International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 3, March - 2014

Optimized Architecture for an Adaptive FIR Filter using Distributed Arithmetic

Sathyabhama. B[1], Sahaana harishankar[2], Preetha[3]
[1] Assistant Professor, Department of Electronics and Communication, Panimalar engineering college, Chennai, India
[2] Sahaana harishankar, Department of Electronics and Communication, Panimalar engineering college, Chennai, India
[3] Preetha, Department of Electronics and Communication, Panimalar engineering college, Chennai, India

Abstract — the architecture is based on distributed arithmetic in which the partial products of filter coefficients are pre-computed. These coefficients are stored in lookup tables (LUTs) and the weighted co-efficient are updated using Offset Binary Coding (OBC).The filtering is done by shift and accumulate operations on these partial products. Thus, it results in lesser area comparatively and this technique can be applied to all FIR based adaptive filter application. Results are obtained using modelsim and altera quartus tool.

Index Terms- adaptive filter, Distributed Arithmetic (DA), finite impulse response (FIR), least Mean Square (LMS), lookup table (LUT), Offset Binary Coding (OBC).

I. INTRODUCTION

Adaptive filters are widely used in many Signal Processing, applications like system identification and modeling, equalization, interference and echo cancelation. However, in many applications such as echo cancelation and system identification, coefficient adaption is needed. Such filters are generally made of Finite –Impulse- Response (FIR) filters whose coefficients are updated as per a minimization criterion. The output of a FIR filter is the weighted sum of present and past input samples and , hence they can be realized using multiply and accumulate(MAC) units .If N is the number of filter taps, a single MAC unit would take N clock cycles to produce one sample of output sequence. Although, multiple MAC units can be utilized in order to increase the speed of the system the system cost increases as the multipliers consume more area.

Distributed Arithmetic (DA) is one of the efficient techniques to realize the higher order filters and it can achieve the high throughputs without using Hardware Multipliers. Distributed Arithmetic is essentially a bit serial operation that produces the sum of products in a fixed number of clock cycles regardless of the number of products to be summed up. The basic idea behind DA is that, the pre-computed partial products are stored in look up tables (LUT), accessing and shift-accumulating. The right ones will generate the output. Furthermore, the Multiplier less architecture of DA makes it the most effective one to realize the higher order filters. A fixed –coefficient filter can be easily realized using DA by storing the partial products of filter coefficients in the LUT. However the DA treatment to adaptive filters will face certain difficulties not encountered in the fixed-coefficient case namely, the two principal operations – filtering and weight updating are manually coupled and so the partial products of the filter coefficients stored in the LUTs are recalculated before the filtering operation. However, few attempts have been made to realize adaptive filters using DA by approximations to standard adaptation algorithms which will degrade the performance.

instant n is given by

$$y[n]=\sum_{i=0}^{N-1} w[i]x[n-i]$$

II. FIR FILTER DESIGN

A. DA (DISTRIBUTED ARITHMETIC) based FIR Filter

Memory based structures are well-suited for many digital signal processing (DSP) algorithms, which involve multiplication with a fixed set of coefficients. For this Distributed Arithmetic architecture is used in our FIR filter design along with the technique of Offset Binary Coding(OBC). DA is an efficient technique for calculation of sum of products or vector dot product or inner product or multiply and accumulate (MAC). MAC operation is very common in all Digital Signal Processing Algorithms. This technique consists of Accumulators, Look Up Tables and Shift registers. According to this technique, all of the multipliers are totally removed and are replaced by array multipliers. The LUTs can be subdivided into many LUTs so that the size of the higher order filters can be reduced. In DA, the cumulative partial products are precomputed and stored in a Look Up Table (LUT) that is addressed by multiplier bits. The output y[n] of an N –tap fir filter with the present input sample x[n] at any

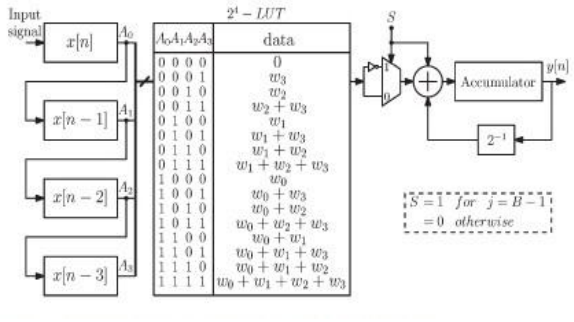

Fig.1. DA based implementation of four tap fir filter

Where w[i] (i=0,1,…,N-1) denotes the weights of the filter .By representing each of the input samples x[n-i] in the two's complement form,we got

$$X[n\text{-}i] = x_{n-i} = -b_{i,B-1} + \sum_{j=1}^{B-1} b_{i,B-1-j}\, 2^{-j} \qquad (2)$$

By substituting (2) into (1) and rearranging, we will get

$$Y[n] = \sum_{j=0}^{B-1} c_{B-1-j}\, 2^{-j}$$

Where

$$c_{B-1-j.} = \sum_{i=0}^{N-1} w_i b_{i,B-1-j} \qquad (j \neq 0)$$

$$c_{B-1} = -\sum_{i=0}^{N-1} w_i\, b_{i,B-1}$$

Thus for the given set of w_i (i=0, 1..., N-1).the terms c_{B-1-j}

Would take only one out of 2^N possible combinatonal, which can be precomputed and stored in LUT.The DA implementation of a four- tap FIR filter is shown in fig.1. The incoming bits of input samples are stored in the registers in the order that, at any instant the bits of the most recent input sample are stored in the top most register while the bits of the oldest sample are stored in the bottommost register. The least significant bits (LSBs) from each of the registers from the address lines to the LUT containing the partial products. The partial products are then shifted and accumulated for "B" number of clock cycles to produce one sample of the output.

III. ADAPTIVE FILTERING TECHNIQUE

The diagram below shows a block diagram in which a sample from a digital input signal x(n) is fed into a device, called an adaptive filter, that computes a corresponding output signal sample y(n) at time n. For the moment, the structure of the adaptive filter is not important, except for the fact that it contains adjustable parameters whose values affect how y(n) is computed.

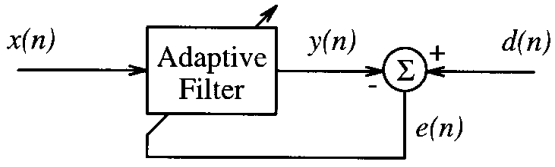

Fig.2. The general adaptive filter.

The output signal is compared to a second signal d(n), called the desired response signal, by subtracting the two samples at time n. This difference signal, given by

$$e(n) = d(n) - y(n)\ ,$$

is known as the error signal. The error signal is fed into a procedure which alters or adapts the parameters of the filter from time n to time (n + 1) in a well-defined manner. This process of adaptation is represented by the oblique arrow that pierces the adaptive filter block in the figure. As the time index n is incremented, it is hoped that the output of the adaptive filter becomes a better and better match to the desired response signal through this adaptation process, such that the magnitude of e(n) decreases over time. In this context, what is meant by "better" is specified by the form of the adaptive algorithm used to adjust the parameters of the adaptive filter.

In the adaptive filtering task, adaptation refers to the method by which the parameters of the system are changed from time index *n* to time index *(n + 1)*. The number and types of parameters within this system depend on the computational structure chosen for the system.

A. DA-Based FIR Filter With the OBC technique

The ROM size in the before mentioned block can be further reduced using the OBC technique [5] as follows. By rewriting (2) as $x_{n-i} = (1/2)[x_{n-i} - (-x_{n-i})]$

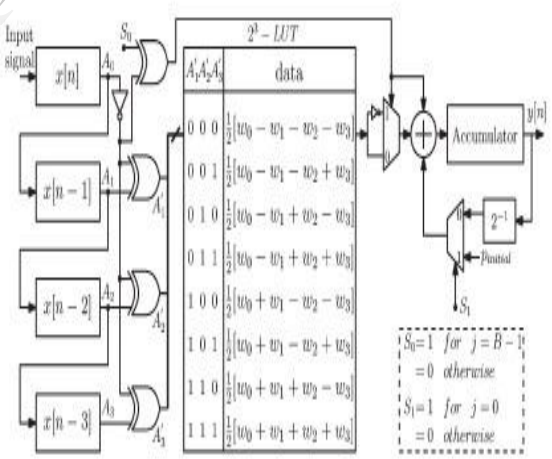

Fig.3. DA based implementation of four tap fir filter using the OBC scheme

Choose

$$d_{i,j} = \begin{cases} -(b_{i,j} - \overline{b_{i,j}}), & j \neq B-1 \\ -(b_{i,B-1} - \overline{b_{i,B-1}}), & j = B-1. \end{cases} \qquad (5)$$

By substituting (4) and (5) into (1) and rearranging we got

$$.y[n] = \sum_{j=0}^{B-1} \Big(\sum_{i=0}^{N-1} \frac{1}{2} w_i d_{i,B-1-j}\Big) 2^{-j} - \Big(\frac{1}{2}\sum_{i=0}^{N-1} w_i\Big)\, 2^{-(B-1)}$$

Define

$$p_j = \sum_{i=0}^{N-1} \frac{1}{2} w_i d_{i,j,} \qquad 0 \le j \le B-1$$

$$P_{initial} = -\frac{1}{2}\sum_{i=0}^{N-1} w_i \qquad (6)$$

We get this

$$y[n] = \sum_{j=0}^{B-1} p_{B-1-j}\, 2^{-j} + p_{initial}\, 2^{-(B-1)} \qquad (7)$$

Now, for the given set of w_i (i=0,1,…,N-1), the terms PB-1-J would take one out of 2^n combinations, half of which would be the mirror image of the other half[4].Hence ,a 2^{N-1} sized ROM can be used, the address of which can be obtained through the EXOR operation of all the LSBs with the LSB of the newest sample. Considering an adaptive filter that processes an input signal x(n) and generates an output signal y(n) as

$$Y(n) = w^T x \qquad (8)$$

Where $w^T = [w_0(n), w_1(n), \ldots w_{N-1}(n)]$ is the coefficient vector and $x^T = [x(n), x(n-1), \ldots, x(n-N+1)]$ is the input sample vector, where N is the number of filter coefficients. If the least mean square (LMS) algorithm is chosen, then each of the weights w_k (k=0,1,…,N-1) can be updated using

$$w_k(n+1) = w_k(n) + \mu e(n) x(n-k) \qquad (9)$$

Where e(n) = d(n) – y(n) is the error signal and μ is the approximate step size.

IV. PROPOSED DA – BLOCK FOR THE ADAPTIVE FILTER

The block schematic of the proposed DA- architecture for efficient implementation of the filtering and weight update operations consists of a register bank to store the incoming input samples, a primary LUT (P-LUT) that stores the combinations of weights that is responsible for the DA filtering operation in every iteration and also a secondary LUT(S-LUT) that will store the combination of input samples. It also consists of a register R_0 that along with S - LUT aids the weight adaptation process, a shift - and – accumulate block of the DA filtering operation, and a combinational logic block that takes care of the weight adaptation process.

ADDRESS	LUT CONTENTS
000	$\frac{1}{2}[w_0(n) - w_1(n) - w_2(n) - w_3(n)]$
001	$\frac{1}{2}[w_0(n) - w_1(n) - w_2(n) + w_3(n)]$
010	$\frac{1}{2}[w_0(n) - w_1(n) + w_2(n) - w_3(n)]$
011	$\frac{1}{2}[w_0(n) - w_1(n) + w_2(n) + w_3(n)]$
100	$\frac{1}{2}[w_0(n) + w_1(n) - w_2(n) - w_3(n)]$
101	$\frac{1}{2}[w_0(n) + w_1(n) - w_2(n) + w_3(n)]$
110	$\frac{1}{2}[w_0(n) + w_1(n) + w_2(n) - w_3(n)]$
111	$\frac{1}{2}[w_0(n) + w_1(n) + w_2(n) + w_3(n)]$

Fig.4. Primary LUT

ADDRESS	LUT CONTENTS
000	$\frac{1}{2}[-x(n-1) - x(n-2) - x(n-3)]$
001	$\frac{1}{2}[-x(n-1) - x(n-2) + x(n-3)]$
010	$\frac{1}{2}[-x(n-1) + x(n-2) - x(n-3)]$
011	$\frac{1}{2}[-x(n-1) + x(n-2) + x(n-3)$
100	$\frac{1}{2}[+x(n-1) - x(n-2) - x(n-3)]$
101	$\frac{1}{2}[+x(n-1) - x(n-2) + x(n-3)]$
110	$\frac{1}{2}[+x(n-1) + x(n-2) - x(n-3)]$
111	$\frac{1}{2}[+x(n-1) + x(n-2) + x(n-3)]$

Fig.5. Secondary LUT

While the P-LUT can store the OBC combinations of the filter weights, the S-LUT stores the OBC combination of input samples except for the term containing the most recent input sample which is stored in register R_0 .The entry of P-LUT at time n addressed by a can be expressed as

$$P_a(n) = \frac{1}{2} w_0(n) + \sum_{k=1}^{N-1} \frac{1}{2} w_k(n) (-1)^{q_{N-1-k}(a)+1}$$

$$= \frac{1}{2} a^T w, \qquad a=0, \quad 1, \quad 2^{N/2}-1 \tag{10}$$

where $\mathbf{w}^T=[w_0(n), w_1(n), \ldots, w_{N-1}(n)]$, $\mathbf{a}^T=[1,(-1)^{q_{N-2}^{(a)}+1}, \ldots,(-1)^{q_0^{(a)}+1}]$, and $q_l^{(a)}$ is the lth bit in the N-bit representation $(q_{N-1}^{(a)}=0)$ of address a. That is,

$$a=\sum_{s=0}^{N-2} q_s^{(a)} 2^s. \tag{11}$$

If the vector w is replaced by the vector x, which is represented as $x^T=[x(n), x(n-1),...x(n-N+1)]$,then a new LUT T(n) is formed with its entry at address location a given as …,

$$T_{(a)}(n)=\frac{1}{2}x(n)+\sum_{k=1}^{N-1} x(n-k)(-1)^{q_{N-1-k}{}^{(a)}+1},$$

$$a=0, 1,.., 2^{N/2}-1 \tag{12}$$

$$T_{(a)}(n)=R_0(n)+s_{(a)}(n) \tag{13}$$

Where R_0 denotes the contents of the register R_0 at time n and $s_{(a)}(n)$ is the entry of the S-LUT at time n addressed by a

$$R_0(n)=\frac{1}{2}x(n) \tag{14}$$

$$S_{(a)}(n)=\sum_{k=1}^{N-1}\frac{1}{2}x(n-k)(-1)^{q_{N-1-k}{}^{(a)}+1} \tag{15}$$

In Fig-4, the contents of P-LUT and S-LUT at time instant n for a four- tap filter are shown. It can be observed that S –LUT stores the OBC combinations (lower half) of input samples except for the sample x(n).In the S-LUT update scheme averaging the S-LUT entry with its next consecutive entry would generate a term that is independent of the oldest input sample. The content of register Ro is then subtracted and added with the result. For example, when N=4,the average of the first and second entries of S-LUT would give $\frac{1}{2}[-x(n-1)-x(n-2)]$, Which is independent of the third term x[n-3] .Now the subtraction and addition of the term of [x{n}] with the result will generate the terms of [-x{n}-x(n-1)-x(n-2)] and ½{x(n)-x(n-1)-x(n-2)},respectively which are then stored in the same consecutive locations of S-LUT. Similarly, the third and fourth locations of S-LUT are updated by subtracting and adding the term1/2[x (n)] to their average and storing the difference and sum in exactly the same locations. In a similar way, all the entries of S-LUT can be updated using the sum and the difference with the term ½[x(n)]. Mathematically, the new entries S_i (n+1) of S-LUT can be obtained from the old entries S_i (n) with the index entry $i \in [0,2^{N-1}-1]$ as

$$S_i(n+1) = (-1)^{i+1} R_0 + 1/2\{S_{2[\frac{i}{2}]}(n) + S_{2[\frac{i}{2}]}(n)\} \tag{16}$$

Address Location	At Time 'n' $x_{n-1}\ x_{n-2}\ x_{n-3}$	At time 'n+1' $x_n\ x_{n-1}\ x_{n-2}$	At Time 'n+2' $x_{n-1}\ x_n\ x_{n-1}$	At Time 'n+3' $x_{n-2}\ x_{n-1}\ x_n$
0	0 0 0	0 0 0	0 0 0	0 0 0
1	0 0 1	1 0 0	0 1 0	0 0 1
2	0 1 0	0 0 1	1 0 0	0 1 0
3	0 1 1	1 0 1	1 1 0	0 1 1
4	1 0 0	0 1 0	0 0 1	1 0 0
5	1 0 1	1 1 0	0 1 1	1 0 1
6	1 1 0	0 1 1	1 0 1	1 1 0
7	1 1 1	1 1 1	1 1 1	1 1 1

Fig.6.contents of s-LUT

Where R_0 (n) is the entry of register R_0 at time n. When the new input sample x(n+1) has arrived ,the right - shifted version of it that is term1/2[x(n+1)] is stored in the register R_0, which is useful for weight adaptation at time n+1.The S-LUT update scheme from time n to n=1 for a 4-Tap filter is shown in fig.5, where the positive and negative terms of the input samples are represented by "1" and "0",respectively.It can be observed that, at time n+1,there are enough contents in the S-LUT but their locations are not in proper order for the weight adaptation. By close observation, it can be seen that the contents of S-LUT are placed in the bit circularly right-shifted addressed locations. For example, if we consider the data (at time n+1) at address location 1(001) given as ½{x(n)+x(n-1)-x(n-2)}which is supposed to be in address location 6(110)-nothing but the circularly right-shifted version of the address bits of location 5(101).

If "1" and "0" are the positive and negative terms in the OBC combinations of input samples, at time n, the contents of S-LUT would look like the binary sequence of the input samples, as shown in fig.6.At time n+1,the contents would be a circularly right-shifted version of that binary sequence . Similarly at time n+2, the contents would be a circularly right-shifted version of the binary sequence at time n+1 and so on. Hence, for accessing the entries of S-LUT in each iteration, instead of physically moving the contents, the address bits to the S-LUT are circularly left-shifted. At time n+3, the sequence once again would be a normal binary sequence as shown in fig. 6.Accessing time can be reduced by maintaining two similar S-LUTs namely, ODD-LUT and EVEN-LUT as shown in fig.7, where the even location entries and odd location entries are stored respectively. $P_{initial}$ Can be stored in a register and can be updated in every iteration easily since

the term $P_{initial}$ is nothing but the additive inverse of the entry of P-LUT at its last address location. The algorithm which explains the overall operation of the proposed filter is shown in fig.8

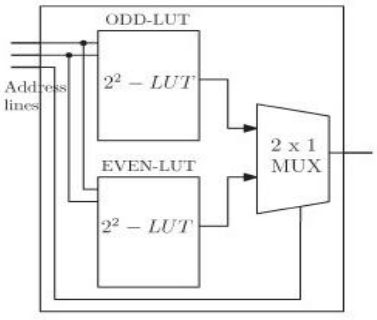

Fig. 7. Two LUT storing odd and even location

IV SIMULATION RESULTS

The following simulation results are obtained using altera modelsim and synthesized using quartus.

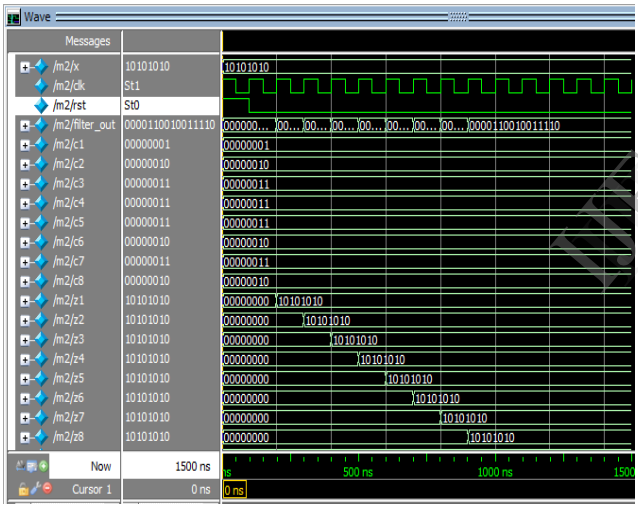

Fig. 8. Fir filter using array multiplier

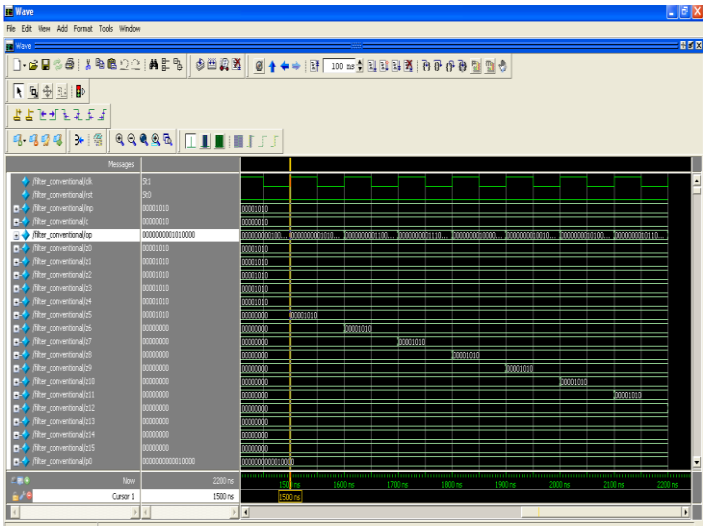

Fig. 9. Fir filter using Distributed Arithmetic

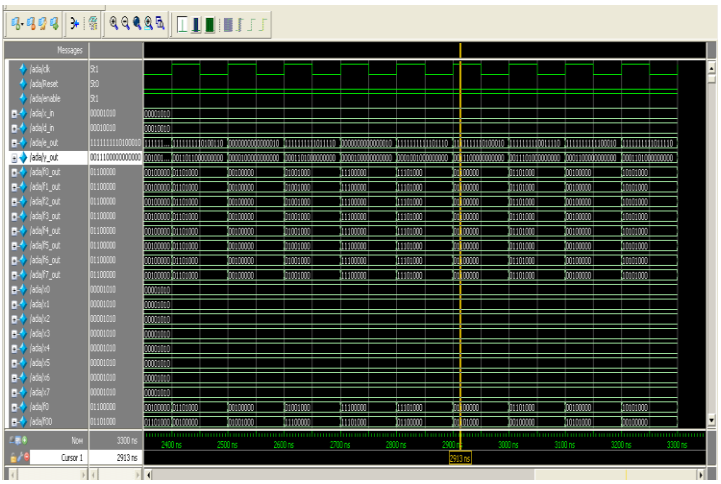

Fig. 10. Adaptive fir filter using Distributed Arithmetic simulated results.

SUMMARY RESULTS (TABLE **1)**

DESCRIPTION	ADAPTIVE FIR FILTER
AREA	Logic elements - 1322
POWER	327.81mw
TIMING	4.327ns

SUMMARY RESULTS (TABLE 2)

DESCRIPTION	FIR FILTER WITH ARRAY MULTIPLIER	FIR FILTER WITH DISTRIBUTED ARITHMETIC
AREA	Logic elements - 19	Logic elements - 288
POWER	324.40mw	320.7mw
TIMING	15.923ns	4.824 ns

V. CONCLUSION

In brief, we have presented an FIR adaptive filter implementation based on DA. In order to reduce the memory requirement this has been implemented . Unlike the technique used in [11], hence the proposed technique uses an S-LUT used to adapt weight process as in [10]. The technique used to store complex combination of input samples and the prev

ious input sample is smartly eliminated in order to update S-LUT. Thus, the proposed structure will utilize very less chip area and can operate at higher throughput when compared to other existing blocks.

REFERENCES

[1] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, "LMS adaptive filters using distributed arithmetic for high throughput," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 7, pp. 1327–1337, Jul. 2005.

[2] R. Guo and L. S. DeBrunner, "Two high-performance adaptive filter implementation schemes using distributed arithmetic," *IEEE Trans. Circuit Syst. II, Exp. Briefs*, vol. 58, no. 9, pp. 600–604, Sep. 2011.

[3] R. Guo and L. S. DeBrunner, "A novel adaptive filter implementation scheme using distributed arithmetic," in *Conf. Rec. 45th ASILOMAR*, Nov. 2011, pp. 160–164.

[4] C. F. N. Cowan and J. Mavor, "New digital adaptive-filter implementation using distributed-arithmetic techniques," *Proc. Inst. Elect. Eng.*, vol. 128, no. 4, pt. F, pp. 225–230, Feb. 1981.

[5] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, "LMS adaptive filters using distributed arithmetic for high throughput," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 7, pp. 1327–1337, Jul. 2005.

[6] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, "A novel high performance distributed arithmetic adaptive filter implementation on an FPGA," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2004, vol. 5, pp. V-161–V-164.

[7] .A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Mag.*, vol. 6, no. 3, pp. 4–19, Jul. 1989.

[8] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, "An FPGA implementation for a high throughput adaptive filter using distributed arithmetic," in *Proc. 12th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2004, pp. 324–325.

[9] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Rizo, "Digital Filter for PCM Encoded Signals," U.S. Patent 3 777 130, Apr. 1973.

[10] A. Peled and B. Lie, "A new hardware realization of digital filters," *IEEE Trans. Acoustics, Sound, Signal Process.*, vol. ASSP-22, no. 4, pp. 456–462, Dec. 1974.

[11] Pocket Guide to DSP Processors, Berkeley Design Technology Inc. (2003, Dec.). [Online]. Available: http://www.bdti.com

[12] Stratix Device Family Data Sheet, Altera Corporation. (2003).[Online].Available:http://www.altera.com/literature/lit-index.html

[13] J. P. Uyemura, *Introduction to VLSI Circuits and Systems*. New York: Wiley, 2001

[14] M. Surya Prakash and R. Shaik, "High performance architecture for LMS based adaptive filter using distributed arithmetic," in *Proc. ICICA*, Mar. 2012, vol. 24, pp. 18–22.