A Ripple Carry Adder based Low Power Architecture of LMS Adaptive Filter

A.S. Sneka Priyaa
PG Scholar
Government College of Technology
Coimbatore

C. Santhi, Ph.D.
Associate Professor
Government College of Technology
Coimbatore

ABSTRACT

The Least Mean Square Adaptive Filter is frequently encountered in wide variety of applications like signal processing, measurement and analysis of continuously changing parameters and signal analysis. The direct form of LMS Adaptive Filter does not support pipelining due to its recursive behavior. Thus modified Delayed LMS Adaptive Filter is preferred in which delays are decomposed so that pipelining is applied to power consuming blocks. Literature survey on the architectures of LMS Adaptive filter reveals that earlier works focused on the implementation using systolic architectures that gave rise to large adaptation delay. This arise a need for designing the LMS Adaptive filter with low adaptation delay. In this project, power efficient hardware architecture of modified delayed LMS Adaptive filter has been designed and implemented. The design of modified Delayed LMS Adaptive Filter is done after implementing pipelined architecture of Error Computation Block and Weight Update Block. The two important blocks involve the use of partial product generator and adder-tree structure which together perform the high-complexity operation. Adder-tree structure uses Carry-look ahead adder whose internal generate and propagate signals contribute to high power consumption. Hence further modification is made in the adder-tree structure by utilizing ripple-carry adder instead of carry-look ahead adder. This modification results in approximately 24% reduction in power relative to existing modified DLMS. Further power optimization is done by replacing adders with 4:2 compressors. The area complexity is also reduced as the number of required 4:2 compressors are less when compared to the requirement of adders. This modification results in 39% reduction in power relative to existing modified DLMS without degradation of steady-state-error performance. This implementation of power-optimized modified DLMS is done using Xilinx ISE Design Suite 14.2.

Keywords

Delayed Least Mean Square (DLMS), Adaptation Delay, Power Optimization, Pipelining.

1. INTRODUCTION

Least mean squares (LMS) algorithms are a class of adaptive filter used to mimic a desired filter by finding the filter coefficients that relate to producing the least mean squares of the error signal. Its simplicity and satisfactory convergence performance made it popular and hence widely used. Various adaptive filtering techniques that are used in signal processing application deals with the theory of adaptation with stationary Signals in addition to the various adaptive algorithms and structures [1] .These adaptive filtering techniques are of high equipment complexity and cost. Thus the LMS algorithm which is a stochastic gradient algorithm iterates each tap weight of the transversal filter in the direction of the instantaneous gradient of the squared error signal with respect to the tap weight [2]. Although the LMS filter is very simple in computational terms, its mathematical analysis is complicated because of its stochastic and nonlinear nature.

The long critical path of the direct-form LMS which is mainly due to an inner-product computation must be reduced by pipelined implementation [3].The proposed digit-serial architecture is highly regular, modular, cell level systolic and above all it is bit-level pipelined .The digit size effect on the sampling rate and adaptation delay of the digit-serial adaptive system has been studied for various word length-size implementations. However the pipeline delays contribute to the high adaptation delay and it becomes more severe for large digit size implementations. This problem is eliminated by implementing DLMS algorithm in systolic architectures to increase the maximum usable frequency [4]-[6].They involve an adaptation delay of N cycles for filter length N, which is quite high for large order filters and the convergence performance degrades considerably for a large adaptation delay. Thus a modified systolic architecture to reduce the adaptation delay is proposed thereby improving the convergence behaviour to near that of the original LMS algorithm [7]. With the use of carry-save arithmetic, the systolic folded pipelined architecture can support very high sampling rates, limited only by the delay of a full adder [8]. But the use of proper and high accurate sequence of functions performing transformations such as associativity, retiming, slowdown, and folding is required.

A transpose-form LMS adaptive filter at any instant depends on the delayed versions of weights and the number of delays in weights varies from 1 to N [9],[10]. The processing speed is very low and can be improved using pipelined blocks which are overcome by 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 [11]. The delay least-mean-square (DLMS) adaptive finite impulse response (FIR) digital filter is based on a new tree-systolic processing element and an optimized tree-level rule. Applying our tree-systolic, a higher convergence rate than that of the conventional DLMS structures can be obtained without sacrificing the properties of the systolic-array architecture [12]. The efficient systolic adaptive FIR digital filter not only operates at the highest throughput in the word-level but also considers finite update of the feedback error. A fine-grained pipelined design to limit the critical path to the maximum of one addition time, supports high sampling frequency [13]. It involves a lot of area overhead for pipelining and higher power consumption than previous architectures, due to its large number of pipeline latches. This is resolved by presenting a modified delayed least mean square (DLMS) adaptive algorithm to achieve lower adaptation-delay [14]. Besides, proposal of an efficient pipelined architecture for the implementation of this adaptive filter is done. It is shown that the proposed DLMS adaptive filter can be implemented by a pipelined inner-product computation unit for calculation of feedback error, and a pipelined weight-update unit consisting of N parallel multiply accumulators to reduce the number of adaptation delays. A 2-bit multiplication cell 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 is presented [15]. Besides, the optimization of design is done 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 [16]. 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.

2. EXISTING MODIFIED DLMS

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

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

Where

$$e_n = d_n - y_n \text{ and } \quad y_n = w^T{}_n\, x_n \qquad (2)$$

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

$$x_n = [x_n, x_{n-1}, \ldots, x_{n-N+1}]^T \qquad (3)$$

$$w_n = [w_n(0), w_n(1), \ldots, w_n(N-1)]^T \qquad (4)$$

d_n is the desired response, y_n is the filter output, and e_n denotes the error computed during the nth iteration. μ is the step-size, and N is the number of weights used in the LMS adaptive filter. In the case of pipelined designs with m pipeline stages, the error e_n becomes available after m cycles, where m is called the “adaptation delay.” The DLMS algorithm therefore uses the delayed error e_{n-m}, i.e., the error corresponding to $(n - m)$th iteration for updating the current weight instead of the recent-most error. The weight-update equation of DLMS adaptive filter is given by

$$w_{n+1} = w_n + \mu \cdot e_{n-m} \cdot x_{n-m} \qquad (5)$$

The block diagram of the DLMS adaptive filter is shown in Figure 3.1, 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.1.Assuming that the latency of computation of error is n1 cycles, the error computed by the structure at the nth cycle is e_{n-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

$$w_{n+1} = w_n + \mu \cdot e_{n-n1} \cdot x_{n-n1} \qquad (6)$$

Where

$$e_{n-n1} = d_{n-n1} - y_{n-n1} \qquad (7)$$

$$y_n = w^T{}_{n-n2}\, x_n \qquad (8)$$

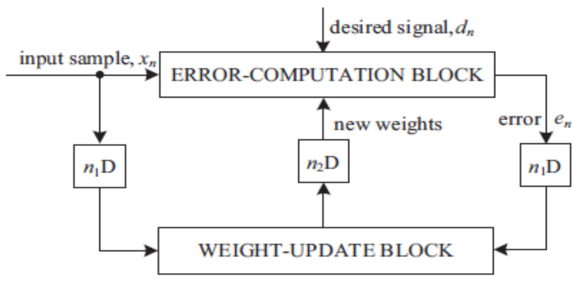


Fig. 1.Modified Delayed LMS Adaptive Filter

There are two main computing blocks in the adaptive filter architecture: 1) the error-computation block, and 2) weight-update block.

2.1 Error Computation Block

The structure for error-computation unit of an N-tap DLMS adaptive filter is shown in Fig. 2. 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 as follows. PPG consists of L/2 number of 2-to-3 decoders and the same number of AND/OR cells (AOC). Each of the 2-to-3 decoders takes a 2-b digit (u1u0) as input and produces three outputs b0 = u0 · u1, b1 = u0 · 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 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., $(u_{L-1}u_{L-2})$ of the input sample, the AOC (L/2 − 1) is fed with w, −2w, and −w as input since $(u_{L-1}u_{L-2})$ can have four possible values 0, 1, −2, and −1.

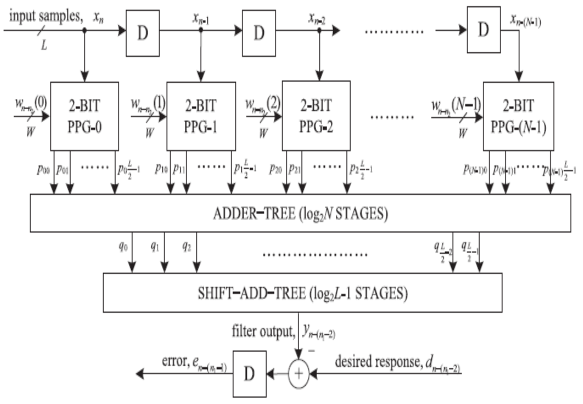


Fig. 2. Error Computation Block

Each AOC consists of three AND cells and two OR cells. 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. 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 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..

2.2 Adder-Tree Structure

The shift-add operation is performed 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, addition of all the N partial products of the same place value is performed from all the N PPGs by one adder tree. All the L/2 partial products generated by each of the N PPGs are thus added by (L/2) binary adder trees. The outputs of the L/2 adder trees are then added by a shift-add tree according to their place values. Each of the binary adder trees require $\log_2$ N stages of adders to add N partial product, and the shift–add tree requires $\log_2$ L − 1 stages of adders to add L/2 output of L/2 binary adder trees. The addition scheme for the error-computation block for a four-tap filter and input word size L = 8 is shown in Figure 3.6. 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. On introducing pipeline latches after every addition, it would require L(N − 1)/2 + L/2 − 1 latches in $\log_2$ N + $\log_2$ L − 1 stages, which would lead to a high adaptation delay and introduce a large overhead of area and power consumption for large values of N and L. The final adder in the shift–add tree contributes to the maximum delay to the critical path. Based on that observation, the pipeline latches that do not contribute significantly to the critical path are identified and could exclude those without any noticeable increase of the critical path.

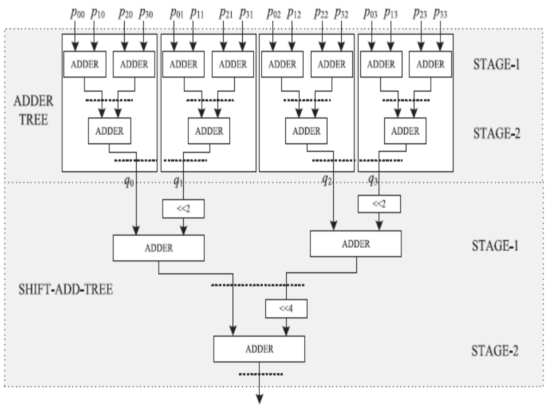


Fig. 3.Adder-Tree Structure

2.3 Weight Update Block

The proposed structure for the weight-update block is shown in Figure 3.7. It performs N multiply-accumulate operations of the form (μ × e) × x_i + w_i to update N filter weights. The step size μ is taken as a negative power of 2 to realize the multiplication with recently available error only by a shift operation. Each of the MAC units therefore performs the multiplication of the shifted value of error with the delayed input samples xi 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 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 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.

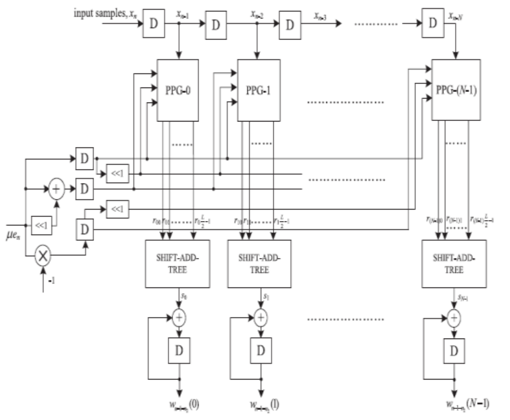


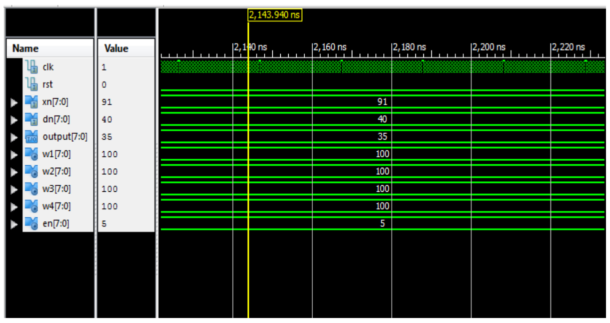
Fig. 4. Weight Update Block

The adaptation delay is decomposed into n1 and n2. The error-computation block generates the delayed error by n1 −1 cycles which is fed to the weight-update block 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.The weight-update block generates 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.

3. POWER OPTIMIZATION

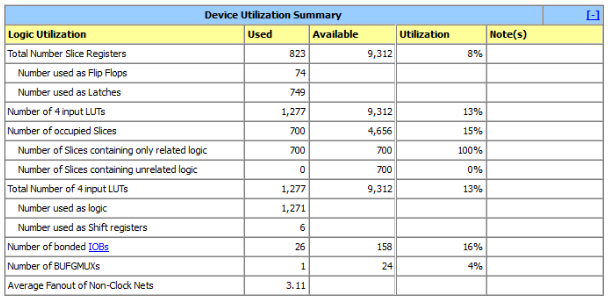
The adder tree and shift–add tree for the computation of y_n can be pruned for further optimization of power complexity. To illustrate the pruning optimization of adder tree and shift–add tree for the computation of filter output, we take a simple example of filter length N = 4, considering the word lengths L and W to be 8. The dot diagram of the adder tree is shown in Fig. 5. Each row of the dot diagram contains 10 dots, which represent the partial products generated by the PPG unit, for W = 8. The four sets of partial products are present 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. 5, 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. There might be one bit difference in the output of adder tree due to pruning. 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 guard bits is closer to or larger than the LSB weight of the output after final truncation, the pruning will affect the overall error more. Power Optimization is effectively done by using ripple carry adder. This minimizes the gate count and hence reduces the area complexity as well.

4. SIMULATION AND ANALYSIS



Name	Value
clk	1
rst	0
xn[7:0]	91
dn[7:0]	40
output[7:0]	35
w1[7:0]	100
w2[7:0]	100
w3[7:0]	100
w4[7:0]	100
en[7:0]	5

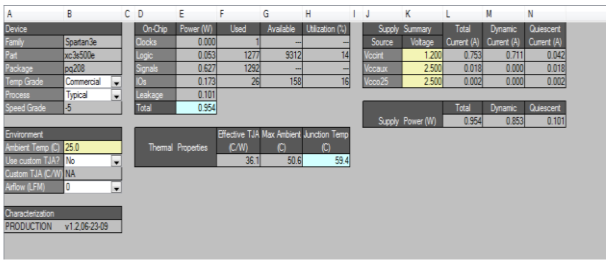
Fig. 5.Simulated Output



Device Utilization Summary [-]

Logic Utilization	Used	Available	Utilization	Note(s)
Total Number Slice Registers	823	9,312	8%	
Number used as Flip Flops	74			
Number used as Latches	749			
Number of 4 input LUTs	1,277	9,312	13%	
Number of occupied Slices	700	4,656	15%	
Number of Slices containing only related logic	700	700	100%	
Number of Slices containing unrelated logic	0	700	0%	
Total Number of 4 input LUTs	1,277	9,312	13%	
Number used as logic	1,271			
Number used as Shift registers	6			
Number of bonded IOBs	26	158	16%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.11			

Fig. 6.Proposed Area



Device	
Family	Spartan3e
Part	xc3s500e
Package	pq208
Temp Grade	Commercial
Process	Typical
Speed Grade	-5

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	1	---	---
Logic	0.053	1277	9312	14
Signals	0.627	1292	---	---
IOs	0.173	26	158	16
Leakage	0.101			
Total	0.954			

Supply Summary Source	Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.753	0.711	0.042
Vccaux	2.500	0.018	0.000	0.018
Vcco25	2.500	0.002	0.000	0.002

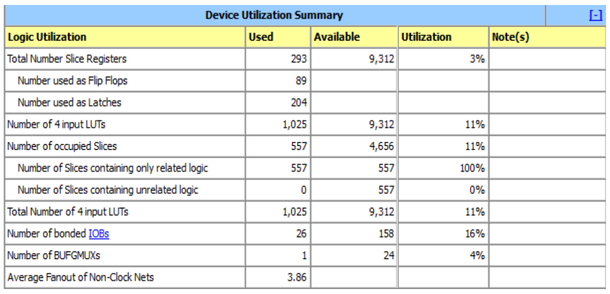
	Total	Dynamic	Quiescent
Supply Power (W)	0.954	0.853	0.101

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	36.1	50.6	59.4

Characterization	
PRODUCTION	v1.2,06-23-09

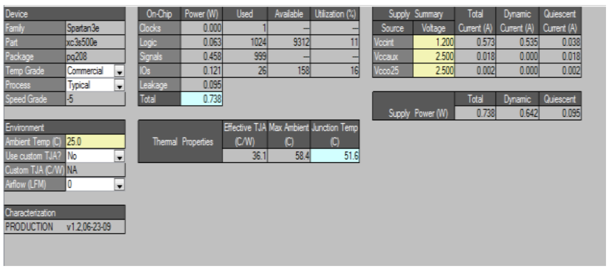
Fig. 7.Proposed Power



Device Utilization Summary [-]

Logic Utilization	Used	Available	Utilization	Note(s)
Total Number Slice Registers	293	9,312	3%	
Number used as Flip Flops	89			
Number used as Latches	204			
Number of 4 input LUTs	1,025	9,312	11%	
Number of occupied Slices	557	4,656	11%	
Number of Slices containing only related logic	557	557	100%	
Number of Slices containing unrelated logic	0	557	0%	
Total Number of 4 input LUTs	1,025	9,312	11%	
Number of bonded IOBs	26	158	16%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.86			

Fig. 8.Modified Area



Device	
Family	Spartan3e
Part	xc3s500e
Package	pq208
Temp Grade	Commercial
Process	Typical
Speed Grade	-5

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	1	---	---
Logic	0.063	1024	9312	11
Signals	0.458	999	---	---
IOs	0.121	26	158	16
Leakage	0.095			
Total	0.738			

Supply Summary Source	Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.573	0.535	0.038
Vccaux	2.500	0.018	0.000	0.018
Vcco25	2.500	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.738	0.642	0.095

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	36.1	58.4	51.6

Characterization	
PRODUCTION	v1.2,06-23-09

Fig. 9.Modified Power

5. INFERENCE

Power Optimization is effectively done by using ripple carry adder. This minimizes the gate count and hence reduces the power complexity by eliminating propagate and generate operations of carry-look ahead adder. Required number of internal signals is less when compared to the requirement in CLA. This in turn reduces the area complexity as well. Further reduction in power is obtained by replacing adders with 4:2 compressors. Area is also reduced as the number of required 4:2 compressors are less when compared to the requirement of full adders as far as implementation is concerned. The power and area analysis of various architectures of DLMS Adaptive Filter are tabulated as in table 1.

Table 1. Power and Area analysis of various architectures

Architecture	Power (W)	Area(slice registers)
CLA	0.954	823
Dot diagram	0.776	544
RCA	0.738	293

6. CONCLUSION

LMS Adaptive filter is preferred for satisfactory convergence performance. In this project, the power efficient hardware architecture of modified Delayed LMS Adaptive Filter has been implemented. The Partial product generator is used for efficient implementation of general multiplications and inner-product computation by common sub expression sharing. Besides, an efficient addition scheme for inner-product computation has been used 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. Also, a replacement strategy of ripple carry addition with 4:2 compressors is done for optimized balanced pipelining across the time-consuming blocks of the structure to reduce the power consumption drastically. The implemented modified DLMS structure involved significantly less power. The implementation of the power optimized modified DLMS architecture is done and the power, area and delay parameters for the same is measured and analyzed using Xilinx ISE Design Suite 14.2. Power is reduced drastically when the full adders are replaced by 4:2 compressors.

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

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

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

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

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

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