IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 4, Issue 1, Ver. II (Jan. 2014), PP 69-74
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Optimization of Adaptive Fir Filter for High Throughput, Low Power & Area Using Distributed Arithmetic

Kalaiarashi.K[1],[1] Mr.Santhakumar.K[2],
[1]*Pg Scholar, Department Of Ece, Nandha Engineering College, Erode.*
[2]*Associate Professor, Department Of Ece, Nandha Engineering College, Erode.*

Abstract: *This brief presents a novel pipelined architecture for low-power, high-throughput, and low-area implementation of adaptive filter based on distributed arithmetic (DA). The throughput rate of the proposed design is significantly increased by parallel lookup table (LUT) update and concurrent implementation of filtering and weight-update operations. The DA-based inner-product computation by conditional signed carry-save accumulation is replaced with CSA Binary to Excess-1 Converter (BEC) in order to reduce the sampling period and area complexity. Reduction of power consumption is achieved in the proposed design by using a fast bit clock for carry-save accumulation but a much slower clock for all other operations. It involves the same number of multiplexors, smaller LUT, and nearly half the number of adders compared to the existing DA-based design.*
Index Terms: *Adaptive filter, binary to excess-1 convertor(BEC) circuit optimization, distributed arithmetic (DA), least mean square (LMS) algorithm.*

I. Introduction

ADAPTIVE filters are widely used in several digital signal processing applications. 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 . 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. Therefore, 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. Hardware-efficient DA-based design of adaptive filter has been suggested using two separate lookup tables (LUTs) for filtering and weight update.

This brief proposes a novel DA-based architecture for low power, low-area, and high-throughput pipelined implementation of adaptive filter with very low adaptation delay. The contributions of this brief are as follows.

1) Throughput rate is significantly increased by a parallel LUT update.
2) Further enhancement of throughput is achieved by concurrent implementation of filtering and weight updating.
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 use of the proposed signed carry-save accumulation also helps to reduce the area complexity of the proposed design.
4) Reduction of power consumption is achieved by using a fast bit clock for carry-save accumulation but a much slower clock for all other operations.
5) The auxiliary control unit for address generation, which is not required in the proposed structure.

II. Review Of Lms Adaptive Algorithms

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 *n*th iteration are updated according to the following equations:

$$\mathrm{w}(n+1) = \mathrm{w}(n) + \mu .\ e(n).\ \mathrm{x}(n) \qquad (1a)$$

where

$$e(n) = d(n) - y(n) \qquad (1b)$$
$$y(n) = \mathrm{w}^{qT}(n).\mathrm{x}(n) \qquad (1c)$$

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

$$\mathrm{x}(n) = [x(n), x(n-1), \ldots, x(n-N+1)]^T \quad (2a)$$
$$\mathrm{w}(n) = [w_0(n), w_1(n), \ldots, w_{N-1}(n)]^T \quad (2b)$$

$d(n)$ 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

$$\mathrm{w}(n+1) = \mathrm{w}(n) + \mu.\ e(n-m).\ \mathrm{x}(n-m). \quad (3)$$

III. Da-Based Approach For 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. For simplicity of presentation, let the inner product of (1c) be given by

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

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

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

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

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

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

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

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

$$y = \left[\sum_{l=1}^{L-1} 2^{-l}.\ y_l \right] - y_0, \quad (8)$$

where

$$y_l = \sum_{k=0}^{N-1} w_{kl} \,.\, x_k$$

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 $l = 0, 1, \ldots, L-1$ can have 2^N possible values. If all the 2^N possible values of y_l are precomputed and stored in a LUT, the partial sums y_l can be read out from the LUT using the bit sequence $\{w_{kl}\}$ as address bits for computing the inner product.

The inner product of (8) can therefore be calculated in L cycles of shift accumulation, followed by LUT-read operations corresponding to L number of bit slices $\{w_{kl}\}$ 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 **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.

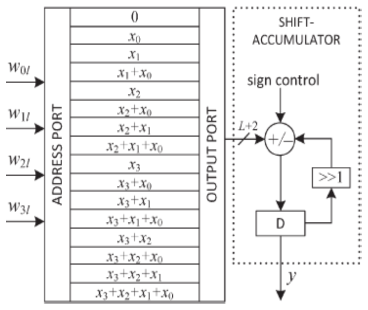


Fig. 1. Conventional DA-based implementation of four-point inner product.

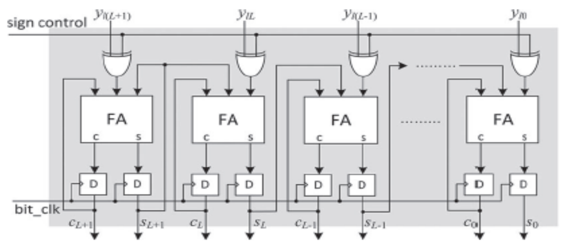


Fig. 2. Carry-save implementation of shift accumulation.

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 (not shown in the figure), 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 $0 \leq k \leq 2^N - 1$. Note that c_k for $0 \leq k \leq 2^N - 1$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. 3. 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.

IV. 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 [3]. Therefore, we describe here the DA-based structures of small- and large-order LMS adaptive filters separately in the next two sections.

A. Structure of Small-Order Adaptive Filter

The structure of DA-based adaptive filter of length $N = 4$ is shown in Fig. 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 [shown in Fig. 5(a)] includes a DA table consisting of an array of 15 registers which stores the partial inner products y_l for $0 < l \leq 15$ and a 16 : 1 multiplexor (MUX) to select the content of one of those registers.

Bit slices of weights $A = \{w_{3l}\, w_{2l}\, w_{1l}\, w_{0l}\}$ for $0 \leq l \leq L - 1$ 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. 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 $e(n)$. 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.

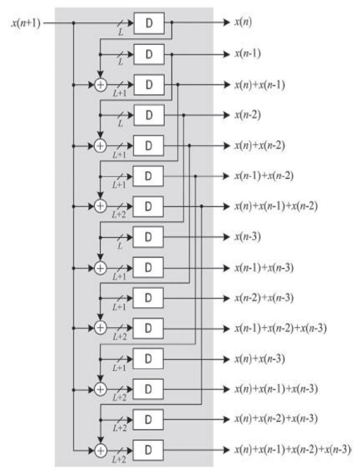


Fig. 3. DA table for generation of possible sums of input samples.

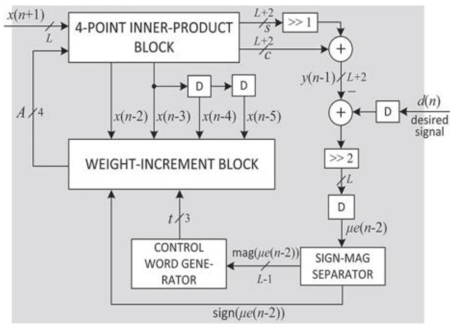
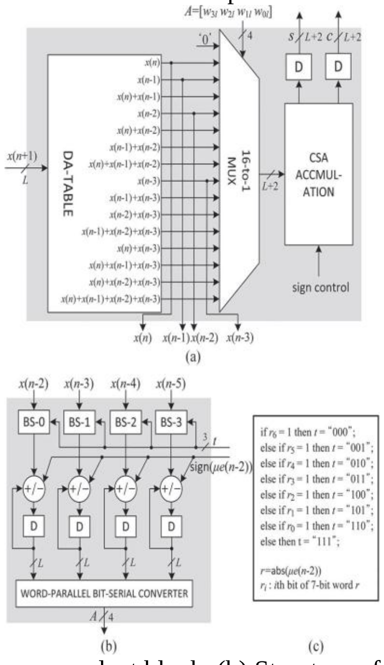


Fig. 4.Structure of DA-based LMS adaptive filter of filter length N = 4.



```
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. 5. (a) Structure of the four-point inner-product block. (b) Structure of the weight-increment block for N = 4. (c) Logic used for generation of control word t for the barrel shifter for L = 8.

The weight-increment unit [shown in and four adder/subtractor cells. Fig. 5(b)] for N = 4 consists of four barrel shifters The barrel shifter shifts the different input values x_k for k = 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.

B. Structure of Large-Order Adaptive Filter

The inner-product computation of (4) can be decomposed into N/P (assuming that $N = PQ$) small adaptive filtering blocks1 of filter length P 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 P-point inner-product computation blocks will accordingly have a weight-increment unit to update P weights. The structure for N = 16 and P = 4 is shown in Fig. 6.The (L + 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. 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 $\mu e(n)$.

V. Proposed Da With Carry Save Adder Bec

Carry Select Adder (CSLA) is one of the fastest adders used in many data-processing processors to perform fast arithmetic functions. From the structure of the CSLA, it is clear that there is scope for reducing the area and power consumption in the CSLA. The basic idea of this work is to use Binary to Excess-1 Converter (BEC) instead of RCA with Cin=1in the regular CSLA to achieve lower area and power consumption The main advantage of this BEC logic comes from the lesser number of logic gates than the n-bit Full Adder (FA) structure. Table 1 shows the comparison of Synthesis results of DA structures in terms of area and power. Area and power are significantly reduced.

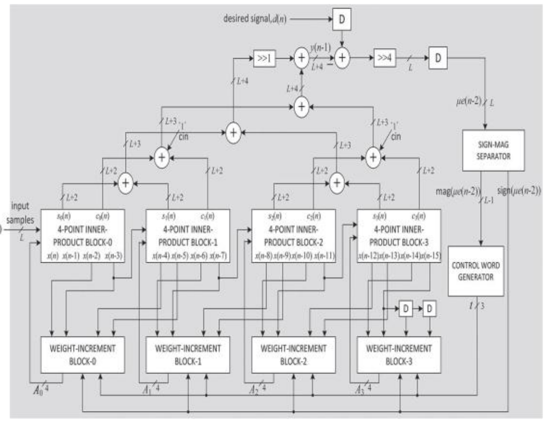


Fig. 6. Structure of DA-based LMS adaptive filter of length N = 16 and P = 4.

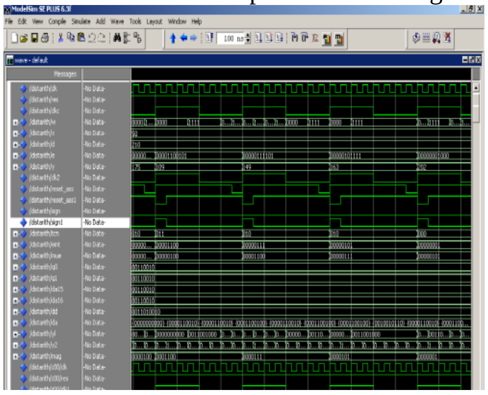


Fig.7 Simulation result of DA-based LMS adaptive filter of filter length N = 4 using ModelSim

Table 1 Comparison of Area and Power using Synthesis results

DA based LMS adaptive filter	AREA(gate count)	POWER(mV)
Ripple carry adder	9431	160.12
Carry save adder	9236	129.10
Carry save adder with BEC	8432	127.61

VI. Conclusion

We have suggested an efficient pipelined architecture for low-power, high-throughput, and low-area implementation of DA-based adaptive filter. Throughput rate is significantly enhanced by parallel LUT update and concurrent processing of filtering operation and weight-update operation. We have alsoproposed a carry-save accumulation BEC scheme of signed partial inner products for the computation of filter output. From the Xilinx synthesis results, we find that the proposed design consumes less power and less area over our previous DA-based FIR adaptive filter in average for filter lengths $N = 16$ and 32. Offset binary coding is popularly used to reduce the LUT size to half for area-efficient implementation of DA 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, 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. 60 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.