An FPGA Implementation of an LMS Self-Adjusting Adaptive Noise Cancellation System for Audio Processing

NIAN ZHANG, TAM LE, SASAN HAGHANI
Department of Electrical and Computer Engineering
University of the District of Columbia
4200 Connecticut Ave, NW, Washington, D.C.
USA
nzhang@udc.edu, tam.le@udc.edu, shaghani@udc.edu

Abstract: - This paper presents the design and implementation of adaptive filter using software/hardware co-design concepts and tools for noise cancellation. The adaptive filter system based on the least-mean-square algorithm was analyzed using the MATLAB/Simulink model, and it later was automatically converted from floating point to fixed point for an Intellectual Property Core. This IP Core was placed in Vivado Synthesis Design for synthesis and implementation. Finally, the debugger was run before the audio file was fed in Zedboard, a development board for the Xilinx Zynq. Experimental results show that the proposed hardware implementation method has a high degree of noise cancellation performance.

Key-Words: - FPGA, Least mean square, Intellectual property (IP), Adaptive filters, Adaptive noise cancellation, Matlab Simulink, Vivado Hlx

1 Introduction

In the last decades, adaptive filters design has been a very active area of research and innovative FPGA implementations [1][2]. Adaptive filters are a neural network based filters that can self-adjust its coefficients based on some optimizing algorithms. Adaptive filters have numerous important real-world applications in a wide range of signal processing, control, and communications fields including: 1) signal detection; 2) echo cancellation; 3) noise cancellation and/or suppression; 3) channel equalization; 4) system identification and inverse modeling of unknown systems; 5) forward and backward predictions and adaptive tracking; and 6) spectral analysis.

Among these applications, the FPGA hardware implementations are extremely important whenever real-time parallel processing is needed [3]. State-of-the-Art technological advances in VLSI fabrications have enable field-programmable gate arrays (FPGA) the platform of hardware implementations, especially when timing requirement is very strict. Such hardware implementations can be realized using hardware description languages such as VHDL or Verilog. Modern FPGA chip design contains numerous resources that are essential for digital signal processing applications such as embedded multipliers, multiply-accumulate units, soft and hard processor cores, and embedded memory blocks [4]. The powerful integration of available hybrid software and hardware co-design and synthesis packages, advanced FPGA boards, Intellectual property (IP) designs, and tools that allow seamless integration between these software and hardware design packages has made FPFA software/hardware co-design a methodology of choice for many of real-time applications.

The rest of the paper is organized as follows. In Section 2, the problem formulation and the neural network model are discussed where an overview of the least mean square (LMS) algorithm is given and the implementation of the design in the FPGA Zynq evaluation kit is described. In Section 3, the details of the software/hardware codesign are discussed in detail. Experimental results are given in Section 4. Finally, the paper is concluded in Section 5.

2 Different Neural Network Models for Adaptive Noise Cancellation Problem

The most commonly-used algorithm to design adaptive filter is called least mean square (LMS) algorithm, originally developed by Widrow and Hoff [5]. The LMS algorithm is based on the principle of the steepest descent algorithms with minimum mean square error. The greatest benefit is that it does not require exact measurements of the gradient vector, nor does it requires matrix inversion. The LMS algorithm is used to solve the Wiener-Hoff equation by searching for the optimal coefficients weights for an adaptive filter. Another main advantage of the LMS algorithm is its computational simplicity, ease of implementation, and unbiased convergence. A block diagram of an adaptive noise cancellation system is shown in Fig. 1.

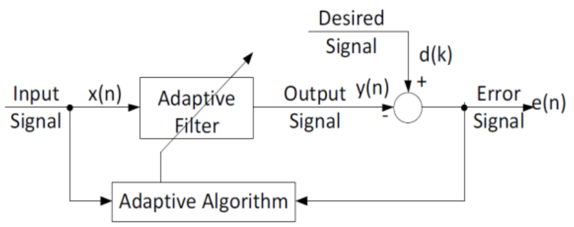


Fig. 1. Block diagram of a typical adaptive noise cancellation system

The vector $X(k)$ denotes an input vector with time delay and x(k) is the input value at time k , that is:

$$X(k) = [x(k)\ \ x(k-1) \ldots x(k-N+1)]^T \qquad (1)$$

The vector $W(k)$ is used to represent the weights applied to the filter coefficients at at time k and is given as:

$$W(k) = [W_0(k)\, W_1(k)\ \ \ldots\ W_{N-1}(k)\,]^T \qquad (2)$$

The step size parameter μ is the step size of the adaptive filter.

$e(k)$ is the error between the desired response $d(k)$ and the output of the filter $y(k)$, i.e., the filtered signal, at time k.

The pseudo code of an LMS algorithm is described in Table 1. The algorithm is adopted to update the coefficients of a finite impulse response (FIR) filter.

Table 1 The pseudo code of an LMS algorithm

1. Calculate the output signal y(k) of the FIR filter. The output of the filter represents an estimate of the desired response. y(k) is the calculated as the convolution of the weight vector and the input vector: $\boldsymbol{y(k) = \sum_{n=0}^{N-1} W_n(k)x(k-n) = W^T(k)x(k)}$ **(3)**
2. The error signal e(k), is estimation error defined as the difference between the estimated response and the desired response. **e(k) = d(k) – y(k) (4)**
3. The error signal and the input signal are applied to the weight update algorithm to updates the filter coefficients.

The LMS algorithm updates its coefficients through the minimization of the mean of the instantaneous squared error denoted by $E[e^2(k)]$. While X(k) and W(k) are assumed to be independent, the LMS algorithm assumes that x(k) and d(k) are wide-sense stationary ergodic processes, and therefore their means and variances are constant. The iterative weight update procedure of the LMS algorithm is shown in the following equation [4].

$$W(k+1) = W(k) + 2\mu * e(k) * X(k) \qquad (5)$$

The selection of the step size parameter μ plays an important role in updating of the system coefficients and thus can affect the system performance. While a relatively small μ value could result in longer convergence time to find an optimal solution, selecting a larger μ value may lead to unstable convergence and a diverge output. For the consideration of stable behavior and convergence, the step size must be a small positive value (μ << 1) and meet the following criteria:

$$0 < \mu < \frac{1}{2 * N * R} \qquad (6)$$

Where N is the number of taps of the filter and R is the input signal covariance matrix defined as

$$R = E(X(n) * X^T(n)) \qquad (7)$$

3 FPGA Implementation

3.1 Zynq Evaluation Kit

Eventually, the complete design will be implemented and exported to work with FPGA Zyn Evaluation Kit [7]. The software design using Simulink LMS Filter block is converted to an Intellectual Property (IP) Core, which is connected with Zynq Processing system and communicate with target interface platform AXI4-Lite. AXI stands for Advanced eXtensible Interface, and the current version is AXI4. The AMBA standard was originally developed by ARM for use in microcontrollers. AXI buses can be used flexibly, and in the general sense are used to connect the processor(s) and other IP blocks in an embedded system.

AXI4-Lite provides a simplified link supporting only one data transfer per connection (no burst). It also is memory-mapped, an address and single data word are transferred. It means data is then written to, or read from, the specified address; in the case of AXI4 bursts, the address specified is for the first data word to be transferred, and the slave must then calculate the addresses for the data words that follow [8]. We also mention using AXI4-Lite as Target Interface, which defined as point-to-point connection for passing data, addresses, and hand-shaking signals between master and slave clients within the system.

3.2 FPGA Implementation

The design and implementation of adaptive filter involves multiple steps and process before the design is completely exported and launched on Hardware (System debugger), as shown in Fig. 2a. Using both Simulink and Vivado Synthesis Design are the final option in Adaptive filter implementation due to the support between two platforms.

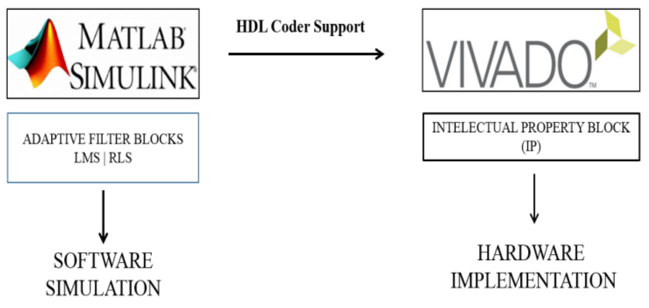


Fig. 2a. Software\Hardware co-design of LMS adaptive filter

A Software Development Kit (SDK) is launched to import necessary drivers and C++ files used for hardware implementation. Generally, whole process is organized with following steps.

Step 1: Design Adaptive Filter LMS using Matlab Simulink DSP Toolbox with HDL Support blocks.. Filter block has to be supported by HDL Coder that it is able to convert to IP Core (**lms_pcore**), which later will be added into Vivado Synthesis Design Environment for Register Transfer Level design

Step 2: Give input samples and simulate the Simulink design [6].

Step 3: Generate RTL design by converting LMS design in Simulink to an Intellectual Property (IP) Core using HDL Coder Support.

Step 4: Target platform interface AXI4-Lite for signal x(k), d(k), and e(k).

Step 5: Design a complete system of Filter based on Zynq board target architecture using Vivado Synthesis Design and VHDL target language. A complete design of IPs core includes lms_pcore, switches and buttons IP core, processing system IP core, and AXI Interconnect altogether wired, synthesized, and implemented.

Step 6: Synthesis, implement, and export the complete design into Hardware.

Step 7: Launch Software Development Kit (SDK) to generate all drives, input C++ files, and input signal needed.

Step 8: Run System Hardware Debugger to Zedboard, and run test.

HDL Coder plays an important role in converting LMS Filter from Simulnk design to an IP Core in Vivado Synthesis Design. HDL Coder is a built-in MathWorks product which enables the synthesizable HDL codes generation from MATLAB functions and Simulink models. It provides a workflow which analyzes a MATLAB/Simulink model and then converts the floating point signals to fixed point signals. This benefits the users on the development of algorithms and models without lower-level HDL code design. The HDL code optimization will give the option of choosing the desired FPGA device so that it can provide specified control during the implementation, implementing data paths, controlling the HDL architecture, and generating hardware resource utilization. Once generated by HDL Coder, the HDL code can be used to create an IP core, as detailed In Fig 2b.

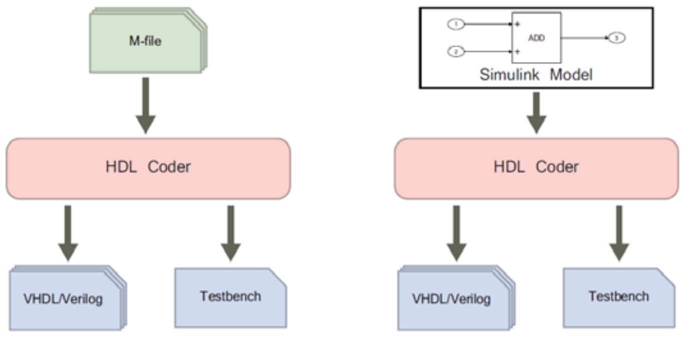


Fig. 2b. HDL Coder Flow

4 Experimental Results

We implemented the FPGA design of the adaptive noise cancellation system based on the least mean square (LMS) algorithm based on the steps described in Section III.

- This design required the latest Simulink library to be properly installed. There exist many type of LMS filters, as shown in Fig. 3. In order to fit our design, the DSP System Toolbox HDL Support LMS Filter is selected. Only this block later will be converted to an IP for synthesis design.

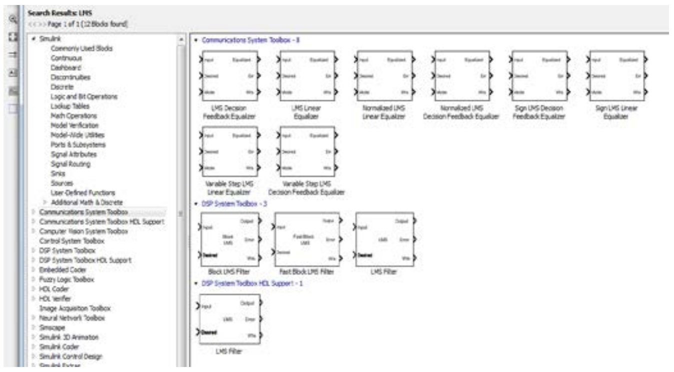


Fig. 3. the LMS Block in Simulink Library

LMS Filter is configurable to fit our design, and we assign signals x(k), d(k), and e(k), to denote the input, the desired signal and the error signal, respectively, as shown in Fig. 4. In order for the HDL Coder to generate HDL codes for the Simulink LMS model, the input type must be fixed-point number. Pair of *DATA Type Conversion blocks* converts the corrupt audio signals and the tonal noise signals to fixed-point signals. These fixed-point signals are then provided to the LMS subsystem. The error signal, **e(k)**, along with the corrupt audio and tonal noise input are transmitted to a scope for visual inspection of the signal. Two blocks called *To Workspace* make the LMS output and the corrupt audio signals to be the Matlab workspace variables for audio playback.

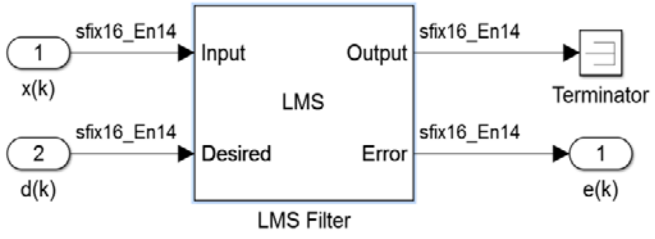


Fig. 4. the LMS Subsystem with parameters chosen. Adaptive Filter coefficients = 16 and step size = 0.13

This step size is then calculated following Eq. (6). The selection of the step size parameter μ plays an important role in updating of the system coefficients and thus has a major impact on the performance of the LMS algorithm. The smaller the μ, the longer it takes for the adaptive filter to converge to the optimal solution. The complete design in Simulink is shown in Fig. 5.

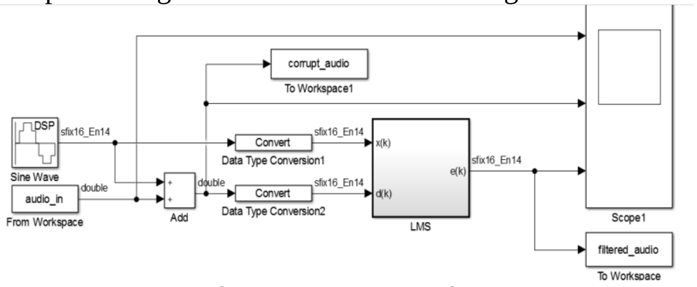


Fig. 5. Complete of Simulink design of LMS Filter

- We simulate Simulink LMS filter design, as shown in Fig. 6. As expected, the Sine wave block generates sinusoidal noise signal, adding with an audio input signal file to generate the total noise, and which will be filtered out via LMS filter block. Each signal is linked with an output scope to observe output waveforms.

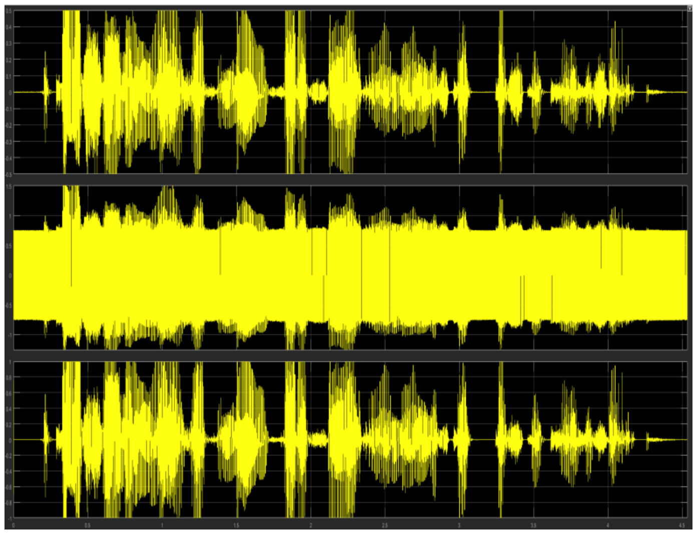
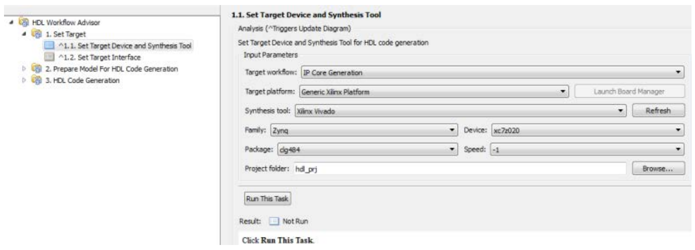


Fig. 6. The first signal represents the audio input (From Workspace). The second one represents the total noise. And the other represent for filtered noise through LMS system. The final output demonstrates the efficiency of LMS filter in noise filtering of this design.

- After Simulink Simulation is successfully run, the LMS filter block will be used for Register Transfer level Design, by converted it to an Intellectual Property (IP) Core using HDL Coder Support. As mention earlier, the HDL code optimization will give the option of choosing the desired FPGA device so that it can provide specified control during the implementation, implementing data paths, controlling the HDL architecture, and generating hardware resource utilization. This platform, as shown in Fig. 7 gives converting target as using Xilinx Vivado synthesis tool, and target device for implementation. The next step describes how to target communication between IP blocks and processing units by applying AXI4-Lite interfaces.

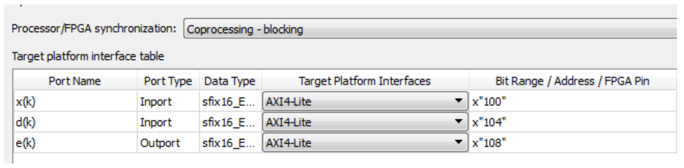

HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
^1.2. Set Target Interface
2. Prepare Model For HDL Code Generation
3. HDL Code Generation

1.1. Set Target Device and Synthesis Tool
Analysis (^Triggers Update Diagram)
Set Target Device and Synthesis Tool for HDL code generation
Input Parameters
Target workflow: IP Core Generation
Target platform: Generic Xilinx Platform
Synthesis tool: Xilinx Vivado
Family: Zynq
Device: xc7z020
Package: clg484
Speed: -1
Project folder: hdl_prj

Result: Not Run
Click **Run This Task**.

Fig. 7. ZedBoard HDL Workflow Advisor Input Parameters (Set Target).

- Another important point is to establish communication stream between each components and IP blocks in Evaluation boards. As mentioned earlier, the target interface uses AXI4-Lite as a point-to-point connection to transmit data, addresses, and hand-shaking signals between master and slave clients.


Processor/FPGA synchronization: Coprocessing - blocking

Target platform interface table

Port Name	Port Type	Data Type	Target Platform Interfaces	Bit Range / Address / FPGA Pin
x(k)	Inport	sfix16_E...	AXI4-Lite	x"100"
d(k)	Inport	sfix16_E...	AXI4-Lite	x"104"
e(k)	Outport	sfix16_E...	AXI4-Lite	x"108"

AXI4-Lite provides a link to support single data transfer per connection (no burst). All the signals x(k), d(k), and e(k) are assigned with AXI4-Lite interfaces, Port type, Data type, and Bit Range, as shown in Fig. 8.

Fig. 8. Target Platform Interface **AXI4-Lite** for x(k), d(k), and e(k), as well as with Port type, Data Type, and Bit Range.

The final IP Core of LMS Filter is transformed into a single Core block that is able to implement into Vivado Synthesis environment, as shown in Fig. 9. The lms_pcore (LMS Filter IP Core) block contains all configurable settings that are set up on previous steps, including AXI4-Lite Interfaces. The next step describes how this block will be imported into Vivado Synthesis Design environment.

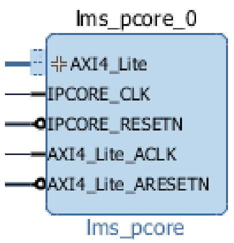


Fig. 9. An Intellectual Property (IP) Core is generated configured with AXI4-Lite Interface

- The IP Core of LMS Filter (lms_pcore) is eventually imported into Vivado Synthesis Design environment, as shown in Fig. 10. Then the LMS Core will go through process of Package IP. This step allows packaging HDL Coder generated IP blocks in IP Package for use in Vivado IP Integrated designs.

The whole IP block design involves multi different IPs, all are connected with AXI4-Lite interfaces. The complete design of LMS is shown in figure below. In this schematic, ninth IP Core blocks are presented: lms IP, AXI Interconnect IP, Processor System Reset IP, LED Controller IP, ZYNQ Processing System IP, Zed Audio IP, NCO (Numerical Controlled Oscillator) IP, AXI_GPIO_0 IP, AXI_GPIO_1 IP.
LMS IP: contains the design and algorithm of LMS Filter.
AXI Interconnect IP: contains the configuration of AXI4-Lite interfaces.
Processor System Reset IP: contains the reset function of Zynq Board.
LED Controller IP: contains the functionality of LEDs.
ZYNQ Processing System IP: contains the logic of Zynq Processing system.
Zed Audio IP: contains the driver of Zedboard Audio map.
Numerical Controlled Oscillator (NCO) IP: contains a digital signal generator which produces a clocked synchronous, discrete-time, and discrete-valued representation of a waveform, i.e. sine waveform.
AXI_GPIO_0 IP, and AXI_GPIO_1 IP: connect to buttons and switches.
The next step is synthesis, implement, and export the complete design into Hardware.

- After the IP block design is complete, it is synthesized, implemented to verify the design requirement, and exported to hardware.

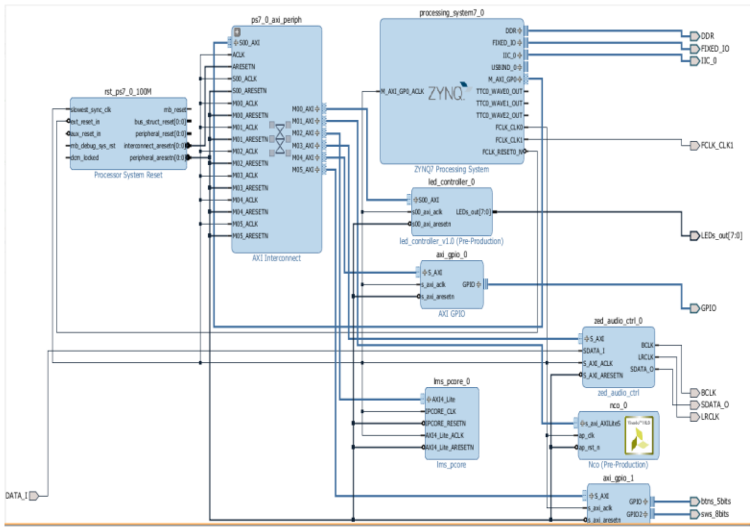


Fig. 10. The complete IP Core design of LMS system

- The Xilinx software development kit is used to create an integrated design environment for various embedded applications by applying all drivers, and C++ files needed to let IP Core design operate and debug. All C++ files and drivers after imported in SDK later will be debugged via System Debugger.

- After the SDK part is finished, System Hardware Debugger is run to debug all necessary files into hardware board, as shown in Fig. 11. This step includes connecting Zynq board with computer for feeding audio in; and speaker will be connected at port out. The board is connected with computer by JART port and PROG USB port. Putty is used as machine to machine communication tool to communicate between computer and Zynq board to give operation command.

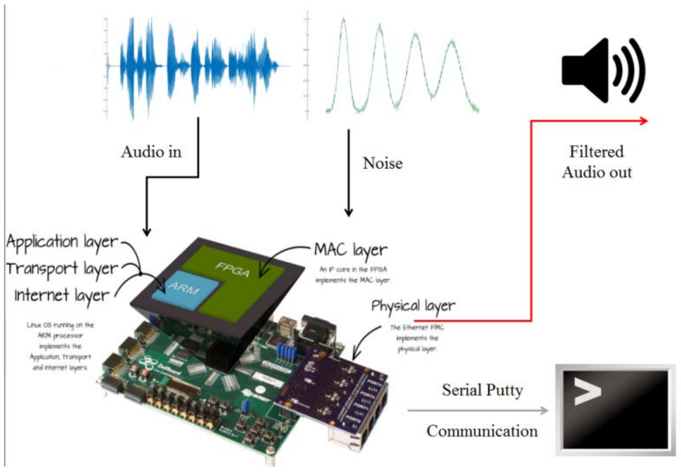


Fig. 11. The overview of hardware implementation

Sinusoidal noise is added to audio input by switching switch on Zed board. Each switch contains a different numerical step size, and an adding of switch at same time will add the higher amplitude noise with higher pitch. Button is used to apply filter algorithm to filter out total noise. The Filter operation is given through Putty Serial Communication with Zed's USB COM port as shown in the Fig. 12.

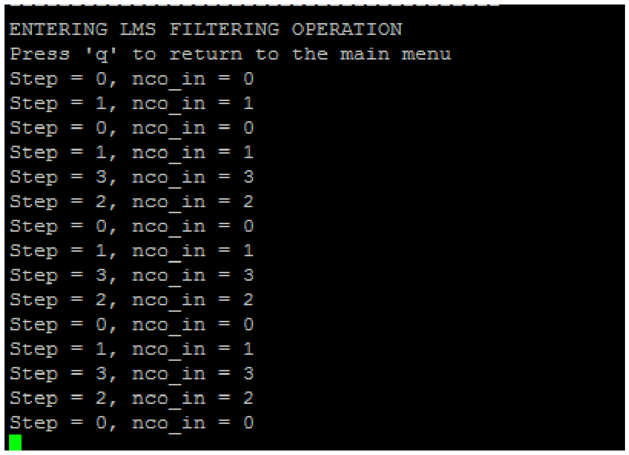

```
ENTERING LMS FILTERING OPERATION
Press 'q' to return to the main menu
Step = 0, nco_in = 0
Step = 1, nco_in = 1
Step = 0, nco_in = 0
Step = 1, nco_in = 1
Step = 3, nco_in = 3
Step = 2, nco_in = 2
Step = 0, nco_in = 0
Step = 1, nco_in = 1
Step = 3, nco_in = 3
Step = 2, nco_in = 2
Step = 0, nco_in = 0
Step = 1, nco_in = 1
Step = 3, nco_in = 3
Step = 2, nco_in = 2
Step = 0, nco_in = 0
```

Fig. 12. Putty display of LMS operation in noise-adding audio signal.

5 Conclusions

In this research an adaptive filter system was successfully completed, and deployed with software/hardware co-design method. The adaptive filter system was analyzed using the MATLAB/Simulink model, and it later was automatically converted from floating point to fixed point for an Intellectual Property Core. This IP Core was placed in Vivado Synthesis Design for synthesis and implementation. Finally, the debugger was run before the audio file was fed in Zedboard. The design method can be applied to any type of FPGA under the Zynq family as long as this design is supported by the DSP-HDL Tool Support. The LMS Filter was processed and implemented to the FPGA board since it is supported by HDL. Experimental results show that the proposed hardware implementation method has a high degree of noise cancellation performance.

Acknowledgment

This work was supported by the National Science Foundation (NSF) grants: HRD #1505509, #1654474, #1533479, and HRD #1435947.

References:

[1] Wagdy H. Mahmoud and Nian Zhang, Software/Hardware Implementation of an Adaptive Noise Cancellation System, *120th ASEE Annual Conference & Exposition*, Atlanta, GA, June 23-26, 2013.

[2] Nian Zhang, Investigation of Fault-Tolerant Adaptive Filtering for Noisy ECG Signals, *2007 IEEE Symposium on Computational Intelligence in Image and Signal Processing (CIISP)*, Honolulu, HI, pp. 177-182, April 1-5, 2007.

[3] M. I. Troparevsky, C. E. D'Attellis, On the convergence of the LMS algorithm in adaptive filtering, *Signal Processing* Vol. 84, pp. 1985-1988, October 2004.

[4] Ahmed Elhossini, Shawki Areibi, Robert Dony, An FPGA Implementation of the LMS Adaptive Filter for Audio Processing, *Proceedings of IEEE International Conference on reconfigurable Computing and FPGS's (ReConFig 2006)*, pp. 1-8, 2006.

[5] A. Rosado-Muñoz, M. Bataller-Mompe, E. Soria-Olivas, C. Scarante, J. F. Guerrero-Martínez, FPGA Implementation of an Adaptive Filter Robust to Impulsive Noise: Two Approaches, *IEEE Transactions on Industrial Electronics*, Vol. 58, No. 3, pp. 860-870, March 2011.

[6] https://www.mathworks.com/help/hdlcoder/examples/basic-hdl-code-generation-with-the-workflow advisor.html

[7] http://www.zynqbook.com/

[8] https://www.xilinx.com/support/answers/66421.html