INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 5, Issue 1 SEP 2015

DESIGN OF POWER EFFICIENT RECONFIGURABLE FIR ADAPTIVE FILTER USING DLMS ALGORITHM

[1]R.JHANSI, PG SCHOLAR IN VLSI, [2]P.VEER NATH ASSOC. PROF, [3]R.DURGA GOPAL ASSOC. PROF,
[1, 2,3] Joginpally B.R Engineering College, Hyderabad, Telangana.

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

1.INTRODUCTION

Filters of some sort are essential to the operation of most electronic circuits. It is therefore in the interest of anyone involved in electronic circuit design to have the ability to develop filter circuits capable of meeting a given set of specifications. In circuit theory, a filter is an electrical network that alters the amplitude and/or phase characteristics of a signal with respect to frequency. Ideally, a filter will not add new frequencies to the input signal, nor will it change the component frequencies of that signal, but it will change the relative amplitudes of the various frequency components and/or their phase relationships. Filters are often used in electronic systems to emphasize signals in certain frequency ranges and reject signals in other frequency ranges. Such a filter has a gain which is dependent on signal frequency.

Filters used for direct filtering can be either Fixed or Adaptive.

1. Fixed filters - The design of fixed filters requires a priori knowledge of both the signal and the noise, i.e. if we know the signal and noise beforehand, we can design a filter that passes frequencies contained in the signal and rejects the frequency band occupied by the noise.

2. Adaptive filters - Adaptive filters, on the other hand, have the ability to adjust their impulse response to filter out the correlated signal in the input. They require little or no a priori knowledge of the signal and noise characteristics. (they require a signal (desired response) that is correlated in some sense to the signal to be estimated).

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. 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, 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, Visvanathanet al. have proposed a modified systolic architecture to reduce the adaptation delay. A transpose-form LMS adaptive filter is suggested in, 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 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.

2. ADAPTATION ALGORITHM

The basic configuration of an adaptive filter, operating in the discrete time domain n, is illustrated in Figure 1. In such a scheme, the input signal is denoted by x(n), the reference signal d(n) represents the desired output signal (that usually includes some noise component), y(n) is the output of the adaptive filter, and the error signal is defined as e(n) =d(n) − y(n). The error signal is used by the adaptation algorithm to update the adaptive filter coefficient vector w(n) according to some performance criterion. In general, the whole adaptation process aims at minimizing some metric of the error signal, forcing the adaptive filter output signal to approximate the reference signal in a statistical sense. There are several adaptation algorithms with different performance criterion. Due to its low complexity and proven robustness, Least Mean Square (LMS) algorithm is used here.

LMS algorithm is a noisy approximation of steepest descent algorithm. It is a gradient type algorithm that updates the coefficient vector by taking a step in the direction of the negative gradient of the objective function.

$$w(n+1) = w(k) - \frac{\mu}{2} \frac{\delta Jw}{\delta w(n)}$$

LMS Algorithm:

For each n

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \cdot e_n \cdot \mathbf{x}_n \quad \text{……..(1)}$$

Where,

$$e_n = d_n - y_n \quad y_n = \mathbf{w}_n^T \cdot \mathbf{x}_n \quad \text{…….(2)}$$

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

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

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 *e(n)* 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

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

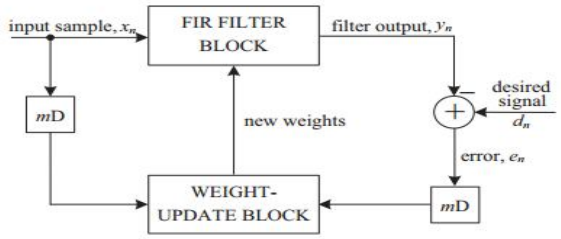

Fig. 1. Structure of the conventional delayed LMS adaptive filter

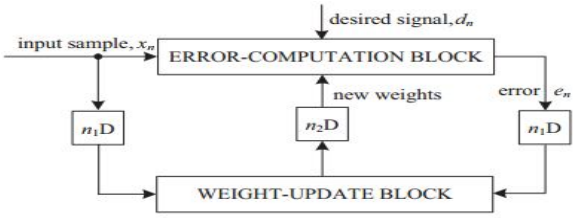

Fig. 2. Structure of the modified delayed LMS adaptive filter

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. 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*1cycles, the error computed by the structure at the *n*th cycle is *en−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

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \cdot e_{n-n_1} \cdot \mathbf{x}_{n-n_1} \quad \text{…(4)}$$

Where,

$$e_{n-n_1} = d_{n-n_1} - y_{n-n_1} \quad \text{…….(5)}$$

$$y_n = \mathbf{w}_{n-n_2}^T \cdot \mathbf{x}_n \quad \text{……(6)}$$

3. PROPOSED ARCHITECTURE

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 in the error-computation block, followed by the weight-update block

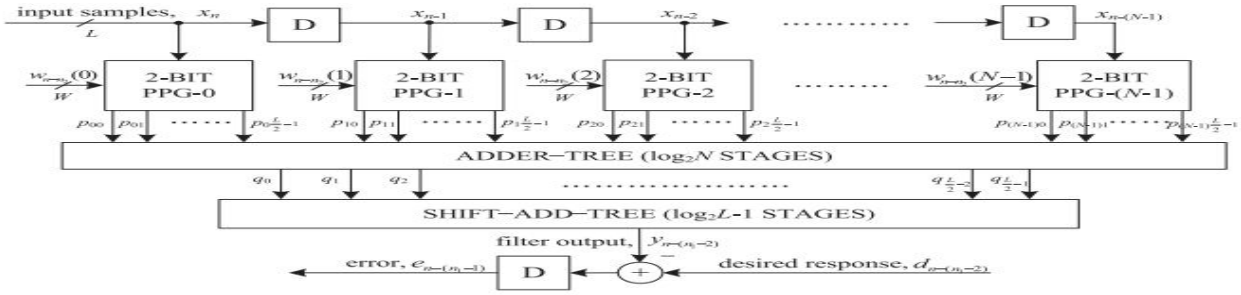

Fig. 4. Proposed structure of the error-computation block.

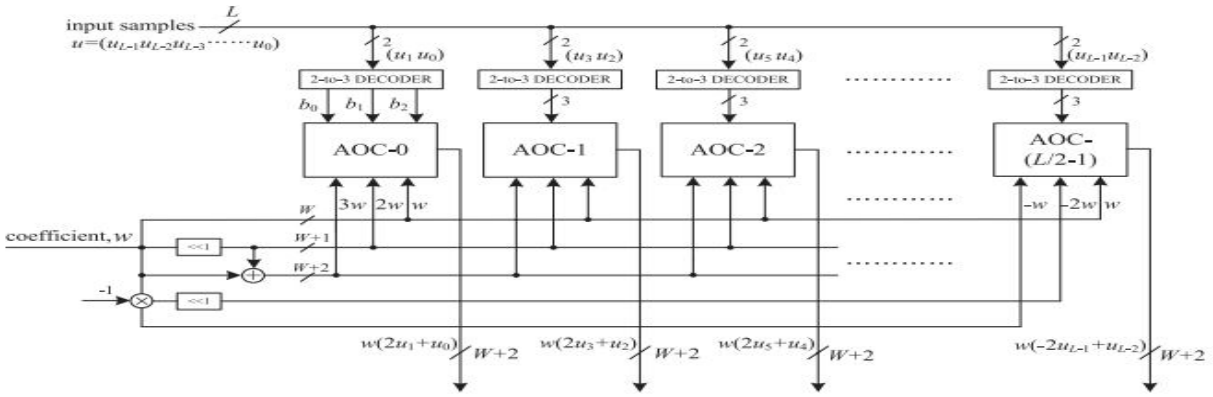

Fig. 5. Proposed structure of PPG. AOC stands for AND/OR cell.

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 sub block 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*1 *u0)* as input and produces three outputs $b0 = u0 \cdot \bar{u}1$, $b1 = \bar{u}0 \cdot u1$,and $b2 = u0 \cdot u1$, such that *b*0 = 1 for *(u*1 *u0)* = 1, *b*1 = 1 for *(u*1*u0)* = 2, and *b*2 = 1 for *(u*1 *u0)* = 3. The decoder output *b*0, *b*1 and *b*2 along with *w*, 2*w*, and 3*w* are fed to an AOC, where *w*, 2*w*, and 3*w* 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*, −2*w*, 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 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.

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.

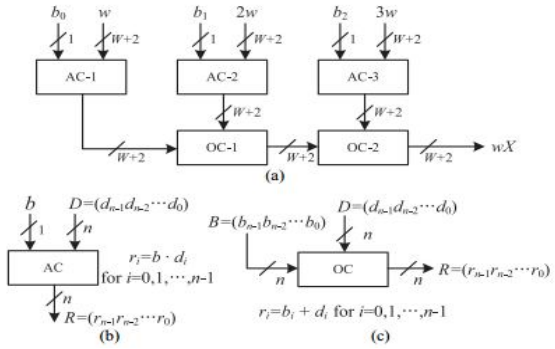

Fig. 6. Structure and function of AND/OR cell. Binary operators · and + in (b) and (c) are implemented using AND and OR gates, respectively

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 $(\mu \times e) \times xi + wi$ 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*

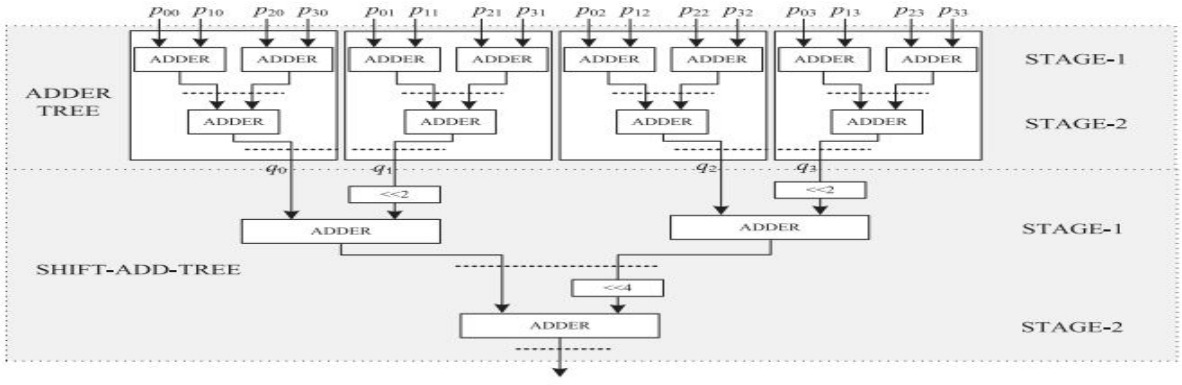

Fig. 7. Adder-structure of the filtering unit for $N = 4$ and $L = 8$.

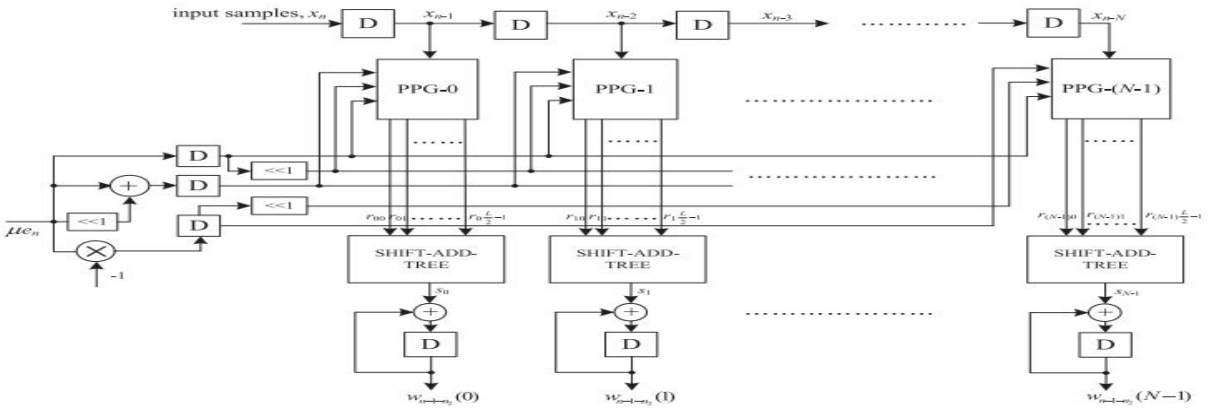

Fig. 8. Proposed structure of the weight-update block.

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 *xi*, where the sub expression $3\mu \times e$ 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 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 $n1$ and $n2$. The error-computation block generates the delayed error by $n1 - 1$ 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 $n1$. In Fig. 8, the weight-update block generates $\mathbf{w}n-1-n2$, and the weights are delayed by $n2 + 1$ 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.,$n1$. Therefore, the delay generated in the weight-update block becomes $n2$. If the locations of pipeline latches are decided as in Table I, $n1$ 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, $n2$ is set to 1 from a latch in the shift-add tree in the weight-update block.

D. Fixed-Point Implementation

A bit level pruning of the adder tree is also proposed to reduce the hardware complexity without noticeable degradation of steady state MSE.

4. SIMULATION RESULTS

Area power delay adaptive filter with low adaptation delay is Verilog coded and simulated on Xilinx to check the desired functionality. The filter specifications are 8 bit data samples, 8 bit filter coefficients. For comparison we have verilog coded the conventional filter structures. Fig. 9 shows the Xilinx snapshots of conventional adaptive filter and fig. 10 shows proposed system. The filter structured in Verilog is synthesized on Xilinx ISE.

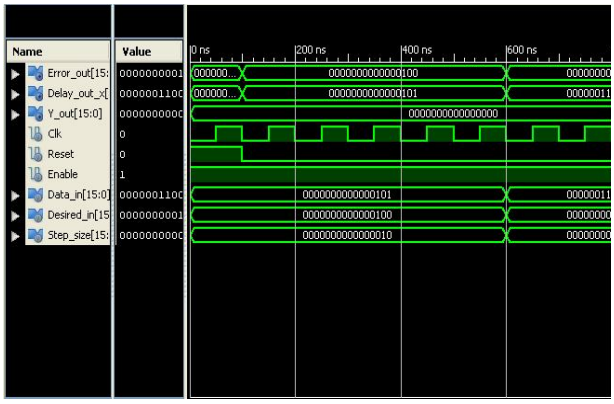

Fig 9: Simulation result of conventional adaptive filter

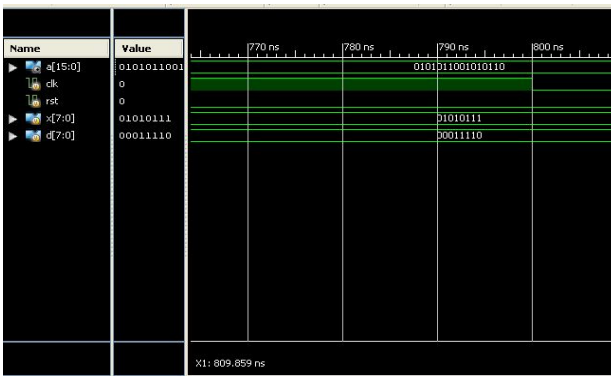

Fig 10: Simulation result of proposed structure

CONCLUSIONS

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 sub expression 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. The delay for conventional system is 19.732 ns and proposed system is 6.473ns.

REFERENCES

[1] Benard Widrow ,S.D. Stearns, ―Adaptive Signal Processing‖ ,2nd Edition ,ISBN 978 -81-317-0532-2 ,2009.

[2] Li Tan , Jean Jiang, ―Digital Signal Processing Fundamentals and Application , 2nd Edition ,ISBN 978-0-12-415893-1,2013.

[3] Antoniou ,A.," Digital Filter",3rd Edition, Tata Mc. Graw Publications, 2001

[4] Parhi K K., "A Systematic Approach For Design Of Digit-Serial Signal Processing Architectures",Circuits and Systems,1991.

[5] Saeid Mehrkanoon, Mahmoud Moghavvemi , “Real time ocular and facial muscle artifacts removal from EEG Signals using LMS Adaptive Algorithm”, International Conference on Intelligence and Advanced System,2007.IEEE

[6] NJ Bershad, JCM Bermudez, “An Affine Combination of Two LMS Adaptive Filter Transient Mean-Squre Analysis”,Signal Processing, IEEE Transactions, May 2008.

[7] K. R. Borisagar, G. R. kulkarni “Simulation and Comparative Analysis of LMS and RLS

Algorithms Using Real Time Speech Input Signal ",GJRE, 2010.

[8] 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.

[9] 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.

[10] 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.

[11] 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.