IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.
PP 01-11
www.iosrjournals.org

Implementation of an improved LMS and RLS adaptive filter with low adaptation delay

Ameena Parveen . M [1] , Joshua Arul Kumar . R [2]
[1] *Pg Scholar , Dept. of ECE , M.A.M College of Engineering , Trichy.*
[2] *Associate Professor , Dept. of ECE , M.A.M College of Engineering , Trichy.*

Abstract: *The filtering process is widely used in several digital signal processing and digital image processing applications. The filter process is to remove the noise in original signal or image. THE LEAST MEAN SQUARE (LMS) adaptive filter is the most popular and most widely used adaptive filter because of its simplicity and also for 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. From the critical path evaluation, it is shown that no pipelining is required for implementing a direct form LMS adaptive filter for most practical cases, and can be realized with a very small adaptation delay in cases where a very high sampling rate is required. Based on these findings, this paper proposes two structures of the LMS adaptive filter having no adaptation delays, and with only one adaptation delay. Moreover, the proposed adaptive filter design is extended by replacing LMS algorithm to RLS (Recursive least squares) algorithm which leads to better performance.*

Keywords : *Adaptive filters, least mean square algorithms, LMS adaptive filter, DLMS adaptive filter.*

I. INTRODUCTION

Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductors and communication technologies were being developed. Before the introduction of VLSI technology most ICs had a limited set of functions they could perform. An electronics circuits might consist of a central processing unit to Read only memory, Random access memory and other Glue logic.

VLSI lets IC makers add all of these into one chip. Structured VLSI design is a modular methodology originated by Carver Mead and Lynn Conway for saving microchip area by minimizing the interconnect fabrics area. This is obtained by repetitive arrangement of rectangular macro blocks which can be interconnected using wiring by abutment. An example is partitioning the layout of an adder into a row of equal bit slices cells. In complex designs this structuring may be achieved by hierarchical nesting.

Structured VLSI design had been popular in the early 1980s, but lost its popularity later because of the advent of placement and routing tools wasting a lot of area by routing, which is tolerated because of the progress of Moore's Law. When introducing the hardware description language KARL in the mid' 1970s, Reiner Hartenstein coined the term "structured VLSI design" (originally as "structured LSI design"), echoing Edsger Dijkstra's structured programming approach by procedure nesting to avoid chaotic spaghetti-structured programs. It has been seen that only 40% power is reduced in case of the adaptive filter when compared to decimation filter.

We have so far concentrated mainly on algorithmic approach of power reduction. Now we proceed to integrate both circuit level and algorithmic level power reduction technique to apply to our problem of hearing aid design. In this chapter we focus on FIR filter which is the primary component of hearing aid design. Here in this design FDF (Folded Direct Form) of FIR filter structure is adopted. We have additionally focused on the clocking strategy to reduce glitches that facilitates power reduction; a novel circuit for latch is also proposed. This clocking strategy in conjunction with the latch is adapted for the FIR filter structure that is used for hearing aid.

II. LEAST MEAN SQUARE ADAPTIVE FILTERS

Adaptive digital filters find wide application in several digital signal processing (DSP) areas, e.g., noise and echo cancelation, system identification, channel estimation, channel equalization, etc. The Least Mean Square 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 [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 ∼ *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*.

III. REVIEW OF DELAYED LMS ALGORITHM

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

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en \cdot \mathbf{x}n \tag{1a}$$

where

$$en = dn - yn\ yn = \mathbf{w}Tn \cdot \mathbf{x}n \tag{1b}$$

where

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

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

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

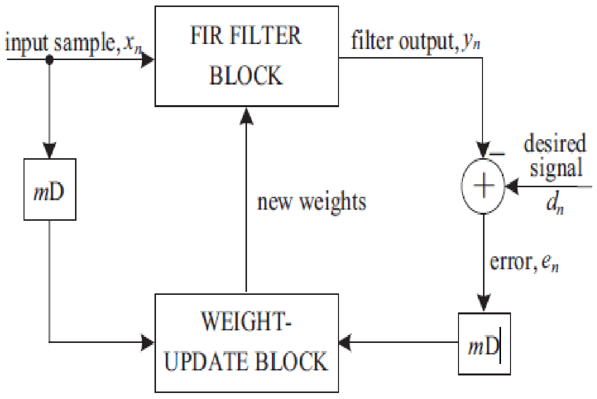


Fig. 1. Structure of the conventional delayed LMS adaptive filter.

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

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en-m \cdot \mathbf{x}n-m. \tag{2a}$$

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.

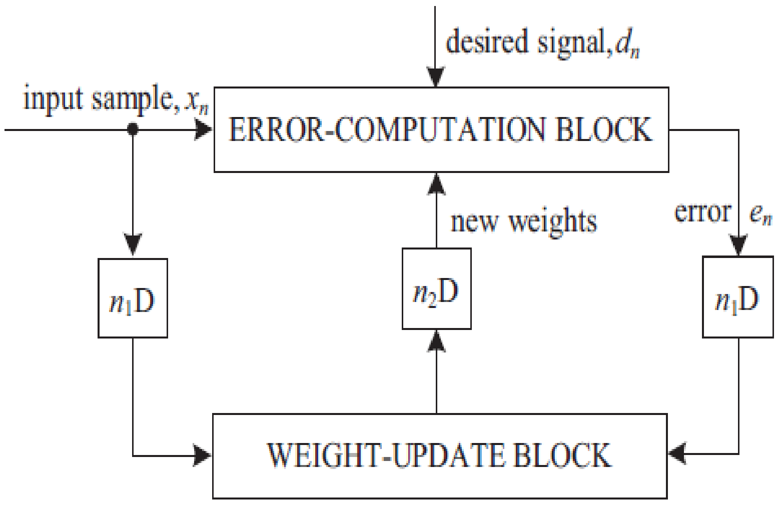


Fig. 2. Structure of the modified delayed LMS adaptive filter.

Assuming that the latency of computation of error is $n1$ cycles, the error 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

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en-n1 \cdot \mathbf{x}n-n1 \tag{3a}$$

where

$$en-n1 = dn-n1 - yn-n1 \tag{3b}$$

and

$$yn = \mathbf{w}T\, n-n2 \cdot \mathbf{x}n. \tag{3c}$$

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 feedforward 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 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 sub block is described in detail.

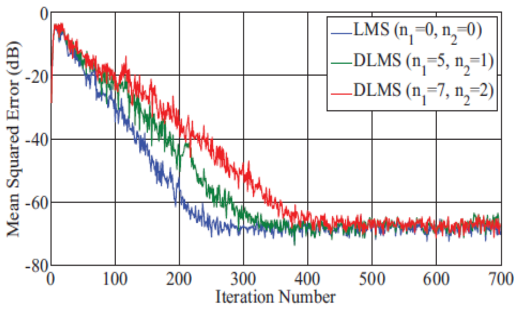


Fig. 3. Convergence performance of system identification with LMS and modified DLMS adaptive filters.

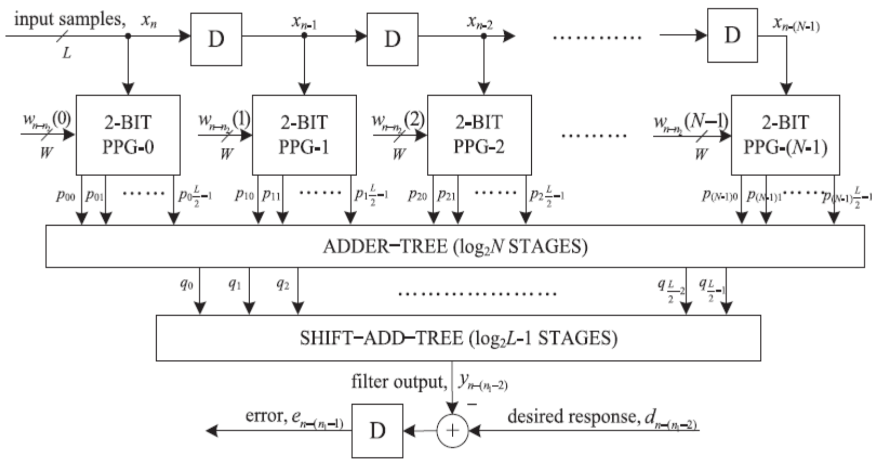


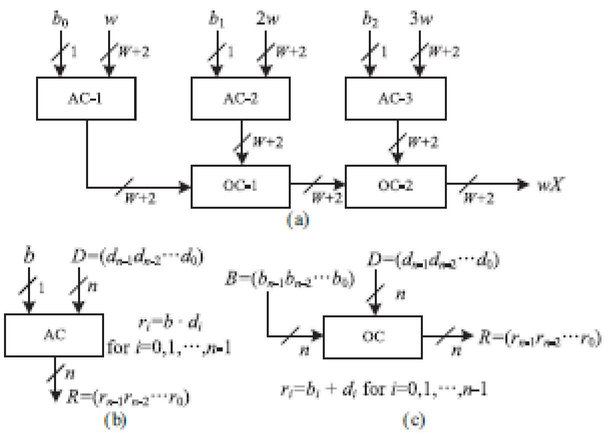
Fig. 4. Proposed structure of the error-computation block.

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 $(u1u0)$ 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.

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. 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, $2w$, and $3w$ corresponding to the decimal values 1, 2, and 3 of the 2-b input $(u1u0)$, respectively. The decoder along with the AOC performs a multiplication of input operand w with a 2-b digit $(u1u0)$, 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.



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

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

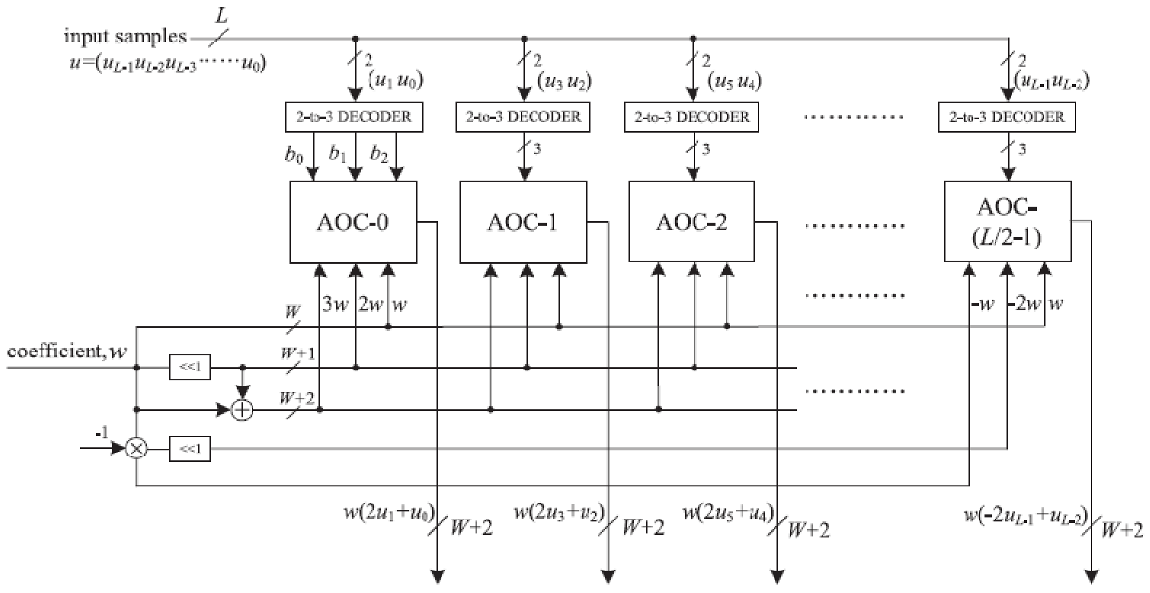


Fig. 6. Proposed structure of PPG. AOC stands for AND/OR cell

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. 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 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.

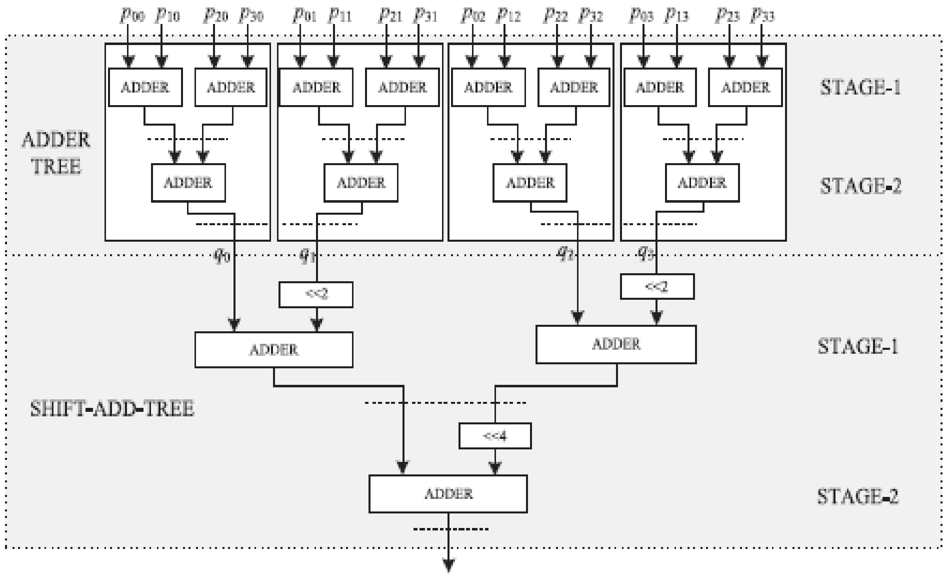

p_{00} p_{10} p_{20} p_{30} p_{01} p_{11} p_{21} p_{31} p_{02} p_{12} p_{22} p_{32} p_{03} p_{13} p_{23} p_{33}
ADDER
ADDER TREE
STAGE-1
STAGE-2
q_0 q_1 q_2 q_3
<<2
<<4
SHIFT-ADD-TREE
STAGE-1
STAGE-2

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

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 *(μ × e) × 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 error value *μ* × *e* with *L/*2, the number of 2-b digits of the input word *xi* , where the sub expression 3*μ*×*e* is shared within the multiplier. Since the scaled error (*μ*×*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. 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 **w***n*−1−*n*2, 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 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.

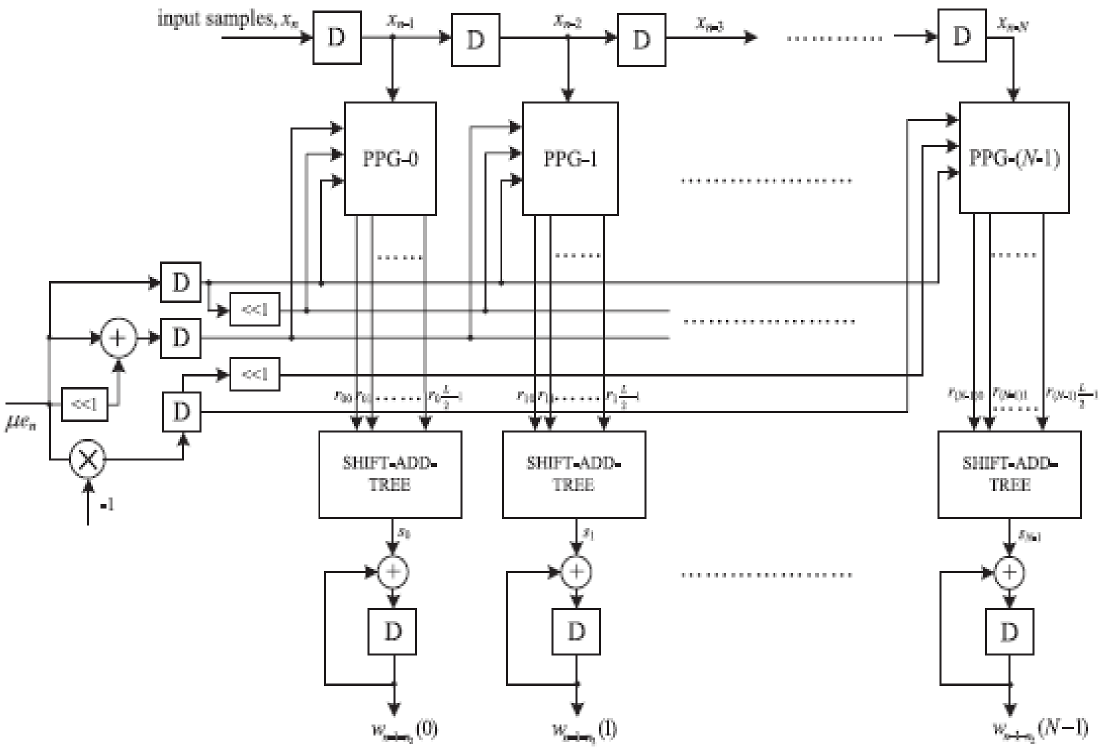


Fig. 8. Proposed structure of the weight-update block.

IV. PROPOSED STRUCTURE

In this section, we discuss area- and power-efficient approaches for the implementation of direct-form LMS adaptive filters with zero and one adaptation delays.

A. Zero Adaptation Delay

As shown in Fig. 2, there are two main computing blocks in the direct-form LMS adaptive filter, namely, (i) the error computation block (shown in Fig. 4) and (ii) the weight-update block (shown in Fig.5). It can be observed in Figs. 4 and 5 that most of the area-intensive components are common in the error-computation and weight-update blocks: the multipliers, weight registers, and tapped-delay line.
The adder tree and subtractor in Fig. 4 and the adders for weight updating in Fig. 5, which constitute only a small part of the circuit, are different in these two computing blocks. For the zero-adaptation delay implementation, the computation of both these blocks is required to be performed in the same cycle. Moreover, since the structure is of the non-pipelined type, weight updating and error computation cannot occur concurrently. Therefore, the multiplications of both these phases could be multiplexed by the same set of multipliers, while the same registers could be used for both these phases if error computation is performed in the first half cycle, while weight update is performed in the second-half cycle. The proposed time-multiplexed zero-adaptation-delay structure for a direct-form N-tap LMS adaptive filter is shown in Fig. 9, which consists of N multipliers. The input samples are fed to the multipliers from a common tapped delay line. The N weight values (stored in N registers) and the estimated error value (after right-shifting by a fixed Number of locations to realize multiplication by the step size μ) are fed to the multipliers as the other input through a 2:1 multiplexer. Apart from this, the proposed structure requires N adders for modification of N weights, and an adder tree to add the output of N multipliers for computation of the filter output. Also, it requires a subtractor to compute the error value and N 2:1 de-multiplexors to move the product values either towards the adder tree or weight-update circuit. All the multiplexors and de-multiplexors are controlled by a clock signal.

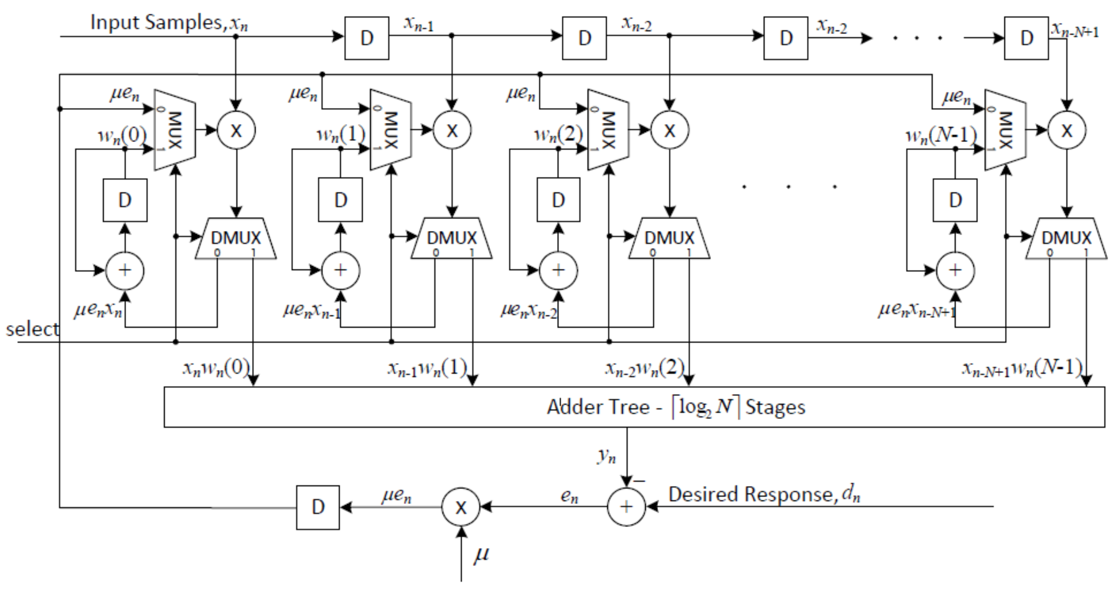


Fig. 9. Proposed structure for zero-adaptation-delay time-multiplexed direct-form LMS adaptive filter.

The registers in the delay line are clocked at the rising edge of the clock pulse and remain unchanged for a complete clock period since the structure is required to take one new sample in every clock cycle. During the first half of each clock period, the weight values stored in different registers are fed to the multiplier through the multiplexors to compute the filter output. The product words are then fed to the adder tree though the de-multiplexors. The filter output is computed by the adder tree and the error value is computed by a subtractor. Then the computed error value is right-shifted to obtain µen and is broadcasted to all N multipliers in the weight-update circuits. Note that the LMS adaptive filter requires at least one delay at a suitable location to break the recursive loop. A delay could be inserted either after the adder tree, after the en computation, or after the µen computation. If the delay is placed just after the adder tree, then the critical path shifts to the weight-updating circuit and gets increased by TADD. Therefore, we should place the delay after computation of en or µen, but preferably after µen computation to reduce the register width. The first half-cycle of each clock period ends with the computation of µen, and during the second half cycle, the µen value is fed to the multipliers though the multiplexors to calculate µenxn and de-multiplexed out to be added to the stored weight values to produce the new weights according to (2a). The computation during the second half of a clock period is completed once a new set of weight values is computed. The updated weight values are used in the first half cycle of the next clock cycle for computation of the filter output and for subsequent error estimation. When the next cycle begins, the weight registers are also updated by the new weight values. Therefore, the weight registers are also clocked at the rising edge of each clock pulse. The time required for error computation is more than that of weight updating. The system clock period could be less if we just perform these operations one after the other in every cycle. This is possible since all the register contents also change once at the beginning of a clock cycle, but we cannot exactly determine when the error computation is over and when weight updating is completed. Therefore, we need to perform the error computation during the first half-cycle and the weight updating during the second half-cycle.

Accordingly, the clock period of the proposed structure is twice the critical-path delay for the error-computation block CERROR,

CERROR = 2[TMULT + (dlog2 Ne + 1)_ + 2TMUX] (4a)

Where TMUX is the time required for multiplexing and demultiplexing.

B. One Adaptation Delay

The proposed structure for a one-adaptation-delay LMS adaptive filter consists of one error-computation unit as shown in Fig. 4 and one weight-update unit as shown in Fig. 5. A pipeline latch is introduced after computation of µen. The multiplication with µ requires only a hardwired shift, since µ is assumed to be a power of 2 fraction. So there is no register overhead in pipelining. Also, the registers in the tapped delay line and filter weights can be shared by the error computation unit and weight-updating unit. The critical path of this structure is the same as CERROR given by,

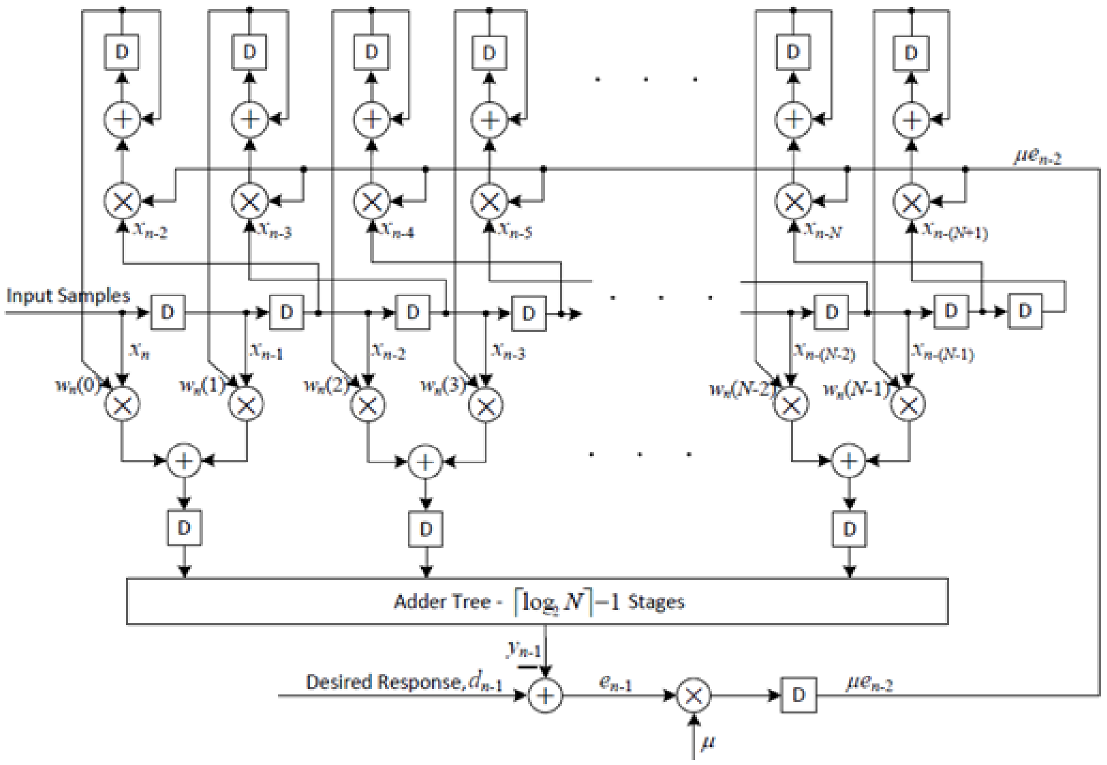


Fig. 10. Proposed structure for one-adaptation-delay direct-form LMS adaptive filter.

T = TMULT + (dlog2 Ne + 1)Δ (4b)

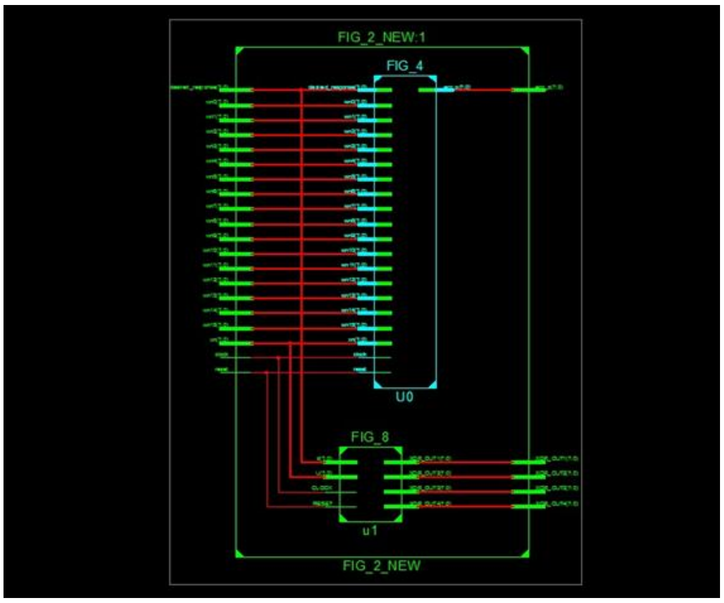
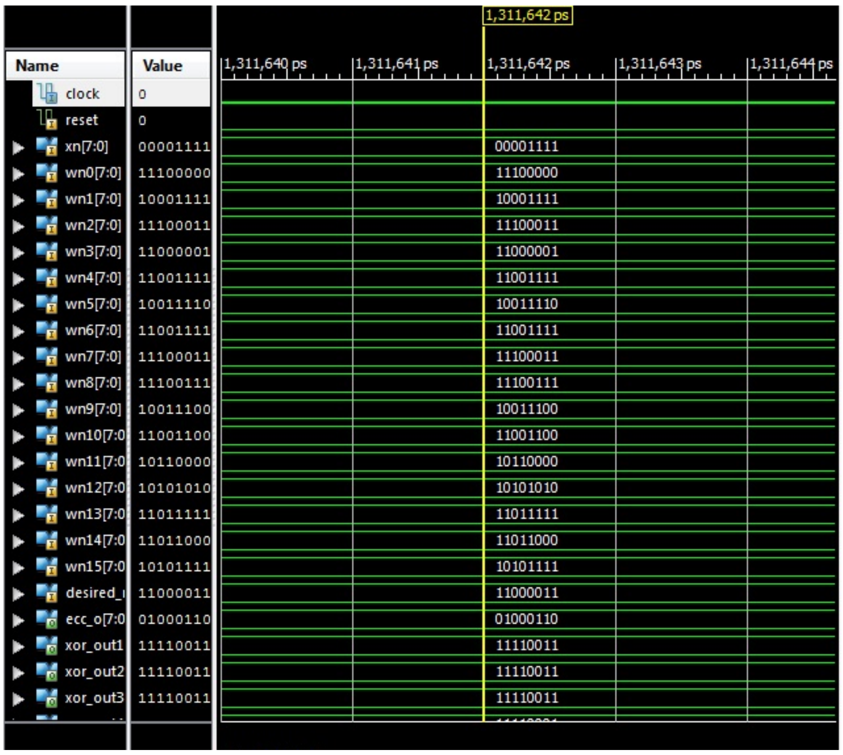


Fig 11: RTL schematic of basic block



Name	Value
clock	0
reset	0
xn[7:0]	00001111
wn0[7:0]	11100000
wn1[7:0]	10001111
wn2[7:0]	11100011
wn3[7:0]	11000001
wn4[7:0]	11001111
wn5[7:0]	10011110
wn6[7:0]	11001111
wn7[7:0]	11100011
wn8[7:0]	11100111
wn9[7:0]	10011100
wn10[7:0	11001100
wn11[7:0	10110000
wn12[7:0	10101010
wn13[7:0	11011111
wn14[7:0	11011000
wn15[7:0	10101111
desired_	11000011
ecc_o[7:0	01000110
xor_out1	11110011
xor_out2	11110011
xor_out3	11110011

Fig 12: Simulation result of basic block

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 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. 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 based on a precise critical-path analysis, we have derived low-complexity architectures for the LMS adaptive filter. The direct-from LMS adaptive filter, however, involves less register complexity and provides much faster convergence than its transpose-form counterpart since the latter inherently performs delayed weight adaptation. We have proposed two different structures of direct-form LMS adaptive filter with (i) zero adaptation delay, (ii) one adaptation delay, Proposed no adaptation delay does not involve any adaptation delay. It has the minimum of MUF among all the structures, but that is adequate to support the highest data rate in current communication systems. It involves the minimum area and the minimum EPS of all the designs. Proposed design with one adaptation delay involve nearly the same (slightly more) EPS than the proposed Design 1 but offer nearly twice or thrice the MUF at the cost of 55:0% and 60:6% more area. However, proposed Design 1 could be the preferred choice instead of proposed Design 2 in most communication applications, since it provides adequate speed performance, and involves significantly less area and EPS.

REFERENCES

[1] B. Widrow and S. D. Stearns, Adaptive Signal Processing. Englewood Cliffs, NJ: Prentice-Hall, 1985.
[2] S. Haykin and B. Widrow, Least-Mean-Square Adaptive Filters. Hoboken, NJ: Wiley-Interscience, 2003.
[3] 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.

[4] M. D. Meyer and D. P. Agrawal, "A modular pipelined implementation of a delayed LMS transversal adaptive filter," in Proc. IEEE International Symposium on Circuits and Systems, May 1990, pp. 1943–1946.

[5] 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 Digit. Signal Process., vol. 48, no. 4, pp. 359–366, Apr. 2001.

[6] 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.

[7] E. Mahfuz, C. Wang, and M. O. Ahmad, "A high-throughput DLMS adaptive algorithm," in Proc. IEEE International Symposium on Circuits and Systems, May 2005, pp. 3753–3756.

[8] P. K. Meher and S. Y. Park, "Low adaptation-delay LMS adaptive filter Part-II: An optimized architecture," in Proc. IEEE International Midwest Symposium on Circuits and Systems, Aug. 2011.

[9] ——, "Area-delay-power efficient fixed-point LMS adaptive filter with low adaptation-delay," J. Very Large Scale Integr. (VLSI) Signal Process., accepted for inclusion in a future issue. [Online]. Available: http://ieeexplore.ieee.org

[10] Y. Yi, R. Woods, L.-K. Ting, and C. F. N. Cowan, "High speed FPGAbased implementations of delayed-LMS filters," J. Very Large Scale Integr. (VLSI) Signal Process., vol. 39, no. 1-2, pp. 113–131, Jan. 2005.

[11] S. Y. Park and P. K. Meher, "Low-power, high-throughput, and low-area adaptive FIR filter based on distributed arithmetic," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 60, no. 6, pp. 346–350, Jun. 2013.

[12] TSMC 90nm General-Purpose CMOS Standard Cell Libraries - tcbn90ghp. [Online]. Available: www.tsmc.com/

[13] TSMC 0.13_m General-Purpose CMOS Standard Cell Libraries - tcb013ghp. [Online]. Available: www.tsmc.com/

[14] 3GPP TS 36.211, Physical Channels and Modulation, ver. 10.0.0 Release 10, Jan. 2011.

[15] P. K. Meher and M. Maheshwari, "A high-speed FIR adaptive filter architecture using a modified delayed LMS algorithm," in Proc. IEEE International Symposium on Circuits and Systems, May 2011, pp. 121–124.

[16] J. Vanus and V. Styskala, "Application of optimal settings of the LMS adaptive filter for speech signal processing," in Proc. IEEE International Multiconference on Computer Science and Information Technology, Oct. 2010, pp. 767–774.

[17] M. Z. U. Rahman, R. A. Shaik, and D. V. R. K. Reddy, "Noise cancellation in ECG signals using computationally simplified adaptive filtering techniques: Application to biotelemetry," Signal Processing: An International Journal (SPIJ), vol. 3, no. 5, pp. 1–12, Nov. 2009.

[18] ——, "Adaptive noise removal in the ECG using the block LMS algorithm," in Proc. IEEE International Conference on Adaptive Science & Technology, Jan. 2009, pp. 380–383.

[19] B. Widrow, J. R. Glover, Jr., J. M. McCool, J. Kaunitz, C. S. Williams, R. H. Hearn, J. R. Zeidler, E. Dong, Jr., and R. C. Goodlin, "Adaptive noise cancelling: Principles and applications," Proc. IEEE, vol. 63, no. 12, pp. 1692–1716, Dec. 1975.

[20] W. A. Harrison, J. S. Lim, and E. Singer, "A new application of adaptive noise cancellation," IEEE Trans. Acoust., Speech, Signal Process., vol. 34, no. 1, pp. 21–27, Feb. 1986.

[21] S. Coleri, M. Ergen, A. Puri, and A. Bahai, "A study of channel estimation in OFDM systems," in Proc. IEEE Vehicular Technology Conference, 2002, pp. 894–898.

[22] J. C. Patra, R. N. Pal, R. Baliarsingh, and G. Panda, "Nonlinear channel equalization for QAM signal constellation using artificial neural networks," IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 29, no. 2, pp. 262–271, Apr. 1999.

[23] D. Xu and J. Chiu, "Design of a high-order FIR digital filtering and variable gain ranging seismic data acquisition system," in Proc. IEEE Southeastcon, Apr. 1993.

[24] M. Mboup, M. Bonnet, and N. Bershad, "LMS coupled adaptive prediction and system identification: A statistical model and transient mean analysis," IEEE Trans. Signal Process., vol. 42, no. 10, pp. 2607–2615, Oct. 1994.

[25] C. Breining, P. Dreiseitel, E. Hansler, A. Mader, B. Nitsch, H. Puder, T. Schertler, G. Schmidt, and J. Tilp, "Acoustic echo control," IEEE Signal Processing Mag., vol. 16, no. 4, pp. 42–69, Jul. 1999.