EFFICIENT IMPLEMENTATION OF LMS ADAPTIVE FILTER USING DISTRIBUTED ARITHMETIC

A thesis submitted towards the partial fulfilment of the requirements
for the award of degree of

MASTER OF TECHNOLOGY
IN
VLSI DESIGN

Submitted by
Priya Bali
Roll. No. 601461019

Under the guidance of
Dr. Ravi Kumar (Assistant Professor, ECED)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
THAPAR UNIVERSITY, PATIALA-147004
PUNJAB, INDIA
JULY-2016

DECLARATION

I hereby declare that the thesis entitled "Efficient implementation of LMS Adaptive Filter using Distributed Arithmetic" is an authentic record of my own work carried out as the requirement for the award of degree of Master of technology in VLSI at Electronics and Communication Engineering Department of Thapar University, Patiala, under the guidance of **Dr. Ravi Kumar (Assistant Professor),** Electronics & Communication Engineering Department during the session 2014-2016.

Date: 8-7-2016

Priya Bali

PRIYA BALI

Roll No: 601461019

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Ravi

Dr. Ravi Kumar

Assistant Professor

ECED

TU, Patiala

Countersigned By:

Dr. Sanjay Sharma

Head of Department

ECED

TU, Patiala

Dean of Academic Affairs

ECED

TU, Patiala

ACKNOWLEDGEMENT

To discover, analyze and to present something innovative is to endeavour on an ungraded path towards and unexplored destination is an onerous adventure unless one gets a true torch bearer to show the right path. I would have never accomplished in completing my work without the cooperation, encouragement and help provided to me by different people. I am getting short of words to reveals my deep regards. I take this opportunity to express my subtle sense of gratitude and respect to all those who supported me through thesis duration. I acknowledge with gratitude and modesty my deficit to**Dr.Ravi Kumar (AssistantProfessor)** Electronics & Communication Engineering Department, Thapar UniversityPatiala under whose direction I had the privilege to complete my thesis. I am really veryfortunate to have the opportunity to work with them.

I convey my candid thanks to **Dr. Sanjay Sharma(Head of Department)**and**Dr. Anil Arora (PG coordinator**) entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

I would like to thank my parents for their years of obstinate love and encourage, they have always desired the finest for me and I admire their determination and sacrifice. Above all I render my thanks to the Almighty who bestowed self-confidence, capability and strength in me to bring this work to completion and not letting me down at the time of dilemma. I am greatly obliged to all my friends, who have amicably applied themselves to the task of helping me with abundant moral support and esteemed suggestions helped in successful completion of the present study.

Priya Bali

ABSTRACT

A LMS adaptive noise canceller has been proposed to remove the maternal heartbeat interference noise from the foetal heartbeat signal by using Distributed arithmetic. DA is capable of computing the inner product of vectors by calculating the partial products with shifting and accumulation. These partial products are stored in look-up table. .Adaptive filter has been proposed with conventional LMS adaptive filter and Sign LMS adaptive filters. It is being found that the noise canceller proposed using sign LMS is more efficient as compared to the conventional LMS adaptive filter. The convergence rate of sign LMS is very fast in comparison to the other adaptive algorithms. The throughput of the sign LMS adaptive filter using distributed arithmetic is high.

The maternal heartbeat signal is stronger as compared to the foetal heartbeat signal. The maternal heartbeat signal is measured from the chest and the foetal heartbeat signal is measured from the abdomen. Since, the maternal heartbeat signal is strong it causes interference in the foetal measured heartbeat signal. This interrupted signal is needed to be recalculated so as to remove noise from the foetal heartbeat signal.

The analysis has been made on the basis convergence rate, varying step size and mean square error of different adaptive algorithms. The convergence rate is measured on behalf of mean square error vs. Iterations consumed to converge the noise canceller. The conventional LMS adaptive filter converges for thousands of iterations while the LMS adaptive filter using distributed arithmetic converges for hundreds of iterations.

INDEX

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 ADAPTIVE FILTER THEORY

An **adaptive filter** has a linear filter whose transfer function is controlled by changing parameters and those parameters are adjusted with the help of necessary algorithm. Since the optimized algorithms are rather complex, almost every adaptive filter is implemented as a digital filter [1]. We have the requirement to use Adaptive filters in some applications as we do not know some of the parameters of the wanting processing operation initially or they are variable in nature. Feedback is used in terms of error signal by closed loop adaptive filter so that its transfer function is refined. The output and input signals of the adaptive filter are iteratively modelled for computations. Adaptive filters use adaptive algorithm so that the filter coefficients are self-adjusted. Cost function is used by closed loop adaptive process which is a used to check the optimum performance of the adaptive filter. Also cost function is fed to an algorithm [2]. This algorithm determines the modifications needed in the filter transfer function so that the cost of the next iteration can be minimized. Mean square error signal is the commonly used cost function. Now days, digital signal processors are using more power. With the increasing power, the use of the adaptive filter has become very common. Many applications like cell phones, camcorders and digital cameras, and medical equipment are using adaptive filters.

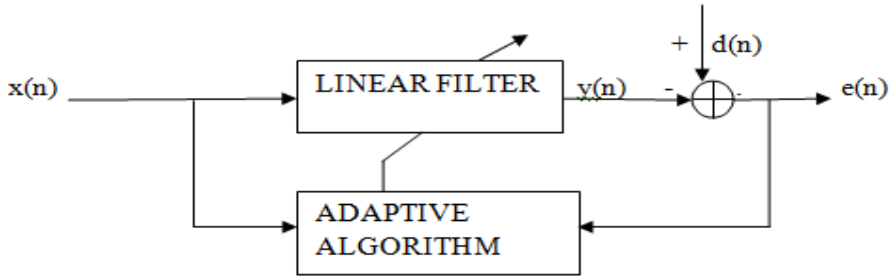

Figure 1. Block diagram of Adaptive filter

Where

- $x(n)$ = input signal of the adaptive filter

- $y(n)$ = output signal of the adaptive filter
- $d(n)$ = desired signal
- $e(n)$ = error signal which is the difference between $d(n)$ and $y(n)$

A block diagram is shown in Figure 1 in which working of adaptive filter is explained. Linear filter is given with a sample from input signal x(n). Output signal y(n) is computed by linear filter at time n [1]. There are adjustable parameters which affects the computation of output y(n). The output signal and desired signal d(n) are compared by the difference the output sample and desired signal sample at time n. The difference is the error signal which is given as

$$e(n) = d(n) \quad y(n) \tag{1}$$

The feedback to the adaptive algorithm is the error signal. Adaptive algorithm adapts or alters the coefficients of the filter at time (n + 1) in a defined manner.

An oblique arrow piercing the linear filter block represents the process of adaptation. With the increasing time index n, it is expected that with this adaptation the output of the filter becomes a good match to the desired signal resulting in the decrease of the magnitude of e(n) with time [2]. A good match depends upon the type of adaptive algorithm used to modify the coefficients of the adaptive filter. Adaptation is the method used to change the coefficients of the adaptive filter at time index (n+1). The number coefficients' number and their type is determined by the computation structure selected. There is a training period that targets the weights so that the required response d(n) can be achieved. When weights become the function of n where n is the time with feedback, adjust the weights to improve y(n) until the training phase is over [2]. After that d(n) will not be applied and y(n) will serve as d(n) only. Training phase is executed time to time.

An adaptive filter is defined by two processes:

1. ***Filtering process*** which involves the convolution of x(n) and w(n). w(n) are the weights of filters.
2. ***Weight adjustment*** which takes e(n) and fed it back.

1.1.1 Applications of Adaptive Filters

Depending upon the versatility of the adaptive filters, there are four types of the adaptive filters. Below listed are the basic types of applications:

1. **System identification:** Figure 2 shows the modelling or identification problem. In this application, the input of the adaptive filter is same as the system. The comparison of output signal *y(n)* of the adaptive filter and desired signal *d(n)*is done to generate an error signal [6]. The weights *w(n)* are adjusted with the help of this error signal *e(n)* and the system is identified.

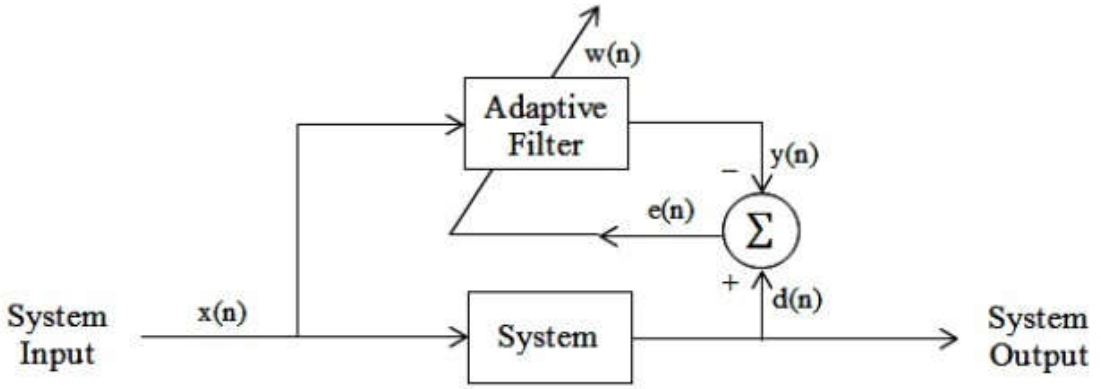

Figure 2. System Identification Model

2. **Inverse Modelling**: The inverse modelling shown in figure 3, also called as *deconvolution,* is used to discover and track the inverse transfer function of the system. In this application, one input *x(n)* is received by the system and it has output *u(n)* whose connection is with the adaptive filter. The desired signal has the delayed form of the input signal [6]. Then the filter output y(n) is compared with the desired signal d(n) producing an error signal. Filter weights are updates using this error signal.

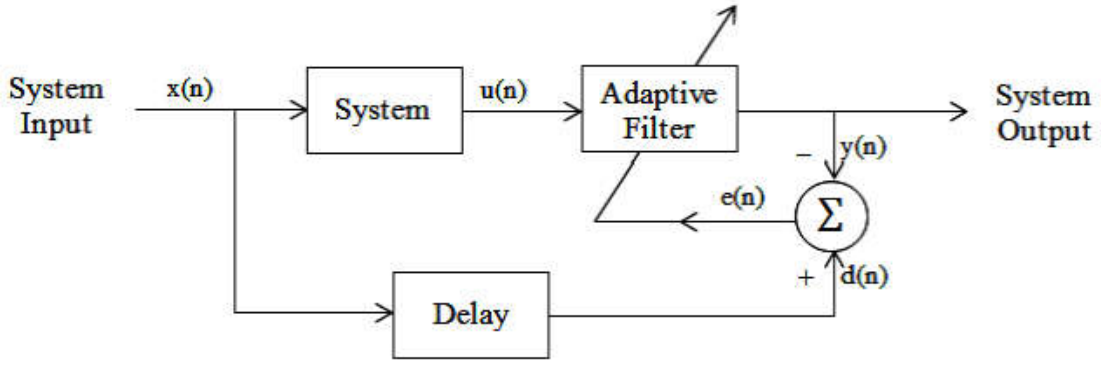

Figure 3. Inverse Modelling

3. **Prediction**: The figure 4 shows the adaptive filter predictor. The aim of the adaptive filter is to produce a random signal to its best. A delay is provided to the input random signal x(n) and these past values of x(n) are given to the adaptive predictor. The output y(n) and desired signal are compared with each other. The desired signal d(n) is the random signal x(n) itself. If the weights of the adaptive filter are adjusted with the help of output of the filter then it behaves as a predictor filter and if the computed error from the comparison of the output signal and desired signal is used to find the new weights of the adaptive filter then it is called an error filter [6].

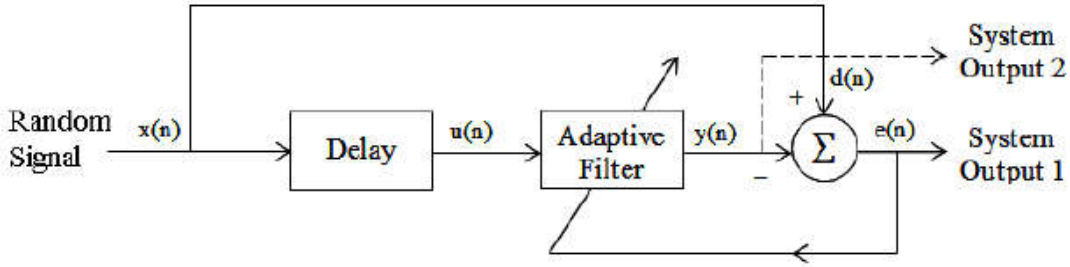

Figure 4. Prediction model

4. **Noise Cancellation**: Figure 5 shows the noise canceller. It consists of a desired signal *d(n)*, which is the pre-eminent noisy signal. A $n_2(n)$ signal is used to corrupt this pre-eminent signal and it is the sum of signal s(n) and noise signal $n_2(n)$. A noise signal $n_1(n)$ is given to the output signal y(n). The pre-eminent signal is corrupted because of this noise source $n_1(n)$. The pre-eminent signal and the output of the adaptive filter are compared and the difference of these two signal produces the error signal e(n). A noise canceller will provide us with an error which is the good match of the original signal by cutting out the interference [6].

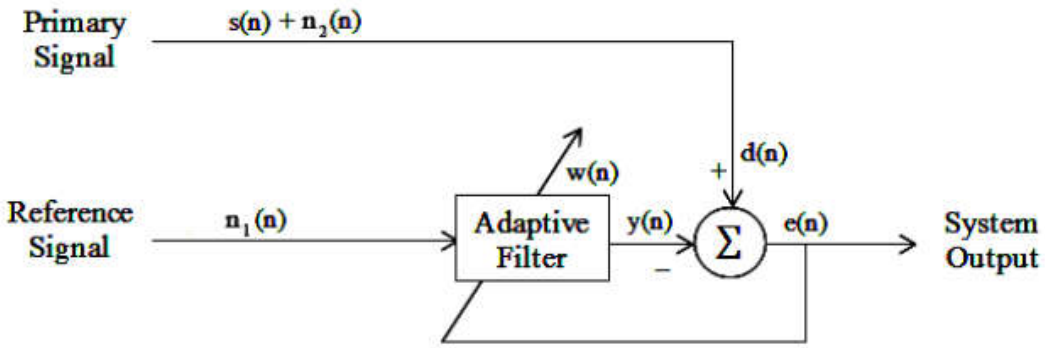

Figure 5. Noise Canceller

1.2 METHODS FOR CALCULATION OF WEIGHTS IN ADAPTIVE FILTERING

1.2.1 Steepest descent method:

A transversal filter is considered whose input is ***x(n)***, tap weights of the filter are $w_0\ w_1\ w_2\ \ldots\ w_{N-1}$ and a desired response d(n) [1].

The filter's weights are

$$w = [w_0\ \ w_1\ \ w_2\ \ \ldots\ \ w_{N-1}]^T \tag{1.2.1}$$

and

input of the filter is

$$x(n) = [x(n)\ x(n\quad 1)\ x(n\quad 2)\ \ldots\ \ x(n\quad N+1)]^T \tag{1.2.2}$$

The MSE (mean squared error) is the mostly used objective function used in adaptive filtering and it is given as:

$$F[e(n)] = \quad (n) = E\left[e^2\text{-}(n)\right] = E[d^2(n)\quad 2d(n)y(n) + y^2(n)] \tag{1.2.3}$$

where cost function is defined by (n) and *E[ ]* defines the expectation.

Y(n) which is the output signal, can be given as:

$$y(n) = \sum_{i=0}^{N}\{w_i(n)x(n\quad i)\} = w^T(n)x(n) \tag{1.2.4}$$

Put value of y(n) from equation 1.2.4 to equation 1.2.3, we get:

$$E[e^2(n)] = \quad (n) = E[d^2(n)\quad 2d(n)w^T(\mathrm{n})\mathrm{x}(\mathrm{n}) + \mathrm{w^T(n)x(n)x^T(n)w(n)}] \tag{1.2.5}$$

$$= E[d^2(n)]\quad 2E[d(n)w^T(n)x(n)] + E[w^T(n)x(n)x^T(n)w(n)] \tag{1.2.6}$$

In FIR filter whose filter coefficients are fixed, MSE can be also written as:

$$E[e^2(n) = \quad (n)$$
$$= E[d^2(n)]\quad 2w^T(n)E[d(n)x(n)]$$
$$+ w^T(n)E[x^T(n)x(n)]w(n) \tag{1.2.7}$$

Between d(n) and x(n), write a N x 1 matrix which is the cross-correlation matrix 'p':

$$p = E[d(n)x(n)] = [p_0\ \ p_1\ \ \ldots\ldots\ \ p_{N-1}]^T \tag{1.2.8}$$

The autocorrelation N X N matrix 'R' is given as:

$$R = E[x^T(n)x(n)] = \begin{matrix} r_{00} & r_{01} & r_{02} & r_{0,N-1} \\ r_{10} & r_{11} & r_{21} & r_{1,N-1} \\ r_{20} & r_{21} & r_{22} & r_{2,N-1} \\ r_{N-1,0} & r_{N-1,1} & r_{N-1,2} & r_{N-1,N-1} \end{matrix}^T \quad (1.2.9)$$

From above equation, if the auto-correlation matrix 'R' and cross-correlation matrix 'P' are known then by controlling the weights w(n), the mean square error function can be found. With the differentiation of mean square error with respect to weights w(n), gradient vector of this MSE can be found having its relation with coefficients as written below:

$$= \frac{\partial}{\partial w} = [\frac{\partial}{\partial w_0} \quad \frac{\partial}{\partial w_1} \quad \frac{\partial}{\partial w_2} \qquad \frac{\partial}{\partial w_N}]^T = \quad 2p + 2Rw \quad (1.2.10)$$

The Weiner solution is found by putting the gradient vector equals to zero and also by taking ***R*** such that it is a non-singular matrix, shown as:

$$w_0 = R^{-1}p \quad (1.2.11)$$

The procedure of the steepest descent algorithm is given below so that the optimum solution can be found [1].

1. To initialize the steepest descent algorithm, make an initial estimate of the coefficients so that they can be optimized for the computation of the minimum Mean square error.

2. Find the true gradient function with respect to the filter coefficients.

3. Update the coefficients by going in the backward direction to the gradient vector found in the previous step.

4. Check whether variations in the coefficients are notable or not by repeating the above steps 2 and 3.

w(n) can be computed by following the above procedure as:

$$w(n+1) = w(n) \quad \mu_n \quad (1.2.12)$$

Where μ defines step-size parameter and is the computed gradient vector by putting $w_0 = w(n)$.

Put the values in equation 1.2.11 to 1.2.12, we get:

$$w(n+1) = w(n) \quad \mu[\ 2p + 2Rw(n)] = w(n) \quad 2\mu[Rw(n) \quad p] \tag{1.2.13}$$

Rearranging the above equation 1.2.13 so as to check the convergence of updated ***w(n)*** is towards ***wo***,

$$w(n+1) = [I \quad 2\mu R]w(n) + 2\mu p \tag{1.2.14}$$

where ***I*** is defined as identity matrix of size NxN.

Put the value of ***p*** from equation 1.2.11 to equation 1.2.14 and subtract ***wo*** from equation 1.2.12, we get:

$$w(n+1) \quad w_0 = [I \quad 2\mu R]w(n) + 2\mu R w_0 \quad w_0 = [I \quad 2\mu R][w(n) \quad w_0] \tag{1.2.15}$$

A vector v(n) is defined by subtracting optimum weight ***wo*** from tap-weight w(n) as:

$$v(n) = w(n) \quad w_0 \tag{1.2.16}$$

and putting equation 1.2.16 in equation 1.2.15, we get:

$$v(n+1) = [I \quad 2\mu R]v(n) \tag{1.2.17}$$

We are using the below written fact for the computation of recursive scalar equations.

$$R = QAQ^T \tag{1.2.18}$$

where Q is the matrix whose columns are ortho-normal Eigen values and A contains the Eigen values $\lambda_0\ \lambda_1\ \lambda_2 \ldots \lambda_{N-1}$ of R as the diagonal values. Putting equation 1.2.18 in equation 1.2.17, we get:

$$v(n+1) = [QQ^T \quad 2\mu QAQ^T]v(n) = Q[I \quad 2\mu A]Q^T \tag{1.2.19}$$

A new support variable is defined as written below:

$$v'(n) = Q^T v(n) \qquad (1.2.20)$$

and equation 1.2.19 is rearranged to get:

$$v'(n+1) = [I \quad 2\mu A]v'(n) \qquad (1.2.21)$$

Equivalent scalar equations which are recursive can be represented by considering interval $i = 0,1, \ldots, N \quad 1$ as:

$$v_i'(n+1) = (1 \quad 2\mu\lambda_i)v_i'(k) = (1 \quad 2\mu\lambda_i)^k v_i'(0) \qquad (1.2.22)$$

Looking at the equation 1.2.15, it is found that the ***w(n)*** is convergent to ***wo***, only if we have vector $v'(n)$ of zeros. By considering equation 1.2.15 and 1.2.21, it is also noticed that the step-size μ should also be selected such that:

$$|1 \quad 2\mu\lambda_i| < 1 \qquad (1.2.23)$$

Equation 1.2.23 describes that:

$$1 < 1 \quad 2\mu\lambda_i < 1 \qquad (1.2.24)$$

Or

$$0 < \mu < \frac{1}{\lambda_i} \qquad (1.2.25)$$

and final condition that guarantees the steepest-descent algorithm's convergence is that the parameter μ i.e. the step size should be:

$$0 < \mu < \frac{1}{\lambda_{max}} \qquad (1.2.26)$$

Where λ_{max} corresponds to the greatest of the Eigen values.

1.2.2 Least mean square algorithm:

The LMS (least mean square) algorithm is the part of linear stochastic gradient family's algorithms. It does two things. Firstly, it does not need to find the true signal stats such as cross-covariance, covariance, and etc. These signal statistics are not regularly present in

practice. Secondly, this algorithm also has a tracking method that allows it to track changes which occur in the signal stats [6].

LMS algorithm is simple and has a good operational stability because relevant correlation and matrix transposition are not required. Because of this simplicity and applicability, LMS algorithm is the principle linear adaptive algorithm.

The LMS algorithm consist two general processes [6]:

1. First is the filtering process which computes the output of the transversal filter with respect to the tap inputs. This computed output is subtracted from a desired signal which generates the error signal.

2. Second is the adaptive process in which filter weights are adjusted adaptively according to the estimated error.

Mean square error is the cost function of LMS algorithm, given as:

$$j(n) = E|e^2(n)| \tag{1.2.27}$$

where *J* represents cost function, $|e^2(n)$ | represents the Euclidean norm of the error by subtracting filter ouput from the desired response.

The estimated error signal is given as:

$$e(n) = d(n) \quad y(n) \tag{1.2.28}$$

Where d(n) shows desired signal and y(n) is represented as output signal.

Filter output is calculated as follows:

$$y(n) = w^T(n)x(n) \tag{1.2.29}$$

Where x(n) is the input vector consisting of all the input samples and w(n) is the tap weight vector. The size of input vector and weight vector is same.

According to the wiener filter equation, it is found that:

$$w(n+1) = w(n) \quad \mu\nabla(n) = w(n) + 2\mu\left[p(n) \quad \hat{R}(n)w(n)\right] \tag{1.2.30}$$

for $n = 0, 1, 2, ...,$ where ∇ represents the estimation of the gradient vector of the mean square error function, the estimation of cross-correlation vector is represented by p between the desired signal and the given input signal, $\hat{R}$ represents input signal's correlation matrix

[6]. A step-size parameter is used to decide the rate of convergence of the algorithm to the calculated minimum error. This step size parameter is represented by μ. To increase the convergence time, μ should be decreased but it makes the algorithm slow. [6]. On the contrary a larger step size increases the speed of the algorithm but slows the convergence and hence, mean square error is increased. The step size μ must fulfil the below written specifications:

$$0 < \mu < \frac{2}{tap \quad input\ power} \tag{1.2.31}$$

Tap input power is as follows:

$$\sum_{k=0}^{M-1} E[|x(n \quad k)|^2] \tag{1.2.32}$$

Resulted estimated gradient is given as:

$$\hat{\nabla}(n) = 2\mu d(n)x(n) \quad 2\mu x(n)[d(n) \quad x^T(n)w(n)] = \quad 2\mu e(n)x(n) \tag{1.2.33}$$

Equation 1.2.30 is updated as:

$$w(n+1) = w(n) + 2\mu e(n)x(n) \tag{1.2.34}$$

Step	**Equations**	*	**+ or -**	/
	Initialization: w(0) = 0	-	-	-
	For n = 1,2,3,...	-	-	-
1	$y(n) = w^T(n)x(n)$	L	L – 1	-
2	$e(n) = d(n) \quad y(n)$	-	1	-
3	$w(n+1) = w(n) + 2\mu e(n)x(n)$	L + 2	L	-
	Total	2L + 2	2L	-

Table 1: Computer complexity of LMS algorithm

1. **Normalized least mean square algorithm:**

The LMS algorithm reviewed in the previous section, it is seen that the correlation 2μe(n)x(n) of tap-weight input is directly proportional to the length of ***x(n)***.

As the length of the ***x(n)*** is increased, a gradient noise amplification is experienced in the LMS algorithm. Normalized least mean square algorithm was build so as to overcome this problem [1].

The increased length of input ***x(n)*** makes almost impossible to select a step size μ that can guaranty the stability of the algorithm. So, a variable step size is provided in NLMS, given by:

$$\mu = \frac{\overline{\mu}}{\delta + \left|\left|x(n)\right|\right|^2} \tag{1.2.35}$$

Where δ represents a small constant, $\overline{\mu}$ represents step size in the range $0 < \overline{\mu} < 2$ and ||x(n)|| shows the Euclidian norm of x(n) [1].

The weight update equation is represented as:

$$w(n+1) = W(n) + 2\mu e(n)x(n) = w(n) + 2\frac{\overline{\mu}}{\delta + \left|\left|x(n)\right|\right|^2} e(n)x(n) \tag{1.2.36}$$

Step	**Equations**	*	**+ or -**	/
	Initialization: w(0) = 0	-	-	-
	For n = 1,2,3,...	-	-	-
1	$y(n) = w^T(n)x(n)$	L	L – 1	-
2	$e(n) = d(n) \quad y(n)$	-	1	-
3	$w(n+1) = \ w(n) + 2\frac{\overline{\mu}}{\delta + \left\|\left|x(n)\right|\right\|^2} e(n)x(n)$	2L + 2	2L	1
	Total	3L + 2	3L	1

Table 2: Computer complexity of NLMS algorithm

2. Leaky least mean sqaure algorithm:

The definition of cost function of leaky LMS is given as:

$$j = e^2(n) + a\sum_{i=0}^{N-1} w_i^2(n) \tag{1.2.37}$$

where α corresponds to the leaky factor having range from 0 to 0.1. Since a is present in the cost function of this algorithm, it is different from standard LMS algorithm. The coefficient overflow problem is diminished in leaky LMS algorithm because both $e^2(n)$ and the filter weights are accounted by the cost function [2]. The filter weights for this algorithm are updated according to the following equation:

$$w(n+1) = (1 \quad \mu a)w(n) + \mu e(n)x(n) \tag{1.2.38}$$

To make leaky LMS algorithm and standard LMS algorithm same, a should be 0. A high steady state error will be resulted because of high leaky factor.

3. **Normalized leaky LMS:**

Modifications are made in leaky LMS algorithm to get a normalized leaky LMS algorithm [1]. The weights are updated as follows [2]:

$$w(n+1) = (1 \quad \mu a)w(n) + \mu e(n)\frac{x(n)}{\left||x(n)|\right|^2} \tag{1.2.39}$$

1. **Sign algorithms:**

Many application of adaptive filters need to implement these algorithms on hardware levels, such as signal processing (DSP) devices, FPGA devices, and ASICs. A simple approach to LMS algorithm is required by such applications [7]. The standard LMS can be simplified using the following sign function:

$$sgn(n) = \begin{cases} 1; & x > 0 \\ 0; & x = 0 \\ 1; & x < 0 \end{cases} \tag{1.2.40}$$

The sign function when applied to standard LMS algorithm results in below written three types:

- **Sign-error LMS algorithm**: sign function is Applied to the error signal $e(n)$. The filter coefficients for this algorithm are updated as:

$$w(n+1) = w(n)\mu sgn\big(e(n)\big)x(n) \tag{1.2.41}$$

  For an error signal $e(n)$ equals to zero, no multiplication operation will be needed by this algorithm. When error signal $e(n)$ is not equal to zero, only one multiplication will be needed by this algorithm [7].

Step	Equations	*	+ or -	/
	Initialization: w(0) = 0	-	-	-
	For n = 1,2,3,...	-	-	-
1	$y(n) = w^T(n)x(n)$	L	L – 1	-
2	$e(n) = d(n) \quad y(n)$	-	1	-
3	$w(n+1) = w(n) + \mu sgn\big(e(n)\big)x(n)$	1	1	-
	Total	L + 1	L + 1	0

Table 3: Computer complexity of sign-error LMS algorithm

- **Sign-data LMS algorithm:** sign function is applied to the input signal x(n). The filter coefficients for the sign-data LMS algorithm are updated by the following equation:

$$w(n+1) = w(n) + \mu e(n) sgn(x(n)) \qquad (1.2.42)$$

When the input signal x(n) is zero, no multiplication operation is needed in this algorithm. When input signal x(n) is not equal to zero, only one multiplication is needed [7].

Step	Equations	*	+ or -	/
	Initialization: w(0) = 0	-	-	-
	For n = 1,2,3,...	-	-	-
1	$y(n) = w^T(n)x(n)$	L	L – 1	-
2	$e(n) = d(n) \quad y(n)$	-	1	-
3	$w(n+1) = w(n) + \mu e(n) sgn(x(n))$	1	1	-
	Total	L + 1	L + 1	0

Table 4: Computer complexity of sign-data LMS algorithm

- **Sign-sign LMS algorithm:** sign function is applied to error signal $e(n)$ as well as input signal x(n). The filter coefficients for this algorithm are updated by following equation [1]:

$$w(n+1) = w(n) + \mu sgn\big(e(n)\big) sgn(x(n)) \qquad (1.2.43)$$

Step	Equations	*	+ or -	/
	Initialization: w(0) = 0	-	-	-
	For n = 1,2,3,...	-	-	-
1	$y(n) = w^T(n)x(n)$	L	L – 1	-
2	$e(n) = d(n) \quad y(n)$	-	1	-
3	$w(n+1) = w(n) + \mu sgn\big(e(n)\big)sgn(x(n))$	1	1	-
	Total	L + 1	L + 1	0

Table 5: Computer complexity of sign-sign LMS algorithm

It can be noticed that when one of the error signal *e*(*n*) or input signal x(n) is equal to zero, no multiplication operation is involved. When none of the error signal *e*(*n*) or input signal x(n) is equal to zero, only one multiplication is involved.

The above sign LMS algorithms have less multiplication operations as compared to other LMS algorithms. The sign LMS can be replaced by shift operations by making the step size μ as a power of 2. Because of this power of 2 , the sign algorithms have only one shift and addition operations [2]. Convergence speed is less as compared to the standard LMS algorithm. Also, steady state error is more in comparison to the standard LMS algorithm.

The sign-sign LMS algorithm is the fastest of them.

1.2.3 RLS algorithm:

The aim of the LMS algorithm is to minimize the mean square error, but on the contrary the RLS (recursive least-squares algorithm) aims on finding, recursively, the filter needed to minimize the cost function. The convergence rate of the RLS algorithm is very fast but considering its problem, RLS algorithm's computation complexity is more [1].

Weighted least squares is the cost function of RLS algorithm, given as:

$$j(n) = \sum_{i=0}^{k} \lambda^{n-1} e^2(n) \qquad (1.2.44)$$

Where $0 < \boldsymbol{\lambda} < 1$ is known as 'forgetting factor'. With this forgetting factor less weight is given exponentially to the older error samples and *e(n)* represents the error which is calculated by subtracing output signal y(n) generated from a transversal filter whose inputs are $x(n), x(n \quad 1), \ldots, x(n \quad M+1)$ from the desired signal *d(n)*.

The error signal is given as:

$$e(n) = d(n) \quad y(n) = d(n) \quad w^T(n \quad 1)x(n) \tag{1.2.45}$$

Where x(n) defines the input vector given as:

$$x(n) = x(n), x(n \quad 1), \dots, x(n \quad M + 1) \tag{1.2.46}$$

w(n) defines the tap-weight vector, given as:

$$w = [w_0(n) \ w_1(n) \ w_2(n) \ \dots \ w_{N-1}(n)]^T \tag{1.2.47}$$

The least cost function value *J(n)* is reached when there will be an optimum value for tap-weights. This optimum value is given as:

$$(n)\widehat{w}(n) = z(n) \tag{1.2.48}$$

The (n) is the MxM matrix and is determined as:

$$(n) = \sum_{i=0}^{k} \lambda^{n-i} x(n) x^T(n) \tag{1.2.49}$$

The z(n) is the *Mx1* cross-correlation matrix defined between the desired signal and input signal of filters as:

$$z(n) = \sum_{i=0}^{k} \lambda^{n-i} x(n) d \ \ (n) \tag{1.2.50}$$

Here, $d^*(n)$ is the complex conjugation.

Matrix inversion Lemma is applied for the computation of the RLS algorithm. The result of applying this matrix inversion lemma is:

$$\bar{a}^1(n) = \lambda^{-1} \ \ \bar{a}^1(n \quad 1) \quad \lambda^{-1} k(n) x^T(n) \ \ \bar{a}^1(n \quad 1) \tag{1.2.51}$$

$\bar{a}^{-1}$ defines the inverse of the correlation matrix, the inverse of forgetting factor is defined by λ^{-1} and k(n) represents the gain.

The gain k(n) is given by:

$$k(n) = \frac{\lambda^{-1} \quad \bar{a}^{1}(n \quad 1)x(n)}{1 + \lambda^{-1}x^{T}(n) \quad \bar{a}^{1}(n \quad 1)x(n)} \tag{1.2.52}$$

The tap-coefficients are calculated with the help of following equation:

$$w(n) = w(n \quad 1) + k(n)e\ (n) \tag{1.2.51}$$

Here, * is used to represent complex conjugation.

Step	Equations	*	+ or -	/
	Initialization: w(0) = 0 and $\bar{a}^{1}(0) = \delta^{-1}$I	-	-	-
	For n = 1,2,3,...	-	-	-
1	$k(n) = \frac{\lambda^{-1} \quad \bar{a}^{1}(n \quad 1)x(n)}{1 + \lambda^{-1}x^{T}(n) \quad \bar{a}^{1}(n \quad 1)x(n)}$	$2L^2 + L$	$2L^2 -$ 2L+1	1
2	$y(n) = w^{T}(n)x(n)$	L	L -1	-
3	e(n) = d(n) – y(n)	-	1	-
4	$w(n) = w(n \quad 1) + k(n)e\ (n)$	L	L	-
5	$\bar{a}^{1}(n) =$ $\lambda^{-1} \quad \bar{a}^{1}(n \quad 1) \quad \lambda^{-1}k(n)x^{T}(n) \quad \bar{a}^{1}(n \quad 1)$	L^2 +L	2L – 1	1
	Total	$3L^2 + 4L$	$2L^2 + 2L$	1

Table 6: Computer complexity of RLS algorithm

1.3 Noise canceller problem:

The adaptive noise canceller is an example of interference cancelling class. The motive of this noise canceller is similar to any adaptive filter. It minimises noise created by the interfering signals. It also aims on cancelling the disruption in an optimal situation [6].

A reference sensor needs an interference signal which is taken and given to the sensor as the reference signal. A primary sensor is detecting the desired signal. A same noise signal is corrupting this signal. An initial response is generated by the adaptive filter and is examined with the desired signal [9]. This comparison computes an error which is given to the filter as feedback. This feedback adjusts the filter coefficients. For an optimum case, the desire signal is the response composed by adaptively.

As discussed before, the motive of the noise canceller is to reduce the interference noise which is corrupting the actual input signal. In the figure above, the *desired signal d(n)*

is the combination of unknown signal *s(n)* and a noise $n_2(n)$ which is achieved from the interference, corrupting the unknown signal. The input to the adaptive filter is *interference* signal $n_1(n)$. Both the noise signals are correlated. An error signal e(n) is generated by subtracting the output of the adaptive filter *y(n)* from the desired signal *d(n)*. Filter weights are adjusted by this generated error so that the noise interference can be minimized. The computed error signal is similar to the desired signal in optimum cases, having minimized interference noise [1].

A good active noise canceller can be achieved by different approaches according to the desired application, having various types of algorithms. Each algorithm has its pros and cons on the basis of application.

1.4 Work Covered in the Thesis:

This work aims to implement several variants of LMS algorithm and evaluate their performance with respect to relevant parameters. A noise canceller is designed in this work. This is an adaptive filter to electrocardiography for foetal signal. Here, we are separating maternal heartbeat from the sensor signal measuring the foetal heartbeat. First, maternal and foetal heartbeats are created. The heart beat rate of foetus is higher than the maternal heartbeat. The amplitude of the maternal heartbeat signal is larger than the amplitude of the foetal heartbeat signal. We are using LMS adaptive filters' types and comparing the obtained results.

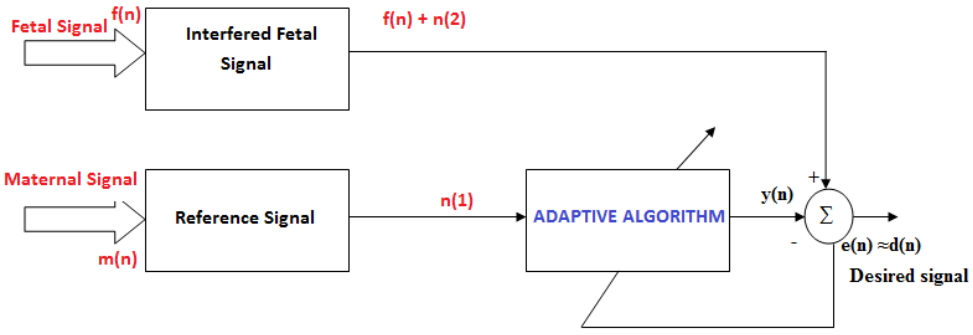

Figure 6: Maternal-foetal adaptive noise canceller

1.5 Research Gaps:

Based upon the literature survey to be presented in the subsequent chapter, following research gaps can be identified:

- There are a lot of scopes to incorporate distributed arithmetic for implementation of popular digital filters.
- One prominent research gap lies in the lack of amalgamation of different arithmetic techniques for multiplier reduction.
- Higher radices are still to be used to evaluate their legacy in multiplier reduction task.
- All these gaps make this field promising for further development of novel paradigms.
- It was also observed that most of the techniques reported in the literature survey are radix-2 techniques.

Specific objectives this thesis can be defined as follows:

- Incorporation of the distributed arithmetic for implementation of popular FIR filters.

1.6 Thesis organization:

This thesis is organized into following five chapters:

1. **Chapter 2** gives the literature survey done to complete the work. The summary of various conference papers and journals is provided giving information about the previous work done by others regarding distributed arithmetic, LMS adaptive filter and the combination of both.
2. **Chapter 3** provided the thorough explanation of the distributed arithmetic. The derivation is given to understand the process of distributed arithmetic. Also, it is explained how to increase the speed of the DA and reduce memory utilization.
3. **Chapter 4** gives the information about the LMS adaptive filters designed using distributed arithmetic with the derivation provided. Different types of LMS algorithms using distributed arithmetic are explained.
4. **Chapter 5** shows the work which is done by us to make an adaptive noise canceller to remove the maternal interference noise from the foetal heartbeat signal using different LMS adaptive algorithms and distributed arithmetic. The comparison of various algorithms is done.
5. **Chapter 6** gives the results and concludes the work done in the thesis. The scope of the work done in future is also shown.

CHAPTER 2
LITERATURE REVIEW

1. ***D. J. Allred et al (2004)***: This paper presents an adaptive filter implemented with a multiplier-free architecture. The filter is implemented using distributed arithmetic (DA) consisting of look-up tables. Multiply-and-accumulate operations are substituted with LUTs accesses. To achieve this, an increase in memory utilization is needed. A series of samples is sent and the adaptive algorithm used is LMS algorithm. An updated look up table is formed with a same auxiliary look up table. FPGA implementation of the system is done. Results show a high-speed and less complex logic LMS adaptive filter is designed with respect to the proposed design [9].

2. ***D. J. Allred et al (2005)***: A LMS adaptive filter with high throughput is presented with the help of distributed arithmetic. DA gives high throughput because it needs bit-serial arithmetic operations and LUTs. These look up tables needs very less cycles per bit, about one cycle, instead of filter length. However, in an adaptive filter, weights are updated after every iteration so the look up tables also needed to be updated again and again to implement a perfect adaptive DA filter. This can impact on the performance of the DA adaptive filtering. An auxiliary LUT with notable addressing can make the DA adaptive filter as efficient as same order DA adaptive filter using DA and throughput is also same. A new hardware of LMS adaptive filter for a increased throughput is given. It is also shown that the area and power usage of DA adaptive is advantageous over DSP microprocessor architectures [10].

3. ***S. Chivapreech et al (2006)***: A transversal adaptive digital filter is designed with the help of LMS adaptive algorithm. The filter is designed on the basis of distributed arithmetic (DA). A multiplier-less desired adaptive digital filter can be designed. VHDL is used to implement the hardware and synthesized on FLEX10K Altera FPGA. The result shows the performance speed and area used of FPGA [11].

4. ***Sudhanshu Baghel et al (2006):*** Fast Block Least Mean Squares is used to propose and implementation an adaptive filter with high. A mulitplierless adaptive filter is designed using Distributed Arithmetic (DA). DA is capable of computing the inner product of vectors by calculating the partial products with shifting and accumulation. These partial products are stored in look-up table. A good computational and area effective implementation of adaptive filter can be proposed with DA. Furthermore, the Field Programmable Gate Arrays' architecture and fundamental blocks of DA map with each other. The results shows that a smaller area can be used for the implementation on DA based adaptive filter on FPGA. About 45% less area is utilized [12].

5. ***Walter Huang et al (2006):*** A new hardware is presented for LMS adaptive filter using conjugate distributed arithmetic (CDA). This can result in high throughput of the filter. In a LMS adaptive filter implemented using traditional distributed arithmetic, the filter weights and sum of possible combination weights are kept in auxiliary look-up-table (LUT). In the Conjugate DA, look up tables consists of input signal samples and their possible sums of combination sums and a well organized update system is used to update the look up table. As the sample arrives, the look up table is updated. The design of Conjugate DA adaptive filter is explained. By observing practical implementations, the conjugate DA has a high throughput with respect to MAC operations. The Conjugate DA adaptive filters have advantages on the basis of area and power utilization [12]

6. ***Daniel Efinger et al (2009):*** An adaptive equalizer is proposed in this paper. Inter-symbol interference is removed by this hardware architecture. Weight adjustment for present tapped delay lines to realize the equalizer filter having a 40 gigabit rate is done either manually or random dithering approaches. If a system has random time-varying performance, tap-weight external adjustment is not possible. If tap-weights are adjusted with random dithering to compute the desired setup then it gives adaptation times more than 1s. Digital approach is given to the solution. Distributed arithmetic and parallelization can help to scale this solution for many bit rates using. Direct implementation of equalizer is done with digital least mean square (LMS) adaptation unit [20].

7. ***Sudhanshu Baghel et al (2011):*** Fast Block Least Mean Squares is used to propose and implementation an adaptive filter with high. A mulitplierless adaptive filter is designed using Distributed Arithmetic (DA). DA is capable of computing the inner product of vectors by calculating the partial products with shifting and accumulation. These partial products are stored in look-up table. A good computational and area effective implementation of adaptive filter can be proposed with DA. Also it is very power efficient. An increase in the memory utilization can help to achieve this Furthermore, the Field Programmable Gate Arrays' architecture and fundamental blocks of DA map with each other. The design of FBLMS algorithm is analyzed. The design is based on adaptive filter. FPGA implementation results conforms that the proposed. The results shows that a smaller area can be used for the implementation on DA based adaptive filter on FPGA. About 52% less area is utilized [12].

8. ***Rui Guo et al (2011):*** This paper presents two methods to on the basis of distributed arithmetic to implement FIR adaptive filter. Distributed arithmetic (DA) is used to perform vector-vector multiplication needed for convolution for the implementation of application having bit-level architectures. The methods proposed in this paper use weights as the addresses to call data from the look up tables. The look up table consist the additions of combinations of delayed and scaled input terms. These methods are different from conventional DA techniques. They have used least-mean-square algorithm to update the coefficients of filer and reduce the mean square error from the desired to obtained output. High speed with low area utilization is achieved and also the results show that the computation complexity is low [23].

9. ***Anirut Trakultritrung et al (2012):*** A least mean square adaptive filter designed with distributed arithmetic is presented in this paper and hardware is realized. The method used here is the coefficient-distributive DA to vary the filter weights with respect to time. The results give more appropriation. The coefficient based DA is using an adder network with multiplexer. In this way, the adaptive filter is made multiplierless and with no look up table. The hardware is designed using VHDL and then synthesized on FPGA for practical adaptive filtering experiment [24].

10. ***M. Surya Prakash et al (2013):*** This paper presents a high-performance scheme for the implementation of a least mean square adaptive filter. The distributed arithmetic is

used to design the architecture is based on distributed. The recalculation of the look up tables is needed because the filter coefficients are variable in nature and they should be updated in each step. This paper is presenting a new method on the basis of offset binary coding to update these look up tables every time. For a unit size k=N/m, where m is representing an integer and number of filter weights is represented by n, it is noticed that the throughout for this method is very high. The area utilization is also very low. For example, a 128-tap FIR adaptive filter with the proposed method of implementation gives 12 times higher throughput (for k = 8) and utilizes almost 26% fewer area comparing with the best present architectures [25].

11. ***Basant K. Mohanty et al (2013)***: A block LMS is implemented in this paper on the basis of an effective distributed-arithmetic (DA). The look up tables are shared in this DA based scheme for the calculation of filter outputs and to update the weights of BLMS algorithm. Since, DA consists of larger number of addresses. This method helps to save the addresses. Also, a scheme is presented for the weight update process in which only one set of look up tables will be updated from the whole sets of look up tables at every iteration. A parallel architecture has been derived on the basis proposed DA formulation; to implement of BLMS adaptive filter. The proposed DA-based LMS consists of L/6 more adders and L number of look up tables giving a throughput is more as compared with the good DA LMS adaptive filter in existence. The present structures need shifter but the method proposed in the paper does not require any shifter but the capacity to use flip-flops is increased by 25%. The main advantage of the method is that the adder count does not increase with block size; and the flip-flop count does not depend on block-size. This helps in reducing the area delay [27].

12. ***Sang Yoon Park et al (2013):*** Distributed arithmetic is used to implement a pipelined novel architecture which gives less power consumption, less area utilization for the adaptive filter and an increased throughput. Method consists of updating the look up tables parallel and simultaneous weight update and filtering operations. This results in the increase of the throughput. A conditional signal accumulation with a carry saver is used instead of the conventional adder-shifter computation of DA to calculate the inner product. This helps in the reduction of sampling time and area complications. The system uses a fast clock for carry-save but a lower clock for remaining operations

to reduce the power usage. Comparing with the present DA based structures; the proposed scheme uses equal number of multiplexors, minor LUT, and almost half of the adders. The results show that the area delay product is low and the power consumption is also less with comparison to the previous DA based adaptive filters [26].

13. ***Soon-Won Kwon et al (2013):*** An equalizer with multiple inputs and multiple output is designed using variable-precision distributed arithmetic (VPDA) for size reduction and to decrease the dynamic power of optical communication receivers of 112 Gb/s dual-polarization quadrature phase-shift-keying (DP-QPSK) for metro applications. A least mean square algorithm is used to reimburse the VPDA MIMO equalizer for channel dispersion and analog to digital converters with many non-idealities of a time dependent successive approximation register (SAR). This results in the designing of reduced area analog architectures. Also, the VPDA MIMO equalizer reduces the power consumption by 45% with respect to fixed resolution counterparts by using the minimal needed resolution to equalize every dispersed symbol [27].

14. ***Harpreet Singh et al (2014):*** This paper emphasizes on the performance of least mean square adaptive filters based on distributed arithmetic by analyzing operating speed, usage of area and memory need. High throughput is achieved with DA technique and fewer resources are utilized. Multipliers are reduced with look up tables to compute the convolution operation. The throughput of the system and resource usage is associated with the tap count in the Distributed arithmetic base unit. DA needs updating of the memory contents and to make this process fast & to lower the memory utilization, these DA base units are important. Memory usage and speed of operation of the filter is analyzed in this paper and their effect on tap count of the base unit is also analyzed. [28].

15. ***T. Pitchaiah et al (2014):*** A pipelined architecture for adaptive filter using DA is presented on the basis of low-area and efficiency in delay on FPGA. Area and power is consumed more in the implementation of Adaptive FIR Filter because they need many multipliers and adders. Shifters and adders are used by the Distributed Arithmetic (DA) technique which substitutes the MAC operations with bit serial behaviour look up tables. The delay efficient adaptive filter can be

designed using DA. DA results in less expensive implementation of the adaptive filter. FPGA resource are analyzed in this paper and studied the synthesis report of Adaptive filter by MAC operations as well as adaptive FIR filter built with DA [29].

16. ***M Surya Prakash et al (2015):*** A Sign-LMS using Distributed Arithmetic (DA) based scheme is implemented in this paper. Inner product of two vectors can be calculated efficiently with DA. The precomputed partial products and their sum of combination are stored in memories which compute the output by shift-accumulation. DA can be easily used for the designing of the finite impulse response (FIR) filters; but in the adaptive filters, the weights are to be updated at each iteration. A memory is used to store these partial products. This memory is updated time to time. The proposed method has a convergence behaviour equivalent to the multiply-and-accumulate (MAC) operation based implementation. Also, the throughput of the Distributed Arithmetic based adaptive filter implementation is superior than the MAC based adaptive filter implementation. It is also noticed that the throughput does not change in accordance with the filter order. Hence, this scheme is good for the implementation of large order filters [30].

CHAPTER 3
DISTRIBUTED ARITHMETIC

3.1 Introduction:

Distributed arithmetic is so called because the arithmetic operations are distributed unclearly. They are not “lumped” in a adequately usual fashion (multipliers are present over there). Digital signal processing is most common computing form in DSP. This computation can be executed easily by DA also. We are motivated to use distributed arithmetic is because of its intense computational efficiency [8]. DA is mostly helpful are in circuit designing but hardware can also be shaped adequately to imply Distributed arithmetic. A careful designing can decrease the gate count in signal processing arithmetic to 50% and up to 80% also.

The Distributed arithmetic is an explicit method for operations including sum of products. This can be done by pre-computing partial products by difference equation and store them in look-up table present in memory. The memory addresses are defined by input signals. To calculate the product, scale-accumulate the partial products present in the memory. Hence, multipliers are not wanted for this method. A multiplier less digital filter can be achieved. Distributed arithmetic is bit serial in nature. The inner product of \two vectors can be calculated within a single step. DA is highly efficient in terms of mechanization. Only disadvantage is has is its slowness since it is bit serial in nature [8]. However, this disadvantage is unreal if each vector has its element proportionate to the number of bits present in every vector. It can be explained with an example that if we input 8 words serially i.e. at the same time on 8 wires and on the other hand 8-bit words are input in parallel, the time required to input the data in both the cases will be same. A parallelism can be introduced to increase the speed. This can be achieved by pairing or portioning the bits of the input word into halves i.e. most significant half or least significant half and these halves into further halves, etc.

3.2 Derivation

A sum of product between two vectors can be given by:

$$y = \sum_{k-1}^{k} A_k X_K \tag{3.1}$$

Where A_k represent fixed coefficients and input words are represented by X_K.

Expressing X_K in 2's complement form, we get:

$$X_K = \quad b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \tag{3.2}$$

Where b_{kn} represents bits 0 or 1, b_{k0} is symbolized as sign bit and $b_{k,\,N-1}$ is LSB.

Putting value of X_K from equation 3.2 to equation 3.1 we get y as:

$$y = \sum_{k=1}^{k} A_k \left[\quad b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right] \tag{3.3}$$

The conventional type of computing inner product is given in equation 3.3. However, interchanging the summations, we get:

$$y = \sum_{k=1}^{k} A_k b_{kn} \tag{3.4}$$

Equation 3.4 will have 2^k possible values because b_{kn} can take value only 0 or 1. These values can be precomputed and stored in memory, instead of calculating them in each step. The memory address can be recognized by the input data and the result of equation 3.4 can be given to the accumulator [23]. The actual result y will be present in memory after N iterations. The configuration is shown in figure 7 and the memory storage is present in table.

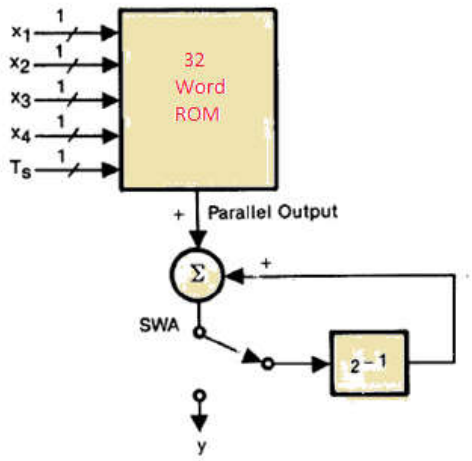

Figure 7: Adder and Subtractor with full memory

	Input code					32- word Memory content
	T_S	b_{1n}	b_{2n}	b_{3n}	b_{4n}	
1<n<N-1	0	0	0	0	0	0
	0	0	0	0	1	A_4
	0	0	0	1	0	A_3
	0	0	0	1	1	$A_3 + A_4$
	0	0	1	0	0	A_2
	0	0	1	0	1	$A_2 + A_4$
	0	0	1	1	0	$A_2 + A_3$
	0	0	1	1	1	$A_2 + A_3 + A_4$
	0	1	0	0	0	A_1
	0	1	0	0	1	$A_1 + A_4$
	0	1	0	1	0	$A_1 + A_3$
	0	1	0	1	1	$A_1 + A_3 + A_4$
	0	1	1	0	0	$A_1 + A_2$
	0	1	1	0	1	$A_1 + A_2 + A_4$
	0	1	1	1	0	$A_1 + A_2 + A_3$
	0	1	1	1	1	$A_1 + A_2 + A_3 + A_4$
n=0	1	0	0	0	0	0
	1	0	0	0	1	$-A_4$
	1	0	0	1	0	$-A_3$
	1	0	0	1	1	$-(A_3 + A_4)$
	1	0	1	0	0	$-A_2$
	1	0	1	0	1	$-(A_2 + A_4)$
	1	0	1	1	0	$-(A_2 + A_3)$
	1	0	1	1	1	$-(A_2 + A_3 + A_4)$
	1	1	0	0	0	$-A_1$
	1	1	0	0	1	$-(A_1 + A_4)$
	1	1	0	1	0	$-(A_1 + A_3)$
	1	1	0	1	1	$-(A_1 + A_3 + A_4)$
	1	1	1	0	0	$-(A_1 + A_2)$
	1	1	1	0	1	$-(A_1 + A_2 + A_4)$
	1	1	1	1	0	$-(A_1 + A_2 + A_3)$
	1	1	1	1	1	$-(A_1 + A_2 + A_3 + A_4)$

Table 7: Storage in 32-word ROM

The memory size having $2(2^k)$ words can be reduced to half by customizing the adder with a subtractor or adder and giving a control line T_s to the adder/subtractor i.e. in figure 8. 32-word configuration can now be shown with 16-word ROM [17]. The memory storage table for this configuration is equal to the upper half of table. However, the memory size can be reduced to half again. To explain this, consider data in offset binary coding from -1 to 1.

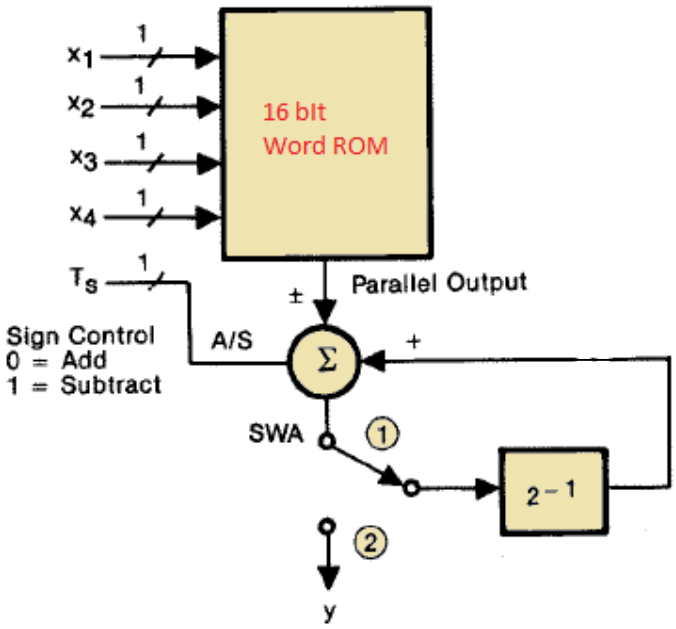

Figure 8. Adder/Subtractor with half memory

In offset binary coding, the X_k is given as:

$$X_k = \frac{1}{2}[X_k \quad (\ X_k)] \tag{3.5}$$

And X_k in 2's complement form is given as:

$$X_k = \overline{b_{k0}} + \sum_{n=1}^{N-1} \overline{b_{kn}} 2^{-n} + 2^{-(N-1)} \tag{3.6}$$

where the bar on the bits shows the complement of these bits. Putting the values of equation 3.2 and 3.6 in equation 3.5, we get:

$$X_k = \frac{1}{2}\left[\ (b_{k0} \quad \bar{b}_{k0)} + \sum_{n=1}^{N-1} (b_{kn} \quad \bar{b}_{kn}) 2^{-n} \quad 2^{-(N-1)} \right] \tag{3.7}$$

Defining new variable so as to simplify equation 3.7

$$c_{kn} = b_{kn} \quad \bar{b}_{kn} \qquad\qquad n \neq 0 \tag{3.8}$$

$$c_{k0} = \ (b_{k0} \quad \bar{b}_{k0}) \tag{3.9}$$

Where c_{kn} can take values± 1, containing n = 0 also.

Putting values of c_{kn} and c_{k0} in equation 3.7, we get:

$$X_k = \frac{1}{2}\left[\sum_{n=0}^{N-1} c_{kn}2^{-n} \quad 2^{-(N-1)}\right] \tag{3.10}$$

Substitute value of X_k from equation 3.10 into equation 3.1, we get:

$$y = \frac{1}{2}\sum_{k-1}^{k} A_k\left[\sum_{n=0}^{N-1} c_{kn}2^{-n} \quad 2^{-(N-1)}\right] = \sum_{n=0}^{N-1} Q(b_n)2^{-n} + 2^{-(N+1)}Q(0) \tag{3.11}$$

Where

$$Q(b_n) = \sum_{k=1}^{k} \frac{A_k}{2} c_{kn} \; and \; Q(0) = \sum_{k=1}^{k} \frac{A_k}{2} \tag{3.12}$$

$Q(b_n)$ can take $2^{(k-1)}$ values. Rapid amalgam of bits decides the sign of the amplitude of $Q(b_n)$. Y is computed by using $2^{-(k-1)}$ word ROM. Q(0) is stored in a one-word register. Also, depending upon the necessity of control-logic gates, adder/subtractor is also need for the computation of the y. An 8-bit word memory example is shown in figure 9, containing$Q(b_n)$.

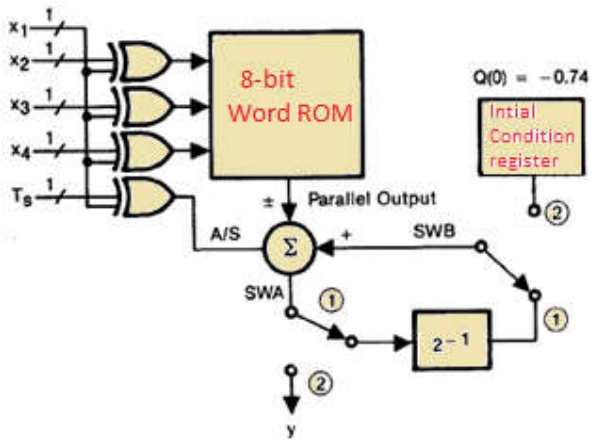

Figure 9. Adder/Subtractor with reduced memory

Input code				8- word Memory content, Q
b_{1n}	b_{2n}	b_{3n}	b_{4n}	
0	0	0	0	$-1/2(A_1 + A_2 + A_3 + A_4)$
0	0	0	1	$-1/2(A_1 + A_2 + A_3 - A_4)$
0	0	1	0	$-1/2(A_1 + A_2 - A_3 + A_4)$
0	0	1	1	$-1/2(A_1 + A_2 - A_3 - A_4)$
0	1	0	0	$-1/2(A_1 - A_2 + A_3 + A_4)$
0	1	0	1	$-1/2(A_1 - A_2 + A_3 - A_4)$
0	1	1	0	$-1/2(A_1 - A_2 - A_3 + A_4)$
0	1	1	1	$-1/2(A_1 - A_2 - A_3 - A_4)$
1	0	0	0	$1/2(A_1 - A_2 - A_3 - A_4)$
1	0	0	1	$1/2(A_1 - A_2 - A_3 + A_4)$
1	0	1	0	$1/2(A_1 - A_2 + A_3 - A_4)$
1	0	1	1	$1/2(A_1 - A_2 + A_3 + A_4)$
1	1	0	0	$1/2(A_1 + A_2 - A_3 - A_4)$
1	1	0	1	$1/2(A_1 + A_2 - A_3 + A_4)$
1	1	1	0	$1/2(A_1 + A_2 + A_3 - A_4)$
1	1	1	1	$1/2(A_1 + A_2 + A_3 + A_4)$

Table 8: Storage in 8-word ROM

It can be noted that the upper half of table--- is same with the lower half with inversed signs. Paying attention to the left side column, we can see that exoring b_{1n} with $b_{2n,}$ $b_{3n,}$ b_{4n}, we can get proper values of Q by pulling them correctly from the address book, and the sign is accepted. However, to get a correct sign, use b_{1n} as an address line for adder/subtractor. Inversion of the add/subtract command is done at the time of sign-bit. For the proper derivation of the add/subtract control line signal, we should apply EXOR gate for the combination of T_s and b_{in} signals.

The memory which stores the value of Q(0) is shown in figure 10. Q(0) must correct the values in ROM which are addressed by the LSBs of X_k via switch swb. Swa switch is synchronous with swb switch [22]. This antiquity can be visualized in equation 3.11. Values which are present in the active session of ROM are added with the previous shifted sum. There is a delay of clock period and switch swa and swb are switched to position 2 for the time period when $T_s = 1$. $Q(b_{N-1})$ is the 1^{st} output from the memory at first clock cycle which is added with Q(0). Similarly, at the time of second clock cycle the $Q(b_{N-1})$ is shifted to right and is added to second output $Q(b_{N-2})$ to compute $Q(b_{N-2}) + [Q(b_{N-1}) + Q(0)]2^{-1}$. The right shifting of the previous computed output is again done at time of third clock cycle so as

to compute $Q(b_{N-3}) + Q(b_{N-2})2^{-1} + [Q(b_{N-2}) + Q(0)]2^{-2}$. This is repeated upto Nth clock cycle untill we sum Q(b_0) to the already calculated and right shited previous quantity to compute $Q(b_0) + Q(b_1)2^{-1} + Q(b_2)2^{-2} + \quad + Q(b_{N-2})2^{-(N-2)} + [Q(b_{N-1}) + Q(b_0)]2^{2-(N-1)}$.

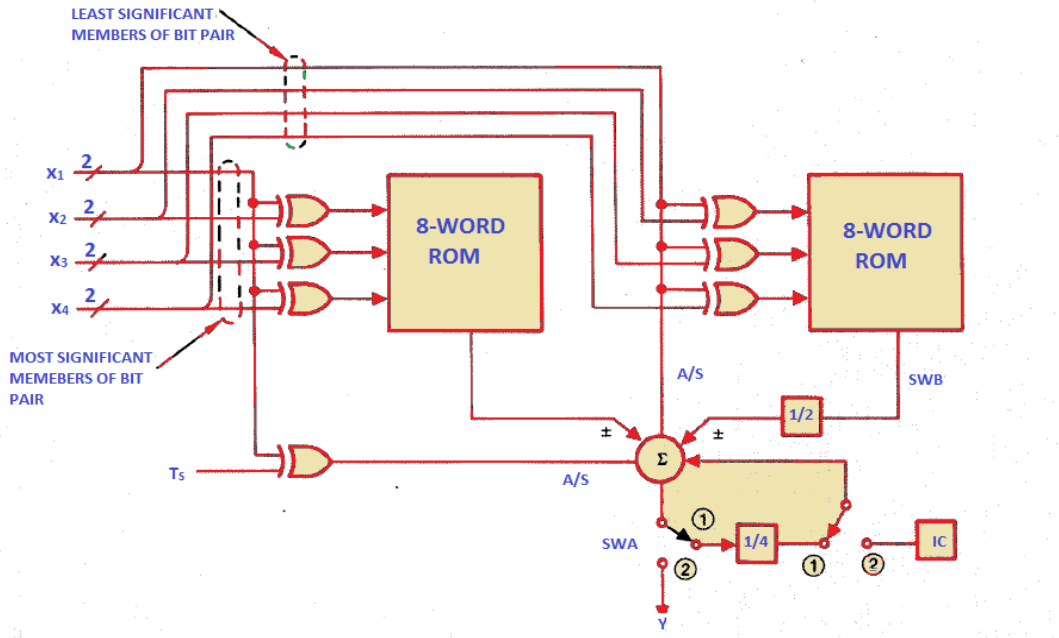

Figure 10. Multi-memory structure

3.2 Increasing the speed of distributed arithmetic:

It can be seen that if we input the data serially, it will result in slow connections. For N-bit input words, we need N-clock periods to compute the dot product. To make a DA processor fast, equivalent k separate products are formed and k is made greater than N. In this way, DA processor will be speedy than a parallel MAC. To add on the speed of distributed arithmetic, two methods can be used:

1. Compromise can be done with by increasing memory utilization and arithmetic operations to increase the speed.
2. Memory can also be increased exponentially.

If we divide the every input word in L parts then the speed can be incremented by L factor. L should be decided such that it is an N's factor [22]. We can use L number of memories and a higher capacity accumulator to combine the results of whole memories. Instead of using

multiple memories if we a single memory then the word capacity will increase by $\frac{1}{2}2^{kL}$ and the word length grows by log_2 bits. Let's explain these approaches with the means of an example. To allow parallelism in equation 3.11, computation should be simple. Break the summation over n into two parts, first is from 0 to (N/L) – 1 and second is from 0 to L-1. Hence, the equation 3.11 is rewritten as:

$$y = 2^{-(N-L)}P_{IC} + \sum_{n=0}^{\left(\frac{N}{L}\right)-1} P(b_k)2^{nL} \tag{3.13}$$

Where

$$P(b_k) = \sum_{k=1}^{k}\sum_{l=0}^{L}\frac{1}{2}A_k 2^{-l}C_{k,nL+1} \tag{3.14}$$

&

$$P_{IC} = 2^{-L}\sum_{k=1}^{k}A_k \tag{3.15}$$

In the previous case, N clock cycles were needed but here N/L clock cycles are used to complete inner product. The total bits which are to be loaded are Nk and the required number of clock cycles to check these bits is N/L. The input lines' number is:

$$p = \frac{total\ bits\ in}{number\ of\ clock\ periods} = \frac{Nk}{N/L} = kL \tag{3.16}$$

Relative cost is given as:

$$w = \frac{N/L}{kL} = \frac{N}{kL^2} \tag{3.17}$$

Number of bits which are to be loaded, L is:

$$L = \left\lceil \sqrt{\frac{N}{wk}} \right\rceil \tag{3.18}$$

These bracket shows to round up the next integer. DA will be more effective when number of input lines is equal to the number of clock required to fill the data. DA is an effective way to compute such equations which are influenced by inner products. Whenever we have customizable designs then DA should be used as a good option.

CHAPTER 4
LMS USING DISTRIBUTED ARITHMETIC

There are many uses of adaptive filters in various applications such as a noise canceller, system identification, etc, and the need for their implementations is increasing in almost each field. Adaptive filters needs many performance factors. These factors include high speed, power efficiency, better convergence, less latency in output, etc. For example, there should be a good tracking mechanism of impulse response in case of video conference, so a good echo-is required with a fast convergence. So, adaptive filters with higher orders should be implemented. [1] To fulfil such requirements, a very efficient algorithm and architecture is wanted [8]. Multipliers, memories, adder/subtractor are needed in the implementation of adaptive filter. We have proposed the structures with fast convergence and less complexity.

The LMS adaptive filter built with distributed arithmetic can be implemented by utilizing adders/subtractors and memories with less multipliers or without mulitpliers, so that a small hardware can be achieved. A Distributed Arithmetic (DA) is a very good method to calculate an inner product of constant vectors, and it is also used in the realization of DCT. It also fits for time varying weight vectors in the adaptive filter. Least Mean Square (LMS) adaptive filter with the help of DA by using offset binary coding is explained below. Convergence speed is degraded because of this LMS filter designing method [8]. However, the sign and sign-sign LMS adaptive filters are proposed with the help of this derivation and they operate on almost same convergence speed by utilizing less iteration.

4.1 Standard LMS using distributed arithmetic:

Below is shown the explained derivation of Least Mean Square (LMS) algorithm using the Distributed Arithmetic. The inputs are represented in 2's complement form [24]. A Nth order adaptive filter is considered with input vector S(k) and N-taps weights vector W(k) as shown:

$$S(K) = [s(k), s(k \quad 1), \dots, s(k \quad N+1)] \tag{4.1}$$

and

$$W(K) = [w_0(k),\ w_1(k), \dots,\ w_{N-1}(k) \tag{4.2}$$

The adaptive filter ouput is given as:

$$Y(K) = S^T(K)W(K) = A^T(K)F^TW(K) \tag{4.3}$$

Where A(K) is the address matrix and F is the scaling factor, defined as:

$$A(K) = \begin{bmatrix} b_0(k) & b_0(k\quad 1) & & b_0(k\quad N+1) \\ b_1(k) & b_1(k\quad 1) & \dots & b_1(k\quad N+1) \\ & & & \\ b_{N-1}(k) & b_{N-1}(k\quad 1) & & b_{N-1}(k\quad N+1) \end{bmatrix} \tag{4.4}$$

And

$$F = \begin{bmatrix} 2^0, 2^1, 2^2, \dots, 2^{-(B-1)} \end{bmatrix} \tag{4.5}$$

where B shows the input signal length.

The input signal and address matrix are related as:

$$s(k) = [b_0(k), b_1(k), b_2(k), b_{B-1}(k)] \tag{4.6}$$

Address vector can be defined from equation 4.4 as:

$$A_i(K) = [b_i(k), b_i(k\quad 1), b_i(k\quad 2), \dots, b_i(k\quad N+1)]^T \tag{4.7}$$

i = 0, 1, 2,..., B-1

The filter update equation is given as:

$$W(K+1) = W(K) + 2\mu e(K)S(K) \tag{4.8}$$

Where e(k) is the error computed by comparing the desired signal d(k) and output signal y(k). Multiplying both sides of equation 4.8 with $A^T(k)$:

$$W(K+1)A^T(k) = W(K) + 2\mu e(K)S(K)\, A^T(k) \tag{4.9}$$

The error signal is given as:

$$e(k) = d(k)\quad y(k) \tag{4.10}$$

Where d(k) is the desired signal.

The adaptive function space is given as:

$$P(k) = A^T(k)W(k) \tag{4.11}$$

And

$$P(k+1) = A^T(k)W(k+1) \tag{4.12}$$

Partial products having relation with $A_i(k)$ are the i^{th} elements of vectors **P**(k) and **P**(k+1). $\mathbf{A_i}(k)$ represents the i^{th} row vector of address matrix $A^T(k)$ [26]. The time index of **P**(k) is analogous to W(k) and **P**(k+1) is analogous to **W**(k+1). An N order input signal will have 2N partial products. B address vectors will select B elements included by **P**(k) and **P**(k+1).

By putting the values of Equation (4.11) and Equation (4.12) to Equation (4.9), we get:

$$P(k+1) = P(k) + 2\mu e(k)A^T(k)A(k)F \tag{4.13}$$

Hence, the output of the filter will be:

$$y = F^T P(k) \tag{4.14}$$

The expectation value of $A^T(k)A(k)F$ can be shown as:

$$E[A^T(k)A(k)F\ ] = 0.25NF \tag{4.15}$$

Replacing $A^T(k)A(k)F$ in equation 4.13 by 0.25NF, we get:

$$P(k+1) = P(k) + 0.5\mu Ne(k)F \tag{4.16}$$

The value 0.5NF in Equation (4.16) can be used as a constant value in the form of integer power of two. Hence, a multiplier less hardware is possible [24]. Therefore, multiplierless adaptive filter can be designed with LMS adaptive algorithm.

The Figure 11 shows the fundamental structure. DA LMS adaptive filter can use calculation block in the fundamental structure, and ROM can be used to realize AFS.

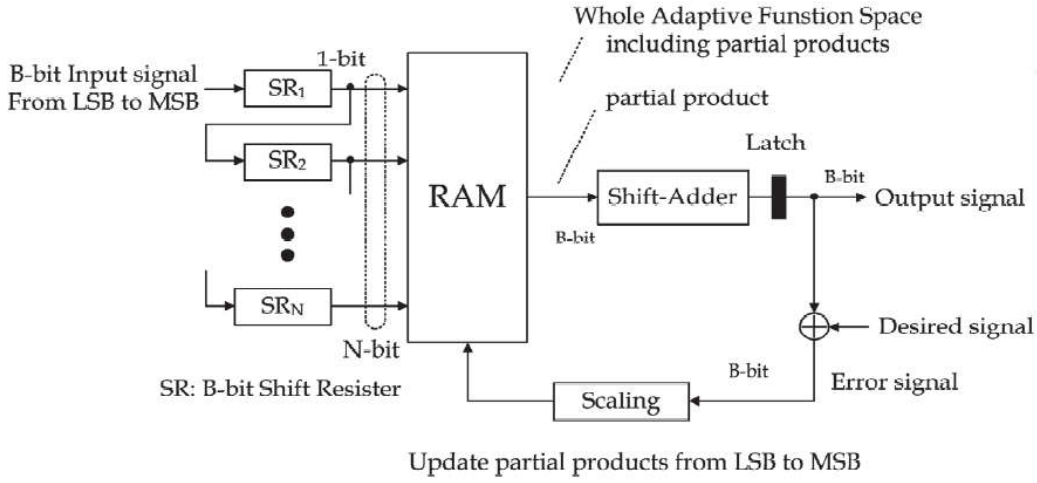

Figure 11. Fundamental structure of LMS algorithm using DA

Further derivations have been done for sign LMS adaptive filters by deriving their adaptive function spaces.

4.2 Sign LMS using distributed arithmetic:

4.2.1 Sign error LMS adaptive filter:

The weights of the sign error LMS are updates by using the following equation:

$$W(k+1) = W(k) + 2\mu sgn\big(e(k)\big)S(k) \tag{4.17}$$

Multiplying both sides of equation 4.17 with $A^T(k)$, we get:

$$W(k+1)A^T(k) = W(k) + 2\mu sgn\big(e(k)\big)S(k)\ A^T(k) \tag{4.18}$$

By substituting equation 4.3 in equation 4.17, we get

$$W(k+1)A^T(k) = W(k) + 2\mu sgn\big(e(k)\big)A(k)A^T(k)F \tag{4.19}$$

Rewriting equation 4.11 and 4.12,

$$P(k) = A^T(k)W(k) \tag{4.20}$$

And

$$P(k+1) = A^T(k)W(k+1) \tag{4.21}$$

These partial products can be used in equation 4.19 and this equation is updates as:

$$P(k+1) = P(k) + 2\ \mu sgn\big(e(k)\big)A^T(k)A(k)F \tag{4.22}$$

The expectation for $A^T(k)A(k)F$ is given in equation 4.15. Applying this expectation in equation 4.22, we get [24]:

$$P(k+1) = P(k) + 0.5\mu sgn\big(e(k)\big)NF \tag{4.23}$$

4.2.2 Sign data LMS adaptive filter:

The weights of the sign error LMS are updates by using the following equation:

$$W(k+1) = W(k) + 2\mu e(k)sgn(S(k)) \tag{4.24}$$

Multiplying both sides of equation 4..24 with $A^T(k)$, we get:

$$W(k+1)A^T(k) = W(k) + 2\mu e(k)sgn(A(k)F)A^T(k) \tag{4.25}$$

By substituting equation 4.3 in equation 4.24, we get

$$W(k+1)A^T(k) = W(k) + 2\mu e(k)sgn(A(k)F)A^T(k) \tag{4.26}$$

Rewriting equation 4.11 and 4.12,

$$P(k) = A^T(k)W(k) \tag{4.27}$$

And

$$P(k+1) = A^T(k)W(k+1) \tag{4.28}$$

These partial products can be used in equation 4.26 and this equation is updates as [24]:

$$P(k+1) = P(k) + 2\,\mu e(k)A^T(k)sgn(A(k)F) \tag{4.29}$$

The equation 4.29 shows the sign data LMS adaptive computation using distributed arithmetic.

4.2.3 Sign-Sign LMS adaptive filter:

The weights of the sign error LMS are updates by using the following equation:

$$W(k+1) = W(k) + 2\mu sgn(e(k))sgn(S(k)) \tag{4.30}$$

Multiplying both sides of equation 4.30 with $A^T(k)$, we get:

$$W(k+1)\mathrm{A^T(k)} = W(k) + 2\mu sgn(e(k))sgn(A(k)F)\mathrm{A^T(k)} \tag{4.31}$$

By substituting equation 4.3 in equation 4.30, we get

$$W(k+1)\mathrm{A^T(k)} = W(k) + 2\mu sgn(e(k))sgn(A(k)\mathrm{F})\mathrm{A^T(k)} \tag{4.32}$$

Rewriting equation 4.11 and 4.12,

$$P(k) = A^T(k)W(k) \tag{4.33}$$

And

$$P(k+1) = A^T(k)W(k+1) \tag{4.34}$$

These partial products can be used in equation 4.32 and this equation is updates as:

$$P(k+1) = P(k) + 2\,\mu sgn(e(k))A^T(k)sgn(A(k)F) \tag{4.35}$$

The equation 4.35 shows the sign-sign LMS adaptive computation using distributed arithmetic [24].

CHAPTER 5

ADAPTIVE NOISE CANCELLER USING LMS ALGORITHM WITH DISTRIBUTED ARITHMETIC

A noise canceller is designed in our work. This is an adaptive filter to electrocardiography for foetal signal. Here, we are separating maternal heartbeat from the sensor signal measuring the foetal heartbeat. First, maternal and foetal heartbeats are created. The heart beat rate of foetus is higher than the maternal heartbeat . The amplitude of the maternal heartbeat signal is larger than the amplitude of the foetal heartbeat signal.

The maternal heartbeat will be measured from the mother's chest. We are removing the maternal heartbeat signal adaptively from the foetal heartbeat signal. Since, the maternal heartbeat signal is stronger than the foetal heartbeat signal obtained from the abdomen; the maternal heartbeat signal is causing interference in the electrocardiogram of the foetus. So our desired signal is the foetal heartbeat signal.

As the adaptive noise canceller needs a reference signal, maternal heartbeat signal is the reference signal here.

The desired signal i.e. the foetal signal f(n) is the combination of foetal heartbeat signal f(n) and a noise $n_2(n)$ which is achieved from the interference maternal signal, corrupting the foetal signal. The input to the adaptive filter is interference signal $n_1(n)$ corrupted because of maternal signal. Both the noise signals are correlated. The output y(n) is the maternal signal. An error signal e(n) is generated by subtracting the output of the adaptive filter y(n) from the desired signal f(n). Filter weights are adjusted by this generated error so that the noise interference can be minimized. The computed error signal is similar to the desired signal in optimum cases, having minimized interference noise.

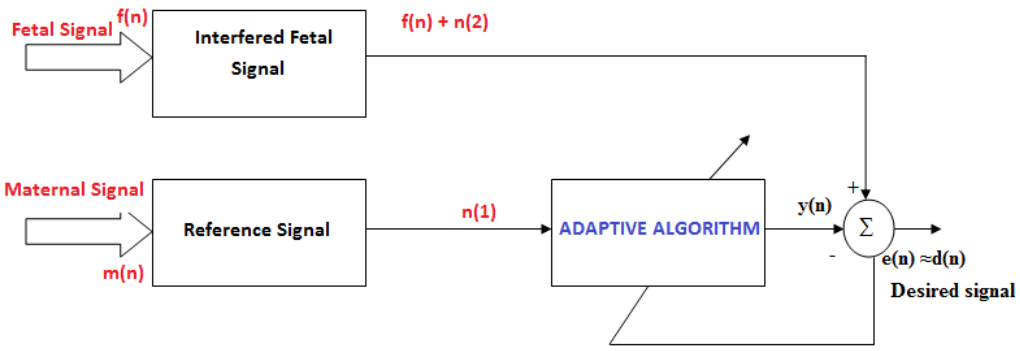

Figure 12. Maternal-foetal adaptive noise canceller

We are using the LMS adaptive filter having 15 coefficients and will analyze the results by changing the step size.

We are designing LMS filter with distributed arithmetic as well as with standard equations also. The following comparisons are made:

1. Comparison between the performance of Standard LMS adaptive filter and Standard LMS adaptive filter using distributed arithmetic.
2. Comparison between the performance of Sign-error LMS adaptive filter and Sign-error LMS adaptive filter using distributed arithmetic.
3. Comparison between the performance of sign-sign LMS adaptive filter and sign-sign LMS adaptive filter using distributed arithmetic.
4. Comparison between the performance of Standard LMS adaptive filter using distributed arithmetic and sign LMS adaptive filter using distributed.

We are achieving above results by choosing different-different values of the step size. The step size is varied to check the convergence rate and SNR of the signals being generated. These results will help us to analyze how the step size should be chosen for different order of filters. We are also checking at what iteration value, our filter is converging having different orders. The waveforms of the generated signals and comparison waveforms are shown.

5.1 Conventional LMS adaptive noise canceller:

The standard LMS plots are analyzed using MATLAB R2014a tool. Below are the figures showing the mean square relation with the iteration value to which the filter converges and the Signal to noise ratio plot with respect to iteration. Also, the plot for desired signal vs. obtained signal is shown. It is being noticed that the convergence of this filter is pretty good. But the filter is slow as it is taking many iterations to converge and gives delay. However the signal to noise ratio is decreasing as the number of iterations are increasing but becomes constant after the filter has converged. This filter is designed for a step size of '0.000001'. As the step size is decreasing, the signal to noise ratio is decreasing and obtained signal follows less to the desired signal. The foetal signal obtained is a good match of the desired signal but the iteration number is going 100000 plus.

5.2 Conventional LMS adaptive noise canceller using Distributed Arithmetic:

According to the distributed arithmetic and LMS adaptive filter using distributed arithmetic studied in chapter 3 and 4, a noise canceller with conventional LMS for removing maternal interference noise from the foetal noise adaptively has been designed. The MSE vs. Iteration

graph for conventional LMS adaptive filter using Distributed arithmetic is shown in figure 17 and figure 18 shows SNR vs. Iteration graph for conventional LMS adaptive filter using distributed arithmetic.

It is being noticed that the convergence speed of this design is quite fast. But the signal to noise ratio is low leading to the degradation of the obtained signal with respect to the desired signal.

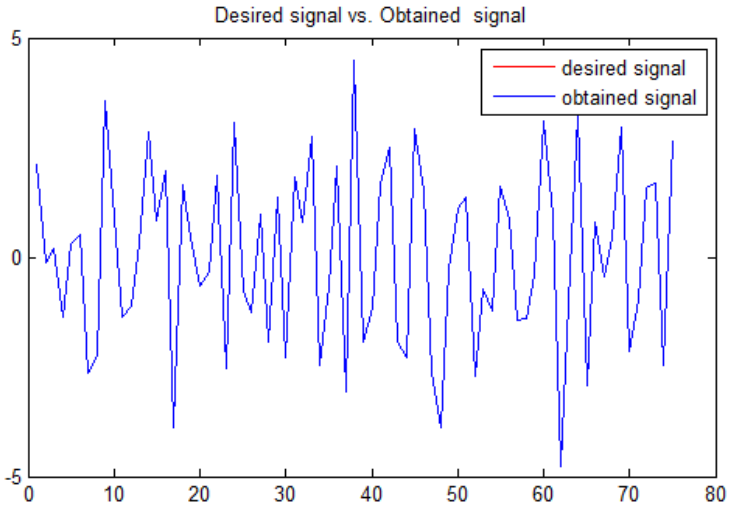

Figure 13: Conventional LMS desired vs. Obtained signal

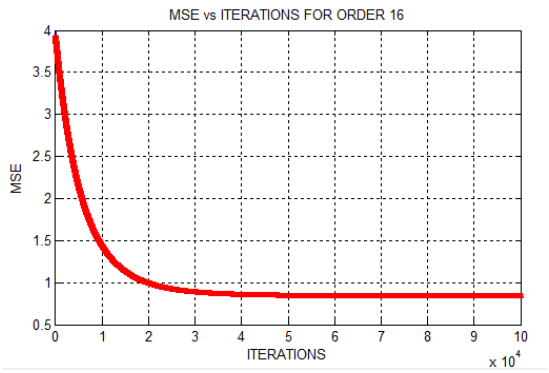

Figure 14: MSE vs. Iterations for conventional LMS

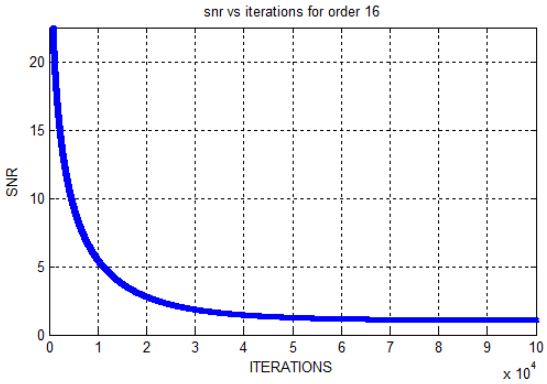

Figure 15: SNR vs. Iterations for conventional LMS

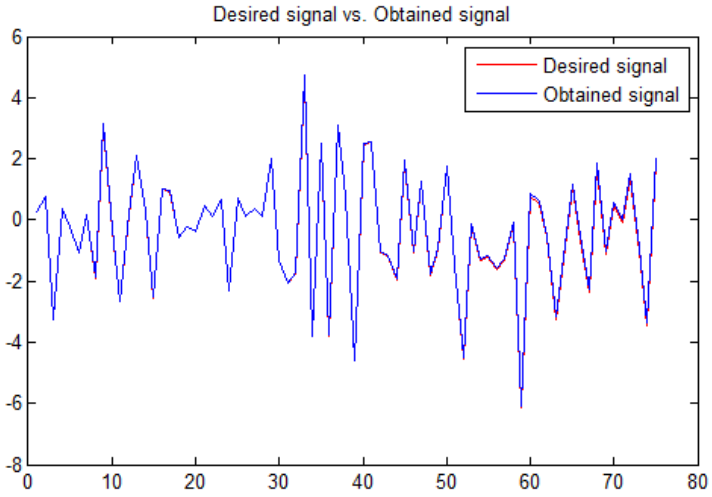

Figure 16: Conventional LMS using DA plot for Obtained signal vs. desired response

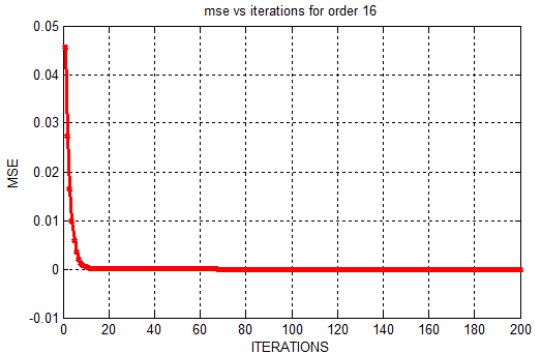

Figure 17: MSE vs. Iterations for conventional LMS using DA

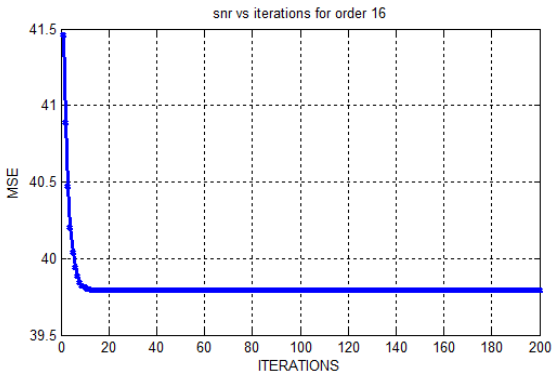

Figure 18: SNR vs. Iterations for conventional LMS using DA

5.3 Sign-error LMS adaptive noise canceller using Distributed Arithmetic:

Sign-error LMS is discussed in section 4.2. From figure 19, it is being seen that the obtained signal is a good match of desired signal but not excellent. The MSE vs. Iteration plot and SNR vs. Iteration plot are shown in figure 20 and 21 respectively.

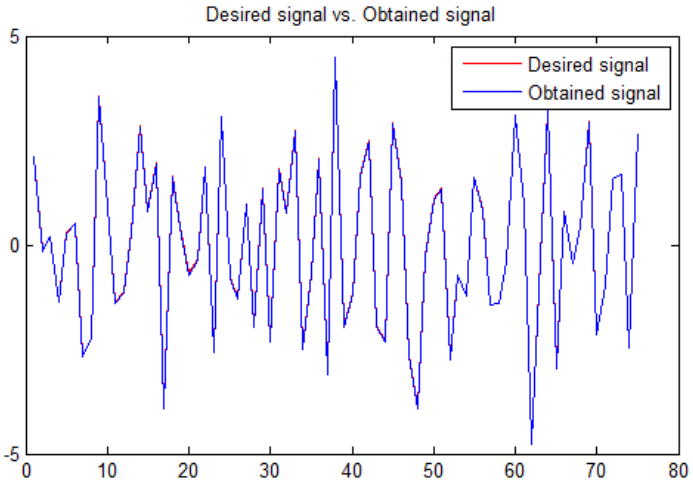

Figure 19: Sign-error LMS using DA plot for Obtained signal vs. Desired response

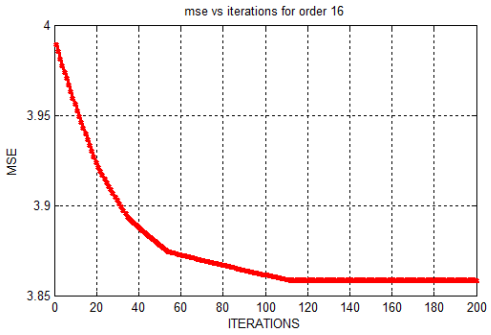

Figure 20: MSE vs. Iterations for sign-error LMS using DA

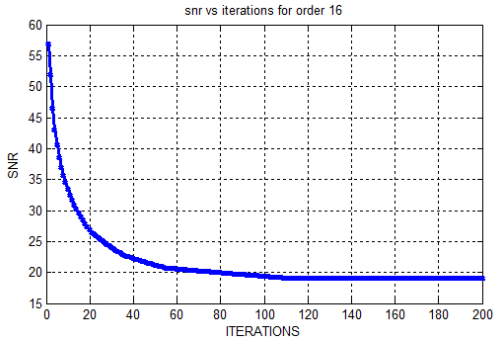

Figure 21: SNR vs. Iterations for sign-error LMS using DA

5.4 Sign-sign LMS adaptive noise canceller using Distributed Arithmetic:

The Desired signal vs. Obtained signal plot is shown in figure 22. The figure 23 shows MSE vs. Iterations plot and figur 24 shows SNR vs. Iterations graph. The obtained signal doesnot follow the desired signal correcly. MSE is high in case of this adaptive filter. Convergence is fair.

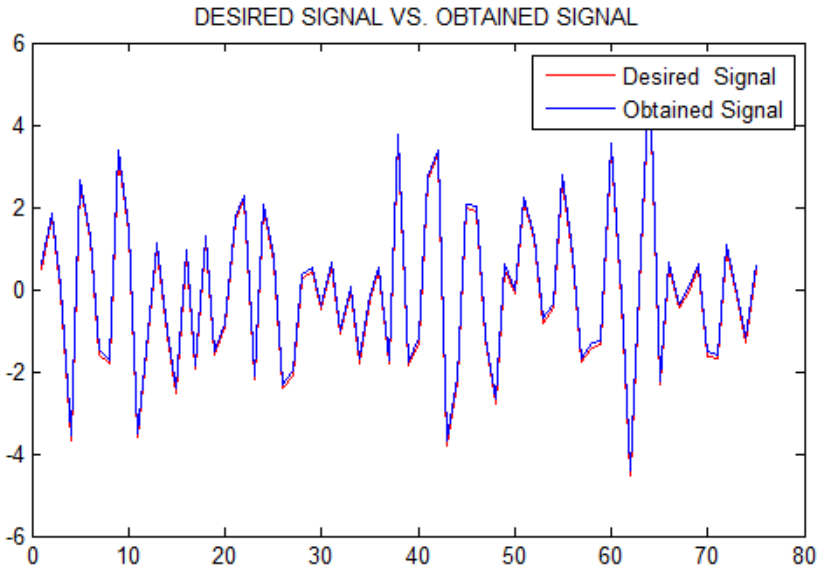

Figure 22: Sign-sign LMS using DA plot for Obtained signal vs. Desired response

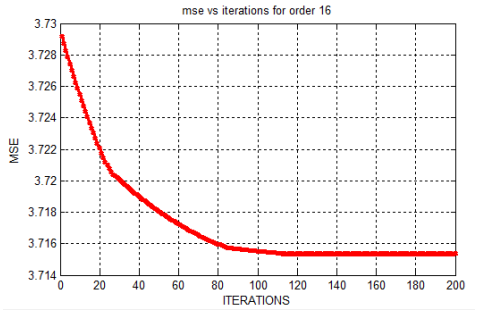

Figure 23: MSE vs. Iterations for sign-sign LMS using DA

Figure 24: SNR vs. Iterations for sign-sign LMS using DA

5.5 Comparison of Conventional LMS adaptive filter and Conventional LMS adaptive filter using Distributed arithmetic:

Figure 25 shows the plot of comparison done in terms of MSE vs. Iterations and figure 26 shows the comparison on the basis of SNR. It can be seen that the convergence rate of LMS adaptive noise canceller designed with distributed arithmetic is very-very high as compared to the convergence rate of normal Conventional LMS adaptive noise canceller. However, the convergence is increased with the trade-off between SNR and MSE. The SNR in case LMS adaptive filter with distributed arithmetic is low as compared to conventional LMS adaptive filter. Also, the mean square error is more in case of distributed arithmetic is high. It can be analyzed that there is a trade-off between convergence rate and signal degradation.

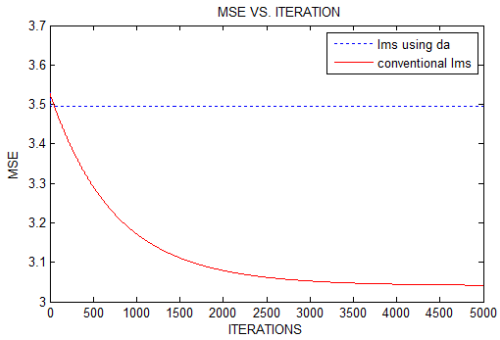

Figure 25: MSE vs. Iteration Graph for comparison of conventional LMS with and without DA

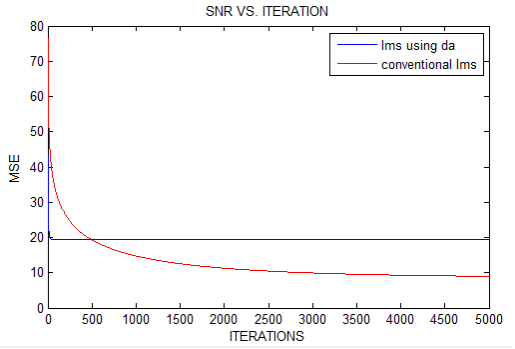

Figure 26: SNR vs. Iteration Graph for comparison of conventional LMS with and without DA

5.6 Comparison of sign error LMS adaptive noise canceller and sign error LMS adaptive noise canceller using distributed arithmetic:

The convergence of Sign-error LMS adaptive noise canceller is good as compared to the sign-error LMS adaptive noise canceller using distributed arithmetic. But the convergence is got with the trade-off in the time taken to converge the filter. Hence, from the graphs shown in figure 27 and 28; it can be seen that the signal quality is good in case of sign-error LMS built using MATLAB commands as the SNR is high but due to the delay, the sign-error LMS using distributed arithmetic can be chosen with the trade-off in signal quality. The convergence of the sign-error LMS using DA is quite satisfactory.

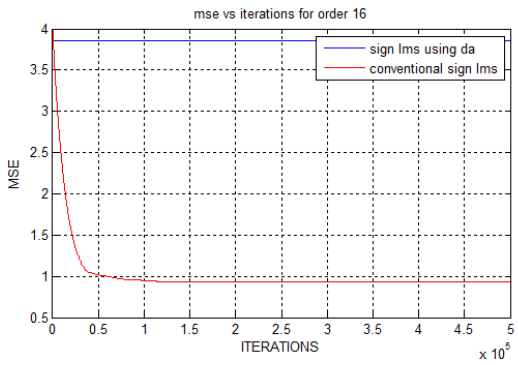

Figure 27: MSE vs. Iteration Graph for comparison of Sign-error LMS with and without DA

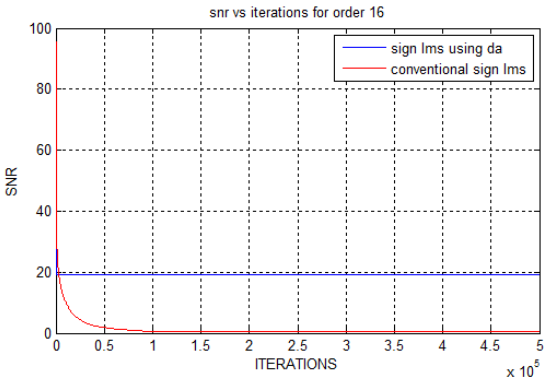

Figure 28: SNR vs. Iteration Graph for comparison of Sign-error with and without DA

5.7 Comparison of sign-sign LMS adaptive noise canceller and sign-sign LMS adaptive noise canceller using distributed arithmetic:

The sign-sign LMS adaptive noise canceller using distributed arithmetic is also very much fast as compared to the other one. The convergence rate is fast. But in case of distributed arithmetic, the Obtained signal is degraded with respect to desired signal because of decreasing SNR in distributed arithmetic. The comparison graph on the basis of MSE vs. Iterations as well as SNR vs. Iterations is shown in figure 29 and figure 30 respectively.

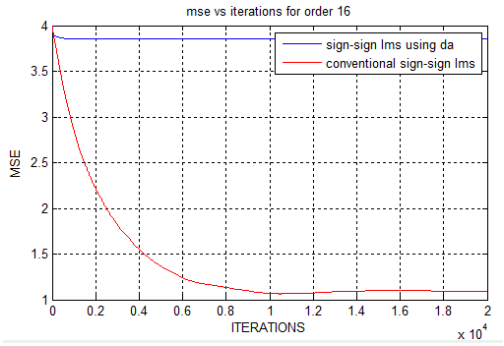

Figure 29: MSE vs. Iteration Graph for comparison of Sign-Sign LMS with and without DA

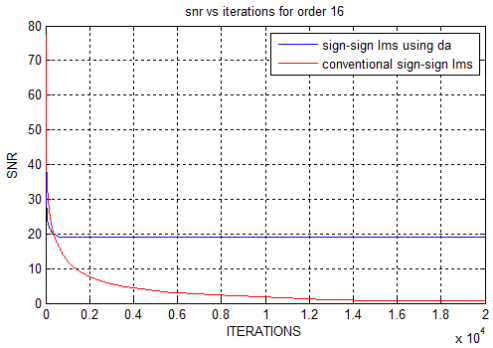

Figure 30: SNR vs. Iteration Graph for comparison of Sign-Sign with and without DA

5.8 Comparison of the Conventional LMS adaptive noise canceller using Distributed arithmetic and sign LMS adaptive noise canceller using distributed arithmetic:

The plots of comparison of Conventional LMS filter using DA and Sign error LMS using DA on the basis of MSE and SNR are shown in figure 31 and figure 32 respectively. Table 9 shows the behaviour of Conventional LMS adaptive algorithm using distributed arithmetic with respect to the order and tells how the step size is affecting the convergence and SNR of the filter. It also helps in approximating the step size value for a particular order. It is being seen that the MSE of both the conventional LMS and sign-error LMS adaptive filter using DA is similar. Table 10 shows the comparison between these filters with a step size of '0.000001'. The values in the table 10 show that SNR of both of the filters for same order is approximately same. But, there is a huge difference between the iteration values of the filters at which they converge. For example, the iteration value at which the conventional LMS adaptive converges for order 4 is 565 but it is 78 for sign-error LMS adaptive filter. Sign-error LMS adaptive filter can be chosen over the conventional adaptive filter because both have similar results. Also, paying attention to the affect of step size on the filter, we analyzed

that the convergence of the filter increases as the step-size decreases and decreases with the increasing step size. The convergence is getting fast with the increase in the filter order.

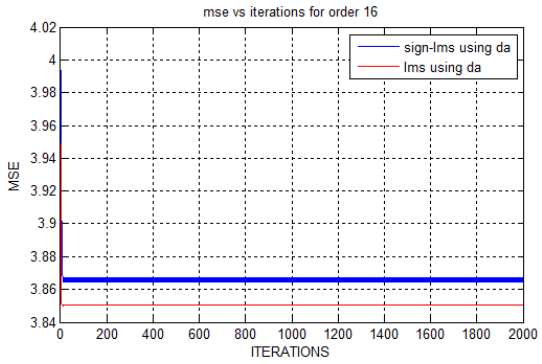

Figure 31: MSE vs. Iteration Graph for comparison of conventional LMS and sign-error LMS using DA

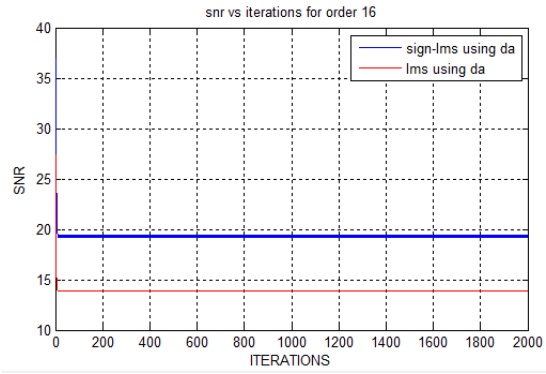

Figure 32: SNR vs. Iteration Graph for comparison of conventional LMS and sign-error LMS using DA

ORDER	STEP SIZE = 0.00001		STEP SIZE = 0.000001	
	Iteration	SNR	Iteration	SNR
4	57	43.225	565	43.6845
8	40	42.0734	282	43,6699
10	30	41.3725	226	43.6589
16	25	39.2110	141	43.6119
18	22	38.5255	126	43.5913
20	20	37.8698	113	43.5688
24	18	36.6511	94	43.5168
26	17	36.0869	87	43.4875
28	17	35.5511	81	43.4559
30	14	35.0410	75	43.4228
40	8	32.8309	57	43.2255
50	7	31.0546	45	42.9854
60	7	29.5853	38	42.7078
70	5	28.3422	32	42.4020
80	4	27.2705	28	42.0734
90	4	26.3334	25	41.7285

100	3	25.5042	23	41.3725

Table 9: Conventional LMS using DA observations

ORDER	STEP SIZE = 0.000001			
	Conventional LMS using DA		Sign LMS using DA	
	Iteration	SNR	Iteration	SNR
4	565	43.6845	78	46.3803
8	282	43,6699	39	43.5873
10	226	43.6589	31	43.5942
16	141	43.6119	19	43.6095
18	126	43.5913	17	43.5046
20	113	43.5688	15	43.6153
24	94	43.5168	13	43.1934
26	87	43.4875	12	43.1421
28	81	43.4559	11	43.1931
30	75	43.4228	10	43.3477
40	57	43.2255	8	42.5803
50	45	42.9854	6	42.7878
60	38	42.7078	5	42.5005
70	32	42.4020	4	42.6649
80	28	42.0734	4	41.5051
90	25	41.7285	3	42.2577
100	23	41.3725	3	41.3425

Table 10: Sign-error LMS using DA observations

5.9 Implementation requirement for LMS adaptive noise cancellers using DA:

The Table—shows the FPGA resource utilization by proposed LMS adaptive filters using distributed arithmetic.

Requirement	Conventional LMS using DA	Sign LMS using DA
Slice LUTs	140	213
Fully used LUTs FF pairs	280	365

B**onded IOBs**	204	204
DSP-48s	35	20

Table 11: Resource utilization

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

The Conventional and sign LMS adaptive filters have been studied and analyzed in chapter 4 and 5. Also, the resulting graphs are shown in chapter 5 and their comparison has been done with the filters implemented without distributed arithmetic.

6.1 Conclusion

The following conclusion has been made according to the analysis done:

1. The LMS adaptive filters have a good convergence but their convergence rate is very slow. They take thousands of iteration to converge leading the latency in the filter output. Fast adaptive filters are difficult to design. However, the SNR of the filter is excellent using LMS adaptive filters. If delay is not an issue in the application, LMS adaptive filter is a good option for the filter implementation. There is a trade-off between delay and mapping of obtained signal to the desired signal.

2. The LMS adaptive filters that have been designed using Distributed arithmetic have a very fast convergence rate as they take less number of iterations to converge. But, the SNR for this filter is somehow reduced as compared to the existing LMS adaptive filters. So there is trade-off signal to noise ratio and mean square error.

3. Conventional LMS adaptive noise canceller using distributed arithmetic is compared with the sign error LMS adaptive noise canceller to check the best from two. The results show that the sing-error LMS adaptive noise canceller is better than the conventional adaptive noise canceller because the convergence rate of this sign error LMS is very fast as compared to the conventional LMS. The observation has been shown in the table 10. The Mean square error for both implementations is similar. Hence, signal quality is not affected by choosing either design method.

4. The convergence is taking many iteration values for lower order filters with smaller step size and approximately 10% less with the higher value of step size for same lower order. The SNR for both the step size is similar. As the filter order is increasing, the iteration value for convergence is decreasing with the decrease in

SNR. Decreasing SNR leads to less mapping of the obtained signal with the desired signal. However, if we are decreasing the step size for higher order filters, the SNR is increased with increase in number of iterations. But the iteration count for these higher orders is similar to the iteration count of lower order filters with higher step-size value. It can be concluded that the lower order adaptive filters can be operated on higher step size value and higher order adaptive filters can be designed for lower value of step size. The analysis is made for adaptive filter designed using distributed arithmetic.

5. By studying table 10 , we observe that the sign-error LMS adaptive noise canceller is faster than the conventional LMS adaptive noise canceller using DA because the sign-error LMS is taking less number of iterations for convergence for step size '0.000001' whereas the conventional LMS adaptive noise canceller is taking almost 10% more iterations. The SNR for both the noise cancellers is almost same which shows that the matching of obtained signal with the desired signal is similar for both the schemes.

6. The implementation requirement for conventional and sign-error LMS adaptive noise canceller has been checked in Xilinx software and the table is presented which shows the resource utilization. It is analyzed from row 4 of table-- that the conventional LMS adaptive noise canceller has more arithmetic operations as compared to the sign-error LMS. But the sign-error LMS adaptive noise canceller utilizes more look up tables than conventional adaptive noise canceller.

7. By studying implementation of LMS adaptive noise canceller with different schemes, the sign-error LMS adaptive noise canceller using DA can be chosen best for removing maternal interference noise from the foetal heartbeat signal because this scheme is fast and have a good convergence because it can operate fast on lower step size also.

6.2 Future Scope:

- From the results analysed, it can be concluded that there is a great future scope for LMS Adaptive filter using distributed arithmetic.

- Higher radices techniques are still to be used for multiplication reduction in adaptive filters.
- Different multiplication techniques will be combined so as to reduce partial products.

REFERENCES

1. B. Widrow et al.. “Stationary and nonstationary learning characteristics of the LMS adaptive fiiter,” Proc. IEEE: vol. 64:pp. 1151-1162, Aug. 1976.
2. S. Haykin, Adaptive Filter Theory, Englewood Cliffs, NJ: Prentice-Hall, 1986
3. G. Long, F. Ling and J.G. Proakis, “The LMS Algorithm with Delayed Coefficient adaptation”, IEEE Trans. Acoust., Speech, Signal Process., vol. 37, no. 9, pp. 1397-1405, sept., 1989.
4. Y.C. Lim, J.B. Evans and B. Liu, “Decomposition of Binary Integers into Signed Power- of-Two Terms”, IEEE Trans. Circuits and Systems, vol. 38, no. 6, pp. 667 672, 1991.
5. Scott C. Douglas, “Exact Expectation Analysis of the LMS Adaptive Filter”, IEEE transactions on signal processing, vol. 43, no. 12, pp. 61-64, 1995
6. B. Farhang-Boroujeny, “Adaptive Filters: Theory and Applications”, Wiley, 1998
7. Y.C. Lim, R. Yang, D. Li and J. Song, “Signed power-of-two term allocation scheme for the design of digital filters”, IEEE Trans. Circuits and Systems, part II, vol. 46, no. 5, pp. 577-584, 1999.
8. K.K. Parhi, “VLSI Digital Signal Processing Systems - Design and Implementation”, John Wiley and Sons Inc., 1999.
9. Daniel J. Allred, Walter Huang, Venkatesh Krishnan, Heejong Yoo, and David V. Anderson, “An FPGA Implementation for a High Throughput Adaptive Filter Using Distributed Arithmetic”, Proc. IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 324 – 325, 2004.
10. Daniel J. Allred, Walter Huang, Venkatesh Krishnan, Heejong Yoo, and David V. Anderson, “LMS adaptive filter using distributed arithmetic for high throughput”, Proc. IEEE Transactions on Circuits and Systems, pp. 1327 –1337, 2005.
11. S. Chivapreecha, A. Jaruvarakul, N. Jaruvarakul, K. Dejhan, “Adaptive Equalization Architecture Using Distributed Arithmetic for Partial Response Channels”, Proc. IEEE International Symposium on Consumer Electronics, pp. 1-5,2006.
12. Walter Huang, Venkatesh Krishnan, and David V. Anderson, “Conjugate Distributed Arithmetic Adaptive FIR Filters and their Hardware Implementation”, proc. IEEE International Symposium on Circuits and Systems, pp. 295-299, 2006

13. Y.J. Yu and Y.C. Lim, “Design of Linear Phase FIR Filters in Sub expression Space Using Mixed Integer Linear Programming”, IEEE Trans. Circuits and Systems part I, vol. 54, no. 10, pp. 2330-2338, 2007.
14. F. Xu, C.H. Chang and C.C. Jong, “Design of Low-Complexity FIR Filters Based on Signed-Powers-of-Two Coefficients With Reusable Common Sub expressions”, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 10, pp. 1898-1907, 2007.
15. S.Y. Park and N.I. Cho, “Design of signed powers-of-two coefficient perfect reconstruction QMF Bank using CORDIC algorithms”, IEEE Trans. Circuits and Systems part I, vol. 53, no. 6, pp. 1254-1264, 2007
16. Fei Xu, Chip Hong Chang, and Ching Chuen Jong, “Design of Low-Complexity FIR Filters Based on Signed-Powers-of-Two Coefficients With Reusable Common Subexpressions”, IEEE Transactions on computer-aided design of integrated circuits and systems, vol. 26, no. 10, pp. 1898 – 1907, 2007
17. Z.G. Feng, K.L. Teo, “A Discrete Filled Function Method for the Design of FIR Filters With Signed-Powers-of-Two Coefficients”, IEEE Trans. Signal Processing, vol. 56, no. 1, pp. 134-139, 2008
18. M. Aktan., A. Yurdakul and G. Dundar, “An Algorithm for the Design of Low-Power Hardware-Efficient FIR Filters”, IEEE Trans. Circuits and Systems part I, vol. 55, no. 6, pp. 1536-1545, 2008.
19. Górriz, J.M.; Ramírez, J.; Cruces-Alvarez, S.; Puntonet, C.G.; Lang, E.W.; Erdogmus, D., “A Novel LMS Algorithm Applied to Adaptive Noise Cancellation”, IEEE Signal Processing Letters, VOL. 16, NO. 1, January 2009.
20. Daniel Efinger, Stefan Payer, Halmo Fischer, “A scalable and hardware-efficient architecture for digitally adaptive electronic dispersion compensation”, Proc. Asia Communications and Photonics conference and Exhibition, pp. 1-11, 2009.
21. Sunav Choudhary, Pritam Mukherjee, Mrityunjoy Chakraborty¤, “A SPT treatment to the Bit Serial Realization of the Sign-LMS based Adaptive Filter”, proc. IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2678 – 2681, 2010
22. Sudhanshu Baghel and Rafiahamed Shaik, “Low Power and Less Complex Implementation of Fast Block LMS Adaptive Filter Using Distributed Arithmetic”, IEEE Students' Technology Symposium (TechSym), pp. 214-219, 2011

23. Rui Guo and Linda S. DeBrunner, “A Novel Adaptive Filter Implementation Scheme Using Distributed Arithmetic”, Conference on Signals, Systems and Computers (ASILOMAR), pp. 160-164, 2011
24. Anirut Trakultritrung, Ekkawin Thanangchusin, Sorawat Chivapreecha, “Distributed Arithmetic LMS Adaptive Filter Implementation without Look-Up Table”, International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), pp. 1 – 4, 2012
25. M. Surya Prakash and Rafi Ahamed Shaik, “Low-Area and High-Throughput Architecture for an Adaptive Filter Using Distributed Arithmetic”, IEEE transactions on circuits and systems, vol. 60, no. 11, pp. 781 – 785, 2013
26. Sang Yoon Park, Pramod Kumar Meher, “Low-Power, High-Throughput, and Low-Area Adaptive FIR Filter Based on Distributed Arithmetic”, Proc. IEEE Circuits and Systems Society, Vol. 60, N0. 7, pp. 346-350, 2013.
27. Basant K. Mohant and Pramod Kumar Meher, “A High-Performance Energy-Efficient Architecture for FIR Adaptive Filter Based on New Distributed Arithmetic Formulation of Block LMS Algorithm”, IEEE transactions on signal processing, Vol. 61, NO. 4, pp. 921 – 932, 2013
28. Harpreet Singh, Gurinder Singh and Tarandip Singh, “Performance Analysis of DA Based Adaptive FIR Filter Using FPGA”, IEEE India Conference (INDICON), pp. 1 – 4, 2014.
29. T. Pitchaiah, Dhana Lakshmi M and P V Sree Devi, “FPGA implementation of low area and delay efficient Adaptive Filter using Distributed Arithmetic”, Proc. Advances in Engineering and Technology Research, pp. 1-5, 2014.
30. P.Priya, Dr.P.Babu, “An Efficient Architecture for the Adaptive Filter Using Delayed LMS Algorithm”, International Conference on Information Communication and Embedded Systems (ICICES), pp. 1-6, 2014
31. Surya Prakash M and Rafi Ahamed Shaik, “A Distributed Arithmetic based approach for the Implementation of the Sign-LMS Adaptive Filter,” International Conference on Signal Processing And Communication Engineering Systems (SPACES), pp. 326-330, 2015

Plagiarism Report

ACTUAL_THESIS.docx

ORIGINALITY REPORT

13%	11%	9%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.intechopen.com Internet Source	2%
2	K. Dejhan. "Adaptive Equalization Architecture Using Distributed Arithmetic for Partial Response Channels", 2006 IEEE International Symposium on Consumer Electronics, 2006 Publication	2%
3	www.comonsens.org Internet Source	1%
4	www.ece.ucdavis.edu Internet Source	1%
5	www.freepatentsonline.com Internet Source	1%
6	164.100.133.70 Internet Source	<1%
7	www.it.cityu.edu.hk Internet Source	<1%
8	scs.etc.tuiasi.ro Internet Source	<1%