International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

DELAY CAPABILITY OF POWERFUL FIXED MOTE FLEXIBLE FILTER LOW ADAPTATION DELAY

C.Rahul Menon[1], T.Gini M.E[2]

[1]II M.E (Applied Electronics), PSN College of Engineering and Technology, Tirunelveli.
[2]Email: menonc.rahul@gmail.com
[2]Asst. Professor, Department of ECE, PSN College of Engineering and Technology, Tirunelveli.
[1]Email: jiniashika@gmail.com

***ABSTRACT* : The filter process mainly used to DSP and DIP real world application. The filter process to remove the noise in original signal or image. So the filter architecture optimized process, to reduce the filter processing time and to increase the performance. Adaptive digital filters find wide application in several digital signal processing (DSP) areas. This paper presents the modified delayed LMS adaptive filter consists of Weight update block with Partial Product Generator (PPG) to achieve a lower adaptation delay and efficient area, power, delay. To achieve lower adaptation delay, initially the transpose form LMS adaptive filter is designed but the output contains large delay due to its inner product process. Here, the pipelining structure is proposed across the time consuming combinational blocks of the structure to reduce the critical path. From the simulation results, we find that the proposed design n offers large efficient output comprises the existing output with large complexities.. We use the digital architecture based VLSI technology to modify the FIR filter architecture. In this architecture, For achieving lower adaptation-delay and area-delay-power efficient implementation, we use a novel partial product generator and to modify the efficient architecture for the implementation of a delayed least mean square adaptive filter using FAP algorithm. And to modify the DFE section. Our proposed work is to modify the filter architecture using FAP Adaptive algorithm and to reduce the delay unit in filter architecture. This algorithm to reduce the pattern count in the partial protect generation architecture and adder operator carry selection process. So it consumes low power. This algorithm used to reduce the path delay and improve the speed compare to proposed algorithm. This FAP algorithm mainly used to modify the filter architecture block.**

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 largeorder 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].

II. REVIEW OF DELAYED LMS ALGORITHM

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

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en \cdot \mathbf{x}n \text{ (1a)}$$

where

$$en = dn - yn \quad yn = \mathbf{w}T\, n \cdot \mathbf{x}n \text{ (1b)}$$

where the input vector **x***n*, and the weight vector **w***n* at the *n*th iteration are, respectively, given by

$$\mathbf{x}n = [xn, xn-1, \ldots, xn-N+1]T$$

$$\mathbf{w}n = [wn(0), wn(1), \ldots, wn(N-1)]T,$$

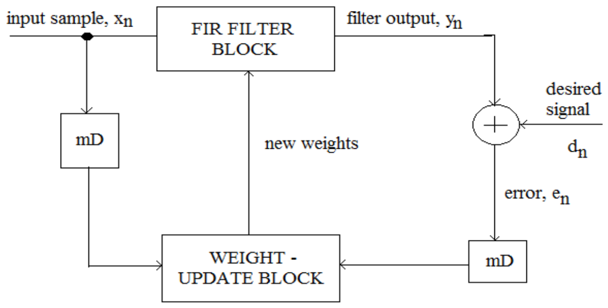


Fig 1 Structure of the conventional delayed LMS adaptive filter

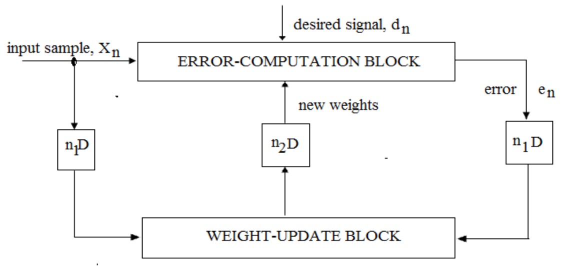


Fig 2 Structure of the modified delayed LMS adaptive filter.

dn is the desired response, *yn* is the filter output, and *en* denotes the error computed during the *n*th 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 *en* becomes available after *m* cycles, where *m* is called the "adaptation delay." The DLMS algorithm therefore uses the delayed error *en−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

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en-m \cdot \mathbf{x}n-m.$$

In this design filter coefficients are loaded in bit-parallel form with no increase in the number of input pins, thereby facilitating and speeding up run-time adaptation to the application environment. Another feature of variable-precision coefficients is that, it can be accommodated easily and flexibly, with no speed penalty. This type of design methods can be applied for the design of application- specific and embedded parallel architectures.

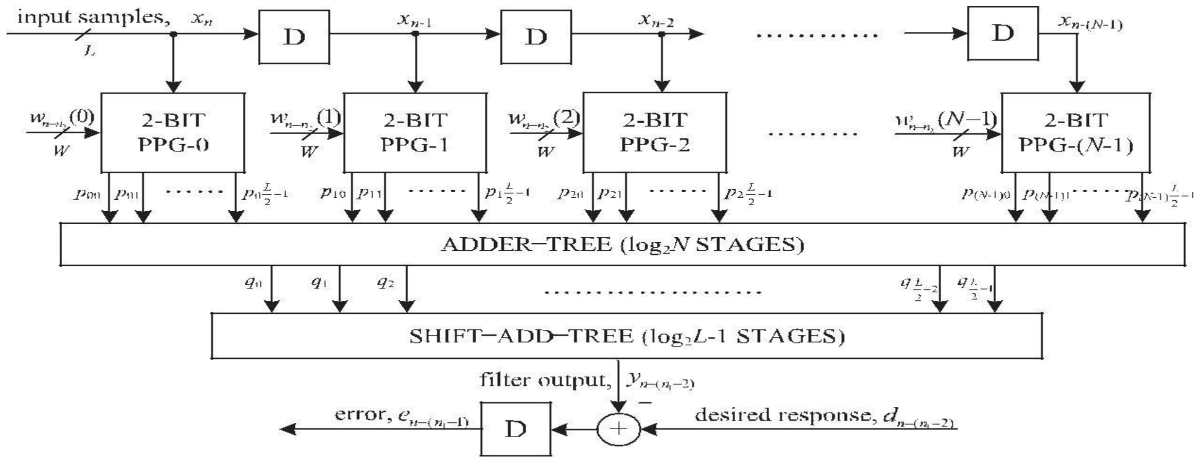
A. Pipelined Structure of the Error-Computation Block

The proposed structure for error-computation unit of an *N* -tap DLMS adaptive filter is shown in Fig. 1. 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. 2. 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 (u1u0) as input and produces three outputs b0 = u 0 • u¯1, b1 = u¯0 • u1, and b2 = u0 • u1, such that b0 = 1 for (u1u0) = 1, b1 = 1 for (u1u0) = 2, and b2 = 1 for (u1u0) = 3. The decoder output b0, b1 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 −1u L −2) of the input sample, the AOC (L /2 − 1) is fed with w, −2w, and −w as input since (u L −1u 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. 3. Each AOC consists of three AND cells and two OR cells. The structure and function of AND cells and

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



input samples, x_n
L
D
x_{n-1}
x_{n-2}
$x_{n-(N-1)}$
2-BIT PPG-0
2-BIT PPG-1
2-BIT PPG-2
2-BIT PPG-(N-1)
ADDER–TREE ($\log_2 N$ STAGES)
SHIFT–ADD–TREE ($\log_2 L$-1 STAGES)
filter output, $y_{n-(n_1-2)}$
error, $e_{n-(n_1-1)}$
desired response, $d_{n-(n_1-2)}$

Fig. 3. Proposed structure of the error-computation block.

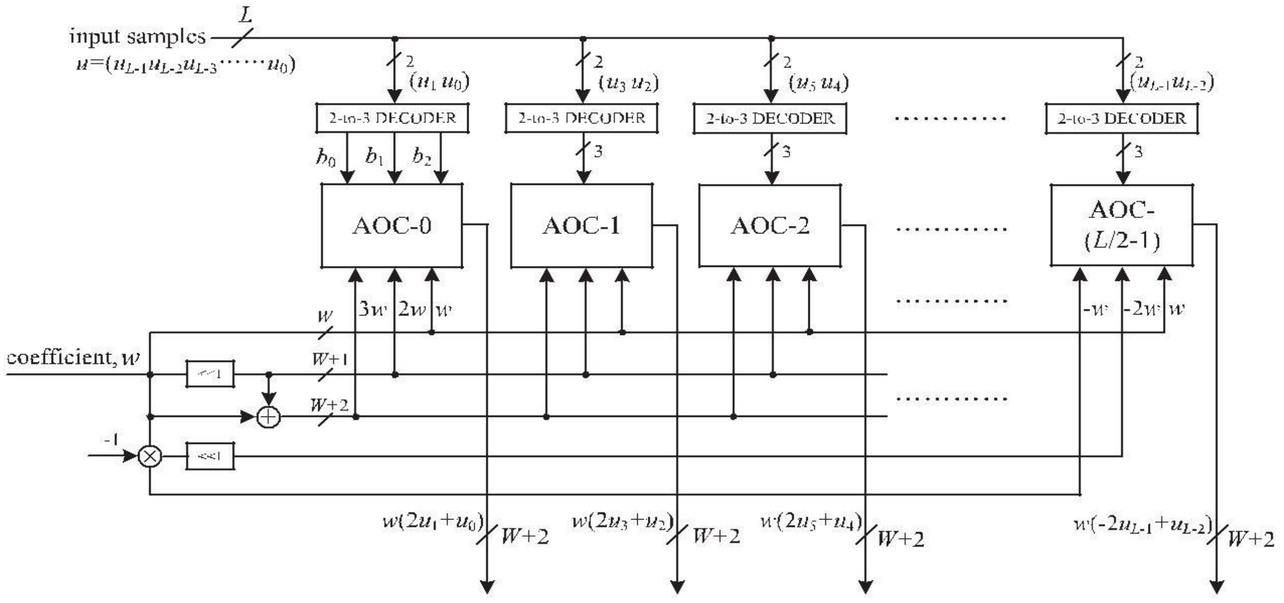


Fig.4.Proposed structure of PPG. AOC stands for AND/OR cell.

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. 4 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 for the error-computation block for a four-tap filter and input word size $L = 8$ is shown in Fig. 5. 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 $\log2 N + \log2 L - 1$ 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].

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 $(\mu \times e) \times xi + wi$ 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 xi followed by the additions with the corresponding old weight values wi . 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 errorvalue $\mu \times e$ with $L/2$, the number of 2-b digits of the input word xi , 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

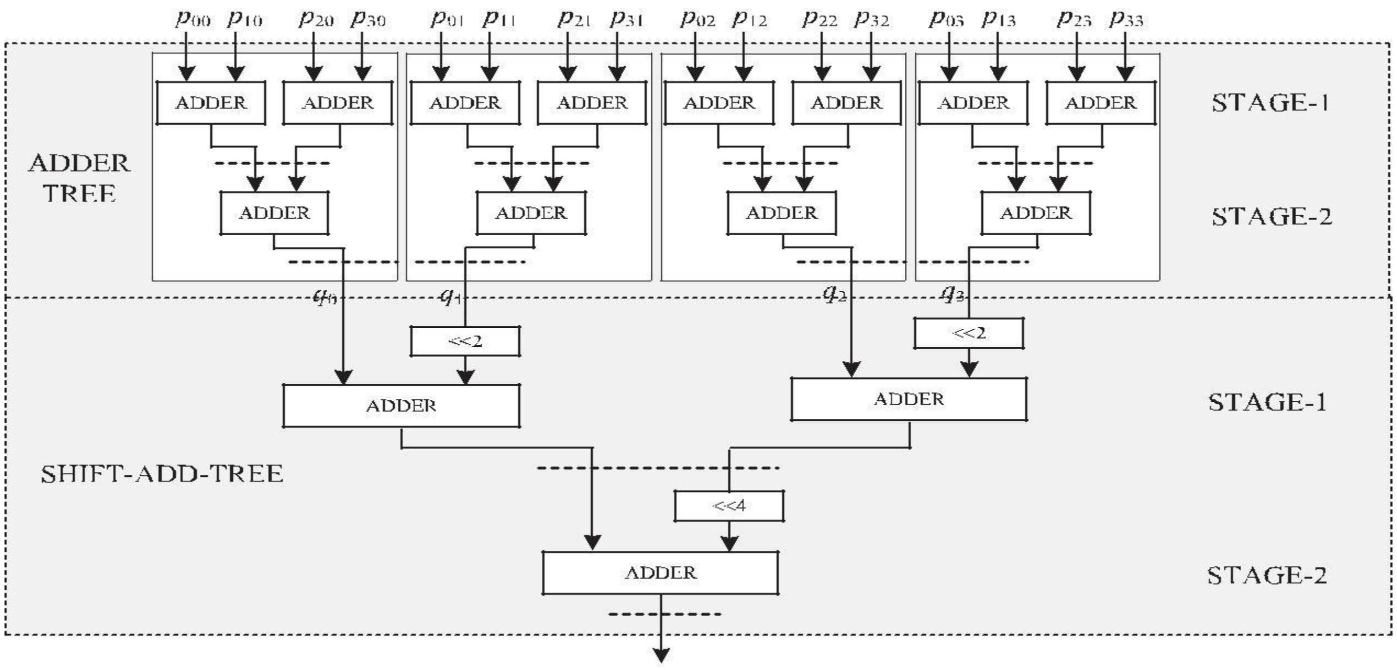


Fig. 5. Adder-structure of the filtering unit for N = 4 and L = 8

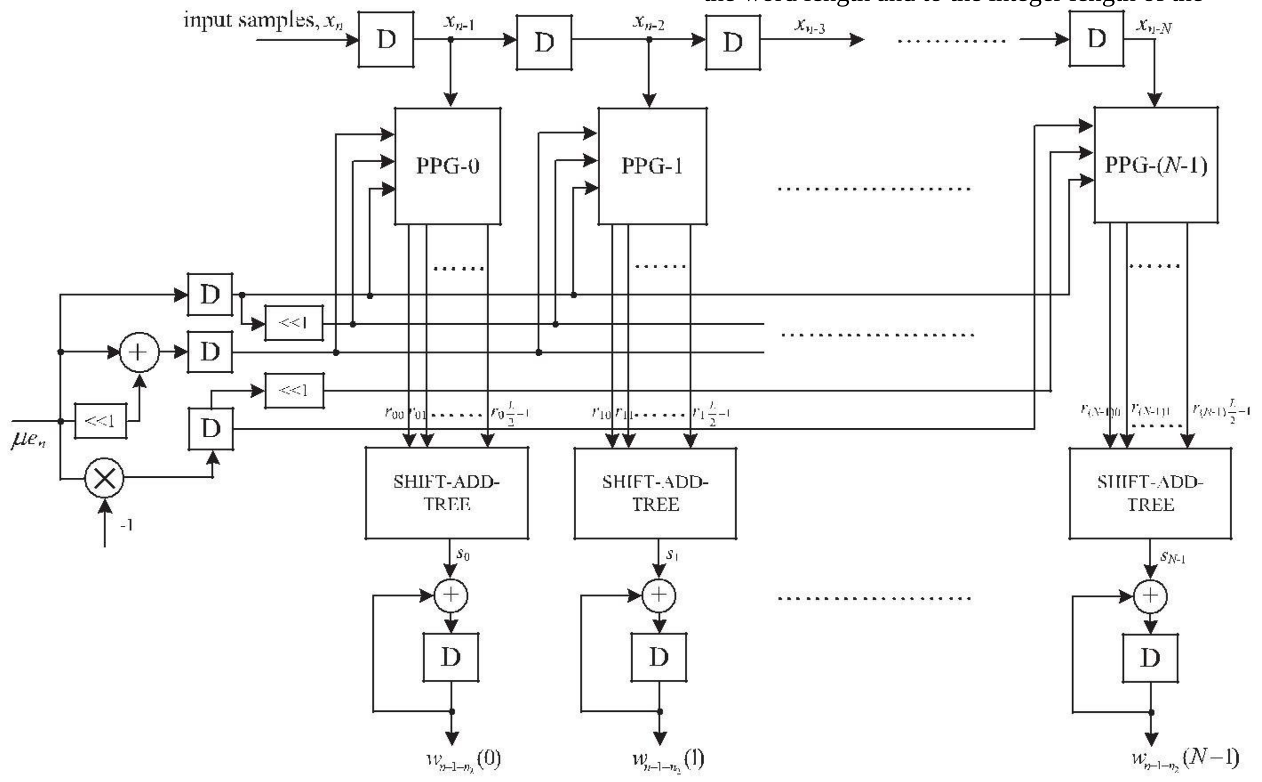


Fig. 6. Proposed structure of the weight-update block.

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. 6 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 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
It is shown in [4] that the convergence of an *N*-tap DLMS
adaptive filter with *n*1 adaptation delay will be ensured if

$0 < \mu < 2\ (\sigma 2\ x\ (N - 2) + 2n1 - 2)\sigma 2\ x$ (5)

where $\sigma 2\ x$ is the average power of input samples.Furthermore, if the value of μ is defined as (power of 2) 2^{-n} , where $n \le W_i + L_i + \log_2 N$, 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.

Adder-Tree Optimization

The adder tree and shift–add tree for the computation of yn can be pruned for further optimization of area, delay, and power complexity. To illustrate the proposed pruning optimiza-tion 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.

ηn 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 k1 as the bit location of MSB of truncated bits and N k2 as the number of rows that are affected by the truncation. In the example of Fig. 11, k1 and k2 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, k2 can be derived using k1 as k2 = _ k1 + 1 for k1 < W, otherwise k2 = W(18) 222 since the number of truncated bits is reduced by 2 for every group of N rows as shown in Fig. 11. Using k1, k2, and N , the k2 −1 k1 k22k1 +1 − 1 bworst = N 2i = N (4k2 − 1). (19) 3j =0 i =2 j
In the example of Fig. 11, bworst amounts to 684. Meanwhile, the LSB weight of the output of adder tree after final truncation is 210 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 p00 to

row p32 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 bworst 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 k1 for N = 8, 16, and 32 when L = W = 16 to show how much the pruning affects the output MSE. When k1 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.

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 implementa-tion 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 implemen-tation 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. Hobo-ken, 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] 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.

[11] 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.

[12] 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.