FAST BLOCK LMS ADAPTIVE FILTER DESIGN

Sonali Dhobale*

P. J. Suryawanshi**

ABSTRACT

This paper proposes a design and adaptive digital filter using Fast Block Least Mean Squares (FBLMS) adaptive algorithm. The filter structure is based on Distributed Arithmetic (DA), which is able to calculate the inner product by shifting, and accumulating of partial products and storing in look-up table, also the desired adaptive digital filter will be multiplier less. Thus a DA based implementation of adaptive filter is highly computational and area efficient. Furthermore, the fundamental building blocks in the DA architecture map well to the architecture of today's Field Programmable Gate Arrays (FPGA). FPGA implementation results conform that the proposed DA based adaptive filter can implement with significantly smaller area.

***General Terms**: Fast Block Least Mean Squares (FBLMS) adaptive algorithm, Distributed Arithmetic (DA), Field Programmable Gate Arrays (FPGA), Look-up-table (LUT), Adaptive noise Canceller (ANC).*

***Keywords**: Fast Fourier transform (FFT), Read Only Memory (ROM), Inverse Fast Fourier Transform (IFFT), FBLMS, DA.*

*Priyadarshini College of Engineering, Nagpur.

**Assistant Professor, Priyadarshini College of Engineering, Nagpur.

1. INTRODUCTION

An Adaptive filters are widely used in the area of signal processing such as echo cancellation, noise cancellation, channel equalization for communications and networking systems. Necessity of adaptive filters implementations is growing up in many fields. An adaptive filter is a filter that self-adjusts its transfer function according to an optimizing algorithm. Because of the complexity of the optimizing algorithms, most adaptive filters are digital filters that perform digital signal processing and adapt their performance based on the input signal. Adaptive filter are required when either the fixed specifications are unknown or time invariant filters cannot satisfy the specifications. Adaptive filters are time-varying since their parameters are continually changing in order to meet a performance requirement.

The hardware implementation requires various of performances such as high speed, low power dissipation, small chip area and good convergence characteristics. However it is difficult to satisfy these characteristics simultaneously, so efficient algorithms and efficient architectures are desired. The design of adaptive filter algorithm is an important part within the design of adaptive filter. Among many adaptive filter algorithms, the least-mean-square (LMS) or the normalized LMS (NLMS) algorithms have been used because of their relatively small computational complexity of 2L, where L is the filter length. The shortcoming of the LMS algorithm is that it has slow convergence rate for colored input signal such as speech. Block processing is an effective approach to reduce the computational complexity and is employed in the LMS. The fast block least mean square (FBLMS) algorithm is one of the fastest and computationally efficient adaptive algorithm since here the process of filtering and adaption is done in frequency domain by using overlap and save method via FFT. The popularity of the FPGA is due to balance that FPGAs provide the designer in terms of flexibility, cost, and time-to-market. In this paper, the design of an FBLMS algorithm based adaptive filter will be analyzed.

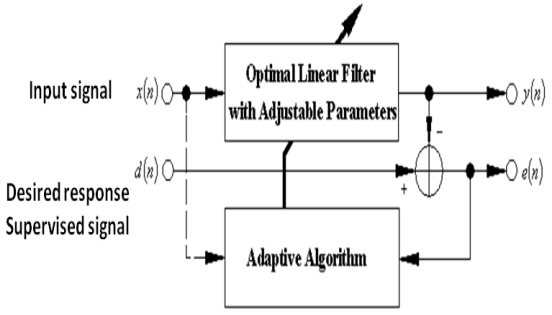

Figure1: Adaptive Filter

2. DISTRIBUTED ARITHMETIC (DA)

DA was first introduced by Croisier et al. and further developed by Peled and Lui. The DA is a direct method for sum of products operations, partial products can pre-compute by difference equation and storing in look-up-table (LUT) contained in memory, input signals are used for addressing. The product can be computed by scaling accumulate of partial products from memory, therefore multipliers do not necessary for this method.

Consider the following inner product of two L dimensional vectors a **and** x, where a is a constant vector, x is the input sample vector, and y is the result.

$$y = \sum_{k=0}^{L-1} \mathbf{a_k x_k} \qquad (1)$$

Using B-bit 2's complement binary representation scaled such that $|x_k| \leq 1$ produces

$$\mathbf{x_k} = -b_{k0} + \sum_{n=1}^{B-1} b_{kn} 2^{-n} \qquad (2)$$

Where b_{kn} are the bits (0 or 1) of x_k , b_{k0} is the most significant bit, and $b_{(B-1)}$ is the least significant bit. Substituting (2) into (1) yields

$$y = \sum_{k=1}^{L-1} \mathbf{a}_k \left[-b_{k0} + \sum_{n=0}^{B-1} b_{kn} 2^{-n} \right] \qquad (3)$$

$$= -\sum_{k=1}^{L-1} \mathbf{a_k} b_{k0} + \sum_{n=1}^{B-1} \left[\sum_{k=1}^{L-1} \mathbf{a}_k b_{kn} \right] 2^{-n} \qquad (4)$$

The computation in distributed arithmetic is represented by (4). The values of b_{kn} are either 0 or 1, resulting in bracketed term in (4) having only 2^B possible values. Since a is a constant vector, the bracketed term can be recomputed and stored in memory using either lookup table (LUT) or ROM. The lookup table is then addressed using the individual bits of input samples, x_k with the final result y computed after B cycles, regardless of lengths of vectors a and x.

3. FBLMS ALGORITHM

A block of samples of the filter input and desired output are collected and then processed together to obtain a block of output samples. A good measure of computational complexity in a block processing system is given by the number of operations required to process one block of data divided by the block length. In this case a computationally efficient implementation of

the block LMS (BLMS) algorithm and a fast BLMS are introduced in frequency domain. Fig2. shows a schematic of a block processing system.

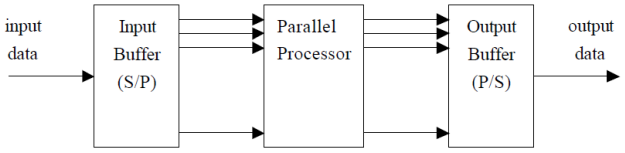

Figure 2 : Block Processing System

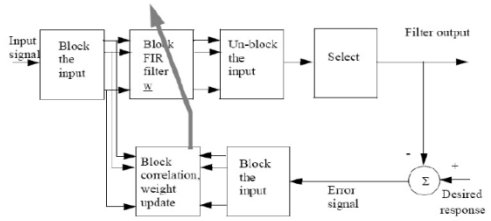

Figure 3: Block digram of Block LMS Adaptive filter

Consider a BLMS based adaptive filter, that takes an input sequence x(n), which is partitioned into non-overlapping blocks of length P each by means of a serial-to-parallel converter, and the blocks of data so produced are applied to an FIR filter of length L, one block at a time. The tap weights of the filter are updated after the collection of each block of data samples, so that the adaptation of the filter proceeds on a block-by-block basis rather than on a sample-by-sample basis as in conventional LMS algorithm.

With the j-th block, (j εZ) consisting of x (j P + r) , r ε Zp = 0,1, ..., P -1, the filter coefficients are updated from block to block as,

$$w(j+1) = w(j) + \mu \sum_{r=0}^{P-1} x(jP+r)e(jP+r) \text{.....(5)}$$

Where $w(j) = [w_0(j) w_1(j) \text{.....} w_{L-1}(j)]^t$ the tap weight vector corresponding to the j-th block,

x (j P + r) = [x (jP +r) x (jP +r -1) ....x (jP + r - L+ l)]^t and e(jP + r) is the output error at n = jP + r , given by, $e(jP+r) = d(jP+r) - y(jP+r)$…….(6)

The sequence d (j P + r) is the so-called desired response available during the initial training period and y (jP + r) is the filter output at n = j P + r , given as,

$y(jP+r) = w^t(j)x(jP+r)$..... (7)

The parameter μ, popularly called the step size parameter is to be chosen as $0 \leq \mu \leq [2/PtrR]$ for convergence of the algorithm. For the lth sub-block within the ith block, $0 \leq l \leq K-1$ i.e., for n = jP+r, r = 0,1, ... P-1, j = iK+ l, the filter output y (n) = w^t (j) X (n) is obtained by

convolving the input data sequence x(n) with the filter coefficient vector w^t (j) and thus can be realized efficiently by the overlap-save method via M = L+ P -1 point FFT, where the first L -1 points come from the previous sub-block, for which the output is to be discarded. Similarly, the weight P-l update term in (5) above, viz., $\sum_{r=0}^{P-1} x(jP+r)e(jP+r)$ can be obtained by the usual circular correlation technique, by employing M point FFT and setting the last P -1 output terms as zero.

4. PROPOSED IMPLEMENTATION

The major computational block in FBLMS algorithm is FFT/IFFT. Each N-point FFT (and IFFT) requires approximately $N \log_2 N$ real multiplications. The throughput of conventional FBLMS based adaptive filters is limited by the computational complexity involved in computation of FFT/IFFT operations. It is possible to enhance the throughput of such systems by employing some efficient techniques to compute FFT/IFFT.

DA is one of the such techniques, in which, by means of a bit level rearrangement of a multiply accumulate terms, FFT can be implemented without multipliers. There are many fast Fourier transform (FFT) algorithm exists like radix-2, Cooley-Tukey, Winograd, Good-Thomas, Rader etc. But using DA, FFT can be efficiently calculated by jointly employing the Good-Thomas and Rader algorithms. Good-Thomas algorithm re-expresses the discrete Fourier transform (DFT) of a size $N = N1N2$ as a two-dimensional $N1N2$ DFT, but only for the case where $N1$ and $N2$ are relatively prime.

It is easily seen from the definition of the DFT that the transform of a length N real sequence X (n) has conjugate symmetry, i.e.

$X(N-K) = X^*(K)$. This property means that half of the transform is redundant and need not be calculated. Rader algorithm provides straightforward way to compute only half of the conjugate symmetric outputs without calculating the others, which is not possible with other algorithms like radix-2, Cooley-Tukey and Winograd. Algorithm presented here first decomposes the one dimensional DFT into a multidimensional DFT using the index map proposed by Good [10]. Next, a method which is based on the index permutation proposed by Rader [11] is used to convert the short DFTs into convolution. This method changes a prime length N DFT of real data into two convolutions of length (N- 1)/2. One convolution is cyclic and the other is cyclic or skew-cyclic.

A. Good-Thomas Index Mapping

The index mapping suggest by Good and Thomas for n is

$$n = N_2 n_1 + N_1 n_2 \bmod N \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases} \quad (8)$$

and as index mapping for k results

$$k = N_2\langle N_2^{-1}\rangle_{N_1} k_1 + N_1\langle N_1^{-1}\rangle_{N_2} k_2 \bmod N \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases} \quad (9)$$

If we substitute the Good-Thomas index map in the equation for DFT matrix it follows

$$X[k_1,k_2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \left(\underbrace{\sum_{n_1=0}^{N_1-1} x[n_1,n_2] W_{N_1}^{n_1 k_1}}_{N_1-\text{point, transform, } x[n_2,k_1]} \right) \quad (10)$$

$$= \underbrace{\sum_{n_2=0}^{N_2-1} x[n_2,k_1] W_{N_2}^{n_2 k_2}}_{N_2-\text{point, transform}} \quad (11)$$

Steps for Good-Thomas FFT Algorithm:

An $N = N1N2$-point DFT can be computed according to following steps :

1) Index transform of input sequence, according to (8).

2) Computation of $N2$ DFTs of length $N1$ using Rader algorithm.

3) Computation of $N1$ DFTs of length $N2$ using Rader algorithm.

4) Index transform of input sequence, according to (9).

Consider the length N = 14, suppose we have $N1 = 7$ and $N2 = 2$ then mapping for the input index according to $n = 2n1+7n2 \bmod 14$ and $k = 4k1+7k2 \bmod 14$ for output index results and using these index transforms we can construct the

signal flow graph as shown in Fig.4. From Fig. 4 we realize that first stage has 2 DFTs each having 7-points and second stage has 7 DFTs each having of length 2. One of the interesting thing here is multiplication with twiddle factors between the stages is not required.

B. Rader Algorithm

Now consider $N1 = 7$ if the data are real we need to calculate only half of the transform. Also, as Rader showed the zero frequency term must be calculated separately.

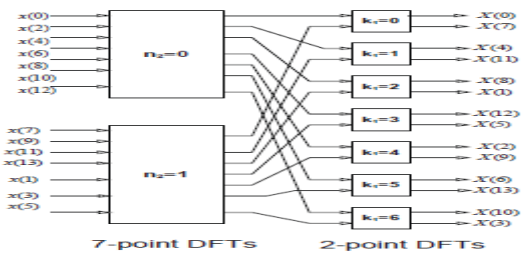

Figure 4: Mapping using Good-Thomas algorithm.

In matrix form, we write

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 1 & 3 & 5 \\ 3 & 6 & 2 & 5 & 1 & 4 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \end{bmatrix} + \begin{bmatrix} x(0) \\ x(0) \\ x(0) \end{bmatrix} \quad (12)$$

Replacing W^k by $W^{(N-k)*}$

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 3^* & 2^* & 1^* \\ 2 & 3^* & 1^* & 1 & 3 & 2^* \\ 3 & 1^* & 2 & 2^* & 1 & 3^* \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \end{bmatrix} + \begin{bmatrix} x(0) \\ x(0) \\ x(0) \end{bmatrix} \quad (13)$$

If real and imaginary parts of W matrix in (13) are separated, a simplification is possible. Consider first the real part using notation in matrix that k stands for $cos(2\pi k/7)$.

The real part of (13) becomes

$$\begin{bmatrix} X_R(1) \\ X_R(2) \\ X_R(3) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \begin{bmatrix} x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{bmatrix} + \begin{bmatrix} x(0) \\ x(0) \\ x(0) \end{bmatrix} \quad (14)$$

Using the notation of k for $sin(2\pi k/7)$ gives, for the imaginary part of (13).

$$\begin{bmatrix} X_I(1) \\ X_I(2) \\ -X_I(3) \end{bmatrix} = \begin{bmatrix} -1 & -2 & 3 \\ -2 & 3 & -1 \\ 3 & -1 & -2 \end{bmatrix} \begin{bmatrix} x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{bmatrix} \quad (15)$$

The (13) and (14) are cyclic convolution relation. Since in our problem we always convolve with the same coefficients (In case of DFT it is twiddle factor matrix), arithmetic efficiency can be improved by precalculating some of the intermediate results. These are stored in table in memory and simply addressed as needed. Using distributed arithmetic this can be implemented efficiently. Here we will present only the structure Fig. 5 best suited to the DFT calculated by cyclic convolution.

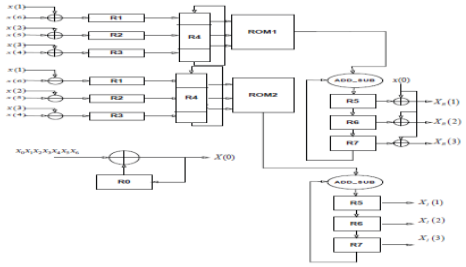

Figure 5: Architecture of FFT using DA

Initially Rl to R7 are cleared to zero and the Xi's are loaded into registers Rl to R3 after addition. Then all Rl to R3 are shifted by one bit, the last bit of each register is in R4. The ROM output will be added to R5. Circular shift of R4 produces at ROM output are added to R5 to R7. when the first cycle is completed content of all Rl to R7 except R4 are right shifted by one bit and the second cycle starts. At B^{th} cycle function at AD D _SU B changed from adder to subtraction and after this Bth cycle the content of R5 to R7 gives final FFT coefficient and zero frequency component can be calculated separately applying accumulate and addition of Xi's as shown in Fig. 2 and the FFT block so obtained has no multipliers at the expanse of increased adder requirement and memory requirement in order to store some precalculated values.

In our proposed architecture of FBLMS algorithm based adaptive filter we replace FFT (and IFFT) block, with an DA based FFT (and IFFT) block. Since the DA based FFT block provides only half of the conjugate symmetric outputs without calculating others (reaming can calculated by conjugating them) and in DA based IFFT block we require to feed only half of the conjugate symmetric coefficients not all, which is not possible in existing FBLMS based adaptive filters since here radix2 based FFT (and IFFT) blocks are employed and in radix2 there is no such facilities are there to calculate only half of the conjugate symmetric outputs hence in this case half of the processed data are redundant. Since in our proposed FBLMS algorithm based adaptive filter we require to calculate only only half of the conjugate symmetric coefficients, under that condition the hardware requirements for our proposed system is approximately half of that of existing one. In our proposed architecture shown in Fig. 5 the computation of frequency domain outputs requires $8(\mathcal{N}2+1)$ number of multiplication and required number of addition is $16N$ +2.5$(N1+N2)$+2, here total number of multiplications are much less than that of required number of multiplications in the existing FBLMS algorithm based adaptive filter at the expanse of increased memory and adder requirement, which drastically reduces the hardware complexity for higher order filters. It

results with an adaptive filter which has high throughput and low power dissipation with reduced area requirement.

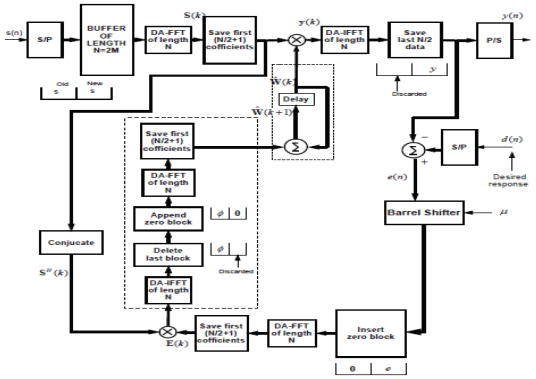

Figure 5: Proposed DA based FBLMS after optimization.

5. APPLICATIONS OF ADAPTIVE FILTERS

Perhaps the most important driving forces behind the developments in adaptive filters throughout their history have been the wide range of applications in which such systems can be used.

1) System Identification:

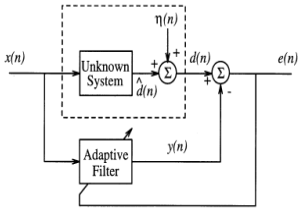

Figure 6: System Identification

Consider Fig. , which shows the general problem of system identification. In this diagram, the system enclosed by dashed lines is a "black box," meaning that the quantities inside are not observable from the outside. Inside this box is (1) an unknown system which represents a general input output relationship and (2) the signal $\eta(n)$, called the observation noise signal because it corrupts the observations of the signal at the output of the unknown system.

Let $\hat{d}(n)$ represent the output of the unknown system with *x(n)* as its input. Then, the desired response signal in this model is

$$d(n) = \hat{d}(n) + \eta(n) \text{..... (16)}$$

Here, the task of the adaptive filter is to accurately represent the signal $\hat{d}(n)$ at its output. If $y(n) = \hat{d}(n)$, then the adaptive filter has accurately modeled or identified the portion of the unknown system that is driven by *x(n)*.

Since the model typically chosen for the adaptive filter is a linear filter, the practical goal of the adaptive filter is to determine the best linear model that describes the input-output relationship of the unknown system. Such a procedure makes the most sense when the unknown system is also a linear model of the same structure as the adaptive filter, as it is possible that $y(n) = \hat{d}(n)$ for some set of adaptive filter parameters. For ease of discussion, let the unknown system and the adaptive filter both be FIR filters, such that

$$d(n) = W_{opt}^{T}(n)X(n) + \eta(n) \quad \ldots\ldots(7)$$

Where, $W_{opt}(n)$ is an optimum set of filter coefficients for the unknown system at time *n*. In this problem formulation, the ideal adaptation procedure would adjust *w(n)* such that $w(n) = w_{opt}(n)$ as $n \to \infty$. In practice, the adaptive filter can only adjust **W***(n)* such that *y(n)* closely approximates $\hat{d}(n)$ over time. The system identification task is at the heart of numerous adaptive filtering applications.

2) Adaptive Noise Cancelling:

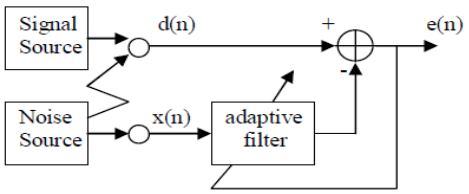

Figure 7: Adaptive Noise Canceller

As the name implies, ANC is a technique used to remove an unwanted noise from a received signal, the operation is controlled in an adaptive way in order to obtain an improved signal-to-noise ratio (SNR) . As shown in Fig.6, an ANC is typically a dual-input, closed-loop adaptive feedback system. The two inputs are: the primary input signal d(n) (the desired signal corrupted by the noise) and the reference signal x(n) (an interfering noise supposed to be uncorrelated with the desired signal but correlated with the noise affecting the desired signal in an unknown way). The adaptive filtering operation achieved the best results when the system output is noise free, which means that the output SNR is infinitely large. Therefore, in the application, on obtaining the best result, we should put the reference sensor into the most appropriate place where the signal component of the primary sensor output is

undetectable in the reference sensor output and the noise component of the primary sensor output is highly correlated with the referencesensor output.

ANC technique has been successfully applied to many applications, such as acoustic noise reduction, adaptive speech enhancement and channel equalization. A medical application that enabled the electroencephalogram (EEG) of the fetal heartbeat of an unborn child to be cleanly extracted from the much-stronger interfering EEG of the maternal heartbeat signal.

6. CONCLUSION

In this paper, we have proposed a new hardware-efficient FBLMS adaptive filters and its implementation details were presented.The concept of DA involves for implementation of FFT block without any hardware multiplier using LUT and adders. Due to reduced hardware complexity the proposed DA based FBLMS adaptive filter is best suitable for implementation of higher order filters in FPGA efficiently with minimum area requirement, low power dissipation.

7. REFERENCES

1. Asit Kumar Subudhi, Biswajit Mishra , Mihir Narayan Mohanty "VLSI Design and Implementation for Adaptive Filter using LMS Algorithm" International Journal of Computer & Communication Technology , Volume-2, Issue-VI, 2011,p84-p87.
2. Sudhanshu Baghel , Rafiahamed Shaik "FPGA Implementation of Fast Block LMS Adaptive Filter Using Distributed Arithmetic for High Throughput" 2011 IEEE,p443-p447.
3. Sudhanshu Baghel and Rafiahamed Shaik "Low Power and Less Complex Implementation of Fast Block LMS Adaptive Filter Using Distributed Arithmetic"2011 IEEE,p214-p219.
4. Amrita , Rajesh Mehra "Embedded Design of an Efficient Noise Canceller for Digital Receivers" International Journal of Engineering Science and Technology (IJEST) Vol. 3 No. 2 Feb 2011,p1252-p1057.
5. Tian Lan, Jinlin Zhang "FPGA Implementation of an Adaptive Noise Canceller" International Symposiums on Information Processing, IEEE2008, p553-558.
6. PATEL Shobhit, PANDYA Killol, BHALANI Jaymin, KOSTA Yogesh"Adaptive noise cancellation using least-mean-square algorithm" International Journal on Science and Technology ,Volume1,Issue1,2010, p019-p028.
7. Tian Lan , Jinlin Zhang "FPGA Implementation of an Adaptive Noise Canceller"2008 IEEE, p553-p558.

8. John G. Prokis & Dimitris G. Manolakis "Digital signal processing " Prentice Hall of India.