REAL TIME IMPLEMENTATION OF FIR FILTER BASED ON TIME DELAY NEURAL NETWORK

Dr. Shefa Abdulrahman Dawwd
Computer Engineering Department, College of Engineering, University of Mosul
Email: shefadawwd@yahoo.com

Abstract

An FPGA real time Implementation of Time Delay Neural Network (TDNN) is presented in this paper. The design and all of the work are geared towards the implementation of the TDNN in a scalable fashion. The TDNN is an adaptive FIR filter with 18-bit input and 18-bit output resolution. In this paper, the filter adapts its tap weights using the Least-Mean-Square (LMS) algorithm and then stores them in FPGA memory cell. The LMS algorithm that is used for weight adaptation is off chip implemented. The input is processed through a digital tapped delay line. The FIR neural network is used for real time adaptive noise cancellation. When the filter order is 10, the filter consumes 1168 Spartan 3E FPGA logic elements.

Keywords: TDNN, FIR neural network, FPGA neural implementation.

تنفيذ الزمن الحقيقي لمرشح نبضة محدد الاستجابة مبني على استخدام شبكة تأخير الزمن العصبية

د.شفاء عبدالرحمن داوؤد
قسم هندسة الحاسبات-كلية الهندسة -جامعة الموصل

الملخص

يعرض هذا البحث تنفيذ الزمن الحقيقي لشبكة تأخير الزمن العصبية بأستخدام مصفوفة البوابات المبرمجة حقليا. تم توجيه التصميم وكل العمل على تنفيذ الشبكة بتقنية قابلة للتوسع. في هذا البحث تعمل شبكة تأخير الزمن العصبية كمرشح متكيف محدد الاستجابة بدقة 18 بت ادخال و 18 بت اخراج. يستخدم المرشح خوارزمية اقل معدل تربيع في تثبيت اوزانه ومن ثم خزن تلك الاوزان في خلايا ذاكرة مصفوفة البوابات المبرمجة حقليا. تم تنفيذ خوارزمية اقل معدل تربيع برمجيا خارج دائرة الرقاقة. يعالج الادخال خلال خط التأخير الرقمي الخاص بالمرشح المصمم. تم استخدام تطبيق ازالة الضوضاء المتكيفة في فحص كفاءة الشبكة العصبية المكافئة لمرشح محدد الاستجابة. لمرشح ذو مرتبة مساوية لل10 تم استغلال 1168 عنصراً منطقياً من حجم رقاقة البوابات المبرمجة حقلياً.

Received 5/6/2009

Accepted 6/10/2009

1- INTRODUCTION

Many modern-day electronic systems must deal with unknown or changing environmental variables such as noise levels, interference, and varying input statistics. These systems frequently use adaptive signal-processing techniques to optimize their performance[1].

The problem of the lack of a priori knowledge of the noise is being treated by such adaptive filtering. The idea is based on the application of time delay neural networks as an adaptive TDNNs inherently posses adaptability properties as well as they are of nonlinear structure, hence they represent a natural choice for the construction of adaptive filters. A TDNN is trained by an identity observer with assumptions that the model of the system is precise and the measurement and the process noise are unknown, regardless whether Gaussian or colored. It is shown by simulations that this concept gives better results when we miss information about the noise statistics, or when the noise is not Gaussian[2].

However, in application domains such as mobile communications or ubiquitous computing, these systems also face severe constraints in power dissipation and circuit die area. In such cases, using programmable digital signal-processing (DSP) chips becomes infeasible. Although analog circuits can implement moderate resolution arithmetic at low power and area, these circuits are limited by other problems such as charge leakage, signal offsets, circuit mismatch, error accumulation, and noise sensitivity[3]. Therefore, using Application Specific Integrated Circuit (ASIC) seems as a compromised solution. Both approaches (analog and digital-ASIC) have fixed topology, resulting in a design suited only for one type of target application. Digital implementation, using FPGAs, allows the redefinition of the topology using the same hardware. FPGAs have traditionally been configured by hardware engineers using a Hardware Description Languages (HDL). Very High Speed Description Language (VHDL) is one of the most efficient description languages used. The disadvantage of an implementation using FPGA over ASIC is the performance. FPGA normally runs slower than ASICs[4].

In this paper an FPGA implementation of TDNN that is used as an adaptive filter is proposed. The rest of paper is organized as follows: in section 2, the FIR filter is introduced. Section 3 presents the structure of TDNN while the algorithm used to train the network is presented in section 4. In section 5, the hardware implementation of the TDNN and its proposed hardware architecture is presented. Finally, in section 6, the experimental results and a comparison with the system presented in [5] where an FPGA chip is used to emulate an adaptive FIR filter of various taps length is performed. Also some conclusions are discussed in this section.

2- FINITE IMPULSE RESPONSE FILTER

Digital filters represent an essential part of digital signal processing. Filters are employed in preprocessing for reducing noise and enhancing characteristic qualities. An example of this is edge enhancement in pictures. Filters are categorized as linear, non-linear or adaptive[6]. Whenever possible, linear filters are preferred since their treatment can be completely put into theory. According to their impulse response, filters are divided into those with limited impulse response (FIR=finite impulse response)) and those with an impulse response of infinite length (IIR=Infinite impulse response).

A FIR filter computes a convolution between an input data stream and a stored weight vector. Fig. 1 shows the architecture of a FIR filter. It comprises a digital delay line, weight cells, digital multipliers and adders[6].

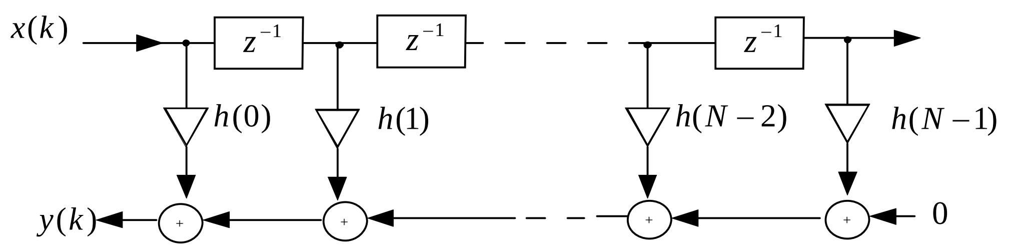


Fig. 1: The architecture of a FIR filter

Real world signal processing problems often require adaptive solutions. Simulations of these solutions, sometimes via trial-and-error, will hopefully identify an algorithm to correctly solve the given problem. The problem such as the lack of a priori knowledge of the noise is being treated by adaptive filtering, simultaneously investigating the statistics of the noise and updating the filter gain upon them [2].

3- TIME DELAY NEURAL NETWORK

The most common feedforward networks are static networks, which have no internal time delays. They respond to a particular input by immediately generating a specific output. However, static networks can respond to temporal patterns if the network inputs are delayed samples of the input signals, i.e., time is treated as another dimension in the problem [7]. Incorporating time as another dimension in neural networks is often referred to as Time Delay Neural Network (TDNN). These networks, trained with the standard back propagation algorithm, have been used as adaptive filters for noise reduction and echo canceling and for chaotic time series prediction [8].

Consider a tapped delay line(Fig. 2), that is, a shift register. Consider also a multilayer perceptron where the tapped outputs of the delay are applied at its input, the output has a finite temporal dependence on the input: y(k)=F[x(k),x(k-1),x(k-2),……..,x(k-τ)] , where F is a typical nonlinearity function. When this function is a weighted sum, then the TDNN is equivalent to a finite impulse response (FIR) filter.

For the filter, the output *y(k)* corresponds to a weighted sum of past delayed values of the input:

$$y(k) = \sum_{n=0}^{\tau} h(n)x(k-n) \qquad (1)$$

The Tapped Delay Line (or TDL), presented at Fig. 2, is a bank of unit delays used at the input, sometimes also at the output, of an ANN. This is a standard element used in adaptive filtering applications of ANNs. In this way the network structure is capable of processing τ number of time samples at the same time instance.

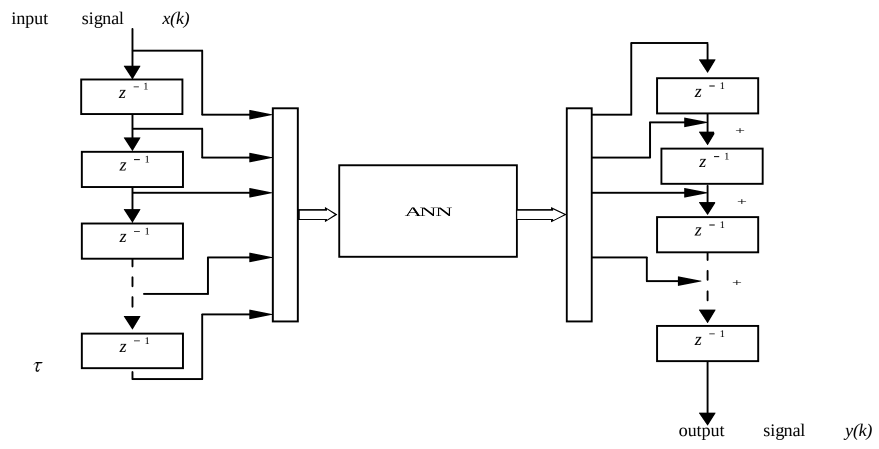


Fig.2: The Tapped Delay Line (or TDL)

The FIR network is a feedforward neural network architecture with internal time delay lines and can be learned by the temporal back-propagation algorithm. It is a modification of the basic multi-layer network in which each weight is replaced by an FIR linear filter. The FIR filter is modeled with a tapped delay line is shown in Fig. 3.

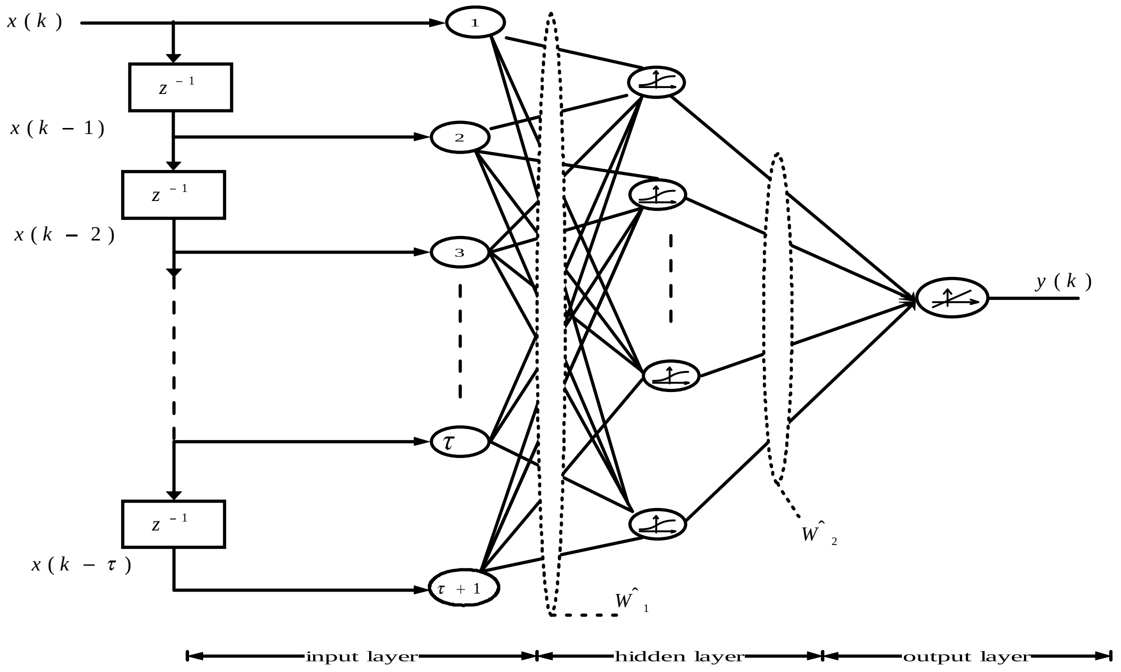


Fig.3: TDNN structure

If the TDNN is formed with two layers(input and output layer, there is no hidden layer), then a 1D FIR network like that shown in Fig. 1 is formulated. The layer weights are adapted for a given problem by using a supervised learning method.

4- TDNN LEARNING ALGORITHM

The network shown in Fig.2 is classified as a network of multilayered feedforward neural network. The back-propagation algorithm is used to train this type of neural network. As shown in Fig. 2, a multilayer feedforward network has an input layer of source nodes and an output layer of neurons (i.e., computation nodes); these two layers connect the network to the outside world. In addition to these two layers, the multilayer perceptron usually has one or more layers of hidden neurons, which are so called because these neurons are not directly accessible. The training is usually accomplished by using a backpropagation (BP) [2] algorithm that involves two phases:
1) Forward Phase. During this phase the free parameters of the network are fixed, and the input signal is propagated through the network of Fig. 2 layer by layer. The forward phase finishes with the computation of an error signal:

$$e[m]=d[m]-y[m] \tag{2}$$

where *d[m]* is the desired response and *y[m]* is the actual output produced by the network in response to the input *x[m]*.
2) Backward Phase. During this second phase, the error signal *e[m]* is propagated through the network of Fig. 2 in the backward direction, hence the name of the algorithm. During this phase the adjustments are applied to the free parameters of the network so as to minimize the error *e[m]* in a statistical sense. To update weights the least mean square (LMS) learning rule is used:

$$w[m+1]=w[m]+\mu*x[m]*e[m] \tag{3}$$

In TDNN, each filter coefficient adaptation uses its present coefficient value, *w[m]*, to add to the product of the step-size parameter, μ, tap input *x[m]* and error output *e[m]*, to obtain an updated version of the filter coefficient *w[m+1]*. Note that the only information needed to update filter coefficients are the tap input and the error term. The step-size parameter *μ* (learning rate) is pre-determined by the system engineer and is a decimal number between 0 and 1[9].

The back-propagation learning algorithm is simple to implement and computationally efficient in that its complexity is linear in the synaptic weights of the network. However, a major limitation of the algorithm is that it does not always converge and can be excruciatingly slow, particularly when we have to deal with a difficult learning task that requires the use of a large network.

5- HARDWARE IMPLEMENTATION OF TDNN

It is the goal of neural network engineers to transfer the progress made into new hardware systems. They are intended to accelerate future developments of algorithms and architectures, and to make possible the use of dedicated neural networks in industrial applications.

Learning algorithm implementation`

The learning algorithms used for modifying weights values using inputs and training data are as an important part of the network system as the architecture itself. Implementation of learning in VLSI systems takes three forms; *off-chip*, *'chip-in-the-loop'* and *on-chip* learning. In *off-chip* learning, weights values are calculated externally by software and are then downloaded to the neural network which is then used only for recall. This is the easiest but least favoured method, since training times can be long. Off-chip learning does have the advantage in that it is easy to change the learning algorithm simply by modification of software. It also allows the use of floating point arithmetic for the algorithms which may not be feasible on a neural network chip. *'Chip-in-the-loop'* training chip may also be considered as an off-chip method since the training algorithm is still run in software. However, in this case the neural network is used in the training loop which removes the need for a software model of the network itself, and compensates for device variability. The main drawback of this method is the communications overhead in continually reading and writing data across the network/host interface. *On-chip* learning must be seen as the most desirable method, since it may open the way to stand-alone neural network chips. The main advantage of running the learning algorithm in hardware is the gain in speed. The *on-chip* learning consumes large chip area for floating point calculations [10] that are usually required to update weights in backward phase. Thus, in this paper *off-chip* learning is used, and the *on-chip* learning can be left to be a future work.

Numerical Representation

In general, neural networks have low-precision requirements, even though the exact specification is algorithm and application dependent. Digital neurohardware can profit from this property by using fixed-point arithmetic with reduced precision. Fixed-point implementations are less complex and less area consuming than floating-point arithmetic and therefore their use helps to reduce system cost [11].

Implementation of TDNNs Activation Function`

Direct implementation for non-linear activation functions is very expensive. There are two practical approaches to approximate non-linear functions with simple FPGA designs. **Piece-wise linear approximation** describes a combination of lines in the form of $y=ax + b$ which is used to approximate the non-linear function. Note that if the coefficients for the lines are chosen to be powers of two, the non-linear functions can be realized by a series of shift and add operations. Many implementations of neuron activation functions use such piece-wise linear approximations. The second method is **lookup tables**, in which uniform samples taken from the center of non-linear function can be stored in a table for look up. The regions outside the center of the non-linear function are still approximated in a piece-wise linear fashion.

`The TDNN consists of two types of activation functions classified as linear and non-linear. Hidden layer neurons have non-linear activation functions, while the output layer activation function is linear (pureline function). Implementation of non-linear activation function was discussed in previous works [12]. The pureline linear activation function is $y=ax$ and $a=1$(see Fig.4), therefore the function weighted sum input will be the activation function output.

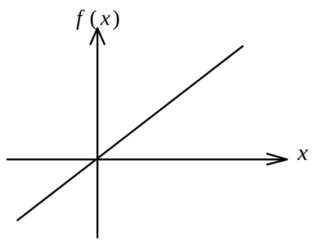


Fig.4: pureline linear activation function

The architecture of FIR network

The FIR network implemented in this work is a TDNN of two layers (input and output layers). The type of the output neuron activation function is pureline linear function. The design and all of the work are geared towards the implementation of the TDNN in a scalable fashion. The scalable design means that the network can become as large as practically possible thus providing a structure for more complex application. In this implementation, the TDNNs are trained *off-chip* and the convergent parameters then fed to the hardware for test and synthesis. The processing elements and the complete architecture of the proposed neural network is shown in Fig.5(a,b).The processing element does the algebraic equations of the electric model of the neuron, that is, the multiplication and sum required in the neuron's internal processing(see Fig. 5).

All PEs are identical. Each PE has one delay register, one weight storage element and one multiplier. The number of multiplier depends on the number of tapes. Therefore, when the number of multipliers exceeds the number of FPGA embedded multipliers, the redundant multipliers are built by exploiting the FPGA chip gates.

The multiplications are the most power dissipating arithmetic operations. Today's FPFAs contain a number of speed optimized signal processing building blocks, such as multipliers, RAM blocks or I/O structures with propagation delays in the range of a few nanoseconds. However, in this system we intended to design the TDNN to be as a part of larger system. Therefore any of embedded hardware multipliers can be left to be used by other parts of the system and a custom hardware multiplier can be designed. Based on that, and in order to minimize the area cost and the total power consumption and also to simplify the implementations, the parallel Booth multiplier technique can be chosen. The simple serial by parallel booth multiplier is particularly well suited for FPGA implementation without carry chains because all of its routing is to nearest neighbors with the exception of the input. The total area of the implemented multiplier highlights the advantages over traditionally implementation of the multipliers specially with the increasing of multiplier operand sizes.

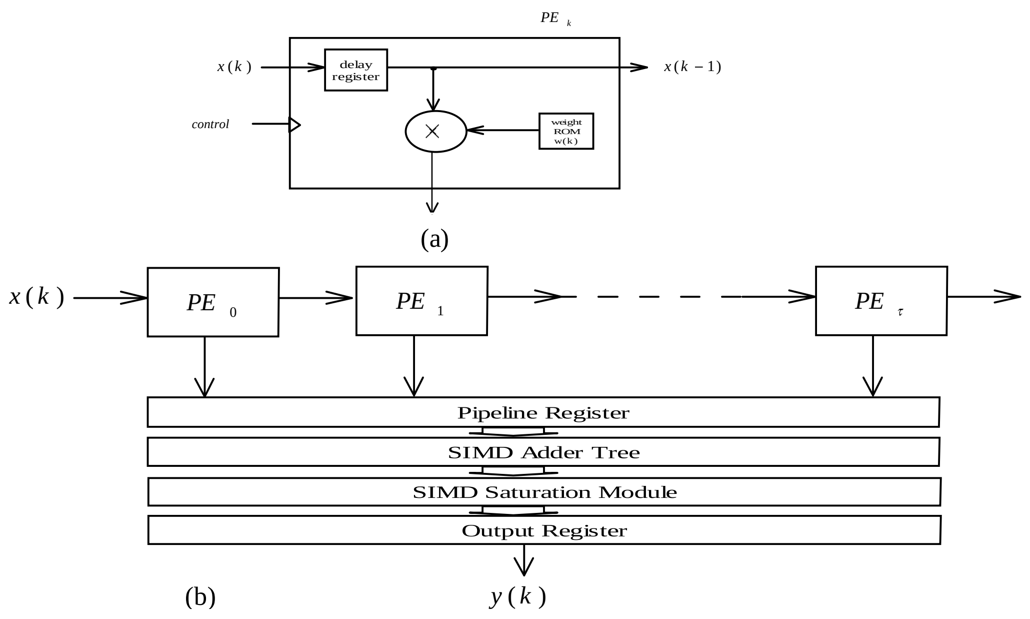


Fig. 5: (a)` A block diagram of the processing element (*PE*) (b) The

To minimize the time required to add all the PEs results, and to reduce the number of adders, the single add instruction, multiple data coming from each PE (SIMD) adder tree [4] is suggested to be used. The SIMD saturation module is used to fix the bit resolution of the adder results.

The minimum word length for each weight and input elements is the one that should preserve the same generalization ability that achieved from software. Since the input parameters word length for FPGA embedded multipliers are (18x18), and to exploit the full precision given by this constrain, the word length for input and tap's weight are selected to be 18-bits. Therefore, each PE has a 18x18 bit multiplier.

6- RESULTS AND CONCLUSIONS

The FIR neural network mentioned in previous sections has been applied as a 1_D adaptive FIR filter which is used to filter a 1_D signal from noise in real time. In this paper an FPGA VLSI architecture that implements the network is proposed. Xilinx Spartan-3E FPGA of 500,000 gates is used for implementation. The FPGA digital hardware model has been designed using Xilinx Foundation environment.

The choice of the order of the FIR filter is experimentally determined. The order that makes the FIR filter gives a reasonable signal to noise ratio (SNR) is adapted. There is no clear mathematical analysis to derive the quantities. Only through experiments may we obtain a feasible solution.

The software part of this paper is implemented on a personal computer works under WindowsXP. The algorithm emulation is carried by using MATLAB. The noise that added to a selected signal, has a mean of zero, a variance of one and standard deviation of one.

In Fig.6a, the original signal is used as a target to train the network, while the training signal (Fig.6b) is the original signal added noise with a SNR equals to 10 dB. In Fig.6c and Fig.6d, the filter output signals are shown when the filter order (no. of taps) equals to 5 and 10, respectively after 200 learning steps. In Fig.6e, the filter used to generate the signal in Fig.6d is considered but after 800 learning steps. In Fig.6f and Fig.6g, the filter output signals are shown when the filter order equals to 10 and the SNR of the input signals are 15 and 20 dB respectively.

In Fig.6h, a test signal of 10 dB SNR that is not used in training is considered as an input to the FIR network of 10 taps trained in 800 epochs. The output test signal is shown in Fig.6i. In Fig.6j and Fig.6k, the output test signals are shown when the input signals are of 15 dB and 20 dB SNR respectively.

The FIR filter has still worked as a low pass filter (LPF) if the noise added to the original signal is reasonable. For example, the original signal properties that shown in Fig. 6a are obviously presented in Fig. 6g which represents the restored signal after adding noise. While the main properties of the original signal are disappeared in Fig. 6c, Fig. 6d and Fig. 6e when the SNR=10dB (the signal shown in Fig. 6b). One also can see that the filter of higher order (no. of tap =10 in Fig. 6d) is better than that of lower order (no. of tap=5 in Fig. 6c). When the properties of the original signal is preserved (Fig. 6h) after adding the noise, the filter can work as a LPF even if the SNR is low (SNR=10dB in Fig. 6i). Briefly, the filter performance is degraded when the SNR decreases, while the filter still works as a LPF even if the input signal is highly corrupted with a noise. Also, it is concluded that the filter is adapted as a low pass filter for any

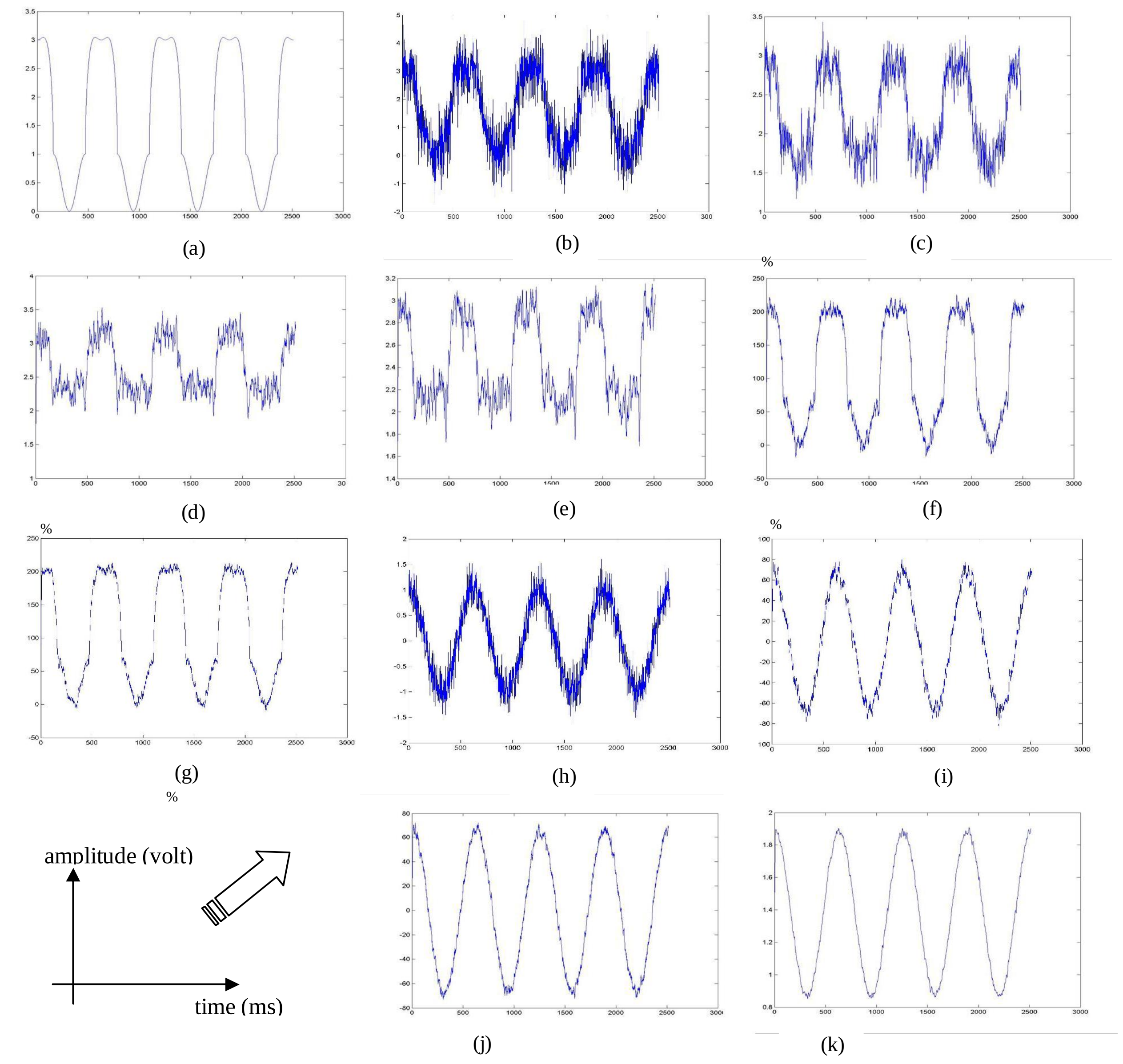


Fig.6: Training and testing signal with the noise influences (a) original target signal (b) the added noise input signal, SNR=10dB (c) The output signal with canceling noise of SNR=10dB, Tap=5,epochs=200 (d) The output signal with canceling noise of SNR=10dB,Tap=10, epochs=200 (e) The output signal with canceling noise of SNR=10dB, ,Tap=10, epochs=800 (f) The output signal with canceling noise of SNR=15dB, ,Tap=10, epochs=800 (g) The output signal with canceling noise of SNR=20dB, ,Tap=10, epochs=800 (h) The test input signal , SNR=10dB, ,Tap=10 (i) The test output signal with canceling noise of SNR=10dB, ,Tap=10, epochs=800 (j) The test output signal with canceling noise of SNR=15dB, ,Tap=10, epochs=800 (k) The test output signal with canceling noise of SNR=20dB,Tap=10, epochs=800.

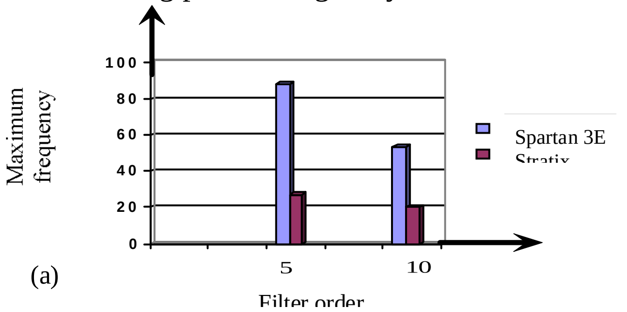
As mentioned earlier, the device chosen is the Spartan 3E, which contains 4656 logic elements, 20 embedded multipliers, and has maximum clock rate of 50MHz. Area and speed are the two main measurements in evaluating the hardware performance of this filter. Fig. 7 shows the varying filter orders vs. speed and area, respectively. Area is measured by number of slices occupied. Speed is measured by the maximum allowable clock frequency. These two factors are compared between the system proposed in this paper and the system of Stratix FPGA device presented in [5]. Each Logic Element (LE) in both Spartan 3E and Stratix families consists of a 4-input lookup table and a flip-flop.

One can see from Fig.7 that the system presented of our paper consumes lower area and can be operated in higher clock frequencies than the system presented in [5]. That is due to the techniques used in calculation components. Also it is due the optimized VHDL codes that used for modeling.

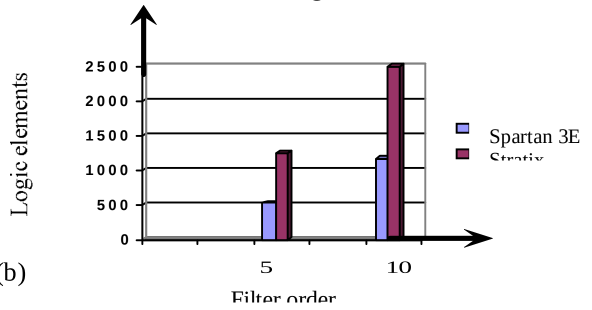
From Fig.7, one can see that the system can be operated in a maximum frequency but of course if the FPGA chip is able to work on this frequency.

Fig .8 shows the timing diagram for the calculations that performed inside the network PEs. One can see that the PEs inputs (x_0,x_1,……………) that represent the input signal samples are fully pipelined. The PEs outputs (y_0,y_1,…………….) are calculated in parallel. Also every clock cycle the network final output signal (z) is generated.

It is believed that a better result can be achieved if the numbers of tapes are increased. Also, more than one layer can be used in the TDNN to improve the filter performance. The LMS algorithm that is used for weight adaptation can also be on-chip implemented, then, the speed of the learning process is greatly increased. These works can be left to be investigated as a future works.



(a)



(b)

Fig 7: (a) Speed vs. FIR filter order (b) Area vs. FIR filter order.

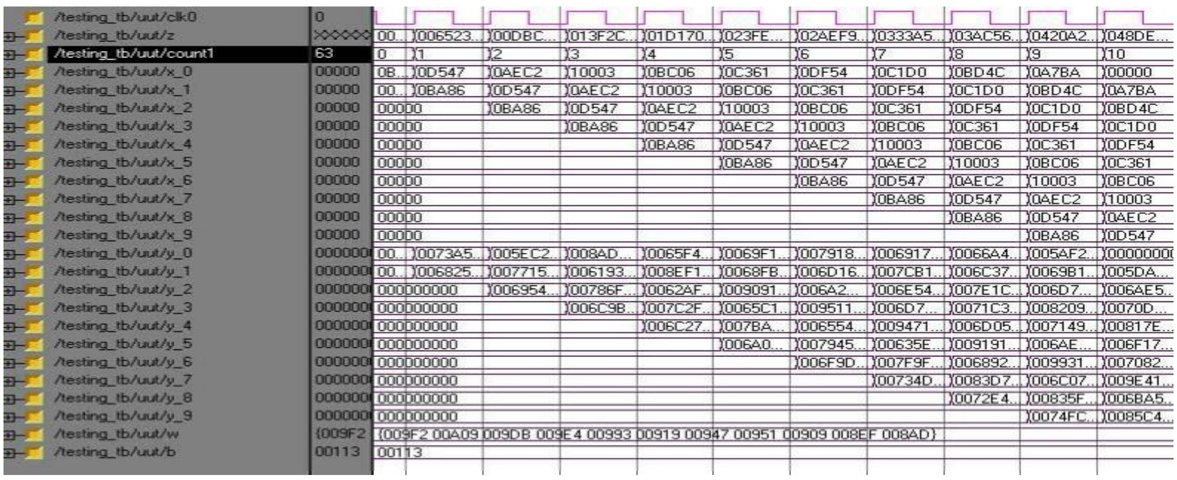

```
/testing_tb/uut/clk0      0
/testing_tb/uut/z         XXXXXX     00..  006523.. 00DBC... 013F2C.. 01D170.. 023FE... 02AEF9.. 0333A5.. 03AC56.. 0420A2.. 048DE...
/testing_tb/uut/count1    63         0     1        2        3        4        5        6        7        8        9        10
/testing_tb/uut/x_0       00000      0B..  0D547    0AEC2    10003    0BC06    0C361    0DF54    0C1D0    0BD4C    0A7BA    00000
/testing_tb/uut/x_1       00000      00..  0BA86    0D547    0AEC2    10003    0BC06    0C361    0DF54    0C1D0    0BD4C    0A7BA
/testing_tb/uut/x_2       00000      00000          0BA86    0D547    0AEC2    10003    0BC06    0C361    0DF54    0C1D0    0BD4C
/testing_tb/uut/x_3       00000      00000                   0BA86    0D547    0AEC2    10003    0BC06    0C361    0DF54    0C1D0
/testing_tb/uut/x_4       00000      00000                            0BA86    0D547    0AEC2    10003    0BC06    0C361    0DF54
/testing_tb/uut/x_5       00000      00000                                     0BA86    0D547    0AEC2    10003    0BC06    0C361
/testing_tb/uut/x_6       00000      00000                                              0BA86    0D547    0AEC2    10003    0BC06
/testing_tb/uut/x_7       00000      00000                                                       0BA86    0D547    0AEC2    10003
/testing_tb/uut/x_8       00000      00000                                                                0BA86    0D547    0AEC2
/testing_tb/uut/x_9       00000      00000                                                                         0BA86    0D547
/testing_tb/uut/y_0       0000000    00..  0073A5.. 005EC2.. 008AD... 0065F4.. 0069F1.. 007918.. 006917.. 0066A4.. 005AF2.. 0000000
/testing_tb/uut/y_1       0000000    00..  006825.. 007715.. 006193.. 008EF1.. 0068FB.. 006D16.. 007CB1.. 006C37.. 0069B1.. 005DA...
/testing_tb/uut/y_2       0000000    000000000      006954.. 00786F.. 0062AF.. 009091.. 006A2... 006E54.. 007E1C.. 006D7... 006AE5..
/testing_tb/uut/y_3       0000000    000000000               006C9B.. 007C2F.. 0065C1.. 009511.. 006D7... 0071C3.. 008209.. 0070D...
/testing_tb/uut/y_4       0000000    000000000                        006C27.. 007BA... 006554.. 009471.. 006D05.. 007149.. 00817E..
/testing_tb/uut/y_5       0000000    000000000                                 006A0... 007945.. 00635E.. 009191.. 006AE... 006F17..
/testing_tb/uut/y_6       0000000    000000000                                          006F9D.. 007F9F.. 006892.. 009931.. 007082..
/testing_tb/uut/y_7       0000000    000000000                                                   00734D.. 0083D7.. 006C07.. 009E41..
/testing_tb/uut/y_8       0000000    000000000                                                            0072E4.. 00835F.. 006BA5..
/testing_tb/uut/y_9       0000000    000000000                                                                     0074FC.. 0085C4..
/testing_tb/uut/w         {009F2     {009F2 00A09 009DB 009E4 00993 00919 00947 00951 00909 008EF 008AD}
/testing_tb/uut/b         00113      00113
```

Fig.8: The timing diagram of the FIR network

7- REFERENCES:

[1] Figueroa M., Hsu D., and Diorio C. "*A Mixed-Signal Approachto High Performance, Low-Power Linear Filters,*" IEEE Journal of Solid-State Circuits,vol. 36, pp. 816- 822, 2001.
[2] Kihas D., Zeljko M., and Branko D., " *Adaptive Filtering based on Recurrent Neural Networks*", Journal of Automatic Control, University of Belgrade, vol.13, no. 1, pp.13-24, 2003.
[3] Miguel Figueroa, Seth Bridges, David Hsu and Chris Diorio,"A 19.2GOPS, 20mW Adaptive FIR Filter",Solid-State Circuits Conference ESSCIRC '03. Proceedings of the 29th European,pp. 509-512,2003.
[4] Meyer U., "*Digital Signal Processing with FieldProgrammable Gate Array*", Springer-Verlag Berlin Heidelberg New York, 2001.
[5] Lin A. Y. and Gugel K. S. , " *Feasibility of Fixed-Point Transversal Adaptive Filters in FPGA Devices with Embedded DSP Blocks*" Proceedings. The 3rd IEEE International Workshop on Volume, Issue, 30 June-2 July 2003 pp. 157 –160, 2003.
[6] Pirsch P., "*Architectures for Digital Signal Processing*", John Wiley & Sons, 1998.
[7] Boozari M., " *State Estimation and Transient Stability Analysis in Power Systems using Artificial Neural Networks*", M. Sc. Thesis, School of Engineering Science,Simon Fraser University, 2004.
[8] Hush D.R. and Horne G., "*Progress in supervised neural networks*", IEEE SignalProcessing Magazine, vol. 10, no. 1, pp. 8-37. 1993.
[9] Douglas S. C. and Meng T. H., "*Linearized least- squares training of multilayer feedforward neural networks*", In Int. Joint Conf. of Neural Networks, pp. 307–
312, 1991.
[10] Jocelyn C., Eric C., and Steven P.,"*VIP: An FPGA-based Processor for Image Processing and Neural Networks*", Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro'96), IEEE, Lausanne, Switzerland, pp. 330-336.1996.
[11] Li X., Moussa M., and Areibi Sh., "*Arithmetic formats for implementing Artificial Neural Networks on FPGAs*" Canadian Journal on Electrical and Computer Engineering, vol. 31, issue 1 ,pp. 31-40,2006.
[12] Tommiska M.T., "*Efficient digital implementation of the sigmoid function for reprogrammable logic*", IEE Proceedings – Computers and Digital Techniques 150, no. 6, pp. 403-411, 2003.
[13]Morris Mano M., *"Computer System Architecture"*, Prentice Hall, Inc., 1993.

The work was carried out at the University of Mosul