International Journal of scientific research and management (IJSRM)
||Volume||3||Issue||1||Pages|| 2010-2014 ||2015||
Website: www.ijsrm.in ISSN (e): 2321-3418

Low-Power and Low-Area Adaptive FIR Filter Based on DA Using FPGA

B.Doss[1], K.Soundararajan[2], Y. Narasimha Murthy[3]

[1]Dept. of Electronics and Communication Engineering, JNTUACEA, India

[2]Principal , KITES Engineering College, India

[3]READER, SSBN Degree and PG College, India

Abstract- **This paper presents an innovative pipelined architecture for the implementation of adaptive filter based on distributed arithmetic (DA) with low-area, low-power. The high throughput rate of the proposed design is achieved by updating the lookup table simultaneously and parallel implementation of filtering and weight-update operations. In order to reduce the sampling period and area complexity, the proposed method uses conditional signed carry-save accumulation for the purpose of DA-based inner-product computation in the place of conventional adder-based shift accumulation. In order to reduce the power consumption, the proposed design uses a faster bit clock for carry-save accumulation but it uses a much slower clock for the remaining operations. The proposed design involves the same number of multiplexers but a smaller LUT and the number of adders used reduces to half when compared to the existing DA-based design.**

Keywords **– Adaptive FIR filter, Distributed Arithmetic, carry-save accumulation, Adaptive LMS algorithm.**

INTRODUCTION

Adaptive filters are the most important and useful blocks in the several digital signal processing applications. The most popular adaptive filter is the tapped-delay line finite impulse response (FIR) whose weights are updated by the well known Widrow-Hoff least mean square (LMS) algorithm. Its popularity is not only due to its simplicity but also due to its better convergence performance [2]when compared to the other algorithms.

The direct form configuration on the forward path of the FIR filter results in a long critical path due to an inner product computation to obtain a filter output. Hence there is a necessity to reduce the critical path of the structure, when the input signal has a high sampling rate so that the critical path could not exceed the sampling period.

Of late, the multiplier less distributed arithmetic (DA) - based technique [4] has gained much popularity because of its high-throughput processing capability and regularity which results in economic and area-time efficient computing structures. Hardware-efficient DA-based design of adaptive filter has been suggested by Allred et al [4] using two separate lookup tables (LUTs) for filtering and weight update. However, the above structure do not support high sampling rate since they involve several cycles for LUT updates for each new sample. In this paper, we have proposed an efficient architecture for high speed DA-based adaptive filter with very low adaptation delay.

This paper proposes an innovative DA-based architecture for low-power, low-area and high-throughput pipelined implementation of adaptive filter with very low adaptation delay. The key points of this paper are as follows:

1.By updating the Look Up Tables (LUTs) concurrently, we can increase the throughput rate significantly.

2.Concurrent implementation of filtering and weight updating further enhances the throughput rate.

3. Conventional adder-based shift accumulation is replaced by a conditional carry-save accumulation of signed partial inner products to reduce the sampling period. The bit cycle period amounts to memory access time plus 1-bit full-adder time by carry-save accumulation.

4.The use of the proposed signed carry-save accumulation also helps to reduce the area complexity of the proposed design.

5. By using a faster clock for carry-save accumulation and a much slower clock for all other operations, we can reduce the power consumption significantly.

II.REVIEW OF ADAPTIVE LMS ALGORITHM

The least mean squares (LMS) algorithm minimizes the cost function by adjusting the filter coefficients. The standard LMS algorithm performs the following operations to update the coefficients of an adaptive filter:

1.It updates the filter coefficients in the following way

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

2. Calculates the error signal $e(n)$ by using the following equation

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

3. Calculates the output signal y(n) from the adaptive filter.

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

where μ is the convergence factor of the adaptive filter, $w(n)$ is the filter coefficients vector, and $x(n)$ is the filter input vector, d(n) is the desired response

In the case of pipelined structures, the feedback error signal $e(n)$ becomes available after certain number of cycles, called "adaptation delay". The pipelined structures therefore uses the delayed error $e(n-m)$ for updating the current weights

adaptation instead of using the most recent error where 'm' is the delay. The weight update equation for such delayed LMS adaptive filter is given by

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

III. DA-BASED APPROACH FOR INNER-PRODUCT COMPUTATION

The LMS adaptive filter needs to perform inner-product computations in each cycle which contributes to most of the critical path. The inner product of the output expression of the LMS adaptive algorithm can be given as

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

Where ω_k and x_k for $0 \le k \le N$-1 form the N-point vector inputs respectively.

Each and every component of the weight vector can be expressed in 2's complement representation assuming L to be the bit length of the weight.

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

Here ω_{kl}denotes the lth bit of ω_k. Substituting eq.(5) in eq.(4) we can write an expanded form for eq.(4) given 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 order to convert the sum-of-products form of the eq.(4) into adistributed form, we have to interchange the order of summations over the indices k and l in eq.(6) , which results in the following equation

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

The inner-product given by the above equation can be computed as

$$y = \left[\sum_{l=1}^{L-1} 2^{-l} * y_l\right] - y_0, \text{ where } y_l = \sum_{k=0}^{N-1} x_k * \omega_{kl} \qquad (9)$$

Since any element of the N-point bit sequence can be either zero or one, the partial sum y_l for l = 0,1,….,L-1 can have 2^N possible values. If all the 2^N possible values of y_l are already computed and stored in a LUT, the partial sums y_l can be taken out from the LUT using the bit sequence $\{\omega_{kl}\}$ as address for computing the inner product.

The inner-product of eq.(8) can therefore be calculated in L cycles of shift accumulation, followed by LUT-read operations corresponding to L number of bit slices $\{\omega_{kl}\}$ for $0 \le l \le$ L-1 as shown in fig.1. Since the shift accumulation in fig.1 involves significant critical path, we perform the shift accumulation using carry-save accumulator as shown in fig.2. The bit slices of vector ω are fed one by one in the LSB to MSB order to the carry-save accumulator. However, the 2's complement of the LUT output needs to be accumulated in the case of MSB slices. Therefore, all the output bits of LUT are passed through the XOR gates with a sign-control which is set to one only when the MSB slice appears as an address. Thus the XOR gates produce the one's complement of the LUT output corresponding to the MSB slice they won't affect the output of other bit slices. Finally, the sum and carry words obtained as result after L clock cycles are added by the final adder and the input carry to the final adder is set to one to account for 2's complement operation of the LUT output corresponding to the MSB slice.

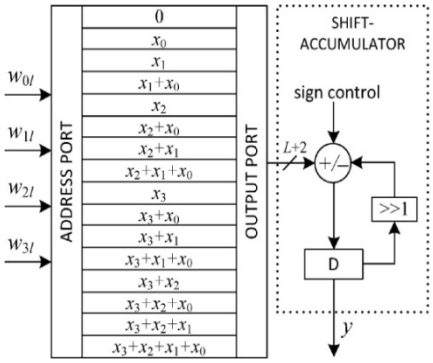

Fig.1 Conventional DA-based implementation of four-point inner product.

The content of the kth LUT location can be expressed as

$$C_k = \sum_{j=0}^{N-1} x_j * k_j \qquad (9)$$

Here k_j is the (j+1)th bit of N-bit binary representation of integer k for $0 \le k \le 2^N$-1. Here C_k for $0 \le k \le 2^N$-1 can be precomputed and stored in a LUT of 2^N words, which is RAM-based. However, instead of storing 2^N words in LUT, we store (2^N-1) registers. An example for such a DA table for N=4 is as shown in fig.3. which contains only 15 registers to store the precomputed sums of input words. Seven new valuesof C_k are computed by seven adders in parallel.

IV. DA-BASED ADAPTIVE FILTER STRUCTURE

In order to compute the large ordered adaptive filters for our convenience, we are decomposing them into small adaptive filtering blocks, since Distributed Arithmetic-based implementation of inner product of long vectors requires a very large LUT [5].Hence we are discussing the 4th order DA-based adaptive filter followed by the large orders in the following sections.

A.Proposed Structure for 4th order Adaptive filter

The proposed structure for 4th order adaptive filter (N=4) is shown in the fig. 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. consists of a DA-table which has an array of 15 registers which is capable of storing the partial inner products ***'y'***, for $0 < l \le 15$ and a 16:1 multiplexer is used to select one of those registers at any particular instant.

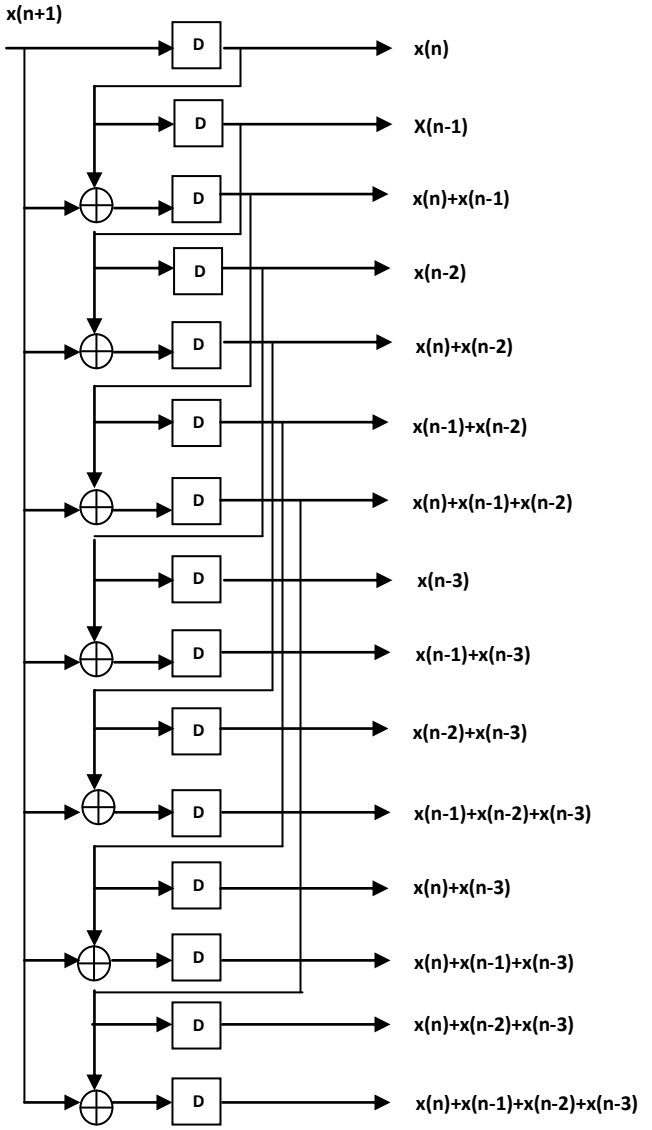

Fig. 2 DA table for generation of possible sums of input samples.

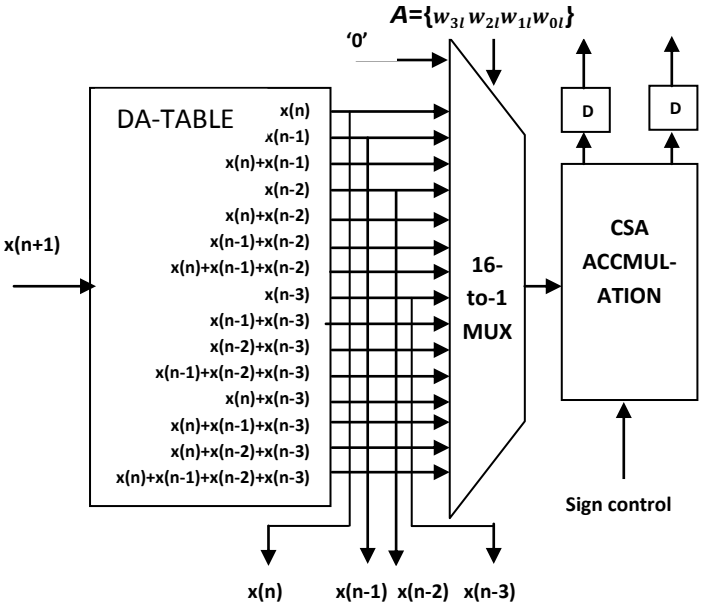

Fig. 3 Structure of four point inner product block

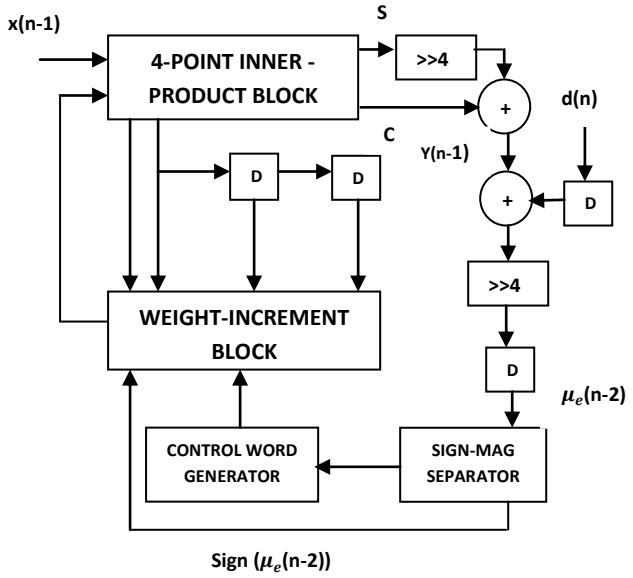

Fig.4 Structure of DA-based LMS adaptive filter of filter length N=4

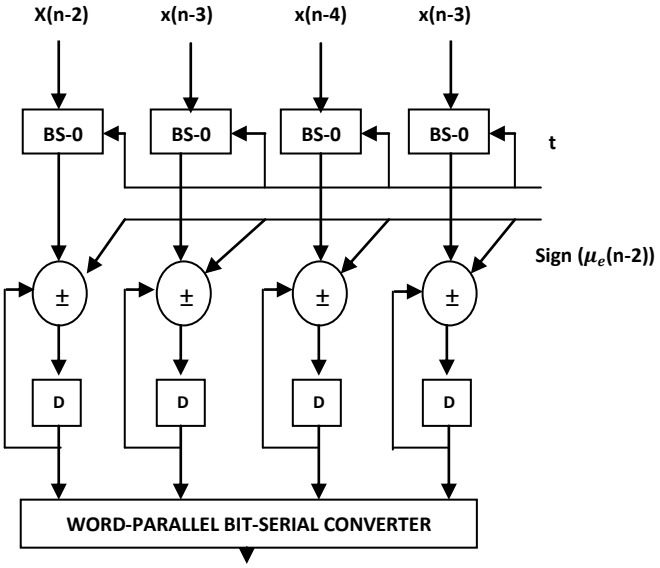

Fig.5 Structure of weight increment block for N=4

Weights A={w_3,w_2,w_1,w_0} for $0 \le l \le L-1$ 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 shift accumulates all the partial inner products and generates a sum word and a carry word of size L+2 bit length each.

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.

The magnitude bits are used by the control word generator to generate the control word *'t'* for the barrel shifter.

d(n)

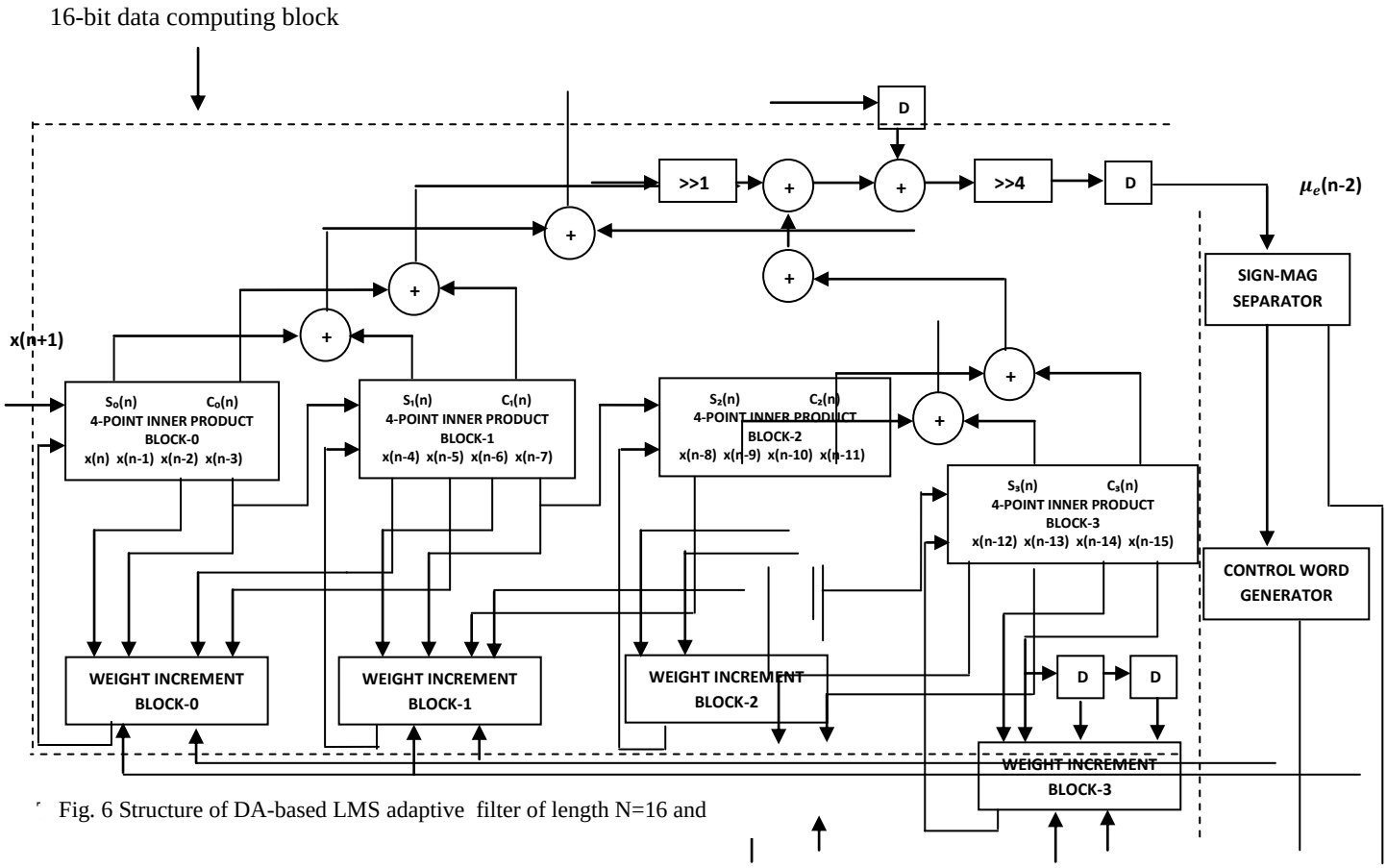

Fig. 6 Structure of DA-based LMS adaptive filter of length N=16 and

The logic used for the generation of control word ***'t'*** for barrel shifter is shown in the fig.7

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 consists of four barrel shifters and four adder/subtractor cells. 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 to or subtracted from the content of the corresponding current value in the weght register. We can extend the structure of DA-based LMS adaptive filter to length N = 16 and N = 32

B. Proposed structure for higher order Adaptive filter with N=16 and N=32

Now, the help of 4th order adaptive FIR filter, we can design adaptive filters of length N=16 and N=32.

In order to implement the 16th order adaptive FIR filter, we are making use of four sets of 4-point inner product blocks and weight increment blocks and they are connected appropriately as shown in the fig.6. The sub block consisting of these 4-point inner product blocks and weight increment blocks is known as 16-bit data computing block since it computes the 16-bit sum and carry words which are used in the further processing steps.

In order to implement the 16th order adaptive FIR filter, we are making use of four sets of 4-point inner product blocks and weight increment blocks and they are connected appropriately as shown in the fig.6. The sub block consisting of these 4-point inner product blocks and weight increment blocks is known as 16-bit data computing block since it computes the 16-bit sum and carry words which are used in the further processing steps.

Four carry-in bits should be added to sum words which are output of four 4-point inner-product blocks. 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. Outputs of sign-magnitude separator and control word generator are fed commonly to all the weight increment blocks. 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, we can extend the 16-bit structure to 32-bit structure by using two 16-bit data computing blocks as shown in the fig.8. By performing the binary addition of 16-bit sum and carry of the two blocks, we are able to generate sum and carry words of length N=32.

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

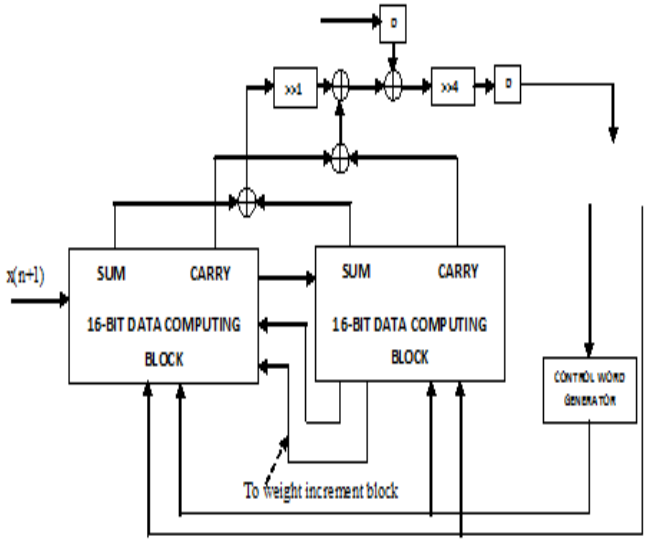

Fig.8 Proposed structure of DA-based LMS adaptive filter of length N=32 and P=4

V. SYNTHESIS RESULTS

COMPARISION OF HARDWARE COMPLEXITIES OF DIFFERENT ARCHITECTURES

Filter Length, *N*	No. of slices	No. of slice flipflops	No. of 4-i/p LUTs
4	252	271	469
8	406	464	733
16	838	912	1509
32	1287	1376	2313
Availability	4656	9312	9312

We have coded the proposed & existing designs in VHDL and synthesized by Xilinx ISE Design tool 13.2 using a 95-nm CMOS library for filter length N = 16 and N = 32 to find the area, time, and power capabilities. The FPGA used is Spartan 3E, device used is XC3S500E, package used is FG320 and speed is -4. In the above table, we have shown the synthesis results in terms of no. of slices, no. of slice flip flops and the no. of 4-input LUTs used. From the synthesis reports, we find that the power consumption was 81 milli watts and remains almost constant even though the order of the adaptive filter is increased to N = 16 and N = 32.

VI. CONCLUSION

Through this paper, we have tried to suggest an efficient and reduced LUT architectures which can provide low-power and low area implementation of DA-based adaptive filter. We have used carry-save accumulation scheme of signed partial inner products for the computation of filter output

Even though we have increased the filter order, we are successful in limiting the power $\mu_e(n-2)$ o 81 milli watts as that of the lower order structures.

FUTURE SCOPE

We can reduce the DA-table structure using Offset binary coding. With this coding technique, we can reduce LUT structure to half of its original size for reducing the physical area for efficient implementation of Distributed Arithmetic.

REFERENCES

[1] 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.

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

[3] 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.

[4] 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.

[5] .R. Guo and L. S. DeBrunner, "A novel adaptive filter implementationscheme using distributed arithmetic," in Proc. Asilomar Conf. Signals,Syst., Comput., Nov. 2011, pp.160–164.