Area-Delay-Power Efficient Fixed-Point LMS Adaptive Filter With Low Adaptation-Delay

Pramod Kumar Meher, *Senior Member, IEEE*, and Sang Yoon Park, *Member, IEEE*

***Abstract*— In this paper, we present an efficient architecture for the implementation of a delayed least mean square adaptive filter. For achieving lower adaptation-delay and area-delay-power efficient implementation, we use a novel partial product generator and propose a strategy for optimized balanced pipelining across the time-consuming combinational blocks of the structure. From synthesis results, we find that the proposed design offers nearly 17% less area-delay product (ADP) and nearly 14% less energy-delay product (EDP) than the best of the existing systolic structures, on average, for filter lengths N = 8, 16, and 32. We propose an efficient fixed-point implementation scheme of the proposed architecture, and derive the expression for steady-state error. We show that the steady-state mean squared error obtained from the analytical result matches with the simulation result. Moreover, we have proposed a bit-level pruning of the proposed architecture, which provides nearly 20% saving in ADP and 9% saving in EDP over the proposed structure before pruning without noticeable degradation of steady-state-error performance.**

***Index Terms*— Adaptive filters, circuit optimization, fixed-point arithmetic, least mean square (LMS) algorithms.**

I. Introduction

THE LEAST MEAN SQUARE (LMS) adaptive filter is the most popular and most widely used adaptive filter, not only because of its simplicity but also because of its satisfactory convergence performance [1], [2]. The direct-form LMS adaptive filter involves a long critical path due to an inner-product computation to obtain the filter output. The critical path is required to be reduced by pipelined implementation when it exceeds the desired sample period. Since the conventional LMS algorithm does not support pipelined implementation because of its recursive behavior, it is modified to a form called the delayed LMS (DLMS) algorithm [3]–[5], which allows pipelined implementation of the filter.

A lot of work has been done to implement the DLMS algorithm in systolic architectures to increase the maximum usable frequency [3], [6], [7] but, they involve an adaptation delay of $\sim N$ cycles for filter length N, which is quite high for large-order filters. Since the convergence performance degrades considerably for a large adaptation delay, Visvanathan *et al.* [8] have proposed a modified systolic architecture to reduce the adaptation delay. A transpose-form LMS adaptive filter is suggested in [9], where the filter output at any instant depends on the delayed versions of weights and the number of delays in weights varies from 1 to N. Van and Feng [10] have proposed a systolic architecture, where they have used relatively large processing elements (PEs) for achieving a lower adaptation delay with the critical path of one MAC operation. Ting *et al.* [11] have proposed a fine-grained pipelined design to limit the critical path to the maximum of one addition time, which supports high sampling frequency, but involves a lot of area overhead for pipelining and higher power consumption than in [10], due to its large number of pipeline latches. Further effort has been made by Meher and Maheshwari [12] to reduce the number of adaptation delays. Meher and Park have proposed a 2-bit multiplication cell, and used that with an efficient adder tree for pipelined inner-product computation to minimize the critical path and silicon area without increasing the number of adaptation delays [13], [14].

The existing work on the DLMS adaptive filter does not discuss the fixed-point implementation issues, e.g., location of radix point, choice of word length, and quantization at various stages of computation, although they directly affect the convergence performance, particularly due to the recursive behavior of the LMS algorithm. Therefore, fixed-point implementation issues are given adequate emphasis in this paper. Besides, we present here the optimization of our previously reported design [13], [14] to reduce the number of pipeline delays along with the area, sampling period, and energy consumption. The proposed design is found to be more efficient in terms of the power-delay product (PDP) and energy-delay product (EDP) compared to the existing structures.

In the next section, we review the DLMS algorithm, and in Section III, we describe the proposed optimized architecture for its implementation. Section IV deals with fixed-point implementation considerations and simulation studies of the convergence of the algorithm. In Section V, we discuss the synthesis of the proposed architecture and comparison with the existing architectures. Conclusions are given in Section VI.

II. Review of Delayed LMS Algorithm

The weights of LMS adaptive filter during the nth iteration are updated according to the following equations [2]:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \cdot e_n \cdot \mathbf{x}_n \tag{1a}$$

where

$$e_n = d_n - y_n \quad y_n = \mathbf{w}_n^T \cdot \mathbf{x}_n \tag{1b}$$

where the input vector $\mathbf{x}_n$, and the weight vector $\mathbf{w}_n$ at the nth iteration are, respectively, given by

$$\mathbf{x}_n = [x_n, x_{n-1}, \ldots, x_{n-N+1}]^T$$
$$\mathbf{w}_n = [w_n(0), w_n(1), \ldots, w_n(N-1)]^T,$$

Manuscript received September 9, 2012; revised December 5, 2012; accepted January 8, 2013. Date of publication February 4, 2013; date of current version January 17, 2014. *(Corresponding author: S. Y. Park.)*

The authors are with the Institute for Infocomm Research, 138632 Singapore (e-mail: pkmeher@i2r.a-star.edu.sg; sypark@i2r.a-star.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2239321

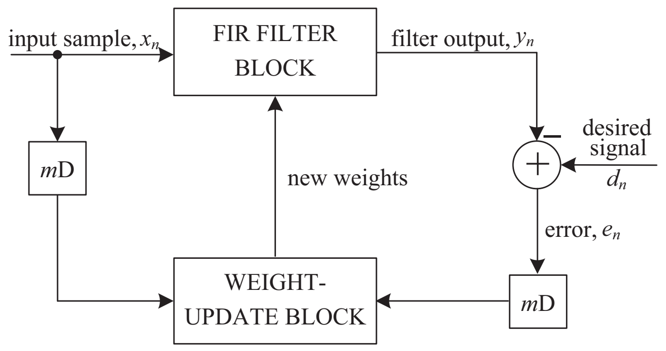


Fig. 1. Structure of the conventional delayed LMS adaptive filter.

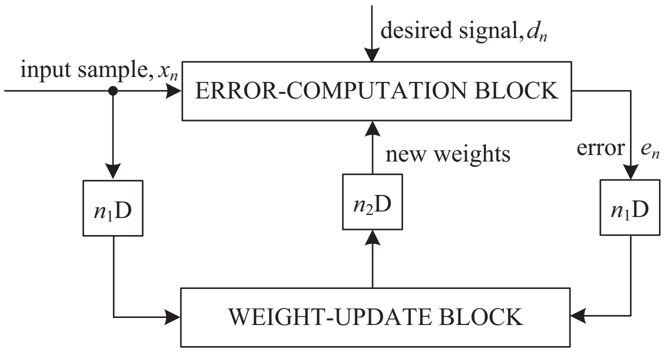


Fig. 2. Structure of the modified delayed LMS adaptive filter.

d_n is the desired response, y_n is the filter output, and e_n denotes the error computed during the nth iteration. μ is the step-size, and N is the number of weights used in the LMS adaptive filter.

In the case of pipelined designs with m pipeline stages, the error e_n becomes available after m cycles, where m is called the "adaptation delay." The DLMS algorithm therefore uses the delayed error e_{n-m}, i.e., the error corresponding to $(n-m)$th iteration for updating the current weight instead of the recent-most error. The weight-update equation of DLMS adaptive filter is given by

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \cdot e_{n-m} \cdot \mathbf{x}_{n-m}. \tag{2}$$

The block diagram of the DLMS adaptive filter is shown in Fig. 1, where the adaptation delay of m cycles amounts to the delay introduced by the whole of adaptive filter structure consisting of finite impulse response (FIR) filtering and the weight-update process.

It is shown in [12] that the adaptation delay of conventional LMS can be decomposed into two parts: one part is the delay introduced by the pipeline stages in FIR filtering, and the other part is due to the delay involved in pipelining the weight-update process. Based on such a decomposition of delay, the DLMS adaptive filter can be implemented by a structure shown in Fig. 2.

Assuming that the latency of computation of error is n_1 cycles, the error computed by the structure at the nth cycle is e_{n-n_1}, which is used with the input samples delayed by n_1 cycles to generate the weight-increment term. The weight-update equation of the modified DLMS algorithm is given by

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \cdot e_{n-n_1} \cdot \mathbf{x}_{n-n_1} \tag{3a}$$

where

$$e_{n-n_1} = d_{n-n_1} - y_{n-n_1} \tag{3b}$$

and

$$y_n = \mathbf{w}^T_{n-n_2} \cdot \mathbf{x}_n. \tag{3c}$$

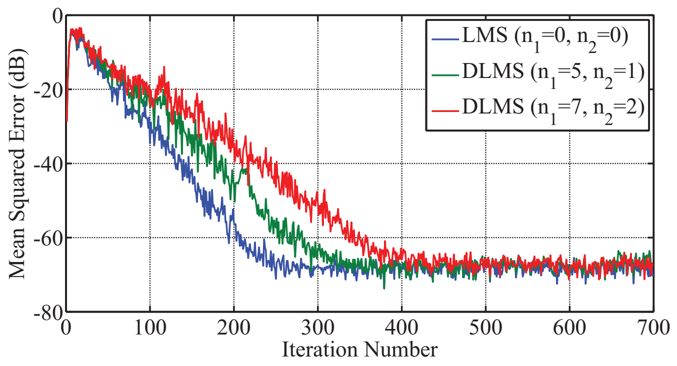


Fig. 3. Convergence performance of system identification with LMS and modified DLMS adaptive filters.

We notice that, during the weight update, the error with n_1 delays is used, while the filtering unit uses the weights delayed by n_2 cycles. The modified DLMS algorithm decouples computations of the error-computation block and the weight-update block and allows us to perform optimal pipelining by feed-forward cut-set retiming of both these sections separately to minimize the number of pipeline stages and adaptation delay.

The adaptive filters with different n_1 and n_2 are simulated for a system identification problem. The 10-tap band-pass filter with impulse response

$$h_n = \frac{\sin(w_H(n-4.5))}{\pi(n-4.5)} - \frac{\sin(w_L(n-4.5))}{\pi(n-4.5)} \quad \text{for } n = 0, 1, 2, \ldots, 9, \text{ otherwise } h_n = 0 \tag{4}$$

is used as the unknown system as in [10]. w_H and w_L represent the high and low cutoff frequencies of the passband, and are set to $w_H = 0.7\pi$ and $w_L = 0.3\pi$, respectively. The step size μ is set to 0.4. A 16-tap adaptive filter identifies the unknown system with Gaussian random input x_n of zero mean and unit variance. In all cases, outputs of known system are of unity power, and contaminated with white Gaussian noise of -70 dB strength. Fig. 3 shows the learning curve of MSE of the error signal e_n by averaging 20 runs for the conventional LMS adaptive filter ($n_1 = 0, n_2 = 0$) and DLMS adaptive filters with ($n_1 = 5, n_2 = 1$) and ($n_1 = 7, n_2 = 2$). It can be seen that, as the total number of delays increases, the convergence is slowed down, while the steady-state MSE remains almost the same in all cases. In this example, the MSE difference between the cases ($n_1 = 5, n_2 = 1$) and ($n_1 = 7, n_2 = 2$) after 2000 iterations is less than 1 dB, on average.

III. Proposed Architecture

As shown in Fig. 2, there are two main computing blocks in the adaptive filter architecture: 1) the error-computation

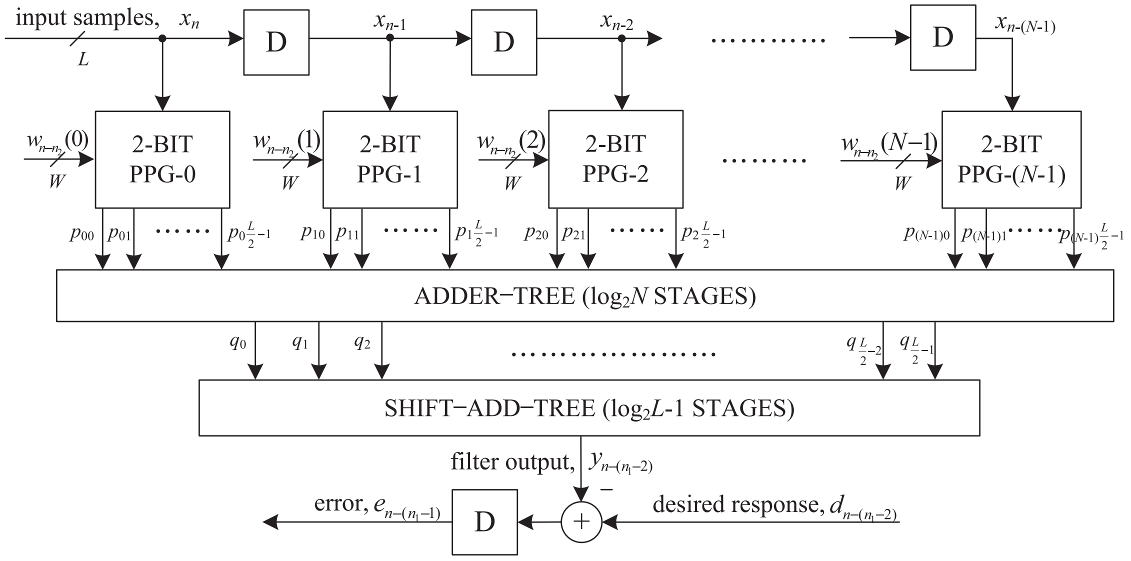


Fig. 4. Proposed structure of the error-computation block.

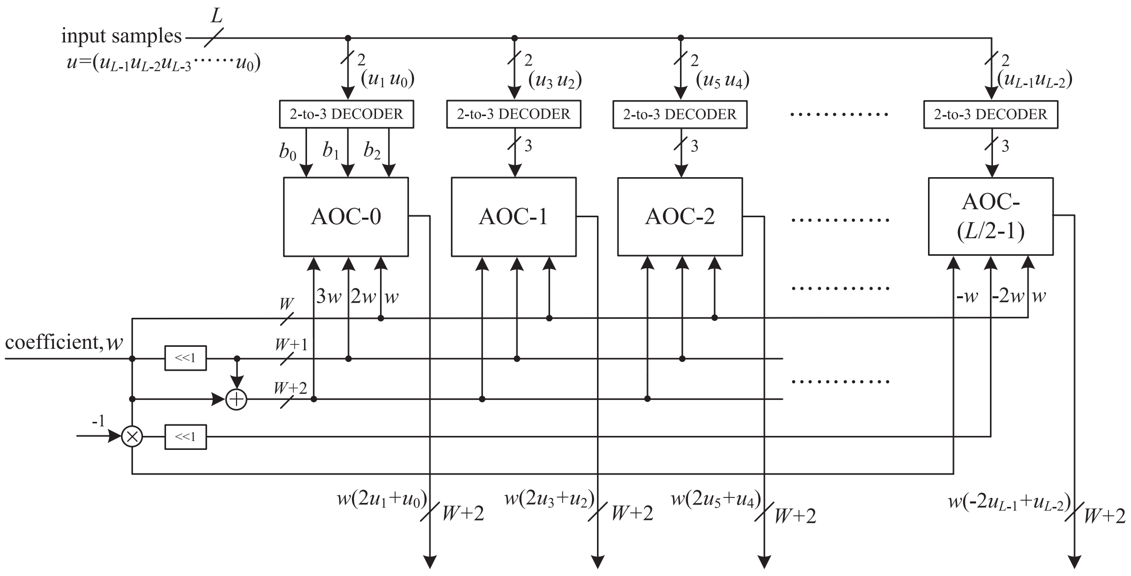


Fig. 5. Proposed structure of PPG. AOC stands for AND/OR cell.

block, and 2) weight-update block. In this Section, we discuss the design strategy of the proposed structure to minimize the adaptation delay in the error-computation block, followed by the weight-update block.

A. Pipelined Structure of the Error-Computation Block

The proposed structure for error-computation unit of an N-tap DLMS adaptive filter is shown in Fig. 4. It consists of N number of 2-b partial product generators (PPG) corresponding to N multipliers and a cluster of $L/2$ binary adder trees, followed by a single shift–add tree. Each subblock is described in detail.

1) Structure of PPG: The structure of each PPG is shown in Fig. 5. It consists of $L/2$ number of 2-to-3 decoders and the same number of AND/OR cells (AOC).[1] Each of the 2-to-3 decoders takes a 2-b digit (u_1u_0) as input and produces three outputs $b_0 = u_0 \cdot \bar{u_1}$, $b_1 = \bar{u_0} \cdot u_1$, and $b_2 = u_0 \cdot u_1$, such that $b_0 = 1$ for $(u_1u_0) = 1$, $b_1 = 1$ for $(u_1u_0) = 2$, and $b_2 = 1$ for $(u_1u_0) = 3$. The decoder output b_0, b_1 and b_2 along with w, $2w$, and $3w$ are fed to an AOC, where w, $2w$, and $3w$ are in 2's complement representation and sign-extended to have $(W + 2)$ bits each. To take care of the sign of the input samples while computing the partial product corresponding to the most significant digit (MSD), i.e., $(u_{L-1}u_{L-2})$ of the input sample, the AOC $(L/2 - 1)$ is fed with w, $-2w$, and $-w$ as input since $(u_{L-1}u_{L-2})$ can have four possible values 0, 1, -2, and -1.

2) Structure of AOCs: The structure and function of an AOC are depicted in Fig. 6. Each AOC consists of three AND cells and two OR cells. The structure and function of AND cells and

[1]We have assumed the word length of the input L to be even, which is valid in most practical cases.

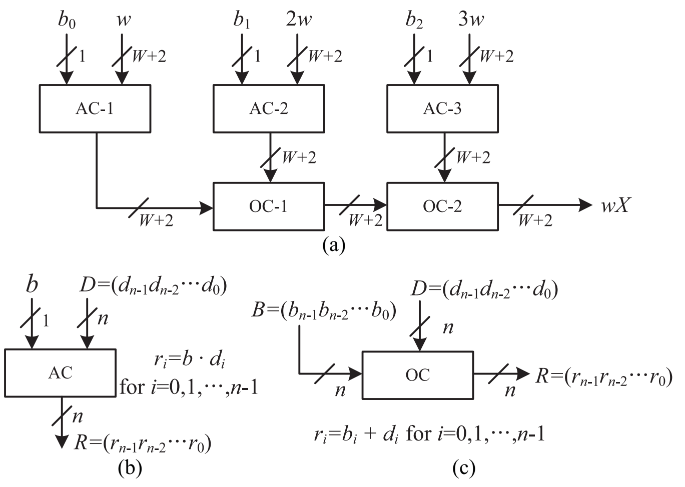

b_0
w
b_1
$2w$
b_2
$3w$
AC-1
AC-2
AC-3
OC-1
OC-2
W+2
wX
(a)
b
$D=(d_{n-1}d_{n-2}\cdots d_0)$
AC
$r_i=b \cdot d_i$ for i=0,1,⋯,n-1
$R=(r_{n-1}r_{n-2}\cdots r_0)$
(b)
$B=(b_{n-1}b_{n-2}\cdots b_0)$
OC
$r_i=b_i+d_i$ for i=0,1,⋯,n-1
(c)

Fig. 6. Structure and function of AND/OR cell. Binary operators · and + in (b) and (c) are implemented using AND and OR gates, respectively.

OR cells are depicted by Fig. 6(b) and (c), respectively. Each AND cell takes an n-bit input D and a single bit input b, and consists of n AND gates. It distributes all the n bits of input D to its n AND gates as one of the inputs. The other inputs of all the n AND gates are fed with the single-bit input b. As shown in Fig. 6(c), each OR cell similarly takes a pair of n-bit input words and has n OR gates. A pair of bits in the same bit position in B and D is fed to the same OR gate.

The output of an AOC is w, $2w$, and $3w$ corresponding to the decimal values 1, 2, and 3 of the 2-b input (u_1u_0), respectively. The decoder along with the AOC performs a multiplication of input operand w with a 2-b digit (u_1u_0), such that the PPG of Fig. 5 performs $L/2$ parallel multiplications of input word w with a 2-b digit to produce $L/2$ partial products of the product word wu.

3) Structure of Adder Tree: Conventionally, we should have performed the shift-add operation on the partial products of each PPG separately to obtain the product value and then added all the N product values to compute the desired inner product. However, the shift-add operation to obtain the product value increases the word length, and consequently increases the adder size of $N-1$ additions of the product values. To avoid such increase in word size of the adders, we add all the N partial products of the same place value from all the N PPGs by one adder tree.

All the $L/2$ partial products generated by each of the N PPGs are thus added by $(L/2)$ binary adder trees. The outputs of the $L/2$ adder trees are then added by a shift-add tree according to their place values. Each of the binary adder trees require $\log_2 N$ stages of adders to add N partial product, and the shift–add tree requires $\log_2 L - 1$ stages of adders to add $L/2$ output of $L/2$ binary adder trees.[2] The addition scheme for the error-computation block for a four-tap filter and input word size $L = 8$ is shown in Fig. 7. For $N = 4$ and $L = 8$, the adder network requires four binary adder trees of two stages each and a two-stage shift–add tree. In this figure, we have shown all possible locations of pipeline latches by dashed lines, to reduce the critical path to one addition time. If we introduce pipeline latches after every addition, it would require $L(N-1)/2 + L/2 - 1$ latches in $\log_2 N + \log_2 L - 1$ stages, which would lead to a high adaptation delay and introduce a large overhead of area and power consumption for large values of N and L. On the other hand, some of those pipeline latches are redundant in the sense that they are not required to maintain a critical path of one addition time. The final adder in the shift–add tree contributes to the maximum delay to the critical path. Based on that observation, we have identified the pipeline latches that do not contribute significantly to the critical path and could exclude those without any noticeable increase of the critical path. The location of pipeline latches for filter lengths $N = 8$, 16, and 32 and for input size $L = 8$ are shown in Table I. The pipelining is performed by a feed-forward cut-set retiming of the error-computation block [15].

[2]When L is not a power of 2, $\log_2 L$ should be replaced by $\lceil \log_2 L \rceil$.

TABLE I
LOCATION OF PIPELINE LATCHES FOR $L = 8$ AND $N = 8$, 16, AND 32

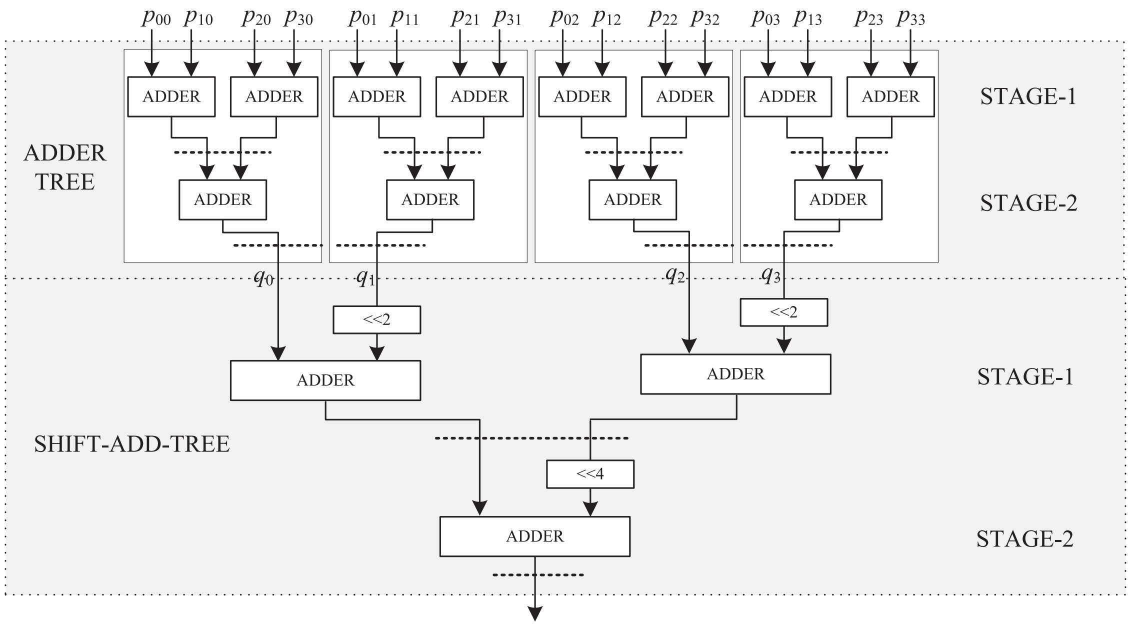
N	Error-Computation Block		Weight-Update Block
	Adder Tree	Shift–add Tree	Shift–add Tree
8	Stage-2	Stage-1 and 2	Stage-1
16	Stage-3	Stage-1 and 2	Stage-1
32	Stage-3	Stage-1 and 2	Stage-2

B. Pipelined Structure of the Weight-Update Block

The proposed structure for the weight-update block is shown in Fig. 8. It performs N multiply-accumulate operations of the form $(\mu \times e) \times x_i + w_i$ to update N filter weights. The step size μ is taken as a negative power of 2 to realize the multiplication with recently available error only by a shift operation. Each of the MAC units therefore performs the multiplication of the shifted value of error with the delayed input samples x_i followed by the additions with the corresponding old weight values w_i. All the N multiplications for the MAC operations are performed by N PPGs, followed by N shift–add trees. Each of the PPGs generates $L/2$ partial products corresponding to the product of the recently shifted error value $\mu \times e$ with $L/2$, the number of 2-b digits of the input word x_i, where the subexpression $3\mu \times e$ is shared within the multiplier. Since the scaled error $(\mu \times e)$ is multiplied with all the N delayed input values in the weight-update block, this subexpression can be shared across all the multipliers as well. This leads to substantial reduction of the adder complexity. The final outputs of MAC units constitute the desired updated weights to be used as inputs to the error-computation block as well as the weight-update block for the next iteration.

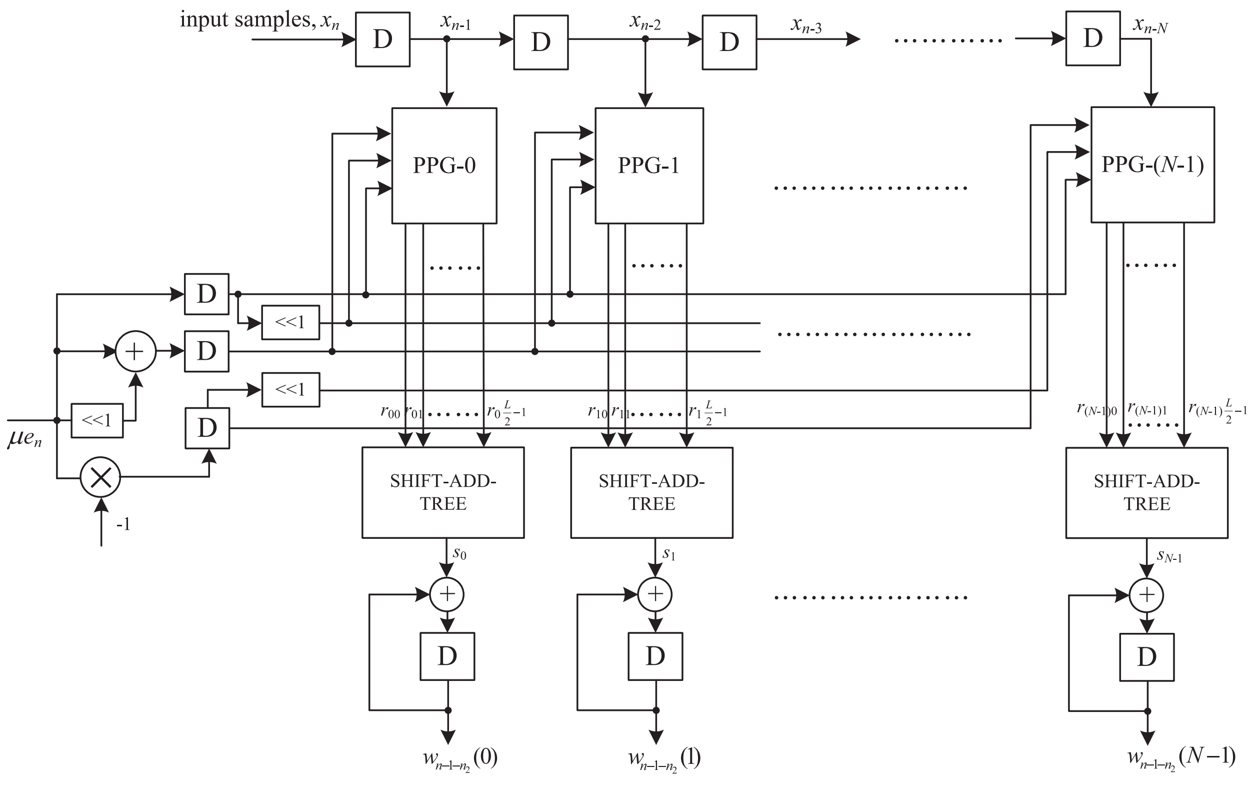
C. Adaptation Delay

As shown in Fig. 2, the adaptation delay is decomposed into n_1 and n_2. The error-computation block generates the delayed error by $n_1 - 1$ cycles as shown in Fig. 4, which is fed to the weight-update block shown in Fig. 8 after scaling by μ; then the input is delayed by 1 cycle before the PPG to make the total delay introduced by FIR filtering be n_1. In Fig. 8, the weight-update block generates $\mathbf{w}_{n-1-n2}$, and the weights are



p_{00} p_{10} p_{20} p_{30} p_{01} p_{11} p_{21} p_{31} p_{02} p_{12} p_{22} p_{32} p_{03} p_{13} p_{23} p_{33}
ADDER TREE
ADDER
STAGE-1
STAGE-2
q_0 q_1 q_2 q_3
<<2
<<4
SHIFT-ADD-TREE

Fig. 7. Adder-structure of the filtering unit for $N = 4$ and $L = 8$.



input samples, x_n
D
$x_{n\text{-}1}$ $x_{n\text{-}2}$ $x_{n\text{-}3}$ $x_{n\text{-}N}$
PPG-0
PPG-1
PPG-(N-1)
<<1
μe_n
-1
r_{00} r_{01} $r_{0\frac{L}{2}-1}$
r_{10} r_{11} $r_{1\frac{L}{2}-1}$
$r_{(N\text{-}1)0}$ $r_{(N\text{-}1)1}$ $r_{(N\text{-}1)\frac{L}{2}-1}$
SHIFT-ADD-TREE
s_0 s_1 $s_{N\text{-}1}$
$w_{n-1-n_2}(0)$
$w_{n-1-n_2}(1)$
$w_{n-1-n_2}(N-1)$

Fig. 8. Proposed structure of the weight-update block.

delayed by $n_2 + 1$ cycles. However, it should be noted that the delay by 1 cycle is due to the latch before the PPG, which is included in the delay of the error-computation block, i.e., n_1. Therefore, the delay generated in the weight-update block becomes n_2. If the locations of pipeline latches are decided as in Table I, n_1 becomes 5, where three latches are in the error-computation block, one latch is after the subtraction in Fig. 4, and the other latch is before PPG in Fig. 8. Also, n_2 is set to 1 from a latch in the shift-add tree in the weight-update block.

IV. Fixed-Point Implementation, Optimization, Simulation, and Analysis

In this section, we discuss the fixed-point implementation and optimization of the proposed DLMS adaptive filter. A bit-level pruning of the adder tree is also proposed to reduce the

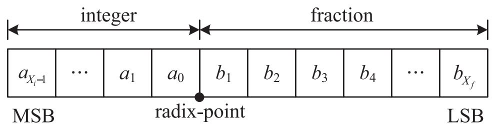


Fig. 9. Fixed-point representation of a binary number (X_i: integer word-length; X_f: fractional word-length).

TABLE II
FIXED-POINT REPRESENTATION OF THE SIGNALS OF THE PROPOSED DLMS ADAPTIVE FILTER ($\mu = 2^{-(L_i+\log_2 N)}$)

Signal Name	Fixed-Point Representation
x	(L, L_i)
w	(W, W_i)
p	$(W+2, W_i+2)$
q	$(W+2+\log_2 N, W_i+2+\log_2 N)$
y, d, e	$(W, W_i+L_i+\log_2 N)$
μe	(W, W_i)
r	$(W+2, W_i+2)$
s	(W, W_i)

x, **w**, **p**, **q**, y, d, and e can be found in the error-computation block of Fig. 4. μe, **r**, and **s** are defined in the weight-update block in Fig. 8. It is to be noted that all the subscripts and time indices of signals are omitted for simplicity of notation.

hardware complexity without noticeable degradation of steady-state MSE.

A. Fixed-Point Design Considerations

For fixed-point implementation, the choice of word lengths and radix points for input samples, weights, and internal signals need to be decided. Fig. 9 shows the fixed-point representation of a binary number. Let (X, X_i) be a fixed-point representation of a binary number where X is the word length and X_i is the integer length. The word length and location of radix point of x_n and w_n in Fig. 4 need to be predetermined by the hardware designer taking the design constraints, such as desired accuracy and hardware complexity, into consideration. Assuming (L, L_i) and (W, W_i), respectively, as the representations of input signals and filter weights, all other signals in Figs. 4 and 8 can be decided as shown in Table II. The signal p_{ij}, which is the output of PPG block (shown in Fig. 4), has at most three times the value of input coefficients. Thus, we can add two more bits to the word length and to the integer length of the coefficients to avoid overflow. The output of each stage in the adder tree in Fig. 7 is one bit more than the size of input signals, so that the fixed-point representation of the output of the adder tree with $\log_2 N$ stages becomes $(W+\log_2 N+2, W_i+\log_2 N+2)$. Accordingly, the output of the shift–add tree would be of the form $(W+L+\log_2 N, W_i+L_i+\log_2 N)$, assuming that no truncation of any least significant bits (LSB) is performed in the adder tree or the shift–add tree. However, the number of bits of the output of the shift–add tree is designed to have W bits. The most significant W bits need to be retained out of $(W+L+\log_2 N)$ bits, which results in the fixed-point representation $(W, W_i+L_i+\log_2 N)$ for y, as shown in Table II. Let the representation of the desired signal d be the same as y, even though its quantization is usually given as the input. For this purpose, the specific scaling/sign extension and truncation/zero padding are required. Since the LMS algorithm performs learning so that y has the same sign as d, the error signal e can also be set to have the same representation as y without overflow after the subtraction.

It is shown in [4] that the convergence of an N-tap DLMS adaptive filter with n_1 adaptation delay will be ensured if

$$0 < \mu < \frac{2}{(\sigma_x^2(N-2)+2n_1-2)\sigma_x^2} \qquad (5)$$

where σ_x^2 is the average power of input samples. Furthermore, if the value of μ is defined as (power of 2) 2^{-n}, where $n \leq W_i+L_i+\log_2 N$, the multiplication with μ is equivalent to the change of location of the radix point. Since the multiplication with μ does not need any arithmetic operation, it does not introduce any truncation error. If we need to use a smaller step size, i.e., $n > W_i+L_i+\log_2 N$, some of the LSBs of e_n need to be truncated. If we assume that $n = L_i+\log_2 N$, i.e., $\mu = 2^{-(L_i+\log_2 N)}$, as in Table II, the representation of μe_n should be (W, W_i) without any truncation. The weight increment term **s** (shown in Fig. 8), which is equivalent to $\mu e_n x_n$, is required to have fixed-point representation $(W+L, W_i+L_i)$. However, only W_i MSBs in the computation of the shift–add tree of the weight-update circuit are to be retained, while the rest of the more significant bits of MSBs need to be discarded. This is in accordance with the assumptions that, as the weights converge toward the optimal value, the weight increment terms become smaller, and the MSB end of error term contains more number of zeros. Also, in our design, $L-L_i$ LSBs of weight increment terms are truncated so that the terms have the same fixed-point representation as the weight values. We also assume that no overflow occurs during the addition for the weight update. Otherwise, the word length of the weights should be increased at every iteration, which is not desirable. The assumption is valid since the weight increment terms are small when the weights are converged. Also when overflow occurs during the training period, the weight updating is not appropriate and will lead to additional iterations to reach convergence. Accordingly, the updated weight can be computed in truncated form (W, W_i) and fed into the error-computation block.

B. Computer Simulation of the Proposed DLMS Filter

The proposed fixed-point DLMS adaptive filter is used for system identification used in Section II. μ is set to 0.5, 0.25, and 0.125 for filter lengths 8, 16, and 32, respectively, such that the multiplication with μ does not require any additional circuits. For the fixed-point simulation, the word length and radix point of the input and coefficient are set to $L = 16$, $L_i = 2$, $W = 16$, $W_i = 0$, and the Gaussian random input x_n of zero mean and unit variance is scaled down to fit in with the representation of $(16, 2)$. The fixed-point data type of all the other signals are obtained from Table II. Each learning curve is averaged over 50 runs to obtain a clean curve. The proposed design was coded in C++ using SystemC fixed-point library for different orders of the band-pass filter, that is, $N = 8$, $N = 16$,

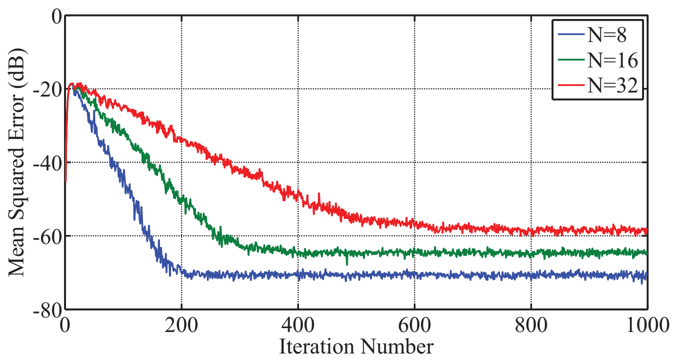


Fig. 10. Mean-squared-error of the fixed-point DLMS filter output for system identification for $N = 8$, 16, and 32.

and $N = 32$. The corresponding convergence behaviors are obtained, as shown in Fig. 10. It is found that, as the filter order increases, not only the convergence becomes slower, but the steady-state MSE also increases.

C. Steady-State Error Estimation

In this section, the MSE of output of the proposed DLMS adaptive filter due to the fixed-point quantization is analyzed. Based on the models introduced in [16] and [17], the MSE of output in the steady state is derived in terms of parameters listed in Table II. Let us denote the primed symbols as the truncated quantities due to the fixed-point representation, so that the input and the desired signals can be written as

$$\mathbf{x}'_n = \mathbf{x}_n + \boldsymbol{\alpha}_n \tag{6}$$
$$d'_n = d_n + \beta_n \tag{7}$$

where $\boldsymbol{\alpha}_n$ and β_n are input quantization noise vector and quantization noise of desired signal, respectively. The weight vector can be written as

$$\mathbf{w}'_n = \mathbf{w}_n + \boldsymbol{\rho}_n \tag{8}$$

where $\boldsymbol{\rho}_n$ is the error vector of current weights due to the finite precision. The output signal y'_n and weight-update equation can accordingly be modified, respectively, to the forms

$$y'_n = \mathbf{w}'^T_n \mathbf{x}'_n + \eta_n \tag{9}$$
$$\mathbf{w}'_{n+1} = \mathbf{w}'_n + \mu e'_n \mathbf{x}'_n + \boldsymbol{\gamma}_n \tag{10}$$

where η_n and $\boldsymbol{\gamma}_n$ are the errors due to the truncation of output from the shift–add tree in the error-computation block and weight-update block, respectively. The steady-state MSE in the fixed-point representation can be expressed as

$$E|d_n - y'_n|^2 = E|e_n|^2 + E|\boldsymbol{\alpha}_n^T \mathbf{w}_n|^2 + E|\eta_n|^2 + E|\boldsymbol{\rho}_n^T \mathbf{x}_n|^2 \tag{11}$$

where $E|\cdot|$ is the operator for mathematical expectation, and the terms e_n, $\boldsymbol{\alpha}_n^T \mathbf{w}_n$, η_n, and $\boldsymbol{\rho}_n^T \mathbf{x}_n$ are assumed to be uncorrelated.

The first term $E|e_n|^2$, where $e_n = d_n - y_n$, is the excess MSE from infinite precision computation, whereas the other three terms are due to finite-precision arithmetic.

TABLE III
Estimated and Simulated Steady-State MSEs of the Fixed-Point DLMS Adaptive Filter ($L = W = 16$)

Filter Length	Step Size (μ)	Simulation	Analysis
$N = 8$	2^{-1}	-71.01 dB	-70.97 dB
$N = 16$	2^{-2}	-64.84 dB	-64.97 dB
$N = 32$	2^{-3}	-58.72 dB	-58.95 dB

The second term can be calculated as

$$E|\boldsymbol{\alpha}_n^T \mathbf{w}_n|^2 = |\mathbf{w}_n^*|^2 (m_{\alpha_n}^2 + \sigma_{\alpha_n}^2) \tag{12}$$

where $\mathbf{w}_n^*$ is the optimal Wiener vector, and m_{α_n} and $\sigma_{\alpha_n}^2$ are defined as the mean and variance of α_n when x_n is truncated to the fixed-point type of (L, L_i), as listed in Table II. α_n can be modeled as a uniform distribution with following mean and variance:

$$m_{\alpha_n} = 2^{-(L-L_i)}/2 \tag{13a}$$
$$\sigma_{\alpha_n}^2 = 2^{-2(L-L_i)}/12. \tag{13b}$$

For the calculation of the third term $E|\eta_n|^2$ in (11), we have used the fact that the output from shift–add tree in the error-computation block is of the type $(W, W_i + L_i + \log_2 N)$ after the final truncation. Therefore

$$E|\eta_n|^2 = m_{\eta_n}^2 + \sigma_{\eta_n}^2. \tag{14}$$

where

$$m_{\eta_n}^2 = 2^{-(W-(W_i+L_i+\log_2 N))}/2 \tag{15a}$$
$$\sigma_{\eta_n}^2 = 2^{-2(W-(W_i+L_i+\log_2 N))}/12. \tag{15b}$$

The last term $E|\boldsymbol{\rho}_n^T \mathbf{x}_n|^2$ in (11) can be obtained by using the derivation proposed in [17] as

$$E|\boldsymbol{\rho}_n^T \mathbf{x}_n|^2 = m_{\gamma_n}^2 \frac{\Sigma_i \Sigma_k \left(\boldsymbol{R}_{ki}^{-1}\right)}{\mu^2} + \frac{N\left(\sigma_{\gamma_n}^2 - m_{\gamma_n}^2\right)}{2\mu} \tag{16}$$

where $\boldsymbol{R}_{ki}$ represents the (k, i)th entry of the matrix $E(\mathbf{x}_n \mathbf{x}_n^T)$. For the weight update in (10), the first operation is to multiply e'_n with μ, which is equivalent to moving only the location of the radix point and, therefore, does not introduce any truncation error. The truncation after multiplication of $\mu e'_n$ with x_n is only required to be considered in order to evaluate γ_n. Then, we have

$$m_{\gamma_n}^2 = 2^{-(W-W_i)}/2 \tag{17a}$$
$$\sigma_{\gamma_n}^2 = 2^{-2(W-W_i)}/12. \tag{17b}$$

For a large μ, the truncation error η_n from the error-computation block becomes the dominant error source, and (11) can be approximated as $E|\eta_n|^2$. The MSE values are estimated from analytical expressions as well as from the simulation results by averaging over 50 experiments. Table III shows that the steady-state MSE computed from analytical expression matches with that of simulation of the proposed architecture for different values of N and μ.

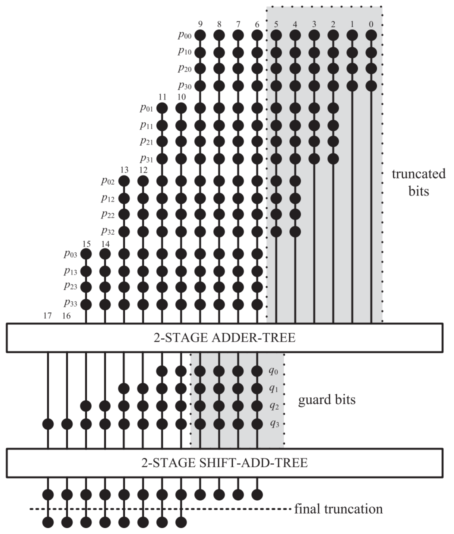


Fig. 11. Dot-diagram for optimization of the adder tree in the case of $N = 4$, $L = 8$, and $W = 8$.

D. Adder-Tree Optimization

The adder tree and shift–add tree for the computation of y_n can be pruned for further optimization of area, delay, and power complexity. To illustrate the proposed pruning optimization of adder tree and shift–add tree for the computation of filter output, we take a simple example of filter length $N = 4$, considering the word lengths L and W to be 8. The dot diagram of the adder tree is shown in Fig. 11. Each row of the dot diagram contains 10 dots, which represent the partial products generated by the PPG unit, for $W = 8$. We have four sets of partial products corresponding to four partial products of each multiplier, since $L = 8$. Each set of partial products of the same weight values contains four terms, since $N = 4$. The final sum without truncation should be 18 b. However, we use only 8 b in the final sum, and the rest 10 b are finally discarded. To reduce the computational complexity, some of the LSBs of inputs of the adder tree can be truncated, while some guard bits can be used to minimize the impact of truncation on the error performance of the adaptive filter. In Fig. 11, four bits are taken as the guard bits and the rest six LSBs are truncated. To have more hardware saving, the bits to be truncated are not generated by the PPGs, so the complexity of PPGs also gets reduced.

η_n defined in (9) increases if we prune the adder tree, and the worst case error is caused when all the truncated bits are 1. For the calculation of the sum of truncated values in the worst case, let us denote k_1 as the bit location of MSB of truncated bits and Nk_2 as the number of rows that are affected by the truncation. In the example of Fig. 11, k_1 and k_2 are set to 5 and 3, respectively, since the bit positions from 0 to 5 are truncated and a total of 12 rows are affected by the truncation for $N = 4$. Also, k_2 can be derived using k_1 as

$$k_2 = \lfloor \frac{k_1}{2} \rfloor + 1 \text{ for } k_1 < \frac{W}{2}, \quad \text{otherwise } k_2 = \frac{W}{2} \tag{18}$$

since the number of truncated bits is reduced by 2 for every group of N rows as shown in Fig. 11. Using k_1, k_2, and N, the sum of truncated values for the worst case can be formulated as

$$b_{\text{worst}} = N \sum_{j=0}^{k_2-1} \sum_{i=2j}^{k_1} 2^i = N \left(k_2 2^{k_1+1} - \frac{1}{3}(4^{k_2} - 1) \right). \tag{19}$$

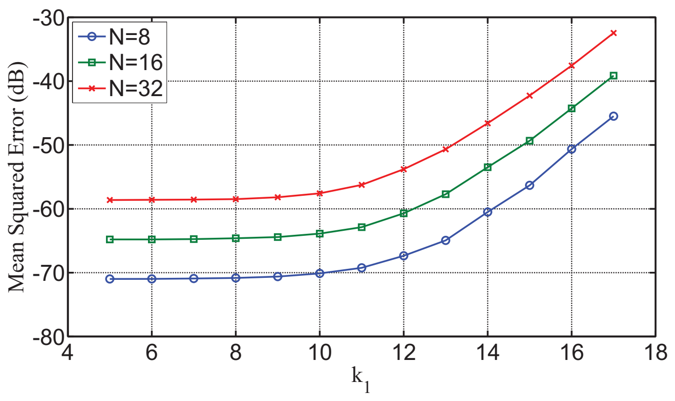


Fig. 12. MSE at the steady-state versus k_1 for $N = 8$, 16, and 32 ($L = W = 16$).

In the example of Fig. 11, b_{worst} amounts to 684. Meanwhile, the LSB weight of the output of adder tree after final truncation is 2^{10} in the example. Therefore, there might be one bit difference in the output of adder tree due to pruning. The truncation error from each row (total 12 rows from row p_{00} to row p_{32} in Fig. 11) has a uniform distribution, and if the individual errors is assumed to be independent of each other, the mean and variance of the total error introduced can be calculated as the sum of means and variances of each random variable. However, it is unlikely that outputs from the same PPG are uncorrelated since it is generated from the same input sample. It would not be straightforward to estimate the distribution of error from the pruning. However, as the value of b_{worst} is closer to or larger than the LSB weight of the output after final truncation, the pruning will affect the overall error more. Fig. 12 illustrates the steady-state MSE in terms of k_1 for $N = 8$, 16, and 32 when $L = W = 16$ to show how much the pruning affects the output MSE. When k_1 is less than 10 for $N = 8$, the MSE deterioration is less than 1 dB compared to the case when the pruning is not applied.

V. Complexity Considerations

The hardware and time complexities of proposed design, those of the structure of [11] and the best of systolic structures [10] are listed in Table IV. The original DLMS structure proposed in [4] is also listed in this table. It is found that the proposed design has a shorter critical path of one addition time as that of [11], and lower adaptation delay than the others. If we consider each multiplier to have $(L - 1)$ adders, then the existing designs involve $16N$ adders, while the proposed one involves $10N + 2$ adders for $L = 8$. Similarly, it involves less number of delay registers compared with others.

We have coded the proposed designs in VHDL and synthesized by the Synopsys Design Compiler using CMOS 65-nm library for different filter orders. The word length of the input samples and weights are chosen to be 8, i.e., $L = W = 8$. The step size μ is chosen to be $1/2^{L_i + \log_2 N}$ to realize

TABLE IV

COMPARISON OF HARDWARE AND TIME COMPLEXITIES OF DIFFERENT ARCHITECTURES FOR $L = 8$

Design	Critical Path	n_1	n_2	Hardware Elements		
				No. of Adders	No. of Multipliers	No. of Registers
Long et al [4]	$T_M + T_A$	$\log_2 N + 1$	0	$2N$	$2N$	$3N + 2\log_2 N + 1$
Ting *et al.* [11]	T_A	$\log_2 N + 5$	3	$2N$	$2N$	$10N + 8$
Van and Feng [10]	$T_M + T_A$	$N/4 + 3$	0	$2N$	$2N$	$5N + 3$
Proposed Design	T_A	5	1	$10N + 2$	0	$2N + 14 + E^\dagger$

$^\dagger E = 24,\ 40$ and 48 for $N = 8,\ 16$ and 32, respectively. Besides, proposed design needs additional $24N$ AND cells and $16N$ OR cells. The 2s complement operator in Figs. 5 and 8 is counted as one adder, and it is assumed that the multiplication with the step size does not need the multiplier over all the structures.

TABLE V

PERFORMANCE COMPARISON OF DLMS ADAPTIVE FILTER BASED ON SYNTHESIS RESULT USING CMOS 65-nm LIBRARY

Design	Filter Length, N	DAT (ns)	Latency (cycles)	Area (sq.μm)	Leakage Power (mW)	EPS (mW$\times$ns)	ADP (sq. μm$\times$ ns)	EDP (mW$\times$ns^2)	ADP Reduction	EDP Reduction
Ting *et al* [11]	8	1.14	8	24204	0.13	18.49	26867	21.08	–	–
	16	1.19	9	48049	0.27	36.43	55737	43.35	–	–
	32	1.25	10	95693	0.54	72.37	116745	90.47	–	–
Van and Feng [10]	8	1.35	5	13796	0.08	7.29	18349	9.84	–	–
	16	1.35	7	27739	0.16	14.29	36893	19.30	–	–
	32	1.35	11	55638	0.32	27.64	73998	37.31	–	–
Proposed Design-I	8	1.14	5	14029	0.07	8.53	15572	9.72	15.13%	1.14%
	16	1.19	5	26660	0.14	14.58	31192	17.34	15.45%	10.10%
	32	1.25	5	48217	0.27	21.00	58824	26.25	20.50%	29.64%
Proposed Design-II	8	0.99	5	12765	0.06	8.42	12382	8.33	32.51%	15.22%
	16	1.15	5	24360	0.13	14.75	27526	16.96	25.38%	12.07%
	32	1.15	5	43233	0.24	21.23	48853	24.41	33.98%	34.55%

DAT: data-arrival time; ADP: area–delay product; EPS: energy per sample; EDP: energy–delay product. ADP and EDP reductions in last two columns are improvements of proposed designs over [10] in percentage. Proposed Design-I: without optimization, Proposed Design-II: after optimization of adder tree with $k_1 = 5$.

its multiplication without any additional circuitry. The word length of all the other signals are determined based on the types listed in Table II. We have also coded structures proposed in [10] and [11] using VHDL, and synthesized using the same library and synthesis options in the Design Compiler for a fair comparison. In Table V, we have shown the synthesis results of the proposed designs and existing designs in terms of data arrival time (DAT), area, energy per sample (EPS), ADP, and EDP obtained for filter lengths $N = 8$, 16, and 32. The proposed design-I before pruning of the adder tree has the same DAT as the design in [11] since the critical paths of both designs are same as T_A as shown in Table IV, while the design in [10] has a longer DAT which is equivalent to $T_A + T_M$. However, the proposed design-II after the pruning of the adder tree has a slightly smaller DAT than the existing designs. Also, the proposed designs could reduce the area by using a PPG based on common subexpression sharing, compared to the existing designs. As shown in Table V, the reduction in area is more significant in the case of $N = 32$ since more sharing can be obtained in the case of large order filters. The proposed designs could achieve less area and more power reduction compared with [11] by removing redundant pipeline latches, which are not required to maintain a critical path of one addition time. It is found that the proposed design-I

TABLE VI

FPGA IMPLEMENTATIONS OF PROPOSED DESIGNS FOR $L = 8$ AND $N = 8$, 16, AND 32

Design	Proposed Design-I		Proposed Design-II	
	NOS	MUF	NOS	MUF
Xilinx Virtex-4 (XC4VSX35-10FF668)				
$N = 8$	1024	148.7	931	151.6
$N = 16$	2036	121.2	1881	124.6
$N = 32$	4036	121.2	3673	124.6
Xilinx Spartan-3A DSP (XC3SD1800A-4FG676)				
$N = 8$	1025	87.2	966	93.8
$N = 16$	2049	70.3	1915	74.8
$N = 32$	4060	70.3	3750	75.7

NOS stands for the number of slices. MUF stands for the maximum usable frequency in [MHz].

involves $\sim 17\%$ less ADP and $\sim 14\%$ less EDP than the best previous work of [10], on average, for filter lengths $N = 8$, 16, and 32. The proposed design-II, similarly, achieves $\sim 31\%$ less ADP and nearly $\sim 21\%$ less EDP than the structure of [10] for the same filters. The optimization of the adder tree of the proposed structure with $k_1 = 5$ offers $\sim 20\%$ less ADP and $\sim 9\%$ less EDP over the structure before optimization of the adder tree.

The proposed designs were also implemented on the field-programmable gate array (FPGA) platform of Xilinx devices. The number of slices (NOS) and the maximum usable frequency (MUF) using two different devices of Spartan-3A (XC3SD1800A-4FG676) and Virtex-4 (XC4VSX35-10FF668) are listed in Table VI. The proposed design-II, after the pruning, offers nearly 11.86% less slice-delay product, which is calculated as the average NOS/MUF, for $N = 8$, 16, 32, and two devices.

VI. Conclusion

We proposed an area–delay-power efficient low adaptation-delay architecture for fixed-point implementation of LMS adaptive filter. We used a novel PPG for efficient implementation of general multiplications and inner-product computation by common subexpression sharing. Besides, we have proposed an efficient addition scheme for inner-product computation to reduce the adaptation delay significantly in order to achieve faster convergence performance and to reduce the critical path to support high input-sampling rates. Aside from this, we proposed a strategy for optimized balanced pipelining across the time-consuming blocks of the structure to reduce the adaptation delay and power consumption, as well. The proposed structure involved significantly less adaptation delay and provided significant saving of ADP and EDP compared to the existing structures. We proposed a fixed-point implementation of the proposed architecture, and derived the expression for steady-state error. We found that the steady-state MSE obtained from the analytical result matched well with the simulation result. We also discussed a pruning scheme that provides nearly 20% saving in the ADP and 9% saving in EDP over the proposed structure before pruning, without a noticeable degradation of steady-state error performance. The highest sampling rate that could be supported by the ASIC implementation of the proposed design ranged from about 870 to 1010 MHz for filter orders 8 to 32. When the adaptive filter is required to be operated at a lower sampling rate, one can use the proposed design with a clock slower than the maximum usable frequency and a lower operating voltage to reduce the power consumption further.

References

[1] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1985.

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

[3] M. D. Meyer and D. 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.

[4] G. Long, F. Ling, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 9, pp. 1397–1405, Sep. 1989.

[5] G. Long, F. Ling, and J. G. Proakis, "Corrections to 'The LMS algorithm with delayed coefficient adaptation'," *IEEE Trans. Signal Process.*, vol. 40, no. 1, pp. 230–232, Jan. 1992.

[6] H. Herzberg and R. Haimi-Cohen, "A systolic array realization of an LMS adaptive filter and the effects of delayed adaptation," *IEEE Trans. Signal Process.*, vol. 40, no. 11, pp. 2799–2803, Nov. 1992.

[7] M. D. Meyer and D. P. Agrawal, "A high sampling rate delayed LMS filter architecture," *IEEE Trans. Circuits Syst. II, Analog Digital Signal Process.*, vol. 40, no. 11, pp. 727–729, Nov. 1993.

[8] S. Ramanathan and V. Visvanathan, "A systolic architecture for LMS adaptive filtering with minimal adaptation delay," in *Proc. Int. Conf. Very Large Scale Integr. (VLSI) Design*, Jan. 1996, pp. 286–289.

[9] Y. Yi, R. Woods, L.-K. Ting, and C. F. N. Cowan, "High speed FPGA-based implementations of delayed-LMS filters," *J. Very Large Scale Integr. (VLSI) Signal Process.*, vol. 39, nos. 1–2, pp. 113–131, Jan. 2005.

[10] L. D. Van and W. S. Feng, "An efficient systolic architecture for the DLMS adaptive filter and its applications," *IEEE Trans. Circuits Syst. II, Analog Digital Signal Process.*, vol. 48, no. 4, pp. 359–366, Apr. 2001.

[11] L.-K. Ting, R. Woods, and C. F. N. Cowan, "Virtex FPGA implementation of a pipelined adaptive LMS predictor for electronic support measures receivers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 1, pp. 86–99, Jan. 2005.

[12] P. K. Meher and M. Maheshwari, "A high-speed FIR adaptive filter architecture using a modified delayed LMS algorithm," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2011, pp. 121–124.

[13] P. K. Meher and S. Y. Park, "Low adaptation-delay LMS adaptive filter part-I: Introducing a novel multiplication cell," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Aug. 2011, pp. 1–4.

[14] P. K. Meher and S. Y. Park, "Low adaptation-delay LMS adaptive filter part-II: An optimized architecture," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Aug. 2011, pp. 1–4.

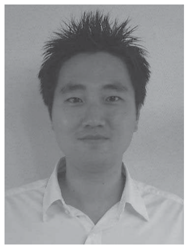
[15] K. K. Parhi, *VLSI Digital Signal Procesing Systems: Design and Implementation*. New York, USA: Wiley, 1999.

[16] C. Caraiscos and B. Liu, "A roundoff error analysis of the LMS adaptive algorithm," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 1, pp. 34–41, Feb. 1984.

[17] R. Rocher, D. Menard, O. Sentieys, and P. Scalart, "Accuracy evaluation of fixed-point LMS algorithm," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, May 2004, pp. 237–240.

Pramod Kumar Meher (SM'03) is currently a Senior Scientist with the Institute for Infocomm Research, Singapore. His research interests include design of dedicated and reconfigurable architectures for computation-intensive algorithms pertaining to signal, image and video processing, communication, bio-informatics and intelligent computing.

Dr. Meher is a Fellow of the Institution of Electronics and Telecommunication Engineers, India. He has served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS during 2008–2011 and as a speaker for the Distinguished Lecturer Program (DLP) of IEEE Circuits Systems Society during 2011–2012. He is continuing to serve as Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and *Journal of Circuits, Systems, and Signal Processing*. He was a recipient of the Samanta Chandrasekhar Award for excellence in research in engineering and technology for the year 1999.



Sang Yoon Park (S'03–M'11) received the B.S., M.S., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science, Seoul National University, Seoul, Korea, in 2000, 2002, and 2006, respectively.

He joined the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, as a Research Fellow in 2007. Since 2008, he has been with the Institute for Infocomm Research, Singapore, where he is currently a Research Scientist. His research interests include dedicated/reconfigurable architectures and algorithms for low-power/low-area/high-performance digital signal processing and communication systems.