Implementation of Efficient FIR Filter using Distributed Arithmetic

C.K.Bagyasri, *II Year, M.E-Commuication Systems, ECE Department*

K.Ramakrishnan College of Engineering-Samayapuram, Trichy.

Abstract-*This paper describes an architecture for efficient implementation of FIR filter using distributed arithmetic(DA).In the proposed system the through put rate is increased by using parallel lookup table (LUT) update and weight update update operations. The conventional adder-based shift accumulation for DA-based inner-product computation is replaced by conditional signed carry-save accumulation in order to reduce the sampling period and area complexity. By using a fast bit clock for carry-save accumulation reduction of power consumption is achieved. It involves the same number of multiplexers, smaller LUT, and nearly half the number of adders compared to the existing DA-based design.*

Key Words: *Adaptive filter, circuit optimization, distributed arithmetic (DA), least mean square (LMS) algorithm.*

I. INTRODUCTION

In digital signal processing applications adaptive filters are used. The tapped-delay line finite-impulse response (FIR) filter whose weights are updated by the famous Widrow–Hoff 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. , when the input signal has a high sampling rate, it is necessary to reduce the critical path of the structure so that the critical path could not exceed the sampling period.

In recent years, the multiplier-less distributed arithmetic (DA)-based technique has gained substantial popularity for its high-throughput processing capability and regularity, which result in cost-effective and area–time efficient computing structures. For high-speed DA-based adaptive filter with very low adaptation delay is proposed.

1) A parallel LUT update is used to increase the throughput.

2)The enhancement of throughput is achieved by concurrent implementation of filtering and weight updating.

3)Conditional carry-save accumulation is used instead of conventional adder based shift accumulation.

4)The bit-cycle period amounts to memory access time along with 1-bit full-adder time (instead of ripple carry addition time) by carry-save accumulation. The carry-save accumulation also helps to reduce the area complexity of the proposed design.

5) By using a fast bit clock for carry-save accumulation, reduction of power consumption is achieved.

6) An auxiliary control unit for address generation, is not required in the proposed structure, that is used in the existing structure.

II. LMS ADAPTIVE ALGORITHMS REVIEW

During each cycle, the LMS algorithm computes a filter output and an error value that is equal to the difference between the current filter output and the desired response. The estimated error is then used to update the filter weights in every training cycle. The weights of LMS adaptive filter during the nth iteration are updated according to the following equations:

$$w(n+1)=w(n)+\mu \,.\, e(n)\,.\, x(n) \qquad (1a)$$

where

$$e(n)=d(n)-y(n) \qquad (1b)$$

$$y(n)= w^{q^{T}}(n).x(n) \qquad (1c)$$

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

$$x(n)=[x(n),x(n-1),....,x(n-N+1)]^{T} \qquad (2a)$$

$$w(n)=[w0(n),w1(n),...., w_{N-1}(n)]^{T} \qquad (2b)$$

d(n) is the desired response, and y(n) is the filter output of the nth iteration. e(n) denotes the error computed during the nth iteration, which is used to update the weights, µ is the convergence factor, and N is the filter length.

In the case of pipelined designs, the feedback error e(n) becomes available after certain number of cycles, called the "adaptation delay." The pipelined architectures therefore use the delayed error e(n − m) for updating the current weight instead of the most recent error, where m is the adaptation delay. The weight-update equation of such delayed LMS adaptive filter is given by,

$$w(n+1)=w(n)+\mu\ .e(n-m)\ .x(n-m) \qquad (3)$$

III. PROPOSED DA-BASED INNER-PRODUCT COMPUTATION

The LMS adaptive filter, in each cycle, needs to perform an inner-product computation which contributes to the most of the critical path Let. the inner product of (1c) be given by

$$y = \sum_{k=0}^{N-1} w_k\ .\ x_k \qquad (4)$$

where w_k and$x_k$for $0 \le k \le N-1$ form the N-point vectors corresponding the current weights and most recent N − 1 input, respectively. Assuming L to be the bit width of the weight, each component of the weight vector may be expressed in two's complement representation

$$w_k = -w_{k0} + \sum_{l=1}^{L-1} w_{kl}\ .\ 2^{-l} \qquad (5)$$

where w_{kl}denotes the l th bit of w_k.substituting (5), we can write (4) in an expanded form

$$y = -\sum_{k=0}^{N-1} x_k\ .\ w_{k0} + \sum_{k=0}^{N-1} x_k\ .\left[\sum_{l=1}^{L-1} w_{kl}\ .\ 2^{-l}\right] (6)$$

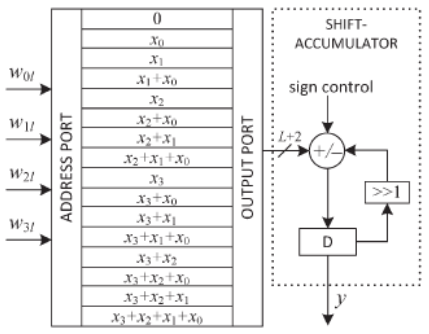

Fig 1:Four point inner product computation implementation

To convert the sum-of-products form of (4) into a distributed form, the order of summations over the indices k and l in (6) can be interchanged to have

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

and the inner product given by (7) can be computed as

$$y = \left[\sum_{l=1}^{L-1} 2^{-l}.yl\right] - y_{0,}$$

$$\text{where } y_{l,} = \sum_{k=0}^{N-1} x_k\ .\ w_{kl} \qquad (8)$$

Since any element of the N-point bit sequence $\{w_{kl}$for $0 \le k \le N-1\}$ can either be zero or one, the partial sum y_l for l = 0, 1, . . . , L − 1 can have 2N possible values. If all the 2N possible values of$y_l$are precomputed and stored in a LUT, the partial sumsy_l can be read out from the LUT using the bit sequence $\{w_{kl}\}$ as address bits for computing the inner product.

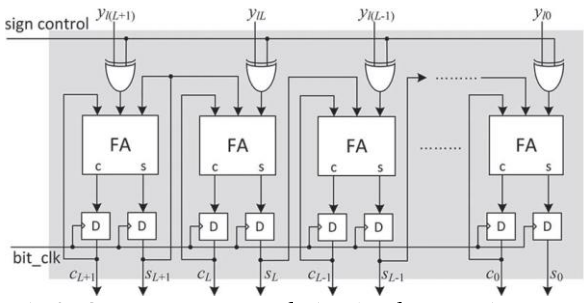

Fig 2: Carry-save accumulation implementation

A significant critical path is involved in Fig 1,so by using Fig 2 we perform the shift accumulation using carry-save accumulator. 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. The XOR gates thus produce the one's complement of the LUT output corresponding to the MSB slice but do not affect the output for other bit slices. Finally, the sum and carry words obtained after L clock cycles are required to be added by a final adder and the input carry of the final adder is required to be set to one to account for the two's complement operation of the LUT output corresponding to the MSB slice.

The content of the kth LUT location can be expressed as

$$c_k = \sum_{j=0}^{N-1} x_j \, . \, k_j \qquad (9)$$

where k_j is the (j + 1)th bit of N-bit binary representation of integer k for 0 ≤ k ≤ $2^N - 1$.Note that c_k for $0 \le k \le 2^N - 1$can be pre computed and stored in RAM-based LUT of 2^Nwords. However, instead of storing 2^N words in LUT, we store ($2^N - 1$) words in a DA table of $2^N - 1$registers.

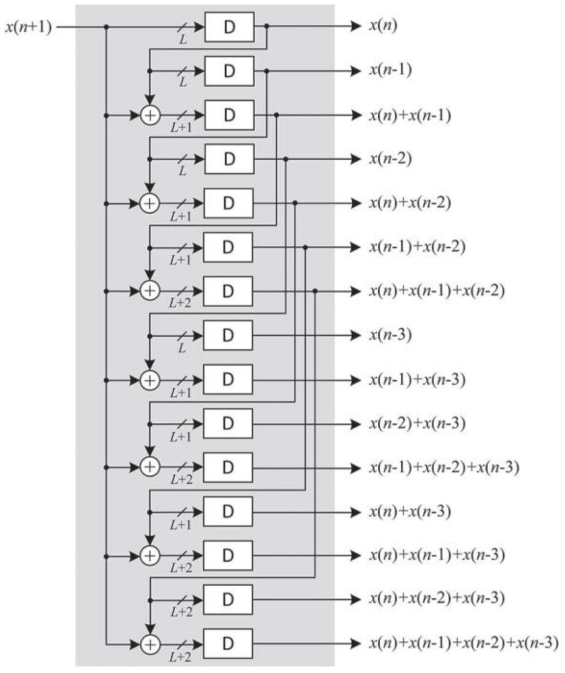

Fig 3:DA table for generation of possible sums of input samples.

An example of such a DA table for N = 4 is shown in Fig. 3. It contains only 15 registers to store the pre computed sums of input words. Seven new values of c_k are computed by seven adders in parallel.

IV.PROPOSED DA-BASED ADAPTIVE FILTER STRUCTURE

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 . Therefore, we describe here the proposed DA-based structures of small- and large-order LMS adaptive filters separately in the next two sections.

A. Proposed Structure of Small-Order Adaptive Filter

The proposed structure of DA-based adaptive filter of length N = 4 is shown in Fig. 4.

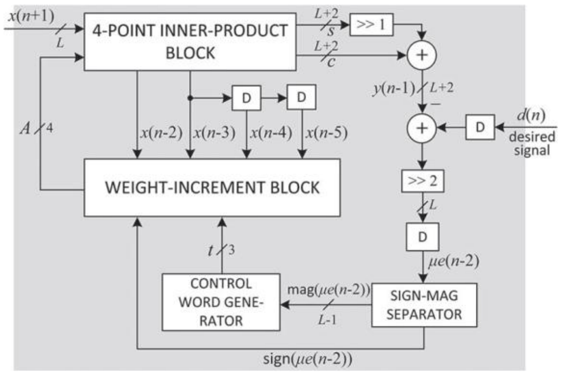

Fig. 4. Proposed structure of DA-based LMS adaptive filter of filter length N = 4.

It consists of a four-point inner-product block and a weight-increment block along with additional circuits for the computation of error value e(n) and control word t for the barrel shifters. The four-point inner-product block includes a DA table consisting of an array of 15 registers which stores the partial inner productsy_l for 0 < l ≤ 15 and a 16 : 1 multiplexer (MUX) to select the content of one of those registers. Bit slices of weights A = {w_{3l} w_{2l} w_{1l} w_{0l}} for 0 ≤ l ≤ 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 The four-point inner-product block [shown in Fig. 5(a)] includes a DA table consisting of an array of 15 registers which stores. 2). After L bit cycles, the carry-save accumulator shift accumulates all the partial inner products and generates a sum word and a carry word of size (L + 2) bit each. The carry and sum words are shifted added with an input carry "1" to generate filter output which is subsequently subtracted from the desired output d(n) to obtain the error e(n).

As in the case in [3], all the bits of the error except the most significant one are ignored, such that multiplication of input x_k by the error is implemented by a right shift through the number of locations given by the number of leading zeros in the magnitude of the error. The magnitude of the computed error is decoded to generate the control word t for the barrel shifter. The logic used for the generation of control word t to be used for the barrel shifter is shown in Fig. 5(c). The convergence factor µ is usually taken to be O(1/N). We have taken µ = 1/N . However, one can take µ as $2^{-i}/N$, where i is a small integer. The number of shifts t in that case is increased by i, and the input to the barrel shifters is pre shifted by i locations accordingly to reduce the hardware complexity.

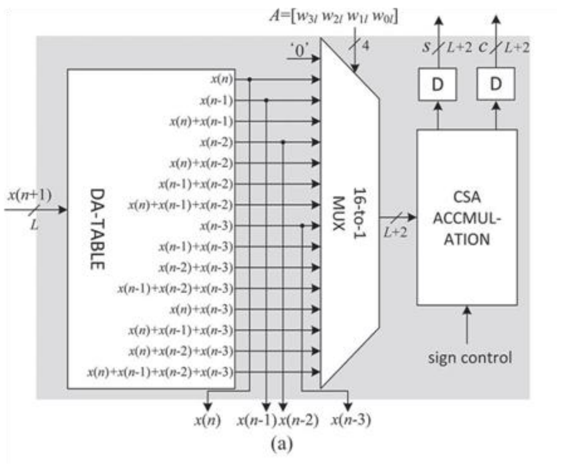

Fig. 5. (a) Structure of the four-point inner-product block.

The weight-increment unit [shown in Fig. 5(b)] for N = 4 consists of four barrel shifters and four adder/subtractor cells. The barrel shifter shifts the different input values ξ_κ for $\kappa = 0, 1, . . ., N - 1$ 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.

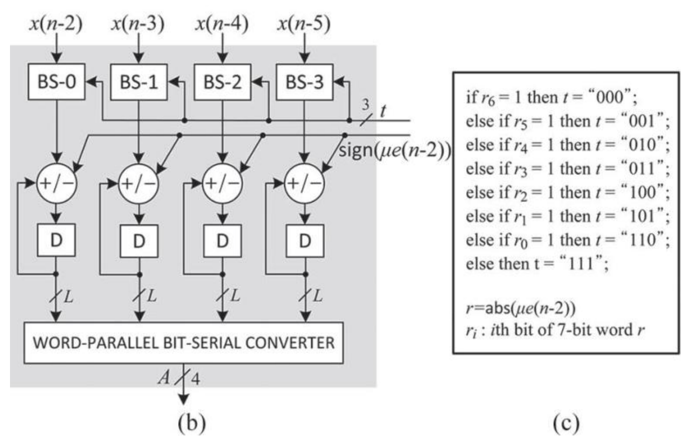
x(n-2) x(n-3) x(n-4) x(n-5)
BS-0
BS-1
BS-2
BS-3
3 t
sign(µe(n-2))
+/-
D
L
WORD-PARALLEL BIT-SERIAL CONVERTER
A 4
(b)
if r6 = 1 then t = "000";
else if r5 = 1 then t = "001";
else if r4 = 1 then t = "010";
else if r3 = 1 then t = "011";
else if r2 = 1 then t = "100";
else if r1 = 1 then t = "101";
else if r0 = 1 then t = "110";
else then t = "111";
r=abs(µe(n-2))
ri : ith bit of 7-bit word r
(c)

Fig. 5. (b) Structure of the weight-increment block for $N = 4$. (c) Logic used for generation of control word τ for the barrel shifter for $\Lambda = 8$.

B. Proposed Structure of Large-Order Adaptive Filter

The inner-product computation of (4) can be decomposed into N/Π (assuming that $N = \Pi\Theta$) small adaptive filtering blocks of filter length Π as

$$y = \sum_{k=0}^{P-1} w_k . x_k + \sum_{k=P}^{2P-1} w_k . x_k . . + \sum_{k=N-P}^{N-1} w_k . x_k \qquad (10)$$

Each of these Π-point inner-product computation blocks will accordingly have a weight-increment unit to update Π weights. The proposed structure for $N = 16$ and $\Pi = 4$ is shown in Fig. 6. It consists of four inner-product blocks of length $\Pi = 4$, which is shown in Fig. 5(a). The $(\Lambda + 2)$-bit sums and carry produced by the four blocks are added by two separate binary adder trees. 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.

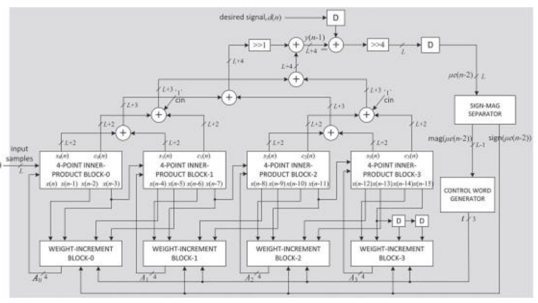

Fig. 6. Proposed structure of DA-based LMS adaptive filter of length $N = 16$ and $\Pi = 4$.

Assuming that $f = 1/N$, we truncate the four LSBs of $\varepsilon(n)$ for $N = 16$ to make the word length of sign-magnitude separator be 4 bit. It should be noted that the truncation does not affect the performance of the adaptive filter very much since the proposed design needs the location of the most significant one of $f\varepsilon(n)$.

V. COMPLEXITY AND SYNTHESIS RESULTS

The proposed design uses two clocks, namely, the bit clock and the byte clock. The duration of the byte clock is the same as the sampling period. The bit clock is used in carry-save accumulation units and word parallel bit-serial converters, while the byte clock is used in the rest of the circuit. The output of the carry save accumulation is obtained as follows:

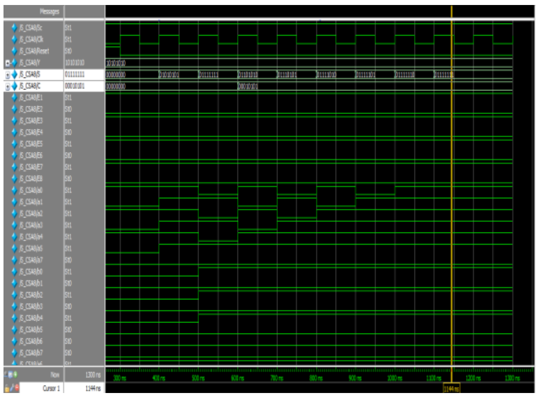

Fig 7. Carry Save Accumulation Simulation Output

The synthesis output using the proposed method is shown as below,

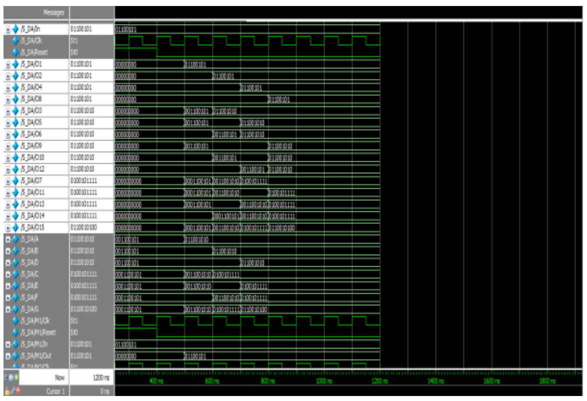

Fig 8. Distributed Arithmetic Simulation Output

From the synthesis results, we find that the proposed design has almost 6.3 times more throughput and 8.7 times less ADP over the design in [3] in average for filter lengths $N = 16$ and 32. Moreover, the proposed design consumes 16.6 times less power than the design in [3]. Since the design in [3] generates the output sample every $3 \cdot 2n-1 + \log_2 \Theta$ clock, it can be seen that its EPS is much higher than that of the proposed design. The proposed design also offers 4.2 times more throughput, 4.6 times less ADP, and 9.5 times less EPS than the design in [4]. Compared to the design in [6], the proposed design involves 13% less power consumption and 29% less ADP.

VI. CONCLUSION

Thus an efficient FIR filter implementation of DA-based adaptive filter was suggested. Throughput rate is significantly enhanced by parallel LUT update and concurrent processing of filtering operation and weight-update operation. We have also proposed a carry-save accumulation scheme of signed partial inner products for the computation of filter output. From the synthesis results, we find that the proposed design consumes 13% less power and 29% less ADP over our previous DA-based FIR adaptive filter in average for filter lengths $N = 16$ and 32.

Compared to the best of other existing designs, our proposed architecture provides 9.5 times less power and 4.6 times less ADP. Offset binary coding is popularly used to reduce the LUT size to half for area-efficient implementation of DA [2], [5],which can be applied to our design as well.

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] 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.

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

[6] 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.

[7] M. D. Meyer and P. Agrawal, "A modular pipelined implementation of a delayed LMS transversal adaptive filter," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 1990, pp. 1943–1946.

[8]K.K. Parhi, "VLSI Digital Signal Processing Systems: Design and
Implementation," John Wiley, 1999.