RESEARCH ARTICLE OPEN ACCESS

Low Power Adaptive FIR Filter Based on Distributed Arithmetic

S Ramanathan[1], Gorty Anand[1], Prasanth Reddy[1], Prof. Sri Adibhatla Sridevi[2]
[1]*(M.Tech, VLSI Design, School of Electronics Engineering (SENSE), VIT University, Vellore)*
[2]*(Associate Professor, Micro & Nanoelectronics, School of Electronics Engineering (SENSE), VIT University, Vellore)*

ABSTRACT
This paper aims at implementation of a low power adaptive FIR filter based on distributed arithmetic (DA) with low power, high throughput, and low area. Least Mean Square (LMS) Algorithm is used to update the weight and decrease the mean square error between the current filter output and the desired response. The pipelined Distributed Arithmetic table reduces switching activity and hence it reduces power. The power consumption is reduced by keeping bit-clock used in carry-save accumulation much faster than clock of rest of the operations. We have implemented it in Quartus II and found that there is a reduction in the total power and the core dynamic power by 31.31% and 100.24% respectively when compared with the architecture without DA table.
Keywords: Adaptive filter, Distributed Arithmetic, Finite Impulse Response, Least Mean Square algorithm, Lookup Table

I. INTRODUCTION

An adaptive filter tries to model the relationship between two real time signals using an iterative approach [1].

Four aspects defines an adaptive filter:

1) The input signals of the filter
2) The impulse response of the filter
3) The filter coefficients
4) The adaptive algorithm that is used to adjust the weights

The LMS algorithm given by Widrow-Hoff is used to update the tapped-delay line FIR filter's weights because of its simplicity and convergence performance provided by it [2].

DA based technique without multipliers [3], are used as they provide high-throughput processing capability and regularity that results in computing structures that are cost-effective and area-time efficient [4].

II. THE ADAPTIVE FILTER

In the block diagram shown in Fig.1 a sample of x(n), the input signal, is fed to adaptive filter and the filter outputs y(n). This output is then compared with d(n), the desired response, and the difference of the two is the error signal e(n) as shown in (1).

$$e(n) = d(n) - y(n) \quad (1)$$

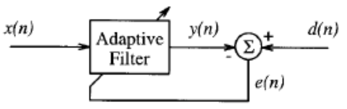


Fig. 1. The general adaptive filter

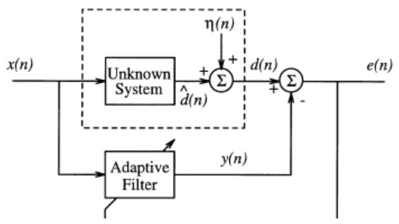


Fig. 2. System identification using adaptive filter

The error signal is fed to the adaptive filter which, in a well-defined manner, updates the coefficients of the filter from time n to time (n + 1). Through this adaptation process, as n increases, the magnitude of e (n) decreases, or in other words, the output of the adaptive filter tends to the desired response signal.

Let x (n) be the input to an unknown system and let đ (n) be its corresponding output. Then, the desired response signal is given by

$$d(n) = \hat{d}(n) + \eta(n) \quad (2)$$

The role of adaptive filter here is to precisely represent đ (n) at its output. It can be said that the adaptive filter driven by x (n), has accurately modeled the unknown system if y (n) = đ (n).

III. LMS ADAPTIVE ALGORITHM

In every cycle, the LMS algorithm calculates an output and the corresponding error value. It then uses the estimated error to update the weights in every cycle. The LMS adaptive filter weights in nth iteration are updated as per the following equations.

$$w(n + 1) = w(n) + \mu.e(n).x(n) \quad (3a)$$

Where

$e(n) = d(n) - y(n)$ (3b)

$y(n) = w^T(n).x(n)$ (3c)

At the nth iteration, x(n) the input vector and w(n) the weight vector, are respectively given by,

$x(n) = [x(n), x(n-1),\ldots,x(n-N+1)]^T$ (4a)

$w(n) = [w_0(n),w_1(n),\ldots,w_{N-1}(n)]^T$ (4b)

In the above equations, d(n) denotes the desired response, y(n) denotes the filter's output in the nth iteration, e(n) is the error computed in the nth iteration and it is used for updating the weights, μ is the convergence-factor and N is the length of the filter.

In pipelined designs, only after certain number of cycles, the feedback error e(n) becomes available, this deley is known as the "adaptation-delay". Hence the pipelined architectures use e(n - m) which is the delayed error in place of recent-most error to update the current weight. Here m denotes the adaptation-delay. The following equation is the weight-update equation of such delayed LMS adaptive filter

$w(n+1) = w(n) + \mu.e(n-m).x(n-m)$ (5)

IV. DA-BASED APPROACH FOR INNER-PRODUCT COMPUTATION

In every cycle, the LMS adaptive filter has to compute an inner-product and this contributes to the most of the critical-path. Let the inner-product of (3c) be given by (6) for simplicity of presentation, and is in fact the arithmetic sum of products which defines the response of linear time-invariant (LTI) network.

$$y = \sum_{k=0}^{N-1} w_k.x_k \quad (6)$$

Where x_k and w_k form the N-point vectors corresponding the recent-most N-1 input and current weights respectively for $0 \le k \le N-1$. Assuming the bit-width of the weight to be L, each component of weight vector can be expressed in 2's complement representation as shown in (7).

$$w_k = -w_{k0} + \sum_{l=1}^{L-1} w_{kl}.2^{-l} \quad (7)$$

where w_{kl} is the lth bit of w_k. Substituting (7) in (6), we get

$$y = \sum_{k=0}^{N-1} x_k [-w_{k0} + \sum_{l=1}^{L-1} w_{kl}.2^{-l}] \quad (8)$$

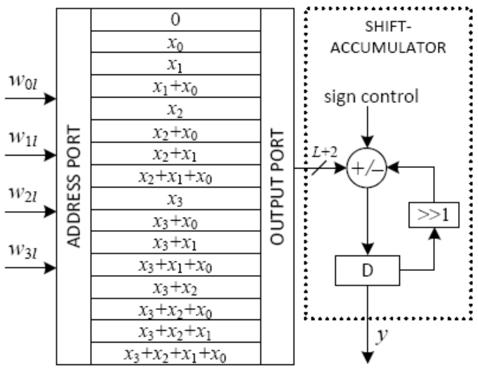


Fig. 3. Conventional DA-based implementation of 4-point inner-product.

rewriting (8), we get

$$y = -\sum_{k=1}^{N} w_{k0}.x_k + \sum_{k=1}^{N} \sum_{l=1}^{L-1} w_{kl}.x_k.2^{-l} \quad (9)$$

expanding (9), we get (10)

$$\begin{aligned} y = & -[w_{10}.x_1 + w_{20}.x_2 + w_{30}.x_3 + \ldots + w_{k0}.x_k] \\ & + [w_{11}.x_1 + w_{21}.x_2 + w_{31}.x_3 + \ldots + w_{k1}.x_k]\, 2^{-1} \\ & + [w_{12}.x_1 + w_{22}.x_2 + w_{32}.x_3 + \ldots + w_{k2}.x_k]\, 2^{-2} \\ & \quad \vdots \\ & + [w_{1(B-2)}.x_1 + w_{2(B-2)}.x_2 + \ldots + w_{k(B-2)}.x_k]\, 2^{-(B-2)} \\ & + [w_{1(B-1)}.x_1 + \ldots + w_{k(B-1)}.x_k]\, 2^{-(B-1)} \end{aligned} \quad (10)$$

Every term inside the brackets is actually a binary AND operation that involves all the bits of the constant and one bit of the input and the plus signs are arithmetic sum operations. An LUT can now be constructed which can be addressed by the same scaled bit of all the input variables and can access the sum of the terms inside each pair of brackets. Fig.3 shows the LUT.

Converting the sum-of-products form of (10) into a distributed form, we get (11)

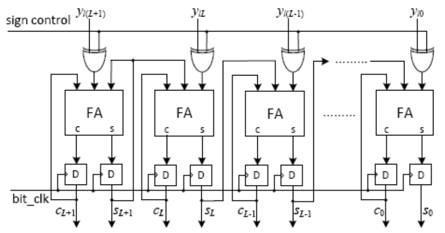


Fig. 4. Carry-save implementation of the shift-accumulation.

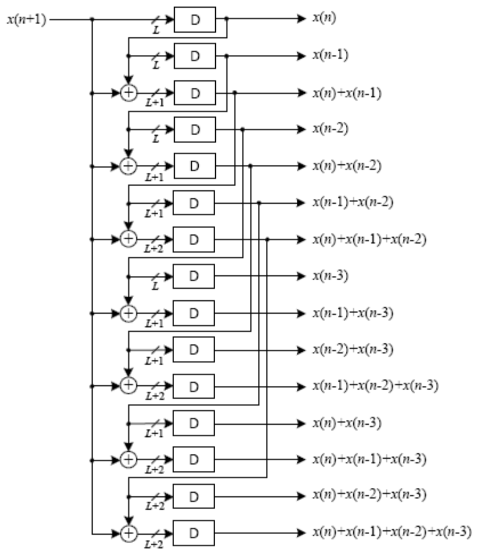


Fig. 5. DA-table for generation of the possible sums of the input samples.

$$y = -\sum_{k=0}^{N-1} w_{k0}.x_k + \sum_{k=0}^{N-1} 2^{-l} . [\sum_{l=1}^{L-1} x_k.w_{kl}] \quad (11)$$

the inner-product given by (11) is computed as

$$y = [\sum_{l=1}^{L-1} 2^{-l}.y_l] - y_0 \text{ where } y_l = \sum_{k=1}^{N-1} x_k.w_{kl} \quad (12)$$

The partial-sum y_l for l = 0, 1,2,…,L-1, can have 2^N possible values, as the elements of the N-point bit-sequence $\{w_{kl}$ for $0 \leq k \leq N - 1\}$ can be either 0 or 1.

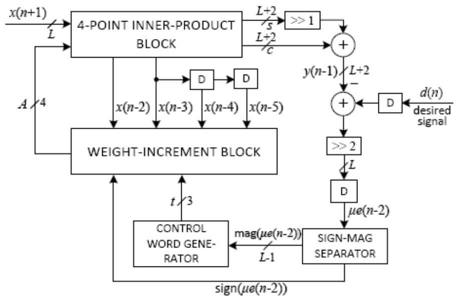


Fig. 6. The structure of DA-based LMS adaptive filter of length N = 4.

Now, using the bit-sequence $\{w_{kl}\}$ as address bits for the computation of inner-product, we can read out the partial sums y_l from the LUT, if we precompute all the 2^N possible values of y_l and store it in the LUT.

Therefore, inner-product of (12) can be calculated be calculated in L cycles of shift-accumulation which is followed by the LUT read operations corresponding to L number of bit-slices $\{w_{kl}\}$ for $0 \leq l \leq$ L-1. This is shown in Fig.3. The shift-accumulation is performed using carry-save accumulator, as the shift-accumulation shown in Fig.3 involves significant critical-path. This is shown in Fig.4. The bit-slices of vector w are given to the carry-save accumulator one after the other in the order LSB to MSB. But in case of MSB slices, the negative or the 2’s complement of the output of the LUT must to be accumulated. This could easily be achieved using XOR gates. Therefore, all the bits of LUT output are fed to XOR gates with sign-control input. If MSB slice appears as address, then alone the sign-control is set to 1. So the XOR gates produce the 1’s complement of the LUT output if MSB slice appears as address and does not affect the output for other cases. Lastly, the sum and carry words that are obtained after L clock cycles are added by an adder and the input carry of this adder is set to 1 to account for the 2’s complement operation of the LUT output for the MSB slice.

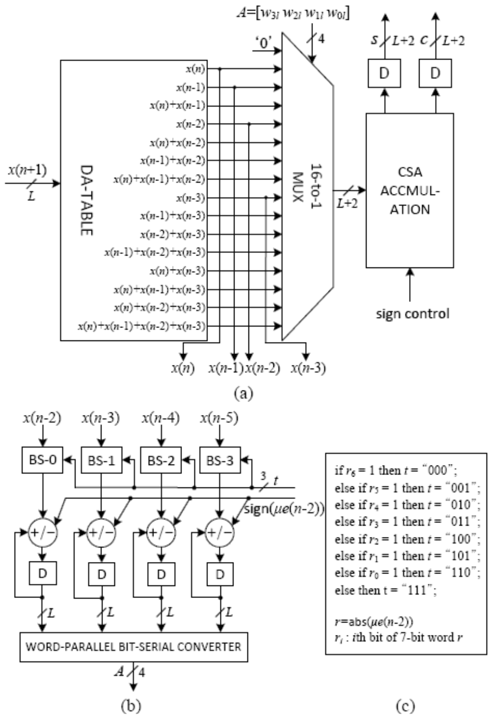
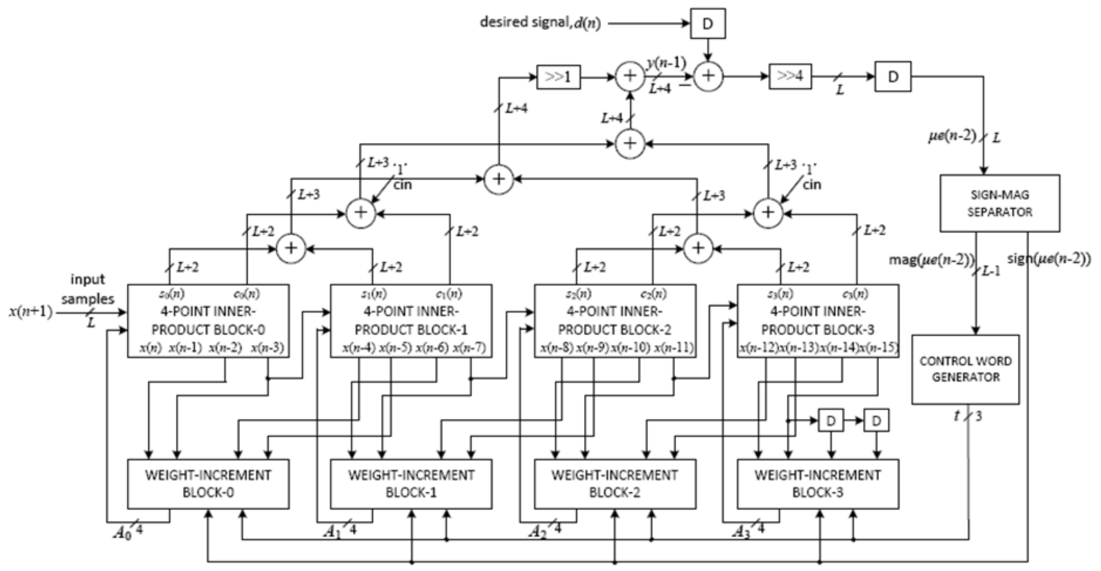


Fig. 7. (a) Structure of 4-point inner-product block. (b) Structure of weight increment block for N = 4. (c) The logic which is used for the generation of the control word t for the barrel-shifter for L = 8.

Fig. 8. Structure of DA-based LMS AF with N = 16 and P = 4.

The content of kth LUT location is expressed as

$$c_k = \sum_{j=0}^{N-1} x_j.k_j \quad (13)$$



where k_j is (j + 1)th bit of the N-bit binary representation of integer k where k lies in the range $0 \le k \le 2^N - 1$. We can pre-compute c_k for $0 \le k \le 2^N - 1$ and store it in a RAM-based LUT of 2^N words. But here we store $(2^N - 1)$ words in a DA-table that consists of $2^N - 1$ registers, in place of storing 2^N words in LUT. Fig.5 shows an example of one such DA-table for N = 4. It has only 15 registers that are used to store the sums of input words which are pre-computed. Seven adders in parallel, compute seven new values of c_k.

Bit-slices of weights A = {w_{3l} w_{2l} w_{1l} w_{0l}} for $0 \le l \le L - 1$ are given to the MUX as control in the order LSB to MSB, and the output of MUX is given to carry-save accumulator which is shown in Fig. 4.

The carry-save accumulator shift-accumulates all the partial-inner-products after L bit-cycles, and generates two words of size (L + 2)-bit each during these L bit-cycles, one for sum and other one for the carry. The sum and carry words are shifted-added with an input carry '1' in order to generate filter output which is subtracted from d(n), the desired output, later to obtain the error value e(n).

By assuming N = PQ, we can now decompose inner-product computation of (6) into N = P small adaptive filtering blocks of length P as shown in (14)

$$y = \sum_{k=0}^{P-1} w_k.x_k + \sum_{k=P}^{2P-1} w_k.x_k + \ldots + \sum_{k=N-P}^{N-1} w_k.x_k \quad (14)$$

affected by the truncation as the design requires the location of most significant 1 of μ.e(n).

V. DA BASED ADAPTIVE FILTER STRUCTURE

Fig.6 shows the structure of DA-based adaptive filter of filter length N = 4. It has a 4-point inner-product and a block for weight-increment, along with the additional circuits required for the computation of e(n), the error value and the control word t for barrel-shifters.

As shown in Fig.7a, the 4-point inner-product block consists of a DA-table that has an array of 15 registers that store the partial-inner-products y_l for l in range $0 < l \le 15$ and to select the content of one of these registers, a 16 : 1 MUX which is used.

Each of the above mentioned P-point inner-product computation blocks, to update P weights, have weight-increment unit. The structure for N = 16 and P = 4 is shown in the Fig.8. It has four inner-product blocks of length P = 4 as shown in Fig.7a. Two separate binary adder-trees are used to add the (L + 2)-bit carry and sum produced by these four blocks. Four carry-in bits are added to the sum words which are the output of four 4-point inner-product blocks. At the first level binary adder-tree of the carry words, two carry-in bits are set as the input carry, as carry words are of twice the weight as compared to sum words. This is same as inclusion of four carry-in to sum words.

To make the length of sign-magnitude separator as L-bit, assuming μ = 1/N, we can truncate the 4 LSBs of error e(n) for N = 16. The performance of adaptive filter is not very much

VI. RESULTS

The Low Power Adaptive FIR Filter Based on Distributed Arithmetic has been implemented in

Quartus II and we have compared it with an architecture without distributed arithmetic table.

TABLE I. POWER CONSUMPTION

Architecture	**Power Consumption**	
	Total Power	**Core Dynamic Power**
Conventional FIR Filter	110.77mW	24.91mW
DA-Based LMS Adaptive FIR Filter	84.36mW	12.44mW

It is found that there is a reduction in the total power and the core dynamic power by 31.31% and 100.24% respectively when compared with the architecture without DA table.

VII. CONCLUSION

We have implemented an efficient pipelined architecture for high-throughput, low-power and low-area DA-based adaptive filter. Throughput rate is quite significantly improved by means of concurrent processing of weight-update and filtering operation and the parallel LUT update. For computation of the filter output, we have used a carry-save accumulation unit for signed partial-inner-products computation.

ACKNOWLEDGEMENTS

We have implemented a Low Power Adaptive FIR Filter Based on Distributed Arithmetic under the supervision of Prof. Sri Adibhatla Sridevi, Associate Professor, SENSE. We thank her for providing proper guidance and timely help which helped us to successfully complete the work.

REFERENCES

Books:

[1]. Douglas S.C. Introduction to Adaptive Filters, Digital Signal Processing Handbook, Ed. Vijay K. Madisetti and Douglas B. Williams, Boca Raton: CRC Press LLC, 1999.

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

Journal Papers:

[3]. 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.

[4]. S. Y. Park And P. K. Meher, "Low-Power, High-Throughput, and Low-Area Adaptive FIR Filter Based On DA" IEEE Transactions on Circuits And Systems—II, Express Briefs, vol. 60, no. 6, June 2013, pp. 346-350.