ISSN 2347 - 3983

International Journal of Emerging Trends in Engineering Research (IJETER), Vol. 3 No.6, Pages : 328 - 335 (2015)
Special Issue of NCTET 2K15 - Held on June 13, 2015 in SV College of Engineering, Tirupati
http://warse.org/IJETER/static/pdf/Issue/NCTET2015sp62.pdf

AN IMPROVED FIXED-POINT LMS & RLS ADAPTIVE FILTER WITH LOW ADAPTATION-DELAY

Dama Pavan Teja[1], Devi Padmaja.T[2]

[1]*PG Scholar, Dept of VLSI, SV College of Engineering, Tirupati,, AP, India,* pavantejadama@gmail.com

[2]*Asst. Professor, Dept of ECE, SV College of Engineering, Tirupati,, AP, India,,* devipadmaja.t@svcolleges.edu.in

ABSTRACT

An improved architecture for the implementation of a delayed least mean square adaptive filter is proposed in this paper. 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. But conventional LMS adaptive filter involves a long critical path due to an inner-product computation to obtain the filter output. That critical path is required to be reduced by pipelined implementation called delayed LMS (DLMS) adaptive filter. The conventional delayed LMS adaptive filter architecture occupies more area, more power wastage and less performance then compare with this proposed architecture. The proposed LMS design offers less area-delay product (ADP) and energy-delay product (EDP). Moreover, the proposed adaptive filter design is extended by replacing LMS algorithm to RLS (Recursive least squares) algorithm which leads to better performance, and also by adding bit-level pruning of the proposed architecture, which improves ADP and EDP further.

Keywords: adaptive filter, LMS, DLMS, RLMS, area-delay product and energy-delay product

I. INTRODUCTION

THE LEAST MEAN SQUARE (LMS) adaptive filter is the most popular and most widely used adaptive filter [1]. 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 [2], 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 but, they involve an adaptation delay of *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 have proposed a modified systolic architecture to reduce the adaptation delay. A transpose-form LMS adaptive filter is suggested in [3], 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*.

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 [4], [5] 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

ISSN 2347 - 3983

International Journal of Emerging Trends in Engineering Research (IJETER), Vol. 3 No.6, Pages : 328 - 335 (2015)
Special Issue of NCTET 2K15 - Held on June 13, 2015 in SV College of Engineering, Tirupati
http://warse.org/IJETER/static/pdf/Issue/NCTET2015sp62.pdf

proposed optimized architecture for its implementation. Section IV deals with fixed-point implementation considerations. Section V contains RLS. Section VI deals with results analysis and Conclusions are given in Section VII.

II. REVIEW OF DELAYED LMS ALGORITHM

LMS adaptive filter is used worldwide because of its easy computation and flexibility. This algorithm is a member of stochastic gradient algorithm , and because of its robustness and low computational complexity it is used worldwide. The algorithm using the steepest distance is as given below.

$$w_{n+1} = w_n + \mu . e_n . x_n \qquad (1a))$$

Where

$$e_n = d_n - y_n \qquad y_n = w_n^T \quad x_n \qquad (1b)$$

Where the input vector $\mathbf{x}_n$, and the weight vector $\mathbf{w}_n$ at the n^{th} iteration are, respectively, given by

$$x_n = [\ x_n, x_{n-1}, \ldots\ldots\ldots\ldots\ldots, x_n - N + 1\ ]^T$$

$$w_n = [\ w_n(0), w_n(1), \ldots\ldots\ldots\ldots\ldots, w_n(N-1)\ ]^T$$

d_n is the desired response, y_n 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

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en-m \cdot \mathbf{x}n-m. \quad (2)$$

The above (1a) (1b) two equations are required output of LMS algorithm where y_n is the filter output and e_n is the error. Figure below shows the block diagram of adaptive filter

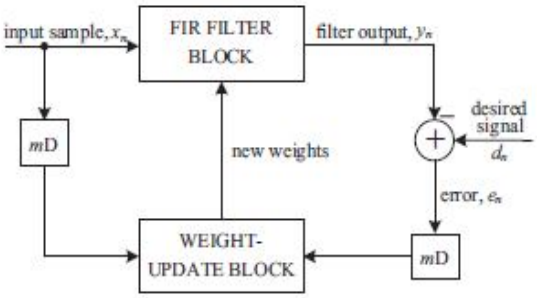


Figure 1: Structure of the conventional delayed LMS adaptive filter.

If the values of d_n and y_n will become equal we will get zero error (e_n). This filter could be used in combination of various other applications. There are number of parameters related to LMS adaptive filter, which could differently play an important role in order to reduce the error. Various applications are also there, which can also be analyzed using LMS filter. 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. 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.

III. PROPOSED ARCHITECTURE

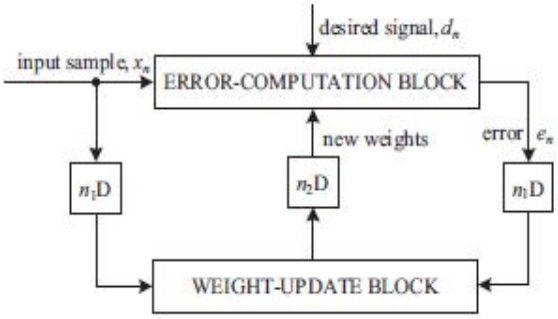


Figure 2: Structure of the modified delayed LMS adaptive filter.

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

ISSN 2347 - 3983

International Journal of Emerging Trends in Engineering Research (IJETER), Vol. 3 No.6, Pages : 328 - 335 (2015)
Special Issue of NCTET 2K15 - Held on June 13, 2015 in SV College of Engineering, Tirupati
http://warse.org/IJETER/static/pdf/Issue/NCTET2015sp62.pdf

computed by the structure at the nth cycle is $en-n1$, which is used with the input samples delayed by $n1$ cycles to generate the weight-increment term. The weight- update equation of the modified DLMS algorithm is given by

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \cdot e_{n-n1} \cdot \mathbf{x}_{n-n1} \quad (3a)$$

Where

$$e_{n-n1} = d_{n-n1} - y_{n-n1} \quad (3b)$$

And

$$y_n = \mathbf{w}^T_{n-n2} \cdot \mathbf{x}_n \quad (3c)$$

We notice that, during the weight update, the error with $n1$ delays is used, while the filtering unit uses the weights delayed by $n2$ 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.

As shown in Fig. 2, there are two main computing blocks in the adaptive filter architecture:
1) The error-computation block, and
2) weight-update block.
In this Section, we discuss the design strategy of the proposed structure to minimize the adaptation delay

A. Pipelined Structure of the Error-Computation Block

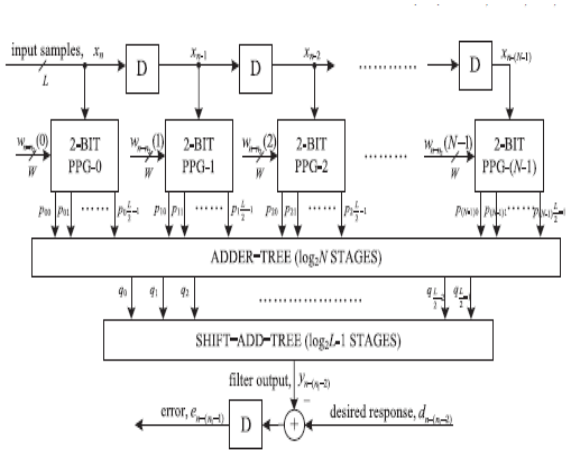


Fig. 3. Proposed structure of the error-computation block.

The proposed structure for error-computation unit of an N-tap DLMS adaptive filter is shown in Fig. 3. 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 sub block is described in detail.

1) Structure of PPG:

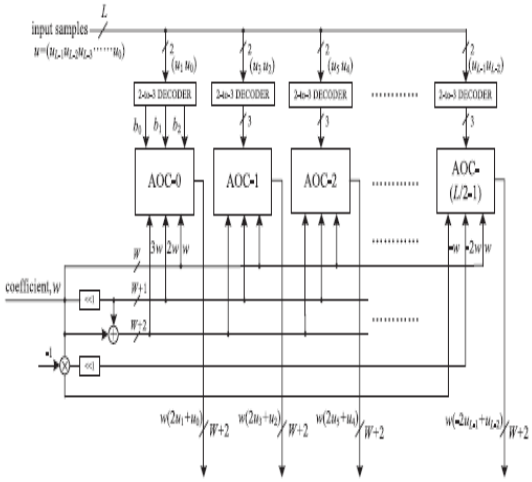


Fig. 4. Proposed structure of PPG.

The structure of each PPG is shown in Fig. 4. 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 $b0 = u0 \cdot . u1$, $b1 = . u0 \cdot u1$, and $b2 = u0 \cdot u1$, such that $b0 = 1$ for $(u1u0) = 1$, $b1 = 1$ for $(u1u0) = 2$, and $b2 = 1$ for $(u1u0) = 3$. The decoder output $b0$, $b1$ and $b2$ 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., $(uL-1uL-2)$ of the input sample, the AOC $(L/2 - 1)$ is fed with w, $-2w$, and $-w$ as input since $(uL-1uL-2)$ can have four possible values 0, 1, −2, and −1.

ISSN 2347 - 3983

International Journal of Emerging Trends in Engineering Research (IJETER), Vol. 3 No.6, Pages : 328 - 335 (2015)
Special Issue of NCTET 2K15 - Held on June 13, 2015 in SV College of Engineering, Tirupati
http://warse.org/IJETER/static/pdf/Issue/NCTET2015sp62.pdf

2) Structure of AOCs:

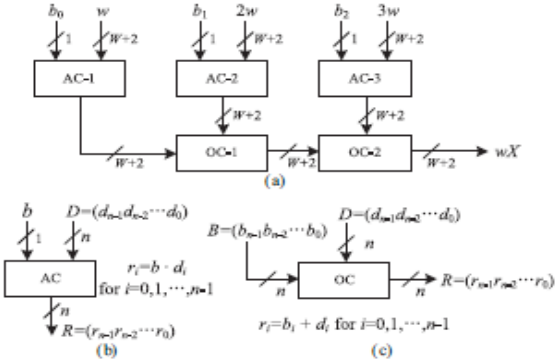


Fig. 5. Structure and function of AND/OR cell. Binary operators · and + in (b) and (c) are implemented using AND and OR gates, respectively.

The structure and function of an AOC are depicted in Fig. 5. Each AOC consists of three AND cells and two OR cells. The structure and function of AND cells and OR cells are depicted by Fig. 5(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. 5(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*, 2*w*, and 3*w* corresponding to the decimal values 1, 2, and 3 of the 2-b input *(u*1*u*0*)*, respectively. The decoder along with the AOC performs a multiplication of input operand *w* with a 2-b digit *(u*1*u*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 log2 *N* stages of adders to add *N* partial product, and the shift–add tree requires log2 *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. 6.

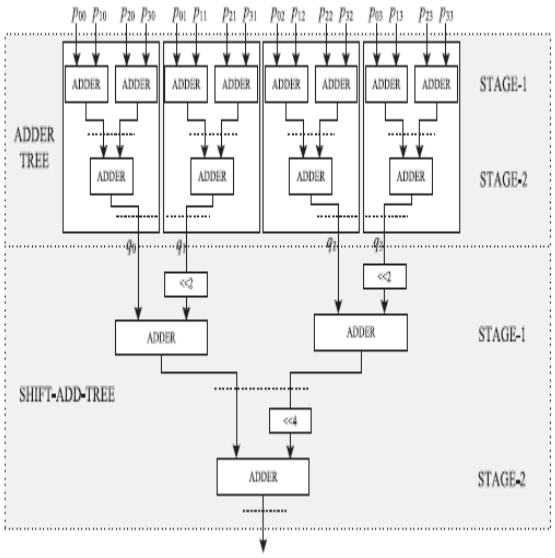


Fig. 6. Adder-structure of the filtering unit for N = 4 and L = 8.

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*

ISSN 2347 - 3983

International Journal of Emerging Trends in Engineering Research (IJETER), Vol. 3 No.6, Pages : 328 - 335 (2015)
Special Issue of NCTET 2K15 - Held on June 13, 2015 in SV College of Engineering, Tirupati
http://warse.org/IJETER/static/pdf/Issue/NCTET2015sp62.pdf

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.

TABLE I

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

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

B. Pipelined Structure of the Weight-Update Block

The proposed structure for the weight-update block is shown in Fig. 7. It performs N multiply-accumulate operations of the form $(\mu \times e) \times xi + wi$ to update N filter weights.

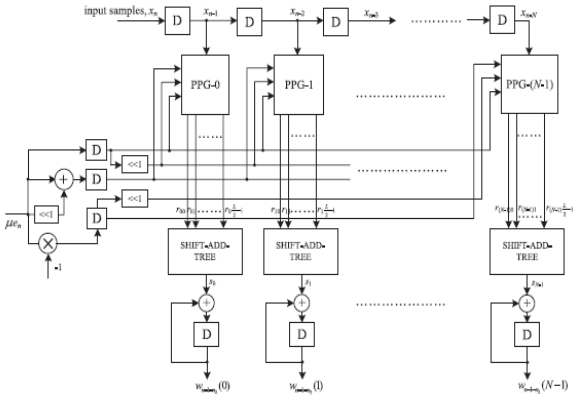


Fig. 7. Proposed structure of the weight-update block.

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 sub expression $3\mu \times e$ is shared within the multiplier. Since the scaled error $(\mu \times e)$ is multiplied with the entire N delayed input values in the weight-update block, this sub expression 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. 3, 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. 7, the weight-update block generates $\mathbf{w}_{n-1-n2}$, and the weights are delayed by n_2+1 cycle. However, it should be noted that the delay by 1 cycle is due to the

ISSN 2347 - 3983

International Journal of Emerging Trends in Engineering Research (IJETER), Vol. 3 No.6, Pages : 328 - 335 (2015)
Special Issue of NCTET 2K15 - Held on June 13, 2015 in SV College of Engineering, Tirupati
http://warse.org/IJETER/static/pdf/Issue/NCTET2015sp62.pdf

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. 3, and the other latch is before PPG in Fig. 7. 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 hardware complexity without noticeable degradation of steady state MSE.

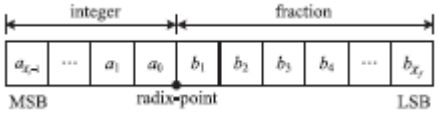


Fig. 8. Fixed-point representation of a binary number (Xi: integer word length; X f : fractional word-length).

TABLE II
FIXED-POINT REPRESENTATION OF THE SIGNALS OF THE PROPOSED

Signal Name	Fixed-Point Representation
x	*(L, Li)*
W	*(W,Wi)*
p	*(W + 2, Wi + 2)*
q	*(W + 2 + log2 N, Wi + 2 + log2 N)*
y, d, e	*(W,Wi + Li + log2 N)*
μe	*(W,Wi)*
r	*(W + 2, Wi + 2)*
s	*(W,Wi)*

x, **w**, **p**, **q**, *y*, *d*, and *e* can be found in the error-computation block of Fig. 3. *μe*, **r**, and **s** are defined in the weight-update block in Fig. 7. It is to be noted that all the subscripts and time indices of signals are omitted for simplicity of notation. For fixed-point implementation, the choice of word lengths and radix points for input samples, weights, and internal signals need to be decided. Fig. 8 shows the fixed-point representation of a binary number. Let *(X, Xi)* be a fixed-point representation of a binary number where *X* is the word length and *Xi* is the integer length. The word length and location of radix point of *xn* and *wn* 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, Li)* and *(W,Wi)*, respectively, as the representations of input signals and filter weights, all other signals in Figs. 3 and 7 can be decided as shown in Table II.

The signal *pi j* , which is the output of PPG block (shown in Fig. 3), 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. 6 is one bit more than the size of input signals, so that the fixed-point representation of the output of the adder tree with log2 *N* stages becomes *(W* + log2 *N* + 2,*Wi* + log2 *N* + 2*)*. Accordingly, the output of the shift–add tree would be of the form *(W+L*+log2 *N,Wi+Li*+ log2 *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* + log2 *N)* bits, which results in the fixed-point representation *(W,Wi* + *Li* +log2 *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.

V. RECURSIVE LEAST SQUARES

The Recursive least squares (RLS) is an adaptive filter which recursively finds the coefficients that minimize a weighted linear least squares cost function relating to the input signals. This is in contrast to other algorithms such as the

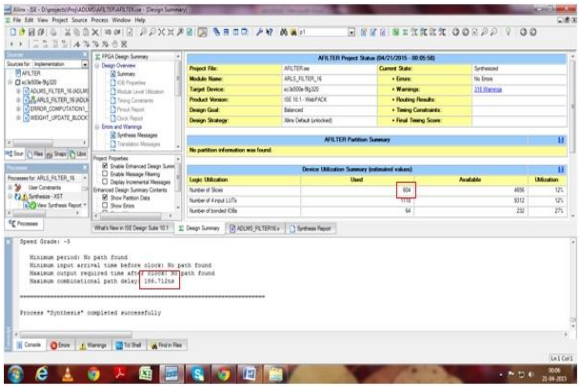
ISSN 2347 - 3983

International Journal of Emerging Trends in Engineering Research (IJETER), Vol. 3 No.6, Pages : 328 - 335 (2015)
Special Issue of NCTET 2K15 - Held on June 13, 2015 in SV College of Engineering, Tirupati
http://warse.org/IJETER/static/pdf/Issue/NCTET2015sp62.pdf

least mean squares (LMS) that aim to reduce the mean square error. In the derivation of the RLS, the input signals are considered deterministic, while for the LMS and similar algorithm they are considered stochastic. Compared to most of its competitors, the RLS exhibits extremely fast convergence. However, this benefit comes at the cost of high computational complexity.Instead of LMS if we use RLMS in the same optimized architecture of proposed adaptive filter which leads to betterment in area, power and delay.

VI. EXPERIMENTAL RESULTS ANALYSIS

Below figure shows area slices and delay of delayed least mean square filter



Below figure shows area slices and delay of delayed Recursive least mean square filter

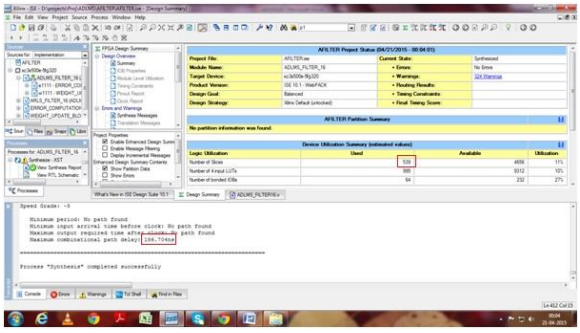


Table III Experiment Results

Adaptive filters	Area	Delay	power
DLMS	604 slices	186.712ns	0.034W
DRLMS	539 slices	186.704ns	0.034W

VII. CONCLUSION

We proposed an area–delay-power economical low adaptation delay design for fixed-point implementation of LMS adaptive filter. We have a tendency to used a unique PPG for economical implementation of general multiplications and inner-product computation by common sub expression sharing. Besides, we've proposed AN economical addition theme for inner-product computation to cut back the variation delay considerably so as to realize quicker convergence performance and to cut back the important path to support high input-sampling rates. Other than this, we have a tendency to propose a method for optimized balanced pipelining across the long blocks of the structure to cut back the variation delay and power consumption, as well. The proposed structure concerned considerably less adaptation delay and provided vital saving of ADP and EDP compared to the prevailing structures. We have a tendency to propose a fixed-point implementation of the proposed design, and derived the expression for steady-state error. We have a tendency to 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 better ADP and EDP over the proposed structure before pruning, without a noticeable degradation of steady-state error performance.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude and thanks to **Smt. N.Suguna, M.Tech** Assistant Professor in Electronics and Communication Engineering department**,** Sri Venkateswara College Of Engineering, Tirupati, for her constant help, valuable guidance and useful suggestions, which helped me in the Completion of the work.

ISSN 2347 - 3983

International Journal of Emerging Trends in Engineering Research (IJETER), Vol. 3 No.6, Pages : 328 - 335 (2015)
Special Issue of NCTET 2K15 - Held on June 13, 2015 in SV College of Engineering, Tirupati
http://warse.org/IJETER/static/pdf/Issue/NCTET2015sp62.pdf

REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.