International Journal on Applications in Information and Communication Engineering
Volume 1: Issue 6: June 2015, pp 36-41. www.aetsjournal.com ISSN (Online) : 2394-6237

Distributed Arithmetic Based Low-Power Adaptive Fir Filter Implementation

J.Jayalakshmi, T.Shanthi

Abstract— Distributed arithmetic is an efficient procedure for computing inner products between a fixed and a variable data vector. Equivalent implementation of four-point inner product and weight increament unit to produce a high throughput. Conditional signed carry-save accumulation is used in order to reduce the sampling period and area complexity. Power is reduced by using fast bit clock for carry-save accumulation but a much slower clock for all other operations. To update the weights by using least mean square(LMS) adaptation and also minimize the mean square error between the estimated and desired output. It reduces the LUT's, occupied slices, gate count for design.

Keywords— Adaptive Filter, Distributed Arithmetic (DA), Finite Impulse Response (FIR), Least Mean Square (LMS), Look-Up Table (LUT), Multiply-Accumulate (MAC), Offset-Binary Coding (OBC).

I. INTRODUCTION

Filter plays a major role for removal of unwanted signal/noise from the original signal, particularly Adaptive FIR filter is simple to attract for many applications, where it is need to minimize computational requirement. In Adaptive filter the signal and/or noise characteristics are often nonstationary and the statistical parameters vary with time. An adaptive filter has an adaptation algorithm, that is meant to monitor the environment and vary the filter transfer function accordingly. It consists of Two process. First one is Filtering process and second one is Adaptation process.

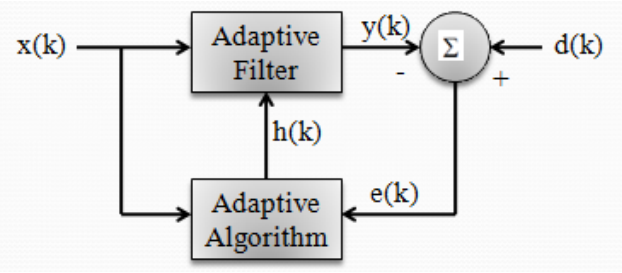

Fig. 1 Block diagram of an adaptive filter

Adaptive filter algorithms are Least Mean Square(LMS), Recursive Least Square(RLS), Normalized Least Mean Square(RLMS). The LMS algorithm are a class of adaptive filter - To mimic a desired filter by finding the filter coefficients that relate to producing the least mean squares of the error signal. The RLS algorithm involve more complicated mathematical operations and require more computational resources than LMS.

The NLMS algorithm is time varying step size algorithm and it is used in system in order to improve voice quality. Finally LMS algorithm is best suitable for reducing the power and area. LMS algorithms are widely used algorithm in adaptive filtering. The main features that attracted the use of the LMS algorithm are low computational complexity. Involves a feedback connection, although LMS might seem very difficult to work due to the randomness, the feedback acts as a low-pass filter or performs averaging so that the randomness can be filtered-out. The main advantage of the LMS algorithm is its simplicity in terms of the memory requirement.

DA is basically a bit-serial computational operation that forms an inner (dot) product of a pair of vectors in single direct step.DA efficiently implements the MAC using basic building blocks (Look Up Tables) in FPGAs. One of the vector operands is fixed, as in a digital filter, distributed arithmetic pre processes the stored data to reduce the computational complexity of the inner product calculation. By using DA instead of traditional way, a huge reduction in area can be happened. Filtering is one of the most widely used complex signal processing operations [1].

Least Mean Square (LMS) algorithm is the most popularly used adaptive filter not only due to its simplicity but also due to its satisfactory convergence performance [2]. DA is one way to implement convolution multiplier-less, where the MAC operations are replaced by a series of LUT accesses and summations. Techniques, such as ROM decomposition [3] and offset-binary coding (OBC) [4] can reduce the LUT size.

II. LMS ADAPTIVE ALGORITHMS

The LMS algorithm computes a filter a filter output and producing the error value e(n), which is equal to the difference between the current filter output and the desired response. In every cycle the filter weights are updated by using e(n).

$$w(n+1) = w(n) + \mu.e(n).x(n) \quad (1)$$

$$w(n+1) = w(n) + \mu.e(n-m).x(n-m) \quad (2)$$

Where,

$$e(n) = d(n) - y(n) \quad (3)$$

$$y(n) = w^{qT}(n) \,.\, x(n) \quad (4)$$

J.Jayalakshmi, PG Scholar (ME -VLSI Design), Department of Electronics and Communication Engineering Kings college of Engineering, Anna university, Thanjavur, Tamilnadu.

T.Shanthi, Assistant Professor , Department of Electronics and Communication Engineering Kings college of Engineering, Anna university, Thanjavur, Tamilnadu.

The input vector x(n) at the n[th] training iteration is given by,

$$x(n) = [x(n), x(n-1), \ldots\ldots\ldots\ldots . x(n-N+1)]^T \quad (5)$$

The weight vector w(n) at the n[th] training iteration is given by,

$$w(n) = [w_0(n), w_1(n), \ldots\ldots\ldots\ldots . w_{N-1}(n)]^T \quad (6)$$

III. Existing System

In existing system, conventional method is implemented for four point inner product operation. It takes large computational operation and also sampling period is increased. The LMS adaptive filter, in each cycle, needs to perform an inner-product computation which contributes to the most of the critical path. For simplicity of presentation, let the inner product is given by,

$$y = \sum_{n=0}^{N-1} c[n].x[n] \quad (7)$$

Where c[n] and x[n] for $0 \le n \le N-1$ form the *N*-point vectors corresponding the current weights and most recent $N-1$ input, respectively. Assuming B to be the bit width of the weight, each component of the weight vector may be expressed in two's complement representation.

$$x[n] = -x[n_0] + \sum_{b=1}^{B-1} x[n_b].2^{-1} \quad (8)$$

where x_{nb} denotes the *b*th bit of x_n . Substituting (8), we can write (7) in an expanded form

$$y = -\sum_{n=0}^{N-1} x[n_0].c[n] + \sum_{b=0}^{B-1} 2^{-b}.\sum_{n=0}^{N-1} x[n_1].c[n] \quad (9)$$

The inner product can be computed as,

$$y = \sum_{b=0}^{B-1} 2^{-b}.y_1 - y_0 \quad (10)$$

Where,

$$y_1 = \sum_{n=0}^{N-1} c[n].x[n_1]$$

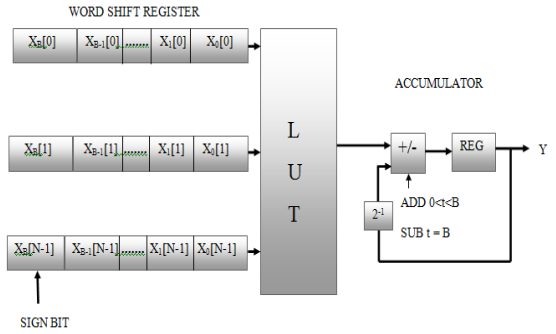

Fig. 2 Conventional DA based implementation of four – point inner product

Since any element of the *N*-point bit sequence $\{x[n_l]$ for $0 \le n \le N-1\}$ can either be zero or one, the partial sum yl for $l = 0, 1, \ldots, L-1$ can have 2*N* possible values. If all the 2*N* possible values of y_l are pre-computed and stored in a LUT, the partial sums y_l can be read out from the LUT using the bit sequence $\{xn_l\}$ as address bits for computing the inner product .

The inner product of (10) can therefore be calculated in L cycles of shift accumulation, followed by LUT-read operations corresponding to L number of bit slices $\{xn_l\}$ for $0 \le l \le L-1$, as shown in Fig. 2.

IV. Proposed System

A. Da Based Adaptive Fir Filter Structure For N=8

The computation of adaptive filters of large orders needs to be decomposed into small adaptive filtering blocks. since DA based implementation of inner product of long vectors requires a very large LUT [5]. The proposed structure of DA-based adaptive filter of length $N = 8$ is shown in Fig. 3.

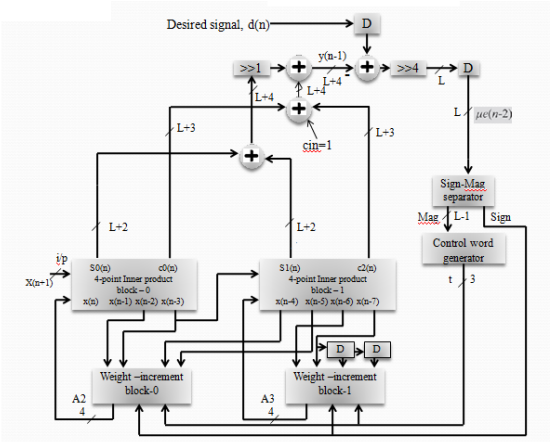

Fig. 3 Proposed structure of DA-based LMS adaptive filter of length $N = 8$.

B. Da Based Adaptive Fir Filter Structure For N=16

The inner-product computation can be decomposed into *N/P* (assuming that $N = PQ$) small adaptive filtering blocks1 of filter length *P* as

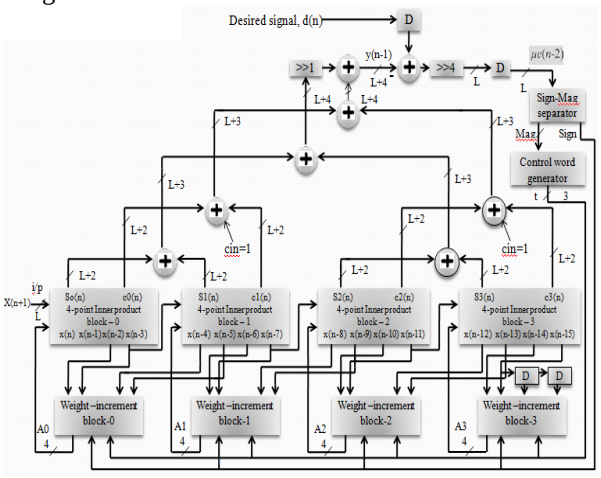

Fig. 4 Structure of DA-based LMS adaptive filter of length $N = 16$ and $P = 4$.

$$y = \sum_{n=0}^{P-1} c[n] . x[n] + \sum_{n=P}^{2P-1} c[n] . x[n] \ldots . + \sum_{n=N-P}^{N-1} c[n] . x[n] \quad (11)$$

Each of these *P*-point inner-product computation blocks will accordingly have a weight-increment unit to update *P*

weights. The proposed structure for $N = 16$ and $P = 4$ is shown in Fig. 4.

The content of the kth LUT location can be expressed as

$$_{k} = \sum_{j=0}^{N-} .k_j \quad (1$$

where k_j is the $(j + 1)$th bit of N-bit binary representation of integer k for $0 \le k \le 2^N - 1$. Note that c_K for $0 \le k \le 2^N - 1$ can be precomputed and stored in RAM-based LUT of 2^N words. However, instead of storing 2^N words in LUT, we store $(2^N - 1)$ words in a DA table of $2^N - 1$ registers. An example of such a DA table for $N = 4$ is shown in Fig. 5. It contains only 15 registers to store the precomputed sums of input words. Seven new values of c_k are computed by seven adders in parallel.

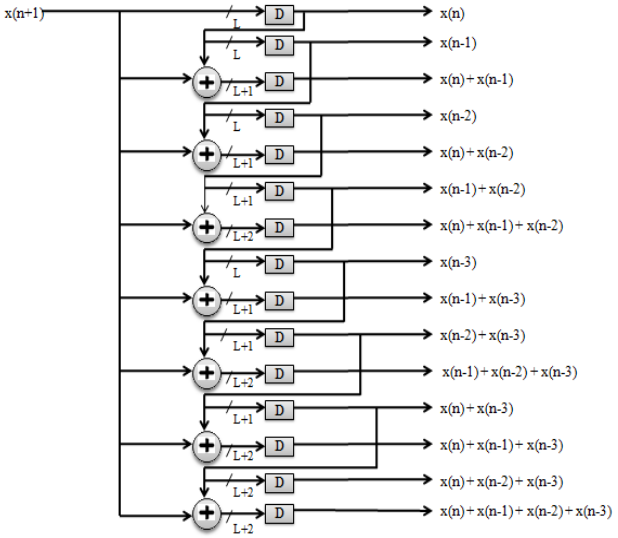

Fig. 5 DA table for generation of possible sums of input sample

C. Four-Point Inner Product

The four-point inner-product block [shown in Fig. 6] includes a DA table consisting of an array of 15 registers which stores the partial inner products yl for $0 < l \le 15$ and a 16 : 1 multiplexer (MUX) to select the content of one of those registers. Bit slices of weights $A = \{w3l\ w2l\ w1l\ w0l\}$ for $0 \le l \le L - 1$ are fed to the MUX as control in LSB-to-MSB order, and the output of the MUX is fed to the carry-save accumulator (shown in Fig. 7).

D. Carry-Save Accmulation

Carry-save arithmetic is frequently used to realize basic arithmetic operations requiring inner product calculations, as multiplication, multiply-add, multiply accumulate, digital filters and cryptography. The bit slices of vector **w** are fed one after the next in the Least Significant Bit (LSB) to the Most Significant Bit (MSB) order to the carry-save accumulator.

However, the negative (two's complement) of the LUT output needs to be accumulated in case of MSB slices. Therefore, all the bits of LUT output are passed through XOR gates with a sign-control input which is set to one only when the MSB slice appears as address.

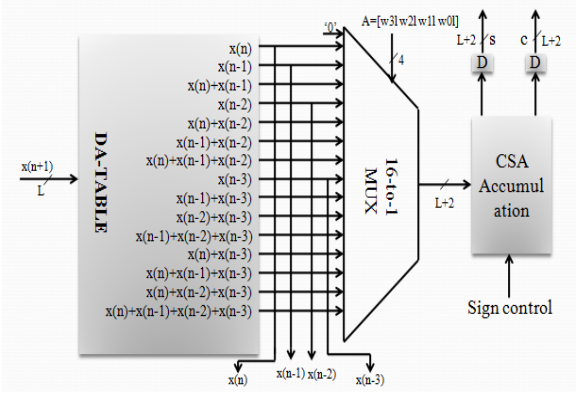

Fig. 6 Structure of the four-point inner-product block for N=4

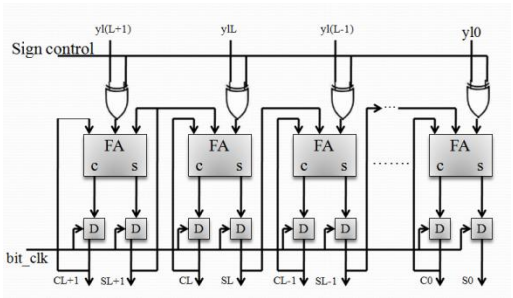

Fig. 7 Carry – save implementation of shift accumulation

These arithmetic operations can be realized by generating partial products, which have to be added up. Carry save arithmetic provides digit parallel adder operations. Using carry-save (CS) arithmetic to add the partial products, to "reduce" them, leads to a tree-like reduction structure with a competitive low critical path latency. Different CS reduction strategies have been proposed, like the well known Wallace and Dadda reduction strategies. We focus on improving these CS reduction strategies to decrease the critical path latency, the area, and the power consumption of the implemented arithmetic unit. Using carry save addition, the delay can be reduced further still.

The idea is to take 3 numbers that we want to add together, x + y + z, and convert it into 2 numbers c + s such that x + y + z = c + s, and do this in O(1) time. The reason why addition cannot be performed in O(1) time is because the carry information must be propagated. In carry save addition, we refrain from directly passing on the carry information until the very last step.

E. Weight Increament Unit

It consists of four barrel shifters and four adder/subtractor cells. The barrel shifter shifts the different input values x_k for $k = 0, 1, \ldots, N - 1$ by appropriate number of locations (determined by the location of the most significant one in the estimated error). The barrel shifter yields the desired increments to be added with or subtracted from the current weights. The sign bit of the error is used as the control for adder/subtractor cells such that, when sign bit is zero or one, the barrel-shifter output is respectively added with or

subtracted from the content of the corresponding current value in the weight register.

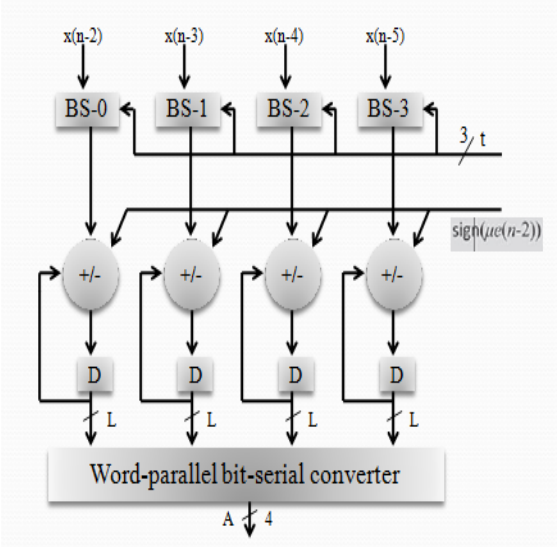

Fig. 8 Structure of the weight-increment block for $N = 4$

```
if r6 = 1 then t = "000";
else if r5 = 1 then t = "001";
else if r4 = 1 then t = "010";
else if r3 = 1 then t = "011";
else if r2 = 1 then t = "100";
else if r1 = 1 then t = "101";
else if r0 = 1 then t = "110";
else then t = "111";

r = abs(μe(n-2))
ri : ith bit of 7-bit word r
```

Fig. 9 Logic used for generation of control word *t* for the barrel shifter.

V. Simulation Results for N=8

A. Conventional Distributed Arithmetic Based Lms Adaptive Filter Implementation

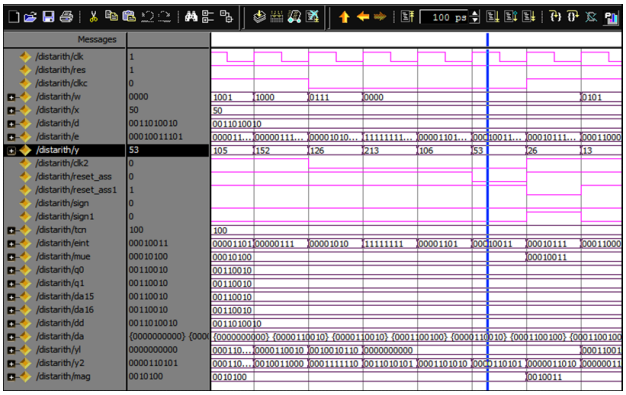

Fig. 10 Simulation of Conventional method(N=8)

B. Conditional Carry Save Accumulation Distributed Arithmetic Based Lms Adaptive Filter Implementation

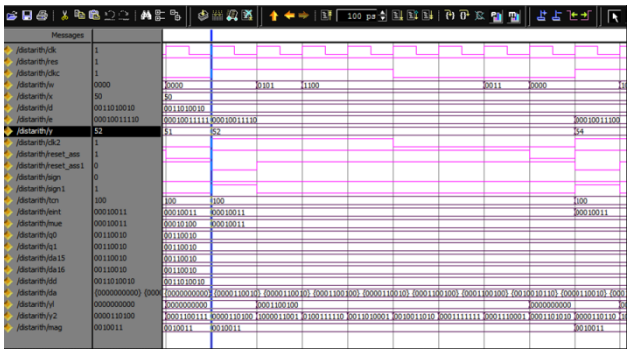

Fig. 11 Simulation of Conditional carry save accumulation method(N=8)

VI. Synthesis Results for N=8

A. Estimated Power For Conventional Distributed Arithmetic Based Lms Adaptive Filter Implementation

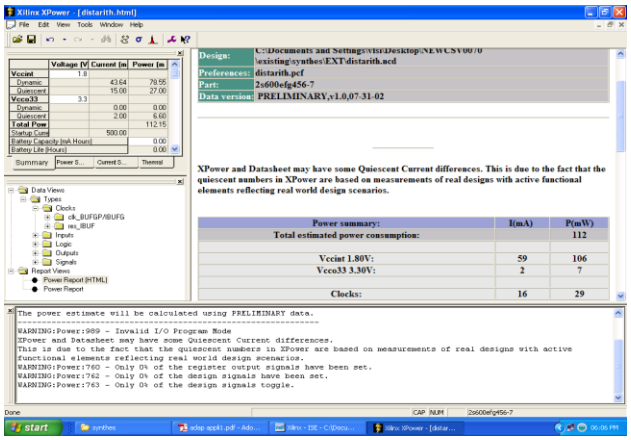

Fig. 12 Synthesize power for Conventional method(N=8)

B. Estimated Power For Conditional Carry Save Accumulation Distributed Arithmetic Based Lms Adaptive Filter Implementation

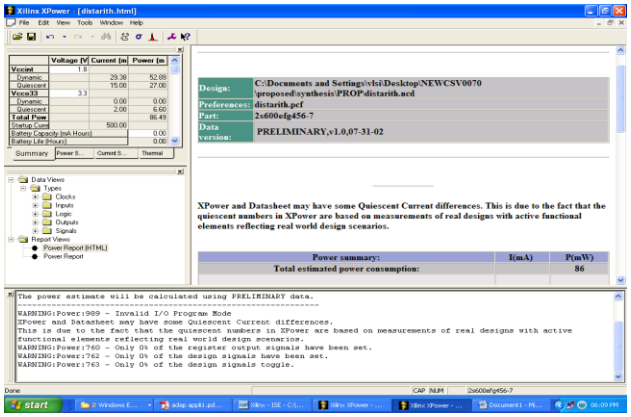

Fig. 13 Synthesize power for Conditional carry – save accumulation method(N=8)

C. Estimated Area For Conventional Distributed Arithmetic Based Lms Adaptive Filter Implementation

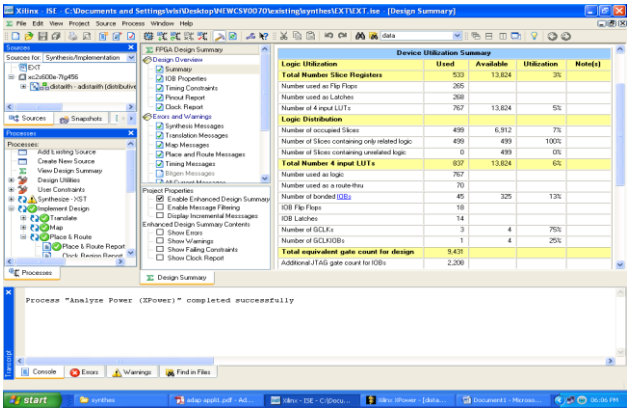

Fig. 14 Synthesize area for Conventional method(N=8)

D. Estimated Area For Conditional Carry Save Accumulation Distributed Arithmetic Based Lms Adaptive Filter Implementation

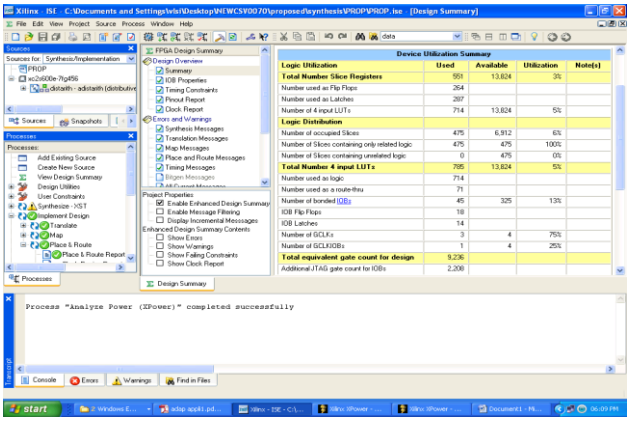

Fig. 15 Synthesize area for Conditional carry – save accumulation method(N=8)

VII. Simulation Results for N=16

A. Conventional Distributed Arithmetic Based Lms Adaptive Filter Implementation

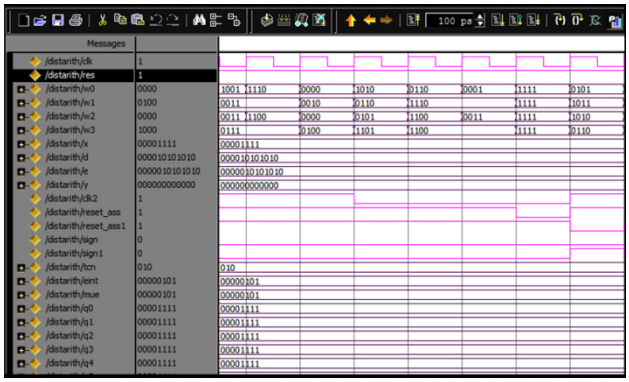

Fig. 16 Simulation of Conventional method(N=16)

B. Conditional Carry Save Accumulation Distributed Arithmetic Based Lms Adaptive Filter Implementation

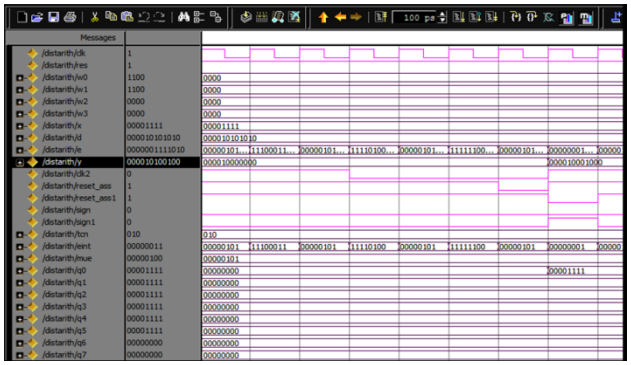

Fig. 17 Simulation of Conditional carry – save accumulation method(N=16)

VIII. Synthesis Results for N=16

A. Estimated Power For Conventional Distributed Arithmetic Based Lms Adaptive Filter Implementation

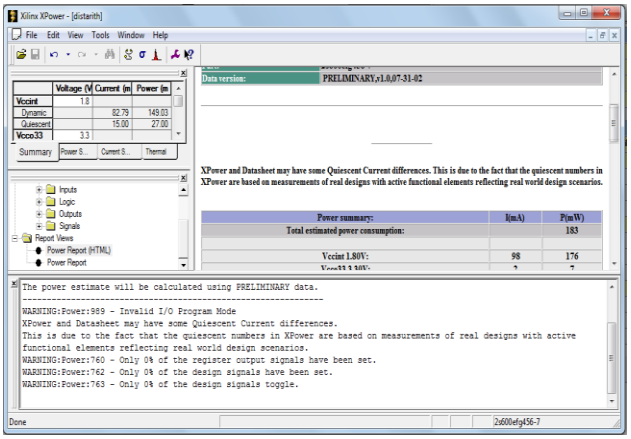

Fig. 18 Synthesize power for Conventional method(N=16)

B. Estimated Power For Conditional Carry Save Accumulation Distributed Arithmetic Based Lms Adaptive Filter Implementation

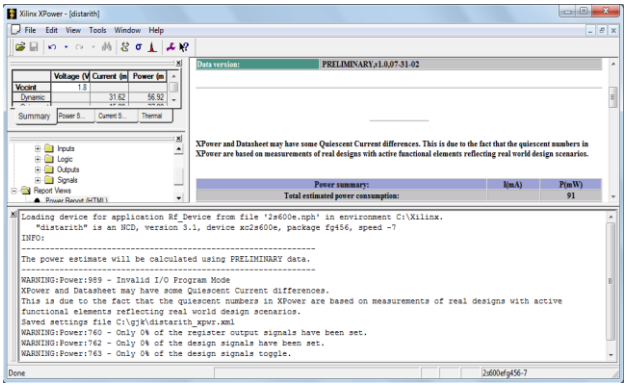

Fig. 19 Synthesize power for Conditional carry – save accumulation method(N=16)

C.Estimated Area For Conventional Distributed Arithmetic Based Lms Adaptive Filter Implementation

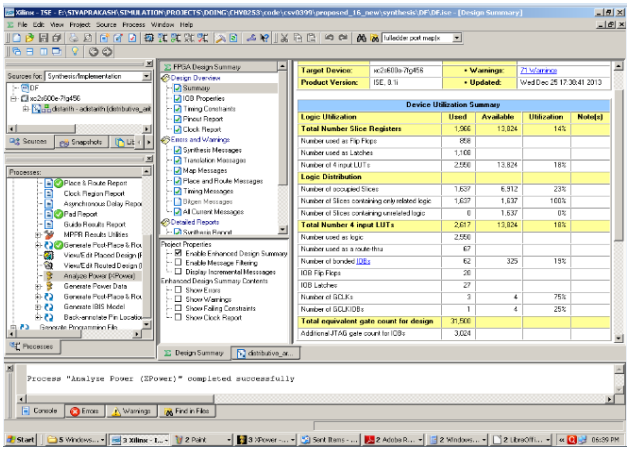

Fig. 20 Synthesize area for Conventional method(N=16)

D.Estimated Area For Conditional Carry Save Accumulation Distributed Arithmetic Based Lms Adaptive Filter Implementation

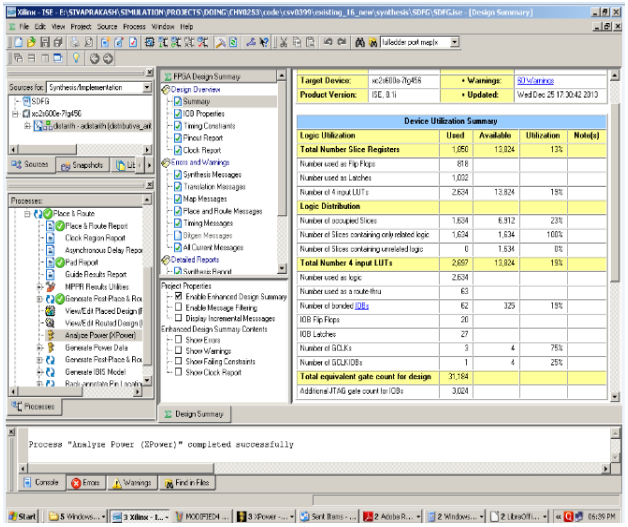

Fig. 21 Synthesize area for Conditional carry – save accumulation method(N=16)

Table I
Comparison B/W Existing And Proposed For N=8

PARAMETERS	EXISTING	PROPOSED
Number Of Lut's	767	714
Number Of Occupied Slices	499	475
Gate Count For Design	9431	9236
Power Consumption	112(Mw)	86(Mw)

Table II
Comparison B/W Existing And Proposed For N=16

PARAMETERS	EXISTING	PROPOSED
Number Of Lut's	2634	2550
Number Of Occupied Slices	1966	1650
Gate Count For Design	31,500	31,184
Power Consumption	183(Mw)	91(Mw)

IX. CONCLUSION

An efficient pipelined architecture for low-power, high-throughput, and low-area implementation of DA-based adaptive filter. From the synthesis results, the proposed design consumes 29% less power and 13% less Area over our previous DA-based FIR adaptive filter in average for filter lengths N = 8 and 16 bit. Comparison of conventional and conditional carry save accumulation for N=8 and N=16.

ACKNOWLEDGMENT

Apart from our efforts, the success of any work depends on the support and guidelines of others. We take this opportunity to express our gratitude to the people who have been supported us in the successful completion of this work. We owe a sincere prayer to the LORD ALMIGHTY for his kind blessings without which this would not have been possible

REFERENCES

[1] S.K. Mitra, "Digital Signal Processing: A Computer–Based Approach", 2nd ed, New York: McGraw–Hill Companies, 2001.
[2] S. Haykin and B. Widrow, "Least-Mean-Square Adaptive Filters", Hoboken, NJ, USA: Wiley, 2003
[3] K.K. Parhi, "VLSI Digital Signal Processing Systems Design and Implementation", John Wiley, 1999.
[4] S.A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review", IEEE ASSP Magazine, vol. 6, pp. 4–19, July, 1989.
[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] R. Guo and L. S. DeBrunner, "Two high-performance adaptive filter implementation schemes using distributed arithmetic", IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 58, no. 9, pp. 600–604, Sep. 2011.
[7] P. K. Meher and S. Y. Park, "High-throughput pipelined realization of adaptive FIR filter based on distributed arithmetic", in VLSI Symp. Tech. Dig., Oct. 2011, pp. 428–433.
[8] R. Guo and L. S. DeBrunner, "A novel adaptive filter implementation scheme using distributed arithmetic", in Proc. Asilomar Conf. Signals, Syst., Comput., Nov. 2011.