International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 15 (2017) pp. 5060-5066
© Research India Publications. http://www.ripublication.com

Area and power efficient OBC DA based adaptive FIR filter

Abhijeet SureshraoShinde[1] and Sriadibhatla Sridevi[2]

[1]PG Student, School of Electronics Engineering, VIT University, Vellore, India.
[2]Associate Professor, School of Electronics Engineering, VIT University, Vellore, India.

[2]Orcid: 0000-0002-6318-3145

Abstract

This paper presents a novel architecture for low-power and low area implementation of adaptive FIR filter based on offset binary coded distributed arithmetic (OBC DA). The DA based LUT of the existing structure is replaced with OBC DA based LUT. Unlike existing DA-based designs the proposed design using DA-OBC method involves the same number of multiplexers but a smaller LUT with less number of delay elements. In addition to this, DA-based inner product computations have been done by using conditional signed carry-save accumulation instead of traditional adder based shift accumulation in order to reduce area complexity. From Synthesis results it is found that the total Power has been decreased by about 21 percent and total area decreased by 16 percent in proposed design when compared to previous DA based adaptive filter.

Keywords: Adaptive filter, least mean square algorithm, circuit optimization, distributed arithmetic.

INTRODUCTION

Adaptive filters are extensively used in many digital signal processing (DSP) applications such as echo and noise cancellation, channelization, system identification, system modelling. Widrow-Hoff 's least mean square (LMS) algorithm based tapped delay line finite impulse response (FIR) filters are popular among them because of their simplicity and better performance in terms of convergence [1]. The process of computation of inner product of the filter involves multipliers which increase the critical path, area and cost of the system. Distributed arithmetic (DA) is an efficient technique for the implementation of higher order filters, which does not involve any multipliers. In DA based filters, Pre computed partial products are stored in a look up table (LUT) and the output is computed by retrieving the appropriate product followed by shift accumulate operation [2-4].

Allredeat al., presented hardware efficient adaptive filter using DA which consist of two Look up tables (LUT) [5,6]. These two LUTs carry out the process of filtering and weight update. Guo et al., proposed two schemes in which only one LUT is used for weight updating and filtering [7,8]. When we go through these designs, we can observe that more number of cycles are required to update the LUTs for every sample and do not support high rate of sampling. The authors in [9], replaced the LUTs with register banks and updated the weights in parallel which improved the through put of the system. However the size of the LUT in DA based filters increases linearly with the order of the filters which in turn increased area and delay. To overcome this limitation, and reduce the size of the LUT, the input data is coded using offset binary coding (OBC) rather than normal binary coding [2].

In OBC DA based filters, the size of the LUT is reduced to half that of normal DA based filters [2]. A new weight updating strategy for the adaptive filter based on OBC is presented in [10]. Authors in [11] proposed an efficient architecture for high-speed DA based adaptive filter with very low adaptation-delay. Later they improved the throughput of the design in [11] by implementing parallel LUTs and conditional carry save accumulation [12]. In the present work, the DA based LUT of [11] is replaced by OBC DA and hence shows more improved hardware design with less area. Instead of 15 delay elements 4 delay elements are used in the proposed OBC-DA technique which results into critical path delay and power reduction.

The organization of the paper is as follows. LMS adaptive algorithm is reviewed in section II. DA technique is described in section III, Existing and Proposed OBC-DA structures are detailed in section IV and V respectively. Synthesis results are presented in section VI. Section VII provides the conclusion.

LMS ADAPTIVE ALGORITHM

The process of evaluation of error value and output of filter is carried out in every cycle using LMS algorithm. Error value is calculated from difference between the current output of the filter and desired response.

Weights of the filters are updated with the help of error which is estimated during each and every cycle. Following equation elaborates the weight update process of LMS adaptive filter at the a^{th} iteration.

$$j(a+1) = j(a) + h \cdot r(a) \cdot k(a) \tag{1.1}$$

where

$$r(a) = q(a) - l(a). \tag{1.2}$$

$$l(a) = k^T(a) \cdot k(a). \tag{1.3}$$

$k(a)$ = Input Vector.

$j(a)$ = Weight Vector.

Now consider $k(a)$ and $j(a)$ at a^{th} iteration. It is given by

$$k(a) = [k(a), k(a-1), ..., k(a-A+1)]^T \tag{2.1}$$

$$j(a) = [j_0(a), j_1(a), ..., j_{A-1}(a)]^T \tag{2.2}$$

Where,

$q(a)$ = Desired Response.

$l(a)$ = Filter output.

$r(a)$ = Error calculated during a^{th} iteration

[It updates the weights]

h = Convergence Factor.

A = Filter Length.

Feedback error $r(a)$ will be available if we do pipelining in our design. This error is called as Adaptation Delay.

Hence pipelined architecture uses $r(a-z)$ which represents the error with delay of 'z'.This 'z' element is called as Adaptation delay. So now weights will be updated with the help of delayed error $r(a-z)$ instead of $r(a)$ which is recent error. Following equation represents weight update equation for delayed LMS adaptive filter.

$$j(a+1) = j(a) + h \cdot r(a-z) \cdot k(a-z) \tag{3}$$

EXISTING DA BASED INNER PRODUCT COMPUTATION

Inner Product computation is important to perform during every cycle in LMS Adaptive Filter which is helpful in critical path.

So inner product for 1.3 is as follows.

$$l = \sum_{x=0}^{A-1} j_x \cdot a_x \tag{4}$$

Here j_x and a_x for $0 \le x \le A-1$ is used to obtain A- point vectors related to present weights and recent most A-1 input.

Y= Width of the bit for weights.

Now 2's complement representation of every object present in weight vector is

$$j_x = -j_{x0} + \sum_{y=1}^{Y-1} j_{xy} \cdot 2^{-y} \tag{5}$$

j_{xy} = y^{th} bit of j_x

Put equation (5) in equation (4) we will get

$$l = -\sum_{x=0}^{A-1} k_x \cdot j_{x0} + \sum_{x=0}^{A-1} k_x \cdot \left[\sum_{y=1}^{Y-1} j_{xy} \cdot 2^{-y} \right] \tag{6}$$

Now bringing equation (4) into Distributed form by changing A and Y.

$$l = -\sum_{x=0}^{A-1} k_x \cdot j_{x0} + \sum_{y=1}^{Y-1} 2^{-y} \cdot \left[\sum_{x=0}^{A-1} k_x \cdot j_{xy} \right] \tag{7}$$

Equation (7) can be represented as,

$$l = \left[\sum_{y=1}^{Y-1} 2^{-y} \cdot l_y \right] - l_0 \tag{8}$$

Where,

$$l_y = \sum_{x=0}^{A-1} k_x \cdot j_{xy}$$

Any element from A point bit sequence is $\left\{ j_{xy} \to for \to 0 \le x \le A-1 \right\}$ have value of either 0 or 1.

Here l_y is represented as a partial sum of our equation

Where

$$y = 0, 1, ..., Y-Z$$

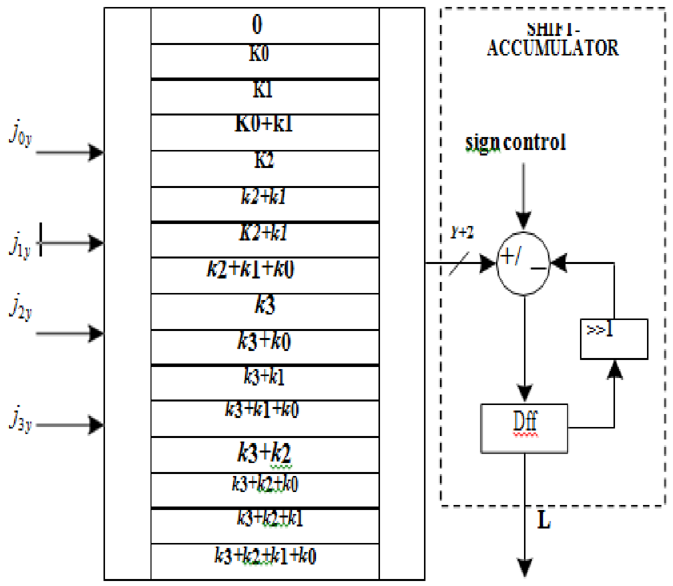

Figure 1: Conventional DA-based implementation of 4-point inner-product.

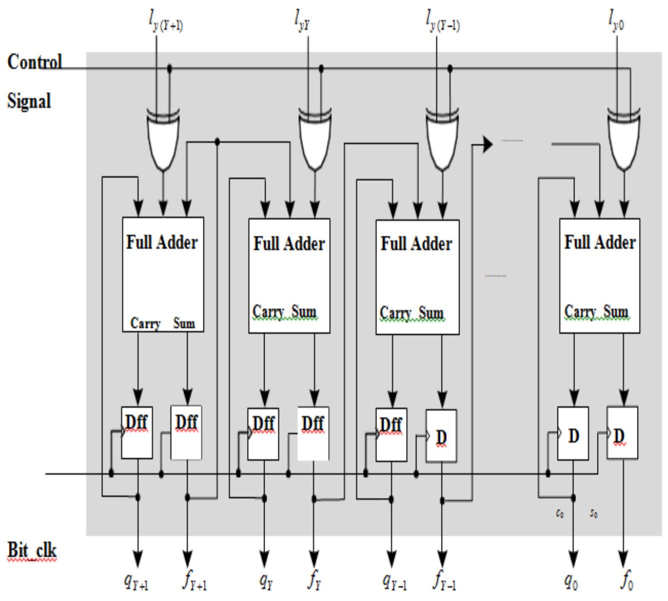

Figure 2: Carry save implementation of shift accumulation

This will have 2^A possible values. If the process calculation of all the l_y value is carried out for 2^A possible values and stored in Look Up Table (LUT) then it is very easy to read all partial sums like l_y from LUT. The read out process from LUT will be easily carried out with the help of $\{j_{xy}\}$ which is used as address bit to calculate inner product. Inner product represented in equation (8) is computed in Z cycles with the help of shift accumulation and LUT operation in read form with L number of bit slices $\{j_{xy}\}$ for $0 \le x \le A-1$ as per fig.1.Critical path is main problem for shift accumulation in fig.1.In fig.2 carry save accumulation is used to perform shift accumulation. Carry save accumulation is fed with bit slices having vector w from LSB to MSB order. $X-OR$ gate is used to carry out $2's$ complement of given negative number for that sign control input of $X-OR$ gate is kept to 1 only when MSB -slice appears as address. Full adder is used to obtain the sum(s) and carry(c) after Y clock cycles where carry to finale adder is set to 1 for $2's$ complement operation. y^{th} location in LUT contain.

$$P_x = \sum_{w=0}^{A-1} k_w \cdot x_w \qquad (9)$$

Where x_w is $(w+1)$ bit of A bit binary representation of integer x for $0 \le x \le 2^A - 1$. P_x for $0 \le x \le 2^A - 1$ is initially calculated and stored in RAM based on LUT of 2^A words. We can store $2^A - 1$ words easily because there are $2^A - 1$ registers in our DA table instead of 2^A words. N=4 DA table shown in fig.(3).It consist of 15 registers to store precomputed value of sums of input words. Use of seven adders results into formation of seven new values that is P_x in parallel.

EXISTING DA BASED ADAPTIVE FILTER STRUCTURE

The computation process of adaptive filter with high orders required to be divided into number of low level filtering blocks, because LUT with high capacity is a basic need of inner product with long vectors for DA-based technique. Hence we are discussing here about higher order DA based adaptive FIR filter.

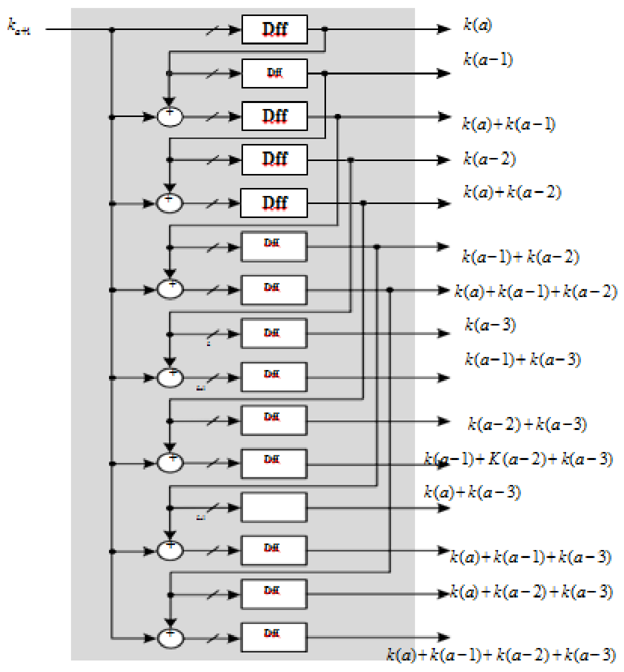

Figure 3: The DA-table for generation of possible sums of input samples

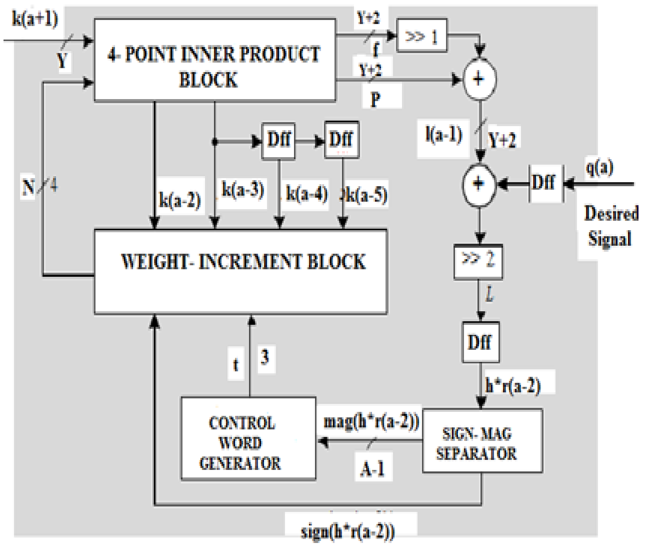

Figure 4: Proposed structure of DA-based LMS adaptive filter of filter length A= 4.

The existing DA based adaptive filter consist of barrel shifter, weight increment block, control word, 4-point inner product block. These all circuits are used for the calculations of error value $r(a)$. The DA table shown in Fig. 3 is having group of 15 registers for storage purpose. These 15 registers used to store the partial inner products y_l for $0 \le l \le 15$. Mux with 16:1 configuration is used for the selection of register content. Weights (Bit Slices) are used as a select lines for 16:1 mux. Carry save accumulator is fed with (fig.2) mux output Carry save accumulator shifts and accumulates it's input provided which is partial inner product and finally generates sum and carry of size $Y+2$.This whole process is carried out in Y bit cycles. Now carry and sum outputs are first shifted then added where carry is considered to be '1' then filter output is generated. This filter output now subtracted from desired output $q(a)$ to obtain error $r(a)$. Here MSB bit when it is '1' immediately avoided. The process of multiplication of error with input x_k is carried out through right shift operation. Right shift depends upon number of zero present in magnitude of error. Resultant error is used to produce control word 'g' which is required for barrel shifter.

Logic for 'g' generation shown in fig(5.3).

$j =$ Convergence Factor $= 0(1/A)$;

We are using $j = 1/A$;

We can also take $j = 2^{\frac{-v}{A}}$

$v =$ Small Integer.

The number of shifts 'g' in this case is incremented by v number of locations. So input to barrel shifter is pre-shifted by v locations to reduce hardware complexity. Weight increment block(fig.5.2) is considered for A= 4 consist of 4 adder .

The logic used for generation of control word 'g' for the barrel-shifter for Y =8 is as follows

If $e_6 = 1$ then g = "000" ;

Else if $e_5 = 1$ then t = "001" ;

Else if $e_4 = 1$ then t = "010" ;

Else if $e_3 = 1$ then t = "011" ;

Else if $e_2 = 1$ then t = "100" ;

Else if $e_1 = 1$ then t = "101" ;

Else if $e_0 = 1$ then t = "110" ;

Else then g = 1 then t = "111" ;

$e = abs(h * r(a-2))$

e_i : ith bit of 7-bit word e

or subtracter, shifter. Barrel shifter shifting input values

x_k for $k = 0,1,....., A-1$;

Barrel shifter logic is responsible for the addition or subtraction of desired increments with current weights.

Sign bit = 1 - Addition ;

= 0 - Subtraction;

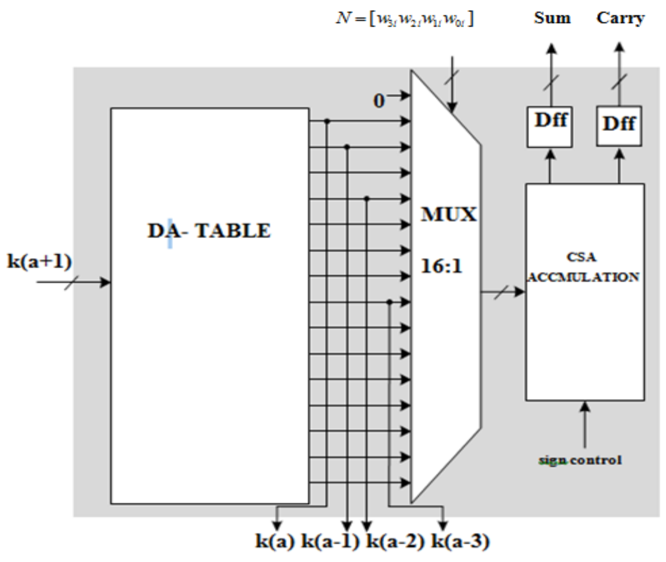

Figure 5: Structure of the 4-point inner-product block.

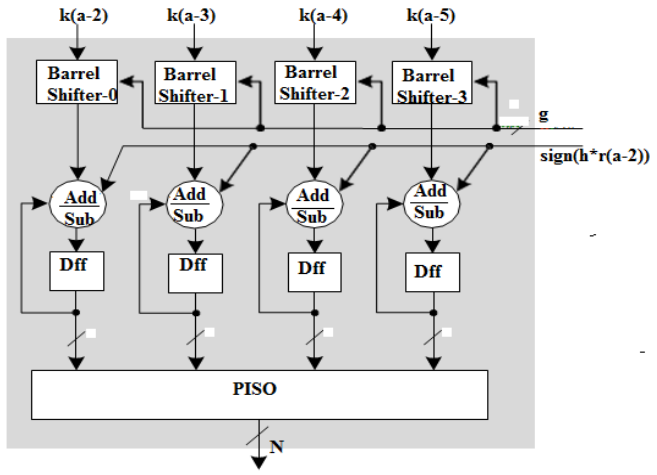

Figure 6: Structure of the weight-increment block for A=4.

The structure of higher order filter of size A=16 is shown in Fig. 7. The design involves four sets of inner product blocks and weight increment blocks. All the blocks are connected in a proper manner in order to give a desired result.

The four 4-point inner product blocks and weight increment blocks all together is known as 16-bit data computing block as this sub block computes 16-bit sum and carry words. These are used in further computations. As in the case of fourth order filter implementation, here also the Y+2 bit sums and carry are produced by the four inner product blocks. And these will be added by using two binary adder trees. The output of four 4-point inner-product blocks i.e. sum words are added with four carry-in bits. Since the carry words are of double the weight compared to the sum words, two carry-in bits are set as input carry at the first level binary adder tree of carry words, which is equivalent to inclusion of four carry-in bits to the sum words. Sign magnitude separator is used to separate sign bits and magnitude bits from the calculated error as in the case of smaller order filter designs. Outputs of sign-magnitude separator and control word generator are fed commonly to all the weight increment blocks. The logic used for control word generator is also same as that of previous logic.

Further, the filtering process is same as that of the 4th order adaptive FIR filter except that the process is performed on 16-bit data instead of 4-bit data. Similarly, the structure can be extended to 32-bit filter implementation. For that purpose two 16-bit data computing blocks which are mentioned earlier have been used. The structure of 32-bit filter is shown in the fig.8. Just by performing the binary addition of 16-bit sum and carry of the two 16-bit data computing blocks, the filter of length A=32 can be realized. The connections between the blocks must be given properly for achieving better results.

PROPOSED OFFSET BINARY CODED DA-BASED ADAPTIVE FILTER (OBC-DA) STRUCTURE

Conventional DA based technique consists of 15 delay elements that increase the area and power consumption. In the proposed approach, for the implementation of DA based adaptive filter, we make use of the OBC. This OBC binary DA-based technique makes use of the 4 delay elements. Hence by using the proposed structure, we can reduce the original DA-table structure about two times which increases the area efficiency of the design twice. We adopt parallel LUTs to achieve low adaptation delay for the filter implementation.

The proposed structure for the DA-table which makes use of four delay elements is shown in the figure.6.

The shift accumulation blocks are replaced with conventional carry save adders to reduce critical path delay. In this way, by using the OBC technique, we can reduce the size of the DA-table structure which in turn reduces the ROM size by a factor of two. Eventually, the design of the adaptive filter with OBC method gives a better chip area reduction.

SYNTHESIS RESULTS

In this section we have implemented the proposed design with input bits length A=8. Design entry for all the modules is made in Verillog. Initially Modelsim tool is used to verify the modules. Each and every module like DA table, weight increment block, inner product block are completely verified on Modelism to understand their functionality before going for Synopsys tool.

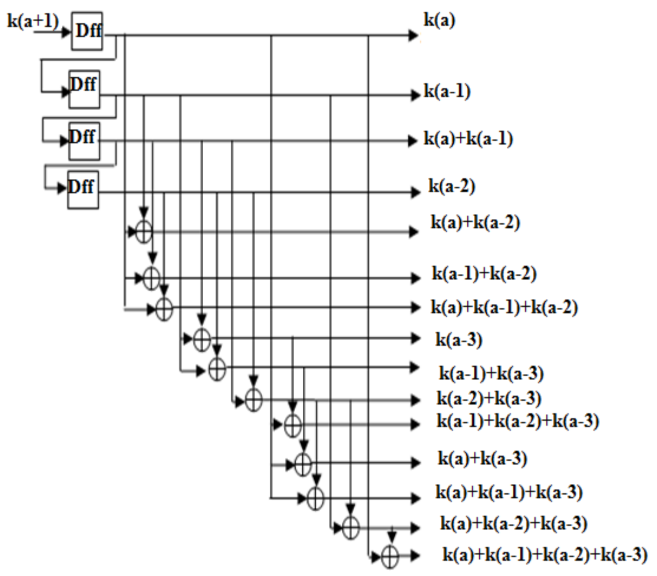

Figure 7: Structure of DA-table with 4 delay elements

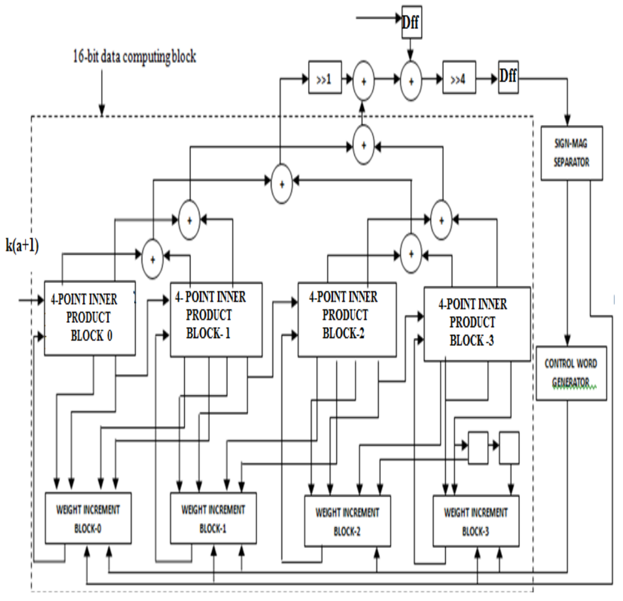

Figure 8: Structure of DA-based LMS adaptive filter of length A=16 and

Simulation and synthesis process is carried out using Synopsys VCS, DC compilers."Saed90nm_typ.db"Library which is 90nm library is used for this whole process. Our main focus is to reduce area and power and the area, power and timing constraints of the tool are defined according to the requirement and provided to the Synoposys tool. The results obtained are tabulated below in Table 1.. From the results it is clearly noticed that the Power and Area are reduced within the proposed design

Table I: Synthesis Report

Design	Filter Length	Power in (μw)	Area In (μm^2)
Existing Design in [11]	16	1.5300	75610.159391
Proposed Design with LUT using 4 Delay Elements	16	1.2647	64837.465268

CONCLUSION

Low power designs have greater importance in efficient hardware implementations. In this brief, OBC DA based architecture for adaptive FIR filters is presented. A four delay element DA table is developed to reduce the size of the delay table. This design achieved power better reduction in power as well as area when compared to the existing DA-based adaptive filter implementations. The power reduction reported is 21% and the area is minimized by 16% and can be a better design for power efficient adaptive filter implementation.

REFERENCES

[1] S. Haykin and B. Widrow, *Least-mean-square adaptive filters*. Wiley- Interscience, Hoboken, NJ, 2003.

[2] S. A. White, “Applications of the distributed arithmetic to digital signal processing: A tutorial review,” *IEEE ASSP Magazine*, vol. 6, no. 3, pp. 5–19, Jul. 1989.

[3] P. K. Meher and S. Y. Park, “High-throughput pipelined realization of adaptive FIR filter based on distributed arithmetic,” in *IFIP/IEEE International Conference on Very Large Scale Integration*, Oct. 2011. pp.428–433.

[4] M. D. Meyer and P. Agrawal, “A modular pipelined implementation of a delayed LMS transversal adaptive filter,” in *IEEE International Symposium on Circuits and Systems*, May 1990, pp. 1943–1946.

[5] 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, SignalProcess.*, May 2004, vol. 5, pp. V-161-V-164.

[6] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, “LMS adaptive filters using distributed

arithmetic for high throughput," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 7, pp. 1327–1337, Jul. 2005.

[7] R. Guo and L. S. DeBrunner, "Two high-performance adaptive filter implementation schemes using distributed arithmetic," *IEEE Transactions on Circuits and Systems-II: Express Briefs*, vol. 58, no. 9, pp. 600–604, Sep. 2011.

[8] R. Guo and L. S. DeBrunner , "A novel adaptive filter implementation scheme using distributed arithmetic," in *Asilomar Conference on Signals, Systems and Computers*, Nov. 2011, pp. 160–164.

[9] 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.

[10] M. Surya Prakash and R. Shaik, "Low-Area and High-Throughput Architecture for an Adaptive Filter Using Distributed Arithmetic ", *IEEE Transactions on Circuits and Systems-II: Express Briefs*, vol. 60, no. 11, pp. 781–604, Nov.. 2013.

[11] P. K. Meher and S. Y. Park, "High-throughput pipelined realization of adaptive FIR filter based on distributed arithmetic," in IFIP/IEEE International Conference on Very Large Scale Integration, pp. 428–433,. Oct 2011.

[12] S. Y. Park P. K. Meher, "Low-Power, High-Throughput, and Low-Area Adaptive FIR Filter Based on Distributed Arithmetic", *IEEE Transactions on Circuits and Systems-II: Express Briefs*, vol. 60, no. 6, pp. 346-350 , June 2013.