, which indicates that the algorithm is stable and follows the variations in speech without losing convergence.

The MSE is illustrated in *Figure 7*. One can notice that the same value of the convergence duration is obtained here by measuring the variation of the MSE (in same conditions).

During the experiments we also studied the effect of the parameters *N* and μ on the performances of the proposed filter structure. Final evaluation is based on the synthesis of the Xilinx compiler (the use of resources on the chip), on SNR_{out} and the convergence time.

The FPGA synthesis of the architecture from *Figure 2* was done for three values of the filter length, as shown in the report generated by the Xilinx FPGA compiler (*Table 1*). In order for the architecture to run on the FPGA platform, the maximum frequency from *Table 1* must be greater than the clock frequency of the FPGA module (*40MHz*).

The results from *Table 1* confirm that:

- the FPGA architecture developed for the adaptive LMS-FIR filter can run on the module NI cRIO-9104, due to the maximum possible frequency that can be used for the algorithm on this module;
- the FPGA resource usage for this architecture is relatively small, therefore the application can be extended for multichannel processing, where several adaptive algorithms, working in parallel and/or independently of one another, filter the signals received on different channels (different frequency bands).

As presented in *Table 1*, the proposed architecture proves to be more effective than the one presented in [5].

The use of FPGA resources also depends on the number of bits used to represent the processed signals and determine the accuracy of processing the signals. Thus, if the use of resources is a priority, then it is advisable to work with a small length of the filter or the accuracy of data representation must be decreased.

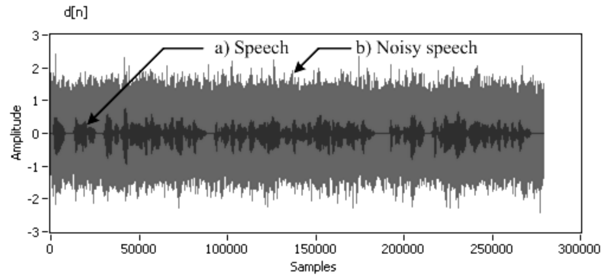


Figure 4. The original speech (a) and the noisy speech (b)

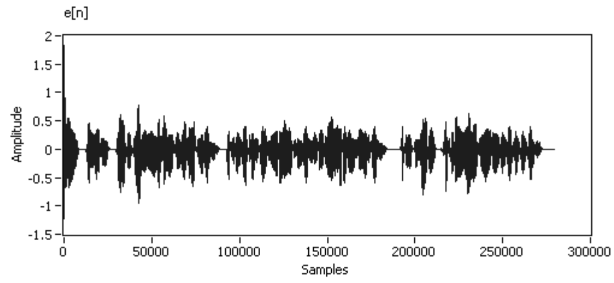


Figure 5. Filtered speech with the adaptive LMS-FIR

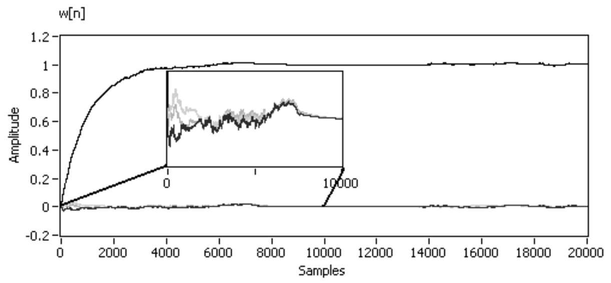


Figure 6. The evolution of the LMS-FIR filter coefficients ($N = 4$, $\mu = 0.002$)

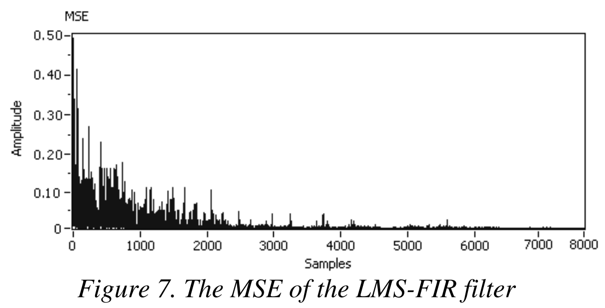


Figure 7. The MSE of the LMS-FIR filter

The performances of the adaptive filtering of a speech signal are summarized in *Table 2*, in which the following can be noticed:

- *SNR_{out}* is significantly improved for a large value of μ;
- *SNR_{out}* decreases as the filter length *N* increases;
- the convergence time increases as the step-size μ decreases, it does not depend on the filter length *N*.

Table 1. LabVIEW FPGA hardware usage provided by the Xilinx Compiler

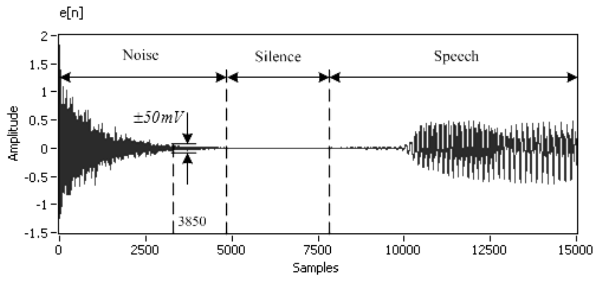
N	Max. Freq. on FPGA (MHz)	Nr. of slices out of *14,336*	Slice Flip Flops out of *28,672*	4 input LUT's of *28,672*	Block RAMs of 96
4	*40.44*	*19%*	*11%*	*13%*	*4.2%*
8	*40.56*	*21.4%*	*13%*	*14.5%*	*4.2%*
16	*40.56*	*25.4%*	*17%*	*16.4%*	*4.2%*

Table 2. The performances of the adaptive filter implemented with the proposed architecture

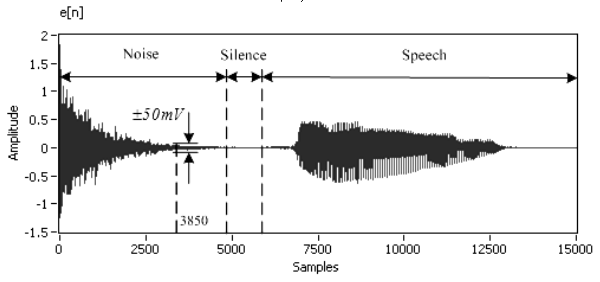
N	μ	SNR [dB]		Convergence (Iteration)
		SNR_{in}	*SNR_{out}*	
4	*0.001*	*-11.54*	*9.96*	*13150*
	0.002		*12.87*	*3850*
	0.01		*18.36*	*1250*
8	*0.001*		*9.94*	*13150*
	0.002		*12.80*	*3850*
	0.01		*17.13*	*1250*
16	*0.001*		*9.90*	*13150*
	0.002		*12.65*	*3850*
	0.01		*15.31*	*1250*

In literature it is known that the step-size μ has a high influence on convergence of the algorithm [1]. In this paper, we developed a simple method to measure the convergence duration. Empirically, we observed that by inserting samples of zero (silence) at the beginning of the original speech signal, the filter response will begin with a transient state. The duration of this transient state represents precisely the convergence duration, which is easily observed and measured. Thus in experiments we inserted *5000* samples of zero, which at the sampling frequency of *16 kHz* means a delay of *0.31 s* in addition to the total duration of the input signal (*17 s*). This delay is not considered as being large, since it is only *1.82 %* of all the duration of the input signal.

Figure 8 shows the beginning part of the speech signal at the output of the adaptive filter, with two different sampling rates: *16 kHz* and *44.1 kHz*. One can notice that the convergence time is the same, when the output noise fells below *±50 mV*. This value is far below the level of the noise superimposed over the speech signal. Therefore, we can state that the convergence time does not depend on the sampling frequency, but with a high bit-rate and a "silence" of some length at the beginning of the process, we can provide the necessary convergence rate prior to receive / process the speech signal.



(a)



(b)

Figure 8. Measuring the convergence time for a speech signal with (a) Fs = 44.1 kHz ; (b) Fs = 16 kHz

V. CONCLUSIONS

This paper proposes a hardware architecture used to implement the LMS-FIR adaptive algorithm on a reconfigurable platform. The architecture is aimed for speech processing using the NI cRIO-9104 FPGA module and the LabVIEW FPGA toolkit.

The proposed architecture was developed to bring better performances in comparison with the structure presented in [5]. The performances of the proposed LMS architecture are demonstrated on the noise cancellation application. The metrics used in the evaluation process are the convergence performance, filtered signal's SNR, MSE and hardware usage. The results yield to high quality of the processed signals, showing at least *20 dB* improvement of *SNR_{out}*, and low MSE values, with efficient use of hardware resources for implementation.

REFERENCES

[1] S. Haykin, *Adaptive Filter Theory*, Fourth Edition, Prentice Hall, Upper Saddle River, N.J., 2002

[2] A. Elhossini, S. Areibi, R. Dony, "An FPGA Implementation of the LMS Adaptive Filter for Audio Processing", *IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006),* pp. 1-8, 2006

[3] T. J. Moir, "FPGA based crosstalk-resistant adaptive decorrelator", *Innovative Computing Technology (INTECH), Third International Conference*, pp. 504 – 509, 2013

[4] P. Shashikala, T.G. Renjith Kumar, H. Subramani, "An FPGA Implementation of the LMS Adaptive Filter for Active Vibration Control", *International Journal of Research in Engineering and Technology*, Vol. 02. No. 10, pp. 1-10, 2013

[5] Szopos, H. Hedesiu, "Labview FPGA Based Noise Cancelling Using the LMS Adaptive Algorithm", *Acta Technica Napocensis – Electronics and Telecommunications*, Vol. 50, No. 4, 2009