International Journal of Knowledge Engineering
ISSN: 0976-5816 & E-ISSN: 0976-5824, Volume 3, Issue 2, 2012, pp.-207-211.
Available online at http://www.bioinfopublication.org/jouarchive.php?opt=&jouid=BPJ0000230

ADAPTIVE NOISE CANCELLATION FILTER USING LMS ALGORITHM ON AN FPGA FOR MILITARY APPLICATIONS

BHARATH K.S., ARA A., RAMANI N., BINDU K. AND HEGDE R.*

Dept. of Telecommunication Engineering, BMS College of Engineering, Bangalore-560 019, Karnataka, India.
*Corresponding Author: Email- rajeshwari.hegde@gmail.com

Received: October 25, 2012; Accepted: November 06, 2012

Abstract- Adaptive filters have gained popularity over the years due to their ability to adapt themselves to different environments without substantial intervention by the user. This paper implements an adaptive noise cancellation filter on a Field Programmable Gate Array (FPGA). The filter is designed using the Least Mean Square (LMS) algorithm due to its computational simplicity, robust behavior when implemented in finite-precision hardware and well understood convergence behavior. To check the correctness and response of the adaptive noise cancellation filter, the LMS algorithm is simulated using the Matlab/Simulink tool. The Simulink model is used as a reference to implement the algorithm using the Xilinx Tool Box. To implement the adaptive filter on hardware, the System Generator (SysGen) tool in the Xilinx block set is used to generate the bit file which is downloaded onto the FPGA through hardware co-simulation. The analysis shows that the FPGA's response is almost identical to that of the simulated responses. Hence, hardware implementation of the adaptive filter using an FPGA is efficient and reliable. This paper presents the adaptive noise cancellation filter using LMS algorithm on an FPGA board suitable for noise cancellation in Pilot helmets for Military Missions.

Citation: Bharath K.S., et al. (2012) Adaptive Noise Cancellation Filter Using LMS Algorithm on an FPGA for Military Applications. International Journal of Knowledge Engineering, ISSN: 0976-5816 & E-ISSN: 0976-5824, Volume 3, Issue 2, pp.-207-211.

Introduction

In most of the noise cancellation applications, the characteristics of the signal changes very fast and hence requires the utilization of adaptive algorithms, which converge rapidly [1]. Hence LMS algorithm has been used for many adaptive filter applications including noise cancellation, echo cancellation etc. because of its simplicity in computation and implementation [2]. An adaptive filter is a computational device that attempts to model the relationship between two signals in real time in an iterative manner. They are often realized either as a set of program instructions running on an arithmetical processing device such as a microprocessor or a digital signal processor, or as a set of logic operations implemented in a Field-Programmable Gate Array (FPGA) or in a VLSI integrated circuit. Adaptive filters are self learning filters. As the signal into the adaptive filter continues, the filter coefficients adjust themselves to achieve the desired result, like identifying an unknown filter or cancelling noise in the input signal.

The [Fig-1] shows the block diagram of the adaptive filter in which an input signal x(n) is fed into the adaptive filter, that computes an output signal y(n). The output signal is compared to a desired signal d(n). The difference between them constitutes the error signal, e(n). This signal is fed into a procedure which alters or adapts the parameters of the filter in a well-defined manner. The output of the adaptive filter becomes a better match to the desired response signal through this adaptation process, such that the magnitude of e(n) decreases over time [3].

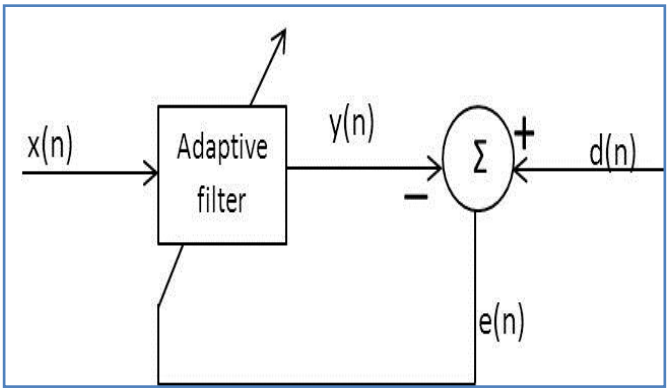

Fig. 1- Adaptive Filter

There are a number of adaptive algorithms developed to update the filter coefficients of an adaptive filter like the Recursive Least Square, QR- RLS, State Walk Model, LMS etc. This paper implements the LMS algorithm, due to its reduced complexity and less computations. This is advantageous in the implementatise cancellation filter on a Field Programmable Gate Array (FPGA) with optimal resource utilization.

The paper is organized as follows. Section II deals with the LMS algorithm. Section III explains the hardware co simulation. Section IV deals with the Hardware specification for the implementation of the filter. Section V deals with the results and discussions. Section VI concludes the paper.

LMS Algorithm

The LMS algorithm is based on the method of steepest descent using the negative gradient of the instantaneous squared error, i.e. J ≈ e2 (n), was devised by Windrow and Hoff to study the pattern-recognition machine in 1959 [4].

An iterative approach is used by the LMS algorithm to adapt the tap weights to the optimum Wiener –Hopf solution.

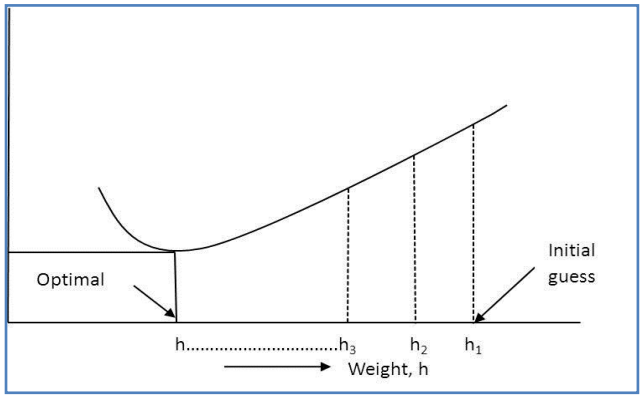

Fig. 2- Gradient search

The algorithm uses a gradient descent, as in [Fig-2] , to estimate a time varying signal. It finds a minimum, if it exists, by taking steps in the negative direction of the gradient. By taking steps in the negative direction of the gradient, it finds a minimum, if it exists, by adjusting the filter coefficients to minimize the error [5]. The filter structure for the basic LMS algorithm is the transversal filter with time varying tap- weight vector h(n), the input stochastic process x (n) and the desired response d(n). The output of the transversal filter is given by

$$y(n) = h^T(n) * x(n) \quad (1)$$

Where x(n) is the vector of the N most recent samples of x(n). The error between the output and the desired response is denoted by e (n), where

$$E(n) = d(n) – y(n) \quad (2)$$

$$E(n) = d(n) - hT(n) * x(n) \quad (3)$$

The purpose of LMS algorithm is to iteratively update the tap weight vector h(n) so as to minimize the performance criteria J.

$$J = \varepsilon[e2(n)] = \varepsilon[(d(n) – y(n))2] \quad (4)$$

Hence, the method of steepest descent algorithm starts from some initial weight h(0), which has some associated mean square error J (h(0)). The tap weight is then updated in a manner which causes the mean square error to move in the direction of steepest decrease on the performance surface. The negative of the gradient of the performance surface J(h) determines the direction. The weight of the filter is updated according to the equation

$$h(n+1) = h(n) – 0.5 * \mu * \blacktriangledown) \quad (5)$$

Where ▼ is the true gradient, given by

$$\blacktriangledown = \eth J/\eth h(n) \quad (6)$$

$$\blacktriangledown = \eth J/\eth h(n) = -2 * p + 2 * R * h \quad (7)$$

Where R is the autocorrelation between the input signal x(n) and p is the cross correlation between the desired response d(n) and input x(n) given by

$$R = \varepsilon[x(n)) * xT(n)] \quad (8)$$

$$P = \varepsilon[x(n) * d(n)] \quad (9)$$

The filter coefficients are hence updated by the following equation

$$h(n+1) = h(n) – 0.5 * \mu * (-2*p + 2*R*h) \quad (10)$$

The drawback of the gradient methods in real time applications of adaptive filters is that the stochastic parameters of the input signals like the mean and variance should be known. In real time scenario, little or no knowledge of the signals is available. Hence, the calculation of these stochastic parameters is not feasible.

The above limitation of the gradient methods is eliminated by the LMS algorithm. The LMS algorithm does not strictly follow the gradient search. It is a more approximate approach to adaptive filtering. The LMS algorithm takes the instantaneous values of the signals into account rather than their stochastic parameters. Hence the computation of the filter coefficients becomes simple even when no knowledge about the signals is available. The equations 11 and 12 give the approximated autocorrelation (R) and the cross correlation (p) functions.

$$R \cong x(n) * x^T(n) \quad (11)$$

$$p \cong d(n) * x(n) \quad (12)$$

Hence, the approximated gradient known as the stochastic gradient is obtained by ignoring the expectation operators:

$$\blacktriangledown_k = -2 * p + 2 * R * w(n) \quad (13)$$

$$\blacktriangledown_k = 2 * (x(n)* x^T(n)) * w(n) - 2* (n)* x(n) \quad (14)$$

$$h(n+1) = h(n)–\mu*[2*x(n)*x^T(n)*h(n)-d(n)*x(n)] \quad (15)$$

But, from equation (1) we have,

$$y(n) = h^T(n) * x(n)$$

$$h(n+1) = h(n)+ 2 * \mu * x(n) * [d(n) – y(n)] \quad (16)$$

But, from equation (2) we have, e(n)= d(n) – y(n)

Therefore,

$$h(n+1) = h(n) + 2*\mu*x(n)*e(n)) \quad (17)$$

Hence, the updated filter coefficient depends on the previous value of the filter coefficient and a scaled down factor of the product of the input signal x(n) and the error signal e(n).

To guarantee the stability and robustness of the algorithm, the step size should be carefully chosen. The upper bound on μ provides an important guide in the selection of a suitable step size for the LMS algorithm. A smaller step size μ is used to prevent instability for a

larger filter length. Also, the step size is inversely proportional to the input signal power P. Therefore, a stronger signal must use a smaller step size, while a weaker signal can use a larger step size.

The autocorrelation matrix is necessary for the convergence [6]. The convergence factor of LMS algorithm is given by

$$0< \mu < 2 / \lambda_{max}$$

Here, λ_{max} is the largest Eigen value of the correlation matrix R [7].

In practice, we may not know λ_{max}, it may be time-varying and hence another result can be obtained by the sum of the Eigen values as follows.

$$\text{tr (R)} = \sum_{i=1}^{M} \lambda i \geq \lambda_{\max}$$

Also, the step size is inversely proportional to the input signal power P which shows that a stronger signal uses a smaller step size, while a weaker signal uses larger step size [8].

$$0 < \mu < 2/(M*P)$$

Hardware Co-Simulation

The SysGen for DSP™ is the industry's leading high-level tool for designing high-performance DSP systems using FPGAs. System Generator for DSP is part of both the DSP and System Editions of ISE® Design Suite.

The engineers with little FPGA design experience can quickly create production quality FPGA implementations of DSP algorithms using System Generator for DSP. DSP modeling, Bit and cycle accurate floating and fixed-point implementation, Automatic code generation of VHDL or Verilog from Simulink, Hardware co-simulation, Xilinx Power Analyzer (XPA) Integration and Hardware / software co-design of embedded systems are the key features of Xilinx SysGen.

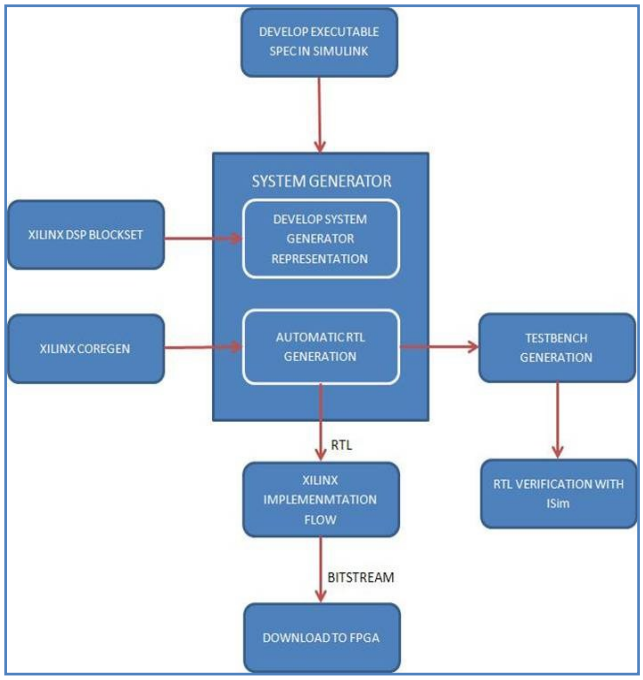

Fig. 3- System Generator Flow

System Generator provides hardware co-simulation, there by incorporating a design running in an FPGA directly into a Simulink simulation. The bit stream is created by the Hardware Co-Simulation compiler and is associated to a block. The Xilinx SysGen tool runs implementation processes like map, translate and place and route with the help of Xilinx ISE in the background, and outputs a bit stream. Since the results for the compiled portion are calculated in hardware while the design is simulated in Simulink, the compiled portion can be tested in actual hardware and can speed up simulation dramatically. [Fig-3] shows the implementation flow of the Xilinx System Generator.

A model can be co-simulated, provided it meets the requirements of the underlying hardware board. The model must include a System Generator token. The system generation token block defines how the model should be compiled into hardware.

The fields on the System Generator token dialog box are automatically configured with settings appropriate for the selected compilation target, When a compilation target is selected.

The configuration bit stream contains the hardware associated with the model, and also contains additional interfacing logic that allows System Generator to communicate with the design using a physical interface between the board and the PC. A memory map interface over which System Generator can read and write values to the input and output ports on the design is included in the logic.

System Generator automatically creates a new hardware co-simulation block once the design is compiled into an FPGA bit stream. The Co-Simulation block created for a 4 tap LMS filter is shown in [Fig-4]. During simulation, a hardware co-simulation block interacts with the underlying FPGA board, automating tasks such as device configuration, data transfers, and clocking. The corresponding data is sent by the block to the appropriate location in hardware after a value is written to one of the block's input ports. Similarly, the data is retrieved by the block from hardware when there is an event on an output port.

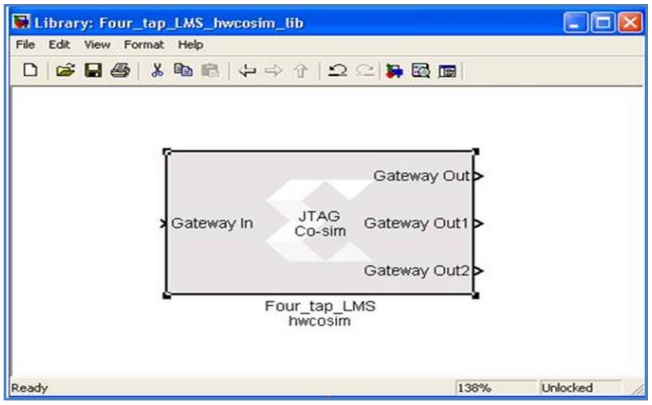

Fig. 4- Hardware Co-Simulation block for 4 tap LMS filter

Hardware Specification

The specific functions such as device-to-device interfacing, signal processing, data communication, timing and control operations, data display and other functions performed by a system are provided by logic devices. They are of two types namely, Fixed logic and Programmable logic.

The advantages of using programmable logic over fixed logic is, it consumes less board space for an equivalent logic, designs can be readily changed without rewiring or replacing components, designs can be implemented faster and are economical.

Logic devices like PAL, PLA GAL and CPLDs are now being replaced with FPGAs. The FPGA is an integrated circuit that contains identical logic cells ranging from 64 to over 10,000 which can be viewed as standard components. A matrix of wires and programmable switches interconnect these individual cells. The array of logic cells and interconnects form a fabric of basic building blocks for logic circuits. These basic blocks combine to create complex designs by creating the desired circuit.

A typical integrated circuit performs a particular function defined at the time of its manufacture, where as the FPGA's function is defined by a program written by someone other than the device manufacturer. FPGAs are essentially like “blank slates” that the end user can program for any logic device. The FPGA architecture is more sophisticated compared to that of the CPLD. It does not use PAL/PLA type arrays and also has much greater densities than CPLDs. A typical FPGA has many times more equivalent gates than a typical CPLD. The logic producing elements in FPGAs are generally much smaller than in CPLDs and there are many more of them.

FPGAs are considered for the hardware implementation of the adaptive noise cancellation filter because they can implement different logic functions by re-programming them. This results in flexibility, that is the same hardware can implement different function unlike ASIC. Different resources are allocated for different functions and therefore the operation is parallel/concurrent in contrast to a DSP processor in which the flow is sequential and they use shared resources. This reduces the delay time which in turn increases the speed and makes it more efficient and accurate. Hence, an FPGA can implement the filter efficiently and at a greater speed.

The Xilinx Virtex®-5 family of FPGAs provides the newest most powerful features in the FPGA market. Since the Virtex-5 family contains five distinct platforms/sub-families, it is the most choice offered by any FPGA family. It is built on a 65-nm state-of-the-art copper process technology and is a programmable alternative to custom ASIC technology. The radiation effects are tolerable compared to that of the FPGAs in the higher families like Vitex-6, Virtex -7, etc. Also, the number of on chip resources is more and also has better resistance to temperature variations compared to that in the lower families like Virtex-3, Virtex-4, etc. Virtex-5 FPGAs offer the best solution for addressing the needs of high-performance logic designers, DSP designers and embedded systems designers with unprecedented logic, hard/soft microprocessor, DSP and other capabilities.

Hence, the paper implements the adaptive noise cancellation filter on XC5VFX130T FPGA of the Virtex-5 family.

Result and Discussion

The bit file generated using Hardware Co-Simulation feature of SysGen for execution on the target board is downloaded onto the board via JTAG interface. The FPGA response is again taken back to Simulink via JTAG, where it is displayed. The output of the adaptive filter and the error signal simulated using Matlab/Simullink and Xilinx block set are shown in [Fig-5a] and [F-g-5b] respectively. The output from the Virtex-5 is shown in [Fig-5c]. The hardware is in good agreement with the simulated responses. The hardware response shows the convergence of the error signal to a minimum value in accordance to the LMS algorithm.

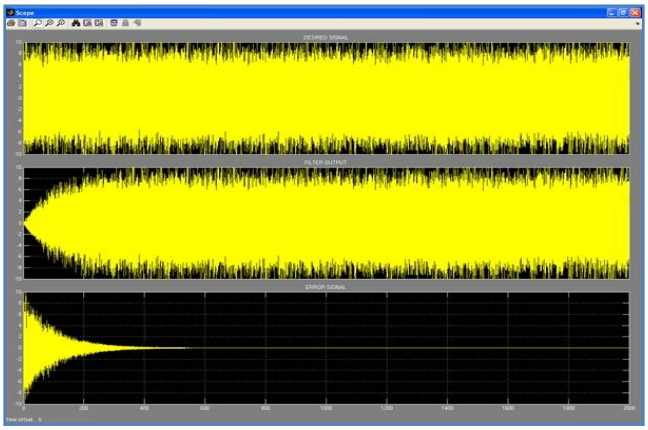

Fig. 5a- Response of the adaptive filter model using simulink block set

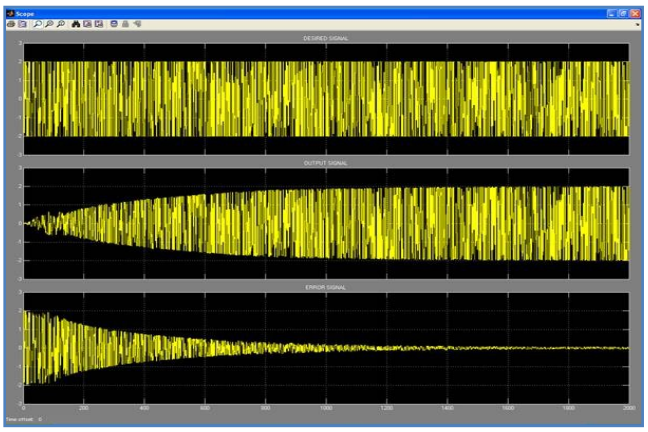

Fig. 5b- Response of the adaptive filter model using xilinx block set

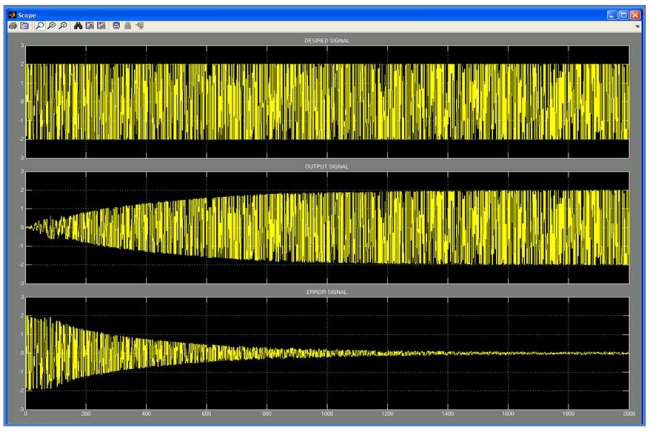

Fig. 5c- Response of the adaptive filter model from xilinx virtex-5 FPGA

Conclusion

In this paper, an FPGA-based adaptive noise cancellation filter using LMS algorithm suitable for military applications is presented. The purpose of this paper being the implementation of LMS filter,

the designed model of LMS algorithm was simulated in Matlab using Simulink block set. After successful simulation of the LMS model, the Algorithm was replicated using the Xilinx toolbox in Simulink. Using the System Generator tool of Xilinx, the Bit-file is generated for downloading onto the Xilinx Virtex-5 FPGA based target board, Xilinx ML510. The results proved that the simulated and implemented responses are in good agreement with each other. Hence, it was proved that the FPGA is a reliable hardware platform to implement the Adaptive noise cancellation filters. Owing to the simplicity of LMS algorithm and the high speed of FPGAs due to concurrency, Adaptive noise cancellation filters using LMS algorithm on FPGAs can be used for critical applications like Noise cancellation in Pilot helmets for Military Missions.

References

[1] Sayed A.H. and Iotfizad M. (2010) *International Journal of Computer and Electrical Engineering*, 2(2).

[2] Górriz J.M., Javier R., Cruces-Alvarez S., Carlos G.P., Elmar W. L., Deniz E. (2009) *IEEE Signal Processing Letters*, 16(1).

[3] Haykin S. (1988) *Adaptive Filter Theory*, Prentice Hall.

[4] Widrow B. and Hoff M.E. (1960) *IRE WESCON Convention Record*, 96-104.

[5] Douglas S.C. (2000) *Convergence Issues in the LMS Adaptive Filter*, CRC Press LLC.

[6] Karni S. and Zeng G. (1989) *IEEE Trans. Circuits Syst.*, 36(7), 1011-1012.

[7] Widrow B. and Stearns S.D. (1985) *Adaptive Signal Processing,* 59.

[8] Kong-Aik Lee and Woon-Seng (2011) *Subband Adaptive Filtering Theory and Implementation*, Wiley Publications.