Omid SHARIFI-TEHRANI

Islamic Azad University, Majlesi branch, Young Researchers Club

Novel hardware-efficient design of LMS-based adaptive FIR filter utilizing Finite State Machine and Block-RAM

Abstract. *A novel pure-hardware design of LMS-based adaptive FIR filter core is proposed which is highly efficient in FPGA area/resource utilization and speed. Unlike HW/SW co-design and other pure-hardware methods, the required area/resource is reduced while keeping the speed in an appropriate level by taking advantage of finite state machine (FSM) and using internal block-rams (BRAM). This model because of being completely general (device independent), gives the ability of implementation on different FPGA brands and thus, is suitable for embedded systems, system-on-programmable-chip (SoPC) and network-on-chip (NoC) applications.*

Streszczenie. *Opisano projekt filtru adapatacyjnego SOI który może byc wykorzystany w technice FPGA. Dla zapewnienia odpowiedniej szybkości zastosowano metodę FSM (finite state machine) i wewnętrzny RAM. Układ może być wykorzystany w systemach typu SoPC (system on programmable chip). (**Sprzętowy projekt filtru adaptacyjnego SOI z mechanizmem FSM**)*

Keywords: FPGA, Adaptive FIR Filter, LMS Algorithm, Embedded Systems.
Słowa kluczowe: filtr adaptacyjny, FPGA.

Introduction

Adaptive signal processing (ASP) is one of the important and most widely used branches of digital signal processing (DSP). Applications such as noise canceling systems, radar, sonar, channel equalizer, channel modeler and etc., require hardware implementation of adaptive processing algorithms, especially adaptive FIR filters with lowest cost and lowest resource utilization. Adaptive FIR filter based on LMS algorithm is highly popular because of its simplicity, low computational cost and linear phase. With the advent of FPGA chips, hardware implementation world of portable and embedded systems was developed dramatically [1]. These chips with the ability of concurrent processing in addition to sequential processing, and having high clock speed, include internal primitives such as DCM blocks, PLL, DLL, HW/SW processor cores, BRAM and etcetera by which the design engineers will be capable of implementing system-on-programmable chip (SoPC), network-on-chip (NoC) and embedded systems with lowest time-to-market and cost [2]. Many works regarding implementation of adaptive FIR filter based on LMS algorithm have been done. Authors in [3] implemented the LMS adaptive filter on Altera Cyclone-II FPGA for active noise control (ANC) application. The filter was designed and synthesized based on conventional multiplier-adder to realize multiply and accumulate (MAC) operation for 64 filter taps with 9-bit signed integer coefficients. Antonio et al. [4] proposed an adaptive FIR filter based on modified LMS algorithm for use in adaptive noise canceller. Mohammad Bahura and Hassan Ezzaidi [5] presented a sequential architecture of a pipelined LMS-based adaptive noise cancellation to remove power line interference (50/60 Hz) from electrocardiogram (ECG). This architecture was implemented on FPGA using XUP Virtex-II-Pro development board and Xilinx system generator (XSG). Authors in [2] proposed an FPGA-based fixed-point standard LMS algorithm core for adaptive signal processing realization in real-time. The LMS core was deigned as an FPGA brand-independent model so could be used in any SoPC applications. They used 50 filter-coefficient taps with 17-bit word length. Rodellar et al. [6] implemented an ANC system based on LMS adaptive FIR filter for robust speech enhancement interface, on both Xilinx Virtex4 and Altera Stratix family. They used a fixed number of word-length (40-bits) for each variable. Authors in [7] presented a modified fixed-point LMS algorithm for hardware implementation. This modified fixed-point LMS algorithm was deduced from theoretical fixed-point LMS algorithm regarding rounding errors. Data entry problems were studied and efficient solving methods were proposed. They implemented a modified 8-weight fixed-point LMS algorithm on FPGA. Chaitanya et al. [8] implemented LMS adaptive filtering algorithm and its variations in software and as well as HW/SW co-design for the NIOS-II soft processor. They claimed an improvement in clock-cycle number required when implementing on HW/SW co-design over a pure software implementation. Ahmed Elhossini [9] presented three different architectures for implementing LMS adaptive filtering algorithm, using a 16-bit fixed-point arithmetic representation for audio processing. The architecture was implemented using the Xilinx multimedia board as an audio processing system. They used Xilinx Virtex-II FPGA chip for implementing pure hardware, pure software and HW/SW co-design architectures using MicroBlaze soft processor core. Jalili et al. [10] implemented an ANC system on the Altera Stratix family. The controller used LMS algorithm for updating adaptive weights array with M=64 (number of weights). A comparison of this implementation with DSP processor implementation, showed remarkable speed-up in FPGA architecture.

But in many of these proposed hardware implementations, a large volume of FPGA internal resources is occupied and thus, less space will remain for other hardware (if any) required to be implemented on FPGA. Moreover, the overall speed will be reduced because of inevitable routing network delay. In some of these designs, HW/SW co-design is used by which, an embedded hardware or software processor core is implemented and some part of the design is moved to software section. Although this method has got a good flexibility, but is not efficient in case of resource utilization and speed [1]. Such implementations are also specific FPGA-brand dependent and not capable of being implemented on other brands, thus lack the generality and independency. Despite the fact that pure hardware implementation is highly efficient, its flexibility is a bit fewer and the synthesis is more complex. FPGA chips include two types of RAM. One is distributed RAM which is synthesized and implemented by internal logics. The other is internal Block-RAM (called BRAM) that is independently embedded in chip wafer during fabrication process. An important note that most of design engineers forget is that if you do not use them you loose them, because they can not be used for anything but RAM and FIFO.

This paper aims to design and implement a pure hardware adaptive FIR filter core based on LMS algorithm by defining FSM and using FPGA internal Block-RAMs, and a model with low resource utilization and high speed which is more efficient regarding other implementations is proposed. Coefficients and input tapped delay line (TDL) instead of being implemented by internal common logics of FPGA, are stored by internal Block-RAMs and thus, logic resource utilization is decreased remarkably. This notable reduction in logic resource utilization makes the proposed model capable of being implemented even on old and obsolete FPGA brands (such as Spartan-II from Xilinx Corporation). During design process of proposed core, some implementation/design problems and bottlenecks were met that could be solved by some tricky key rules which are described in this paper.

LMS algorithm

According to LMS algorithm theory [11] and Fig. 1, the error of filter is calculated by (1).

(1) $$e_n = d_n - y_n$$

(2) $$y_n = W_n^T . U_n$$

The error is simply the desired output (d_n) minus the filter output (y_n). W_n is the filter weight vector and U_n is the filter input vector through taped delay line (TDL). The weight update equation is defined by:

(3) $$W_{n+1} = W_n + \eta . e_n . U_n$$

This is the Widraw-Haff LMS weight update algorithm [11]. η is called learning rate or step-size by which the convergence speed and stability of filter is adjusted. Evaluating the effect of choosing different η values and its relation to eigenvalue spread of input signal is beyond the scope of this article.

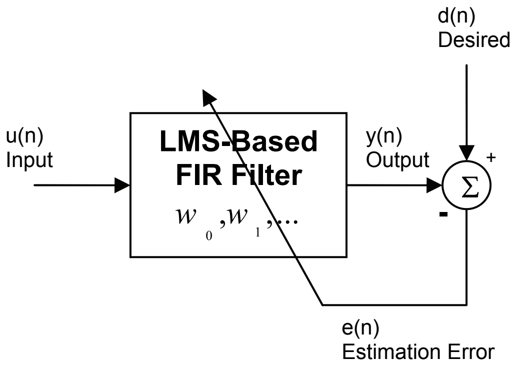

Fig. 1. Block Diagram of Adaptive FIR Filter

Fixed-point standard LMS model

In the designed core, it was considered that analog to digital converter (ADC) unit provides 12-bit signed binary output , so the model for input data-bit allocation was designed in two's complement form with 17-bit length as in Table 1. One bit is for sign bit, 12 bits are for representing integer part of number and one bit fraction-length is dummy and not used for input/output data, but is necessary for weight update. Table 2 demonstrates the LMS weight bit-allocation. As is evident from Table 2, the most bits are assigned for fraction part and it is because filter weights are normally between -1 to +1. The output of LMS filter is calculated by equation (2) and is accomplished by FPGA internal MULT18X18 multiplier block, thus the LMS output will be 34-bit long. To fit in 17-bit long format and having minimum discarding error, Y is truncated from 31 down to 15 bit-indexes. As mentioned beforehand, weights are updated based on equation (3), hence, the resulting length for W is 51-bit long. Again to fit in 17-bit long format and having minimum discarding error, W is truncated from 33 down to 17 bit-indexes. The overall entity of proposed model is depicted in Fig. 2.

Table 1. Input Data Bit-Allocation

Sign Bit	Guard Bits	Word Length	Fraction Length
1	3	12	1

Table 2. Weights Bit-Allocation

Sign Bit	Word Length	Fraction Length
1	1	15

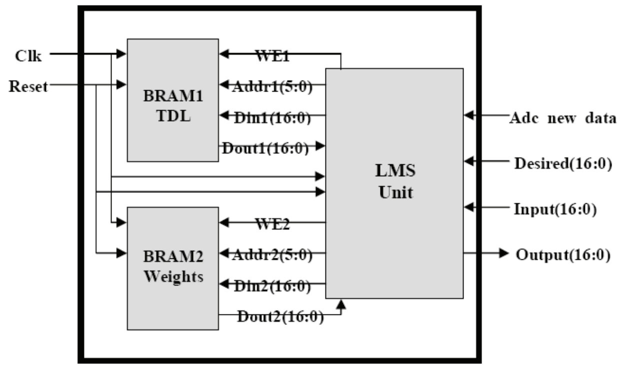

Fig. 2. Entity of Proposed Adaptive FIR Filter Core

Problems, bottlenecks and solutions

There are some key tips and tricks in VHDL programming which many of digital design engineers forget to use them [12]. If these rules are obeyed, the derived hardware will be less complex and much easier to be synthesized and implemented. Also resource utilization will be reduced remarkably. Some of these rules are illustrated here. The first key rule is about using PROCESS and consists of three tips:

1- Include all input signals in the sensitivity list.

2- Include ELSE branch for IF statements.

3- Assign a value to every signal in every branch.

Since VHDL standard implies that if a signal is not assigned in a process, it will keep its previous value, hence, a latch will be inferred to keep the previous value, resulting in more resource utilization and not an optimum design. Obeying the mentioned tips will prevent inferring latches. An efficient and easy way of implementing these tips is using the FSM style proposed by Pong. P. Chu in [12]. In this article the same FSM style has been used which results in low area/resource utilization by preventing latch inferring.

The second key rule is about using FPGA internal Block-RAM called BRAM. There are two types of RAM; Block-RAM and distributed RAM. The latter is derived by internal common gates, but the first one is embedded into FPGA chip as hardware core during fabrication process. This is so important to know that if you do not use BRAMs, you loose them because they can not be used for anything else but RAM. Thus, instead of using internal common gates of FPGA to store TDL elements and weight elements which results in more area/resource utilization, we can use pre-prepared internal BRAM units. In this paper two BRAMs have been used for storing and retrieving TDL and weight elements leading to an efficient and optimum design. In this way, a bottleneck was encountered. Since processes with the same clock in sensitivity list are synchronous and concurrent, and on the other hand, VHDL standard implies that signals in a process accept their new value at the end of the process and not within the process, hence the design met latency and delay problem. The first solution was introducing dummy states in FSM to ensure safe data retrieving from BRAMs. But the main drawback of this method is requiring higher clock speed to compensate for these dummy states time-wasting. The second solution was using a double (2x) clock speed for BRAMs, with the main disadvantage of introducing two clock domains and violating synchronous design methodology. The final solution was using the global clock for all the design parts and triggering BRAMs processes by falling edge of clock instead of rising edge. This will ensure that after the main process sends read command and address to be read to the BRAMs, for the next rising edge of clock, the data of BRAMs will be stable and safe to be read by main process, because the falling edge of clock has triggered the BRAMs processes in advance of the current rising edge and there is enough time for reading/writing procedure. This method prevents us from introducing extra states (dummy states), using high speed clock or violating synchronous design methodology by using extra clock domains. This method also leads to very low area/resource utilization.

Another important issue is how to use (describe) BRAM. There are three methods for defining BRAM;

1- Using CoreGen program
2- HDL instantiation
3- Behavioural HDL inference template

The first two are FPGA-brand specific (here Xilinx FPGA) and thus are not general. But the third is a semi-device-independent behavioral description [12]. Because of making the proposed core, FPGA-brand independent so that can be directly implemented on any FPGA (Xilinx, Altera, Actel, etc.) and also because of behavioral description clarity, the third method was used in this article.

Finite state machine

The designed finite state machine that controls data path is depicted in Fig. 3. The conditions under which, the states change are illustrated in Table 3.

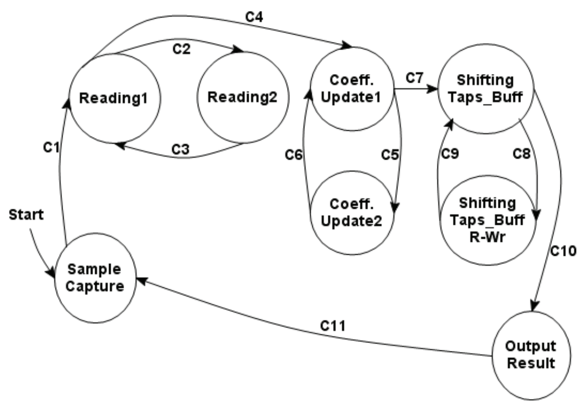

Fig. 3. Proposed Finite State Machine (FSM)

Table 3. State Change Conditions

State	Condition Description
C1	If Sample Capture is Done
C2	If BRAMs Current Address Fetch, Decode and Load, Completed
C3	Current Input and Weight Read, Completed
C4	All Tap Delay Line (TDL) Reading, Completed
C5	BRAMs Current Address Fetch, Decode and Load, Completed
C6	Current Weight Updating and Writing it to BRAM, Completed
C7	All Weights Updating, Completed
C8	BRAMs Current Address Fetch, Decode and Load, Completed
C9	Current Input Insertion-Shifting, Completed
C10	One-Step Shifting of All TDL, Completed
C11	Output Calculation and Loading, Completed

The FSM includes eight major states as described below.

Sample Capture: if adc_new_data=1, catches new sample from ADC unit.

Reading1: Address fetch, decode and load of both TDL (BRAM1) and coefficients (BRAM2).

Reading2: BRAMs-Data retrieve. Calculating output of filter (Y) and error (E).

Coef_Update1: Address fetch, decode and load of both TDL (BRAM1) and coefficients (BRAM2).

Coeff_Update2: BRAMs-Data retrieve. Calculating weight update and assigning new values to weights.

Shifting-Taps-Buff: address fetch, decode and load of TDL-BRAM (BRAM1).

Shifting-Taps-Buff-R-Wr: TDL-BRAM-Data retrieve. Shifting all TDL one step.

Output Result: puts the calculated filter result at the output ports and goes to Sample Capture state.

Simulation of proposed model

For simulation and verification of designed core, MATLAB Simulink was used to create the essential test-data as can be seen in Fig. 4. The Simulink file is consist of signal generators, 12-bit ADC quantize and etcetera. Test-data is created with Simulink and then is exported to MATLAB workspace. An M-file is designed to write the results in separate text files to use in ModelSim simulator software. The VHDL core file and test-bench file are compiled in ModelSim as VHDL93 standard. At the end of simulation, the output data which has been written in a text file is brought to MATLAB workspace and is converted from double to int16. Then the required plots are derived. Data are chosen from database provided in [2].

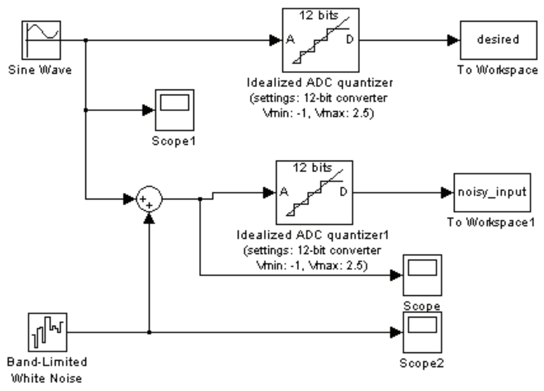

Fig. 4. Test Data Creation in MATLAB Simulink

Two simulations are provided here to measure how well the new proposed model works. Fig. 5 and Fig. 6 show residual error for both the new proposed model and the previous model developed in [1]. Tables 4 and 5 illustrate the performance comparison. η was considered to be $1/2^{15}$. As is evidence, new proposed method has better performance improvement as well as better (lower) resource utilization as will be shown then.

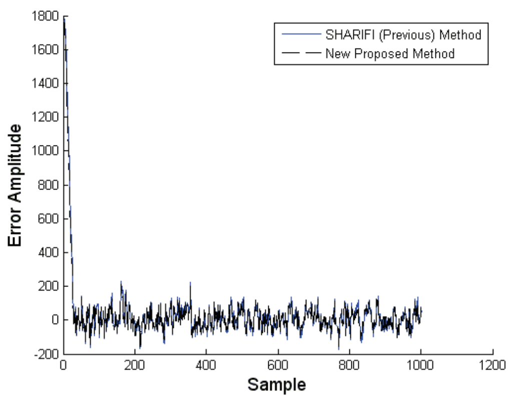

Fig. 5. Residual Error (Learning Curve) of a 200 hertz sine input signal which is degraded by band limited white noise with 0.015 noise power (PSD).

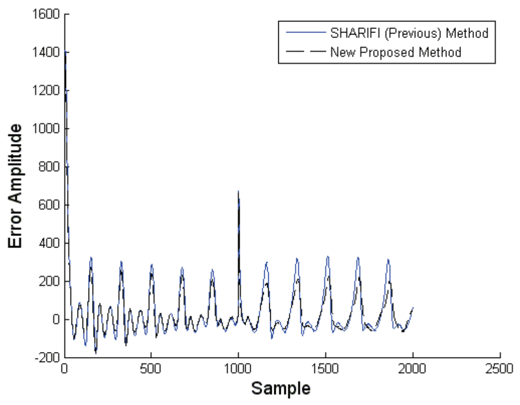

Fig. 6. Residual Error (Learning Curve) of a 200 hertz sine input signal which is first degraded by a 600 hertz sine signal with amplitude of 0.5 and phase of $\pi/3$, and after some time by a 400 hertz sine signal with amplitude of 0.5 and phase of $\pi/4$.

Table 4. Performance Comparison of Fig. 5

Input SNR	Output SNR		SNR Enhancement	
	Previous Method	New Method	Previous Method	New Method
1.0831dB	8.7515dB	9.4794dB	7.6684dB	8.3963dB

Table 5. Performance Comparison of Fig. 6

Input SNR	Output SNR		SNR Enhancement	
	Previous Method	New Method	Previous Method	New Method
1.1399dB	5.9211dB	6.7650dB	4.7812dB	5.6251dB

Synthesis-implementation results

The proposed adaptive FIR filter core is synthesized and implemented on different FPGA families using Xilinx ISE software version 10. Resource utilization results are given in following tables for comparison with other methods and better judgment.

Table 6. Resource Utilization on XC2V250-6fg256

	Proposed Method	Method in Ref [9]
Slice Registers	2%	23%
4 input LUTs	7%	10%
Occupied Slices	7%	14%
RAMB16s	8%	0%
MULT18X18s	8%	59%
Maximum Frequency (MHz)	178	65

Table 7. Resource Utilization on XC4VSX25-12ff668

	Proposed Method	Method in Ref [6]
Slice Registers	1%	24%
4 input LUTs	1%	24%
Occupied Slices	1%	25%
FIFO16/RAMB1s	1%	18%
DSP48s	1%	18%
Maximum Frequency (MHz)	124	92

Table 8. Resource Utilization on XC2VP30-7fg676

	Proposed Method	Method in Ref [5]
Slice Registers	1%	2.4%
4 input LUTs	1%	2%
Occupied Slices	1%	2.1%
Bonded IOBs	12%	8.8%
RAMB16s	1%	1%
MULT18X1s	1%	1.4%
Maximum Frequency (MHz)	231	163

Table 9. Resource Utilization on XC2S300E-6ft256

	Proposed Method	Method in Ref [4]
Slice Registers	1%	57%
4 input LUTs	14%	Not Available
Occupied Slices	15%	Not Available
Block RAMs	25%	Not Available
GCLKs	25%	Not Available
Maximum Frequency (MHz)	40	50

Table 10. Resource Utilization on EP2C35U484C8-Cyclone II

	Proposed Method	Method in Ref [3]
Logic Elements	<1%	8.3%
Combinational Functions	<1%	Not Available
Dedicated Logic Registers	<1%	Not Available
Pins	17%	Not Available
Memory Bits	<1%	Not Available
Embedded Multiplier 9-bit Elements	6%	Not Available
PLLs	0%	Not Available
Maximum Frequency (MHz)	57	50

Table 11. Resource Utilization on EP2S180F1508C4-Stratix II

	Proposed Method	Method in Ref [10]
Combinational ALUTs	<1%	<1%
Dedicated Logic Registers	<1%	<1%
Pins	5%	5%
Block Memory Bits	<1%	0%
DSP Block 9-bit Elements	<1%	17%
PLLs	0%	0%
DLLs	0%	0%
Maximum Frequency (MHz)	115	118

Table 12. Resource Utilization on EP2S15F484C3-STRATIX II

	Proposed Method	Method in Ref [6]
Combinational ALUTs	2%	55%
Dedicated Logic Registers	<1%	31%
Block Memory Bits	<1%	1.65%
DSP Block 9-bit Elements	4%	56%
PLLs	0%	0%
DLLs	0%	0%
Maximum Frequency (MHz)	117	121

Table 13. Resource Utilization on XC3S1600E-5fg320

	Proposed Method	Method in Ref [1]
Slice Registers	1%	9%
4 input LUTs	1%	11%
Occupied Slices	1%	21%
Bonded IOBs	21%	21%
RAMB16s	5%	0
MULT18X18 SIOs	5%	5%
BUFGMUXs	4%	4%
Maximum Frequency (MHz)	186	50

Discussion on results

As is evident from simulation results, the proposed core has good performance in SNR enhancement as well as fast convergence speed. According to synthesis resource utilization results, the proposed core utilizes very low area/resources on FPGA which is one of the main aims of this article. Low resource utilization, fast convergence speed and FPGA-brand independency make this core very suitable for SoC, SoPC and NoC designs. The generic parameters of core give hardware designers the ability of customizing core by adjusting number of coefficients, length of coefficients, length of inputs and learning rate.

REFERENCES

[1] O. Sharifi-Tehrani and M. Ashourian, An FPGA-Based Implementation of ADALINE Neural Network with Low Resource Utilization and Fast Convergence, *PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review)*, 86 (2010), No. 12, 288-292.

[2] O. Sharifi-Tehrani, M. Ashourian and P. Moallem, An FPGA-Based Implementation of Fixed-Point Standard-LMS Algorithm with Low Resource Utilization and Fast Convergence, *Inter. Rev. on Comp. and Soft. (IReCOS)*, 5 (2010), No. 4, 436-444.

[3] V.R. Mustafa, et al., Design and implementation of least mean square adaptive filter on Altera Cyclone II field programmable gate array for active noise control, *IEEE Sym. on Industrial Electronics Applications - ISIEA*, Kuala Lumpur, Malaysia, (2009), 479-484.

[4] A. Di Stefano, A. Scaglione and C. Giaconia, Efficient FPGA implementation of an adaptive noise canceller, *7th Inter. Workshop on Computer Architecture for Machine Perception*, Italy, (2005), 87-89.

[5] M. Bahoura and H.Ezzaidi, FPGA-implementation of a sequential adaptive noise canceller using Xilinx system generator, *Inter. Conf. on Microelectronics - ICM*, Marrakech, (2009), 213-216.

[6] V. Rodellar, A. Alvarez and E. Martinez, FPGA implementation of an adaptive noise canceller for robust speech enhancement interfaces, *4th Southern Conf. on Programmable Logic*, San Carlos de Bariloche, (2008), 13-18.

[7] H. Zheng-wei and X. Zhi-yuan, Modification of theoretical fixed-point LMS algorithm for implementation in hardware. *2nd Inter. Sym. on Electrical Commerce and Security - ISECS*, Vol. 2 China, IEEE Computer Society, USA, (2009), 174-178.

[8] K.S. Chaitanya, P. Muralidhar and C.C. Rama Rao, Implementation of reconfigurable adaptive filtering algorithms. *Inter. Conf. on Signal Processing Systems - ICSPS*, Singapore, (2009), 287-291.

[9] A. Elhossini, S. Areibi and R. Dony, An FPGA implementation of the LMS adaptive filter for audio processing, *Intern. Conf. on Reconfigurable Computing and FPGA's*, San Luis Potosi, (2006), 1-8.

[10] A. Jalili, S. G. Boroujeny and M. Eshghi, Design and implementation of a fast active noise control system on FPGA, *Mediterranean Conf. on Control & Automation*, Athens, (2007), 1-4.

[11] T. Adali, S. Haykin, Adaptive Signal Processing-Next Generation Solutions, *Wiley-IEEE Publication*, New York, (2010).

[12] P.P. Chu, FPGA Prototyping by VHDL Examples, *Wiley Publication*, New York, (2008).

***Authors**: Omid Sharifi-Tehrani, D.Sc. candidate in electrical engineering, he received M.Sc. degree in electrical engineering from IAU Najafabad branch in 2010 and B.Sc. degree from IAU Majlesi branch in 2007. His research interests are adaptive signal processing, hardware implementations and artificial intelligence. Address: Iran, Isfahan, Keshavarz Blvd., Golzare 4 st., Meisam st., No. 41, P.O box: 8178676361. E-mails: omidsht@sel.iaun.ac.ir , omid-sharifi-tehrani@ieee.org , omidsht@yahoo.com*