International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 01,January-2015

Implementation of Adaptive FIR Filter with Area Optimization Based on DA and OBC

B. Doss
Department Of ECE,JNTUA CEA
Anantapur, India

K. Soundararajan
Department of ECE, KITES Engg, College
Hyderabad, India

Y. Narasimha Murthy
Reader, SSBN Degree & PG College
Anantapur, India

Abstract— **In this paper an efficient pipelined architecture of an adaptive FIR filter based on distributed arithmetic (DA) is discussed. On comparing with existing DA based adaptive filter implementations the proposed architecture has achieved "low-complexity". For achieving this advantage Offset Binary Coding (OBC) has been used in this filter design. By doing so, the number of LUTs used has been decreased from 16 to 8. 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. Unlike existing DA-based designs the proposed design involves the same number of multiplexers but a smaller LUT and nearly half of the number of adders.**

Keywords*— *Adaptive FIR filter, Distributed Arithmetic, carry-save accumulation, Adaptive LMS algorithm, offset Binary Coding.

I. INTRODUCTION

Now-a-days in numerous DSP applications adaptive filters are the most important and useful blocks. Because of its simplicity and satisfactory convergence performance, the tapped-delay line finite impulse response (FIR) is the most popularly used adaptive filter. The weight updating in this filter follows Widrow-Hoff least mean square (LMS) algorithm [1]. A long critical path on the forward path of FIR filter design will be resulted in the direct form realization. It is because of the computation of an inner product to obtain a filter output. So it is the case while dealing with input signals of high sampling rate, the reduction in the critical path of the structure is necessary.

In the current scenario some distributed arithmetic techniques such as multiplier less algorithms [2] have gained much popularity. This structure has more advantages such as high throughput and because of its regularity it is cost-effective and area efficient. Allred et al [3] proposed hardware-efficient DA-based design which uses two different lookup tables (LUTs) for updating weight and filter coefficients. Sang Yoon Park and Pramod Kumar proposed an architecture [4] for implementing adaptive FIR filters. But the architecture is complex for higher order filters.

So in this brief, low-complex architecture is proposed for implementing adaptive FIR filter. Such a simpler design is possible with the help of "Offset Binary Coding". Offset binary coding is a digital coding scheme. It is also known as "excess-k". In this coding scheme minimum negative value is represented by all zeros and the maximum positive value is represented by all-ones. This coding scheme has no standards, but the offset K=2^ (n-1) is most often used for an n-bit binary word. Here the "zero" value is represented by all zeros except the most significant bit, which is similar to two's complement notation except the most significant bit is inverted.

Offset binary coding finds more applications in Digital Signal Processing (DSP) [5]. The basic units of DSP chips are analog to digital (A/D) and digital to analog (D/A) converters. These are unipolar so cannot process bipolar signals. For that biasing the analog signals with a DC offset is needed. The result is in offset binary format. Offset binary format cannot be handled directly by most of the standard computer CPU chips. They typically require some data conversion techniques to process offset binary. But without requiring any data conversion DSP chips can handle offset binary. Just by inverting the most significant bit offset binary can be converted into two's complement form.

II. CONCEPT OF ADAPTIVE LMS ALGORITHM

The LMS algorithm computes both the filter output and an error value during each cycle. Usually the difference between current filter output and desired responses is termed as error value. Then the filter coefficients are updated with an estimated error value in every training cycle. The filter coefficients are updated during the *n*th iteration according to the following equations:

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

Error signal is given as

$$e(n) = d(n) - y(n) \qquad (2)$$

Output signal of the adaptive filter is

$$y(n) = w^{q^T}(n) * x(n) \qquad (3)$$

Where $w(n)$ is the filter coefficients vector, and $x(n)$ is the filter input vector, d(n) is the desired response, μ is convergence factor. The feedback error $e(n)$ will be available This is called "adaptation delay". So the delayed error is used for updating the current weight in pipelined structures instead of the most recent error, the parameter 'm' is the adaptation delay. The equation that governs such weight update in adaptive filter using LMS algorithm is given by

$$w(n+1) = w(n) + \mu * e(n-m) * x(n-m) \qquad (4)$$

Here x*(n)* is filter input vector and *w(n)* is weight vector.

III. DA-BASED APPROACH FOR INNER-PRODUCT COMPUTATON

In the LMS adaptive filter, inner-product computations of the critical path in each cycle are performed by using the following expression.

$$y = \sum_{k=0}^{N-1} \omega_k * x_k \qquad (5)$$

Where ω_k and x_k form the *N*-point vectors corresponding to the current weights and the most recent *N*-1 input values respectively for $0 \leq k \leq N$-1. Let *L* be the bit width of the weight and each component of the weight vector in its 2's complement representation is expressed as:

$$\omega_k = -\omega_{k0} + \sum_{l=1}^{L-1} \omega_{kl} * 2^{-l} \qquad (6)$$

Where ω_{kl} denotes the l^{th} bit of ω_k.Equation (5) can be written as

$$y = -\sum_{k=0}^{N-1} x_k * \omega_{k0} + \sum_{k=0}^{N-1} x_k * \left[\sum_{l=1}^{L-1} \omega_{kl} * 2^{-l}\right] \qquad (7)$$

In the above equation by altering the summations order of *k* and *l* indices it can be converted to sum-of-products form.

$$y = -\sum_{k=0}^{N-1} x_k * \omega_{k0} + \sum_{l=1}^{N-1} 2^{-l} * \left[\sum_{l=1}^{L-1} x_k * \omega_{kl}\right] \qquad (8)$$

From the above equation the inner-product can be calculated as

$$y = \left[\sum_{l=1}^{L-1} 2^{-l} * y_l\right] - y_{0,}\ where\ y_l = \sum_{k=0}^{N-1} x_k * \omega_{kl} \qquad (9)$$

The partial sum y_l for l = 0,1,….,L-1 will have 2^N possible values. The inner-product of above equation will be calculated for L cycles of shift accumulation; thereafter for L number of bit slices where $0 \leq l \leq$ L-1 corresponding LUT-read operations will be performed as shown in fig.1. And the shift accumulation is performed using carry-save accumulator as it involves significant critical path.

In carry-save accumulation the weight vector *ω* bit slices are given one after the next in the order LSB to MSB. In the case of MSB slices the accumulation of output in its 2's complement form is needed. Therefore, all the bits of LUT output are passed through the XOR gates with a sign-control input as the result is represented in two's complement format. The output will be set to one if the appeared address is MSB slice.

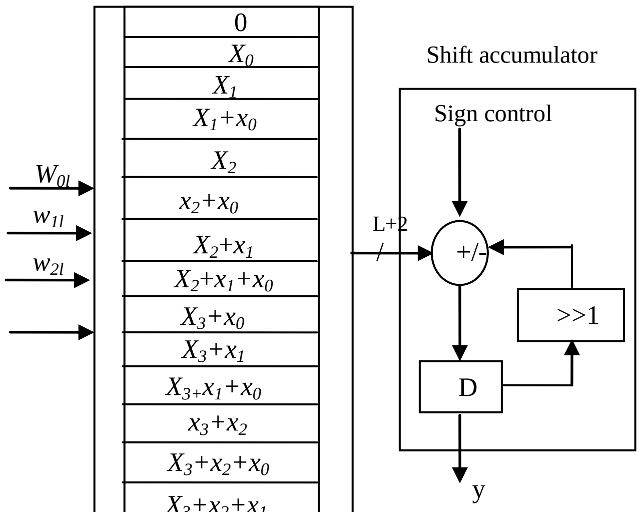

Fig. 1. 4-point inner product computation in Conventional DA-based implementation

Thus the XOR gates produce the LUT output of MSB slice in one's complement form without affecting the output of other bit slices. At last both the outputs of XOR gate (i.e. sum and carry words) resulted after L number of clock cycles have been added by the final adder. And the input carry to the final adder is set to one to account for performing 2's complement operation of the LUT output of the MSB slice.

The equation that gives data in kth LUT location is as follows

$$C_k = \sum_{j=0}^{N-1} x_j * k_j \qquad (10)$$

In the above expression k_j is the (j+1)th bit of *N*-bit binary representation of integer *k* for $0 \leq k \leq 2^N$-1. Here $C_k$for $0 \leq k \leq 2^N$*-1* can be recomputed and stored in RAM-based LUT of 2^N words. However, only *(2^N-1)* registers are stored in LUT instead of storing 2^N words.

IV. EXISTING DA-BASED ADAPTIVE FILTER STRUCTURE

For our convenience we can decompose the computations of the large ordered adaptive filters into small adaptive filtering block. This makes the job of computing inner product which is the most important task in DA-based implementation of adaptive filters easier as the inner product computation of long vectors requires a very large LUT [3].

The existing DA-based LMS adaptive filter design requires 16 delay element DA-table structure. The 4th order adaptive filter (N=4) implementation involves a four-point inner product block, weight increment block and the circuit for generating error ***'e'***, and a control word generator that generates the control word ***'t'*** for the barrel shifters as shown in fig 2.

But the same structure can be implemented with the help of 8-delay elements. Since with the help of basic elements x(n), x(n-1), x(n-2), x(n-3) the remaining structure can be implemented. It can be achieved with the help of Offset Binary Coding (OBC).

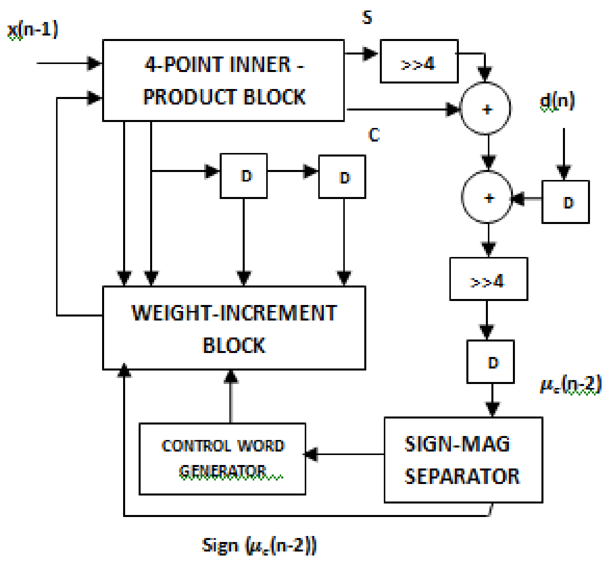

Fig. 2. Structure of DA-based LMS adaptive filter of filter length N=4

V. PROPOSED OFFSET BINARY CODING BASED ADAPTIVE FILTER DESSIGN

In the proposed approach, for the implementation of DA-based adaptive filter, we make use of the DA-table structure that makes use of the 8 delay elements. Hence by using the proposed structure, we can reduce the original DA-table structure by two times, which increases the area efficiency of the design twice. The proposed structure for the DA-table which makes use of eight delay elements is shown in the figure. 3.

The proposed structure for 4th order adaptive filter (N=4) is shown in the fig 2. Generally, it consists of a four-point inner product block, a weight increment block and the circuit for generating error ***'e'***, and a control word generator that generates the control word ***'t'*** for the barrel shifters.

The four point inner product block which was shown in fig. 5 , consists of a DA-table which has an array of 15 registers. It is capable of storing the partial inner products ***'y'***, for $0 < l \leq 15$ and a 16:1 multiplexer is used to select one of those registers at any particular instant. Weights A= $\{w_3, w_2, w_1, w_0\}$ for $0 \leq l \leq L\text{-}1$ are fed to the multiplexer as control bits in the LSB-MSB order. The output of the MUX is then fed to the carry-save accumulation block. After L clock cycles, the carry-save accumulation block accumulates all the partial inner products and generates a sum word and a carry word of size L+2 bit length each.

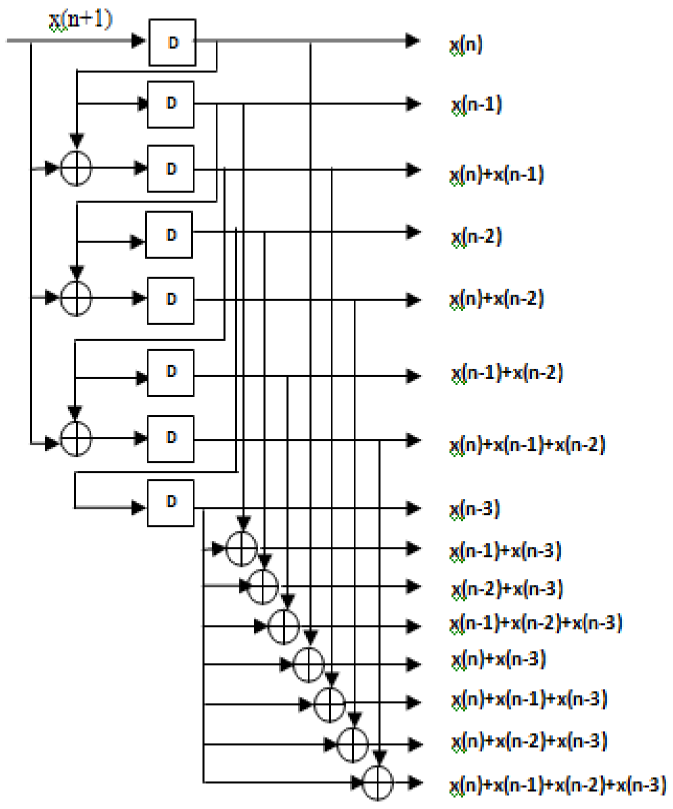

Fig. 3. Proposed structure of DA table with 8 delay elements

The sum word is shifted right by one position right and added to the carry to generate the filter output ***y(n-1)*** which is then subtracted from the desired signal d(n) to obtain the error ***e(n)***. After that the sign-magnitude separator is used to separate the sign bit and magnitude bits from the obtained error.

If r_6=1 then t= "000";
Else if r_5=1 then t="001";
Else if r_4=1 then t="010";
Else if r_3=1 then t="011";
Else if r_2=1 then t="100";
Else if r_1=1 then t="101";
Else if r_0=1 then t="110";
Else then t="111"
r=abs (μ_e(n-2)
r_i= *i* th bit of 7-bit word r

Fig.4. Logic used for the generation of control word t for the barrel shifter for L=8

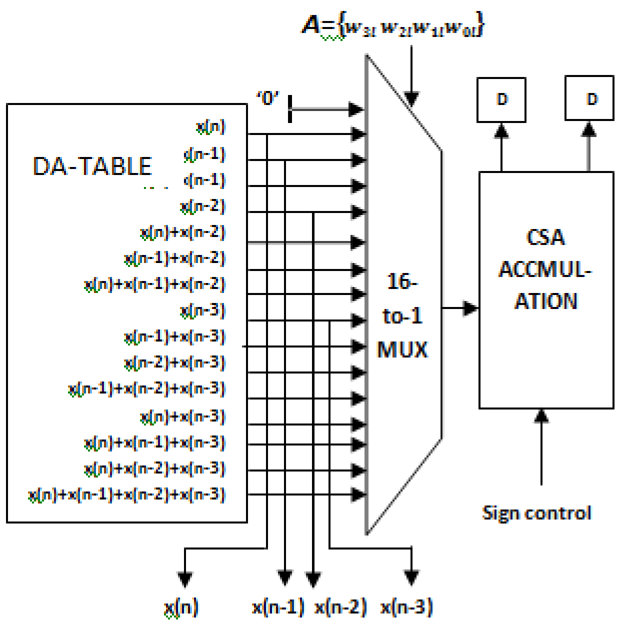

Fig. 5. Structure of four point inner product block

The magnitude bits are used by the control word generator to generate the control word 't' for the barrel shifter. The logic used for the generation of control word 't' for barrel shifter is shown in the figure 4.

The convergence factor 'μ' is taken as 1/N. Generally, we can take μ as 2^{-i}/N, where 'i' is a small integer. The weight increment unit for N=4 consist of four barrel shifters and four adder/subtractor cells. The structure of weight increment cell is shown in figure 6.

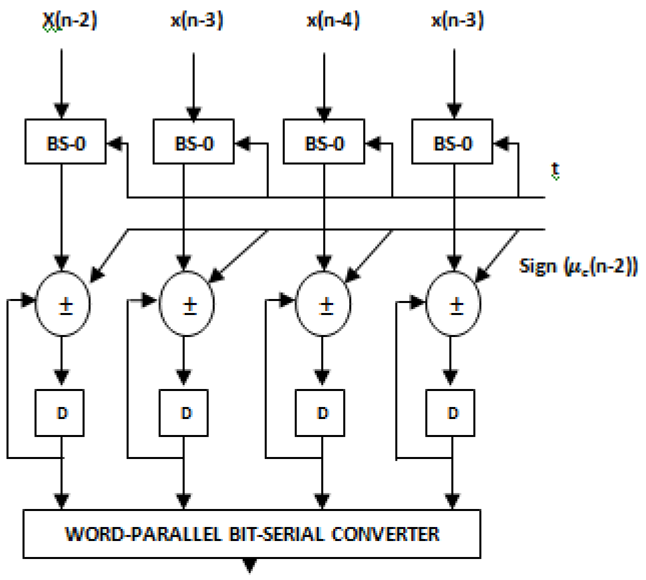

Fig. 6. Structure of weight increment block for N=4

In general Barrel Shifter is the most helpful unit in Digital Signal Processing blocks. Mostly it is used in multiplication and division operations. It performs multiplication just by right shifting the data in registers by required number of times. That means to multiply the data by 2 times, the data is shifted right by one position. Similarly division can be performed by left shifting the data as per requirement. Thus the barrel shifter reduces the complexity of operations there by area optimization has been achieved too.

Here the barrel shifter shifts the input values $\boldsymbol{x_k}$, where k=0,1,….N-1 by defined number of locations. Barrel shifter yields the number of increments to be added or subtracted from the present weights. The sign bit from the sign-magnitude separator is used as the control for adder/subtractor cells such that depending on the value of the sign bit whether it is zero or one, the barrel shifter output is respectively added or subtracted from the content of the weight corresponding current value in register.

Thus the fourth order filter implementation is done successfully with 8-DA table structure. It reduces the complexity as the basic terms like x(n), x(n-1), x(n-2), x(n-3) are more enough to generate all the other terms. So higher order filters can also be easily implemented in hardware as this design achieves low-complexity.

A. *Higher Order Filter Implementation*

The structure of higher order filter of size N=16 is shown in figure 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 L+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 N=32 can be realized. The connections between the blocks must be given properly for achieving better results.

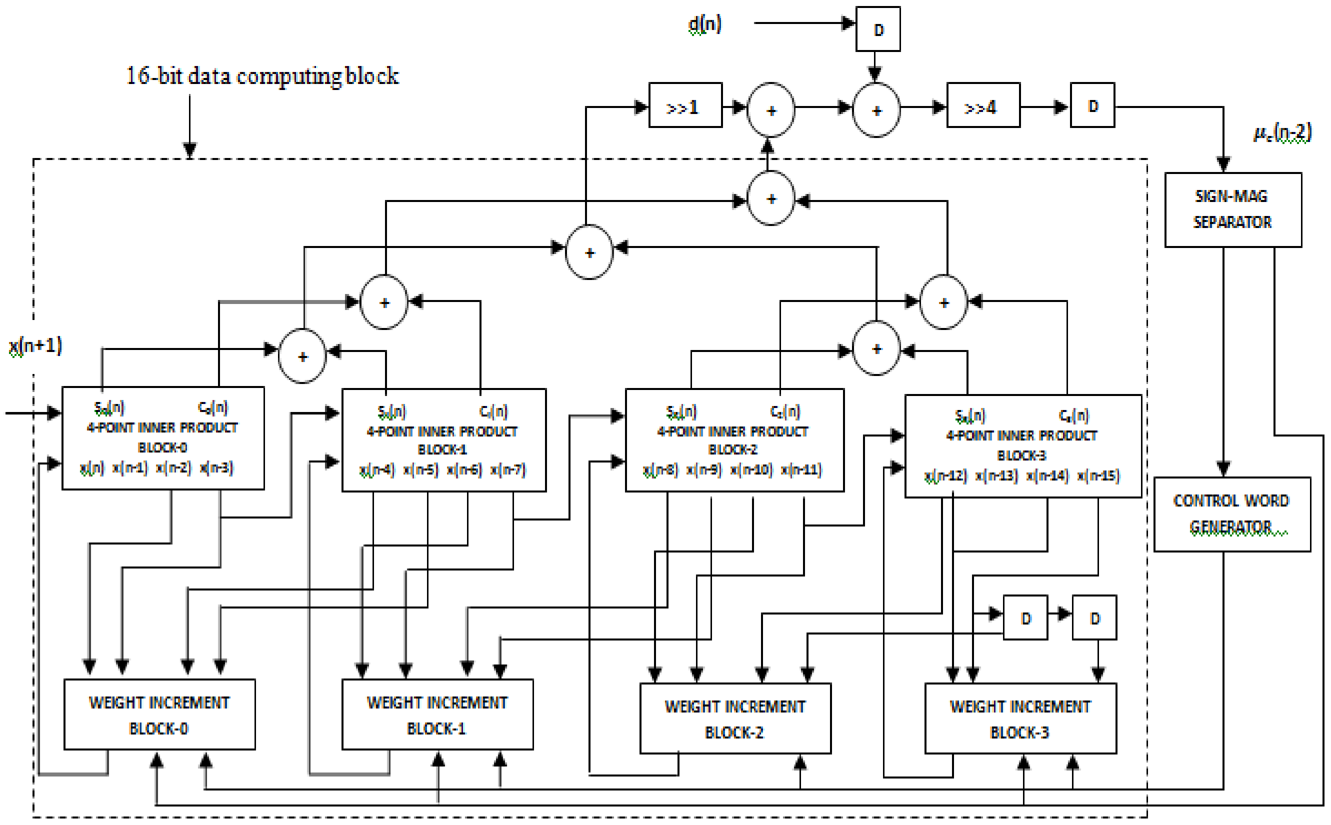
16-bit data computing block
d(n)
D
>>1
+
>>4
$\mu_e(n-2)$
SIGN-MAG SEPARATOR
CONTROL WORD GENERATOR
x(n+1)
$S_0(n)$ $C_0(n)$ 4-POINT INNER PRODUCT BLOCK-0 x(n) x(n-1) x(n-2) x(n-3)
$S_1(n)$ $C_1(n)$ 4-POINT INNER PRODUCT BLOCK-1 x(n-4) x(n-5) x(n-6) x(n-7)
$S_2(n)$ $C_2(n)$ 4-POINT INNER PRODUCT BLOCK-2 x(n-8) x(n-9) x(n-10) x(n-11)
$S_3(n)$ $C_3(n)$ 4-POINT INNER PRODUCT BLOCK-3 x(n-12) x(n-13) x(n-14) x(n-15)
WEIGHT INCREMENT BLOCK-0
WEIGHT INCREMENT BLOCK-1
WEIGHT INCREMENT BLOCK-2
WEIGHT INCREMENT BLOCK-3

Fig.7. Structure of DA-based LMS adaptive filter of length N=16 and P=4

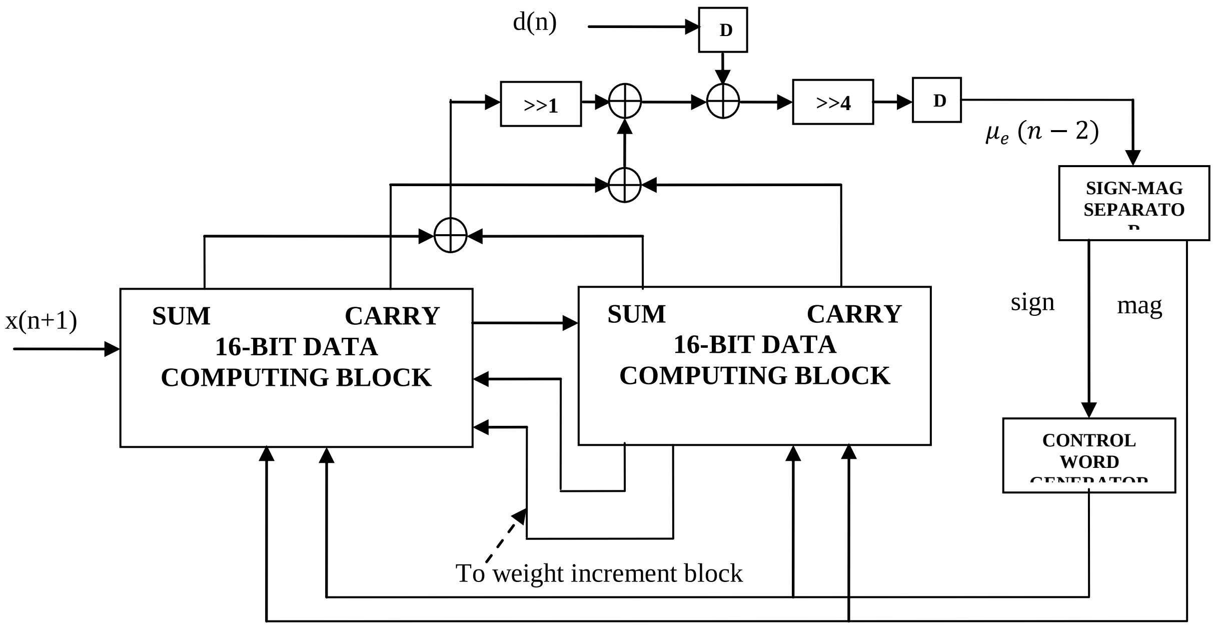
d(n)
D
>>1
>>4
$\mu_e(n-2)$
SIGN-MAG SEPARATOR
sign
mag
CONTROL WORD GENERATOR
x(n+1)
SUM CARRY 16-BIT DATA COMPUTING BLOCK
SUM CARRY 16-BIT DATA COMPUTING BLOCK
To weight increment block

Fig. 8. Structure of DA-based LMS adaptive filter of length N=32 and P=4

VI. SYNTHESIS RESULTS

TABLE I. SYNTHESIS REPORT

Design	**Filter Length, *N***	**No. of slices**	**No. of slice flip-flops**	**No. of 4-i/p LUTs**
	Availability	***4656***	***9312***	***9312***
Existing Design with an LUT using 16-delay elements	4	252	271	469
	8	406	464	733
	16	838	912	1509
	32	1287	1376	2313
Proposed Design with an LUT using 8-delay elements	4	267	204	494
	8	437	330	783
	16	901	644	1609
	32	1382	974	2463

In this section we have implemented the proposed design with VHDL and synthesized our design using Xilinx ISE 13.2. The results obtained are tabulated above. From the above results it is clearly noticed that the hardware complexities are reduced significantly.

VII. CONCLUSION

Low-complexity design implementations have greater importance in efficient hardware implementations. In this brief an architecture for the implementation of adaptive FIR filters is proposed. This design achieved low-complexity compared to existing DA-based implementations.

REFERENCES

[1] S. Haykin and B. Widrow, Least-Mean-Square Adaptive Filters. Hoboken, NJ, USA: Wiley, 2003.

[2] S. A. White, “Applications of the distributed arithmetic to digital signal processing: A tutorial review,” IEEE ASSP Mag., vol. 6, no. 3, pp. 4–19, Jul. 1989.

[3] 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.

[4] Sang Yoon Park and Promod Kumar Meher, “Low- power, High-Throughput, and Low-area Adaptive FIR Filter Based on Distributed Arithmetic,” IEEE Trans. On Circuits and Systems-II, Express Briefs, Vol.60, no.6, pp.346-350, June 2013.R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[5] W. Huang and D.V. Anderson, “Modified Sliding-Block Distributed Arithmetic with Offset Binary Coding for Adaptive Filters,” *Journal of Signal Processing Systems*, April 13, 2010.M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.