

Implementation of high-speed–low-power adaptive finite impulse response filter with novel architecture

Manish Jaiswal, Sandeep Sharma, Anuj Sharma

Department of ECE and AEI, Dehradun Institute of Technology, Dehradun 248001, Uttarakhand, India
E-mail: manishjeswal@hotmail.com

Published in *The Journal of Engineering*; Received on 4th August 2014; Accepted on 7th January 2015

Abstract: An energy efficient high-speed adaptive finite impulse response filter with novel architecture is developed. Synthesis results along with novel architecture on different complementary metal–oxide semiconductor (CMOS) families are presented. Analysis is performed using Artix-7, Spartan-6 and Virtex-4 for most popular adaptive least mean square filter for different orders such as $N=8, 16, 32$. The presented work is done using MATLAB (2013b) and Xilinx (14.2). From the synthesis results, it can be found that CMOS (28 nm) achieves the lowest power and critical path delay compared to others, and thus proves its efficiency in terms of energy. Different parameters are considered such as look up tables and input–output blocks, along with their optimised results.

1 Introduction

Convergence analysis of the adaptive least mean square (LMS) filter along with its simplicity make this filter the most popular for wide implementation in real-world applications. Echo cancellation, noise cancellation and channel estimation are the different streams of digital signal processing in which adaptive filtering is applicable [1, 2]. In the literature of LMS filter implementation and architectural analysis, it has been found that large critical path is the main reason for adaptation delay. Therefore, one way to reduce these critical paths is the process of pipelining. However, the recursive behaviour [2] of the LMS algorithm limits the pipelining. The next known algorithm which supports pipelining is delayed LMS [3] which utilises the delayed version of calculated output to generate new weights. However, recent analysis of the critical path of the LMS finite impulse response (FIR) filter has shown that in real applications the requirement of pipelining is obsolete in direct form implementation of above. In real applications where a very high sampling rate is required, the LMS filter can be implemented with small adaptation delay. The LMS adaptive filter still gains importance among the different complementary metal–oxide semiconductor (CMOS) families in terms of energy efficiency, as presented in this paper.

To improve the speed of the LMS filter and to reduce the delay for making it faster, combinatorial logic optimisation as well as compression is applied and the critical path analysis is revised. Different forms of LMS filters are suggested in [4, 5], in which the delayed error signal is used for the governance of the coefficient update block of the filter at a particular time instant and the delay in the coefficients is in the range of $1-N$. When the novel architecture along with the algorithm is implemented on the different CMOS families, it is found to be more energy efficient. The filter weights are improved in each sampling period by the calculation of the difference between the filter response and the desired response for the algorithm implementation. The coefficients of the filter are updated by the equations

$$w(n+1) = w(n) + \mu x(n)r(n) \quad (1)$$

$$r(n) = d(n) - w(n)^T x(n) \quad (2)$$

In the above equations, $x(n)$ is the input vector and $w(n)$ is the coefficient update vector at a particular iteration, for example, at the n th iteration, μ is the step size, which plays an important role in

the performance analysis of the LMS adaptive filter

$$X(n) = [x(n), x(n-1), \dots, x(n-N+1)]^T \quad (3)$$

$$w(n) = [w_n(0), w_n(1), \dots, w_n(N-1)]^T \quad (4)$$

where the desired response is denoted by $d(n)$, the response of the filter is denoted by $w(n)^T x(n)$ and $r(n)$ is the difference between the desired and the filtered response at the n th iteration. Conventional implementation of the above equation is shown in Fig. 1.

In the previous literature of the LMS algorithm, it is assumed that the arithmetic operations can only be started when all the inputs are present, which is the main cause of the long critical loops [6–9]. In one multiply–add operation, the addition is always possible after the multiplication.

This assumes that the critical path of one multiply–add operation is equal to T_{MA} , which is given as

$$T_{MA} = T_{MULTIPLY} + T_{ADDITION} \quad (5)$$

Therefore, the direct form realisation of the LMS filter has the critical path as $T_{MA} = 2T_{MULTIPLY} + (N+1)T_{ADDITION}$ which is very high for N -order filters. With a different approach, we can assume that as soon as the least significant bit (LSB) is present, arithmetic and-OR operations are started and the delay in propagation is taken $T_{MA} = T_{MULTIPLY} + T_{FAC} + T_{FAS}$, where $T_{MULTIPLY}$ is one multiplication time and T_{FAC} and T_{FAS} are the time involved in carry and sum [10] which is much less than above.

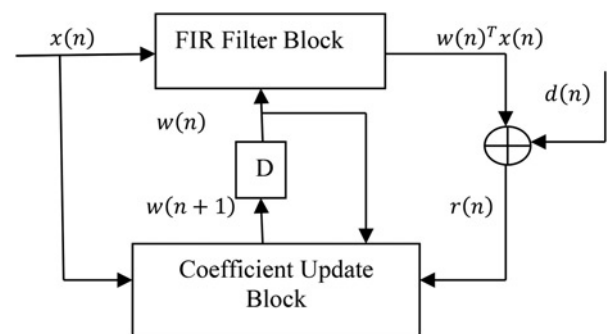


Fig. 1 Conventional adaptive LMS filter

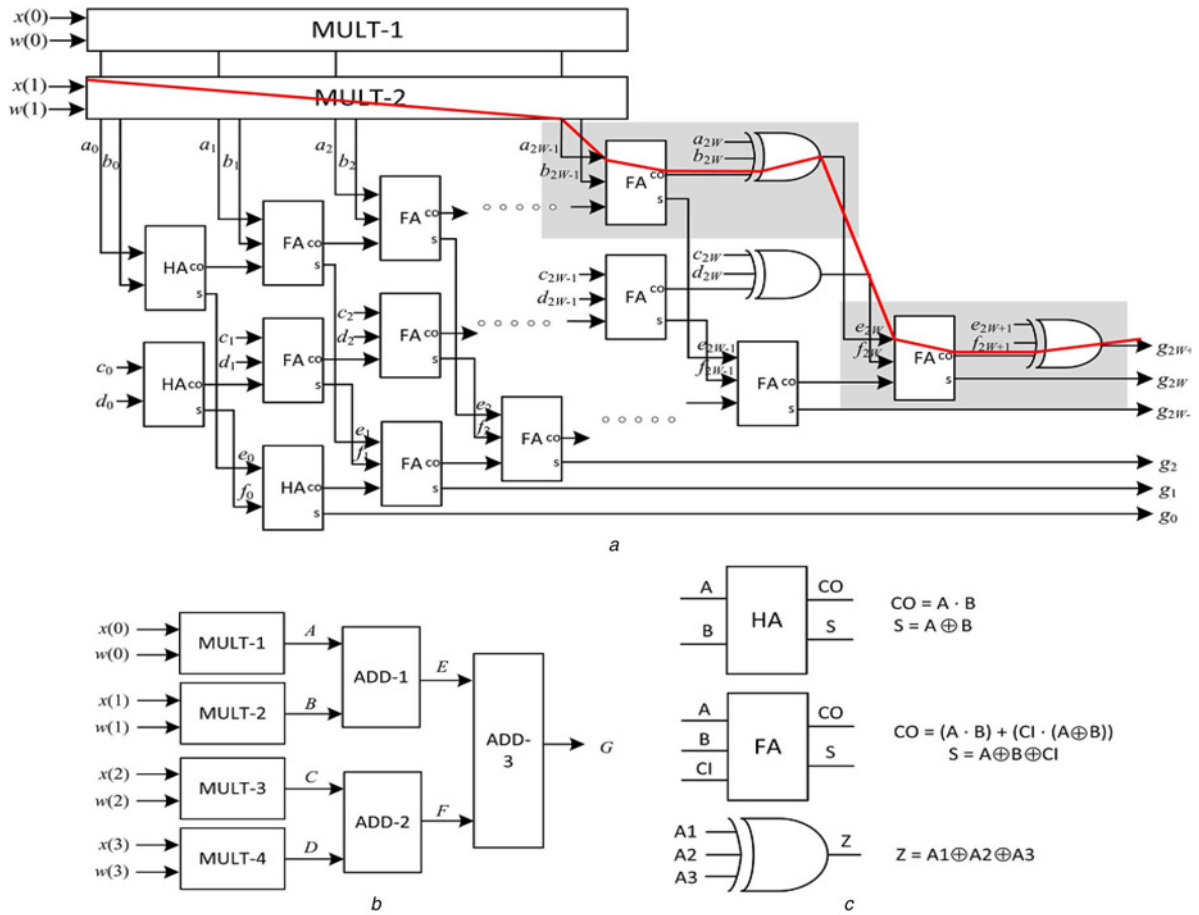


Fig. 2 Inner product critical path computation for $N = 4$

The rest of this paper is organised as follows. In the next section, the critical path analysis is discussed. The synthesis results of different CMOS families and simulation for different orders are presented in Section 3. The architectures for high-speed filter are presented in Section 4. The conclusion is given in Section 5. In Section 6, future scope is given.

2 LMS filter and critical path analysis

In the LMS adaptive filter shown in Fig. 1, the critical path for direct implementation is given as the sum of two terms

$$T = T_{\text{ERROR}} + T_{\text{UPDATE}} \quad (6)$$

in which T_{ERROR} is the time taken in error calculation and T_{UPDATE} is the time taken in coefficient update. For two stages of pipelining, it is given as

$$T = \max(T_{\text{ERROR}}, T_{\text{UPDATE}}) \quad (7)$$

The inner product computation for a length of 4 is given as $w(0)x(n) + w(1)x(n-1) + w(2)x(n-2) + w(3)x(n-3)$ and is shown in Fig. 2. In this, all the multiplications start in a synchronous way and addition starts as soon as the LSBs of the multiplication are present.

In full adder, because of the presence of the 3-input XOR gate, delay is involved and denoted as T_{FAS} . In general, in [10] it is shown that the total delay because of inner product computation is given as

$$T_{\text{INNER}} = T_{\text{MULTIPLY}} + [\log_2 N] \Delta \quad (8)$$

$$\Delta = T_{\text{FAS}} + T_{\text{FAC}} \quad (9)$$

Similarly, in the error computation and coefficient update the critical path is given by

$$T_{\text{ERROR}} = T_{\text{MULTIPLY}} + ([\log_2 N] + 1) \Delta \quad (10)$$

$$T_{\text{UPDATE}} = T_{\text{MULTIPLY}} + \Delta \quad (11)$$

Using all the above equations of the critical path analysis of the LMS filter, the critical path can be found as in (12), which is less than (5)

$$T = \max(T_{\text{MULTIPLY}} + ([\log_2 N] + 1) \Delta, T_{\text{MULTIPLY}} + \Delta) \quad (12)$$

Application of pipelining, in which we place latches in feed-forward data paths, can further reduce this critical path. In the next sections, pipelining is introduced in the error computational block.

3 Implementation and simulation results

The simulated results are presented here for fixed point adaptive LMS filters implemented with MATLAB [11–13] and synthesised using Xilinx 14.2. Different field programmable gate array (FPGA) families (28, 40 and 90 nm) are taken for different orders of LMS filter ($N=8, 16$ and 32) and the results are compared in Table 1. Different parameters such as input–output blocks (IOBs), power requirement look up tables (LUTs), area (number of slices) and combinational path delay (CPD) are considered and the comparative analysis with optimised combinational path delay at the cost of increased LUTs is presented. Reduction of the critical path is possible only if we put a pipelining after each addition by which complexity improves. As the order of filter increases in all

Table 1 Comparative view of parameters for different FPGAs (28, 45 and 60 nm)

CMOS family	Filter order, N	Number of slice registers	DSP	Power, mW	Slice LUTs	Input/output blocks	CPD, ns	Optimised CPD, ns	CPD, ns without constraint
Artix-7 (28 nm)	8	282	19	1.8	319	75	6.457	0.459	25.6
	16	550	35	2.1	517	74	6.429	0.959	32.77
	32	1086	67	3.5	1170	74	8.111	1.569	46.55
Spartan-6 (45 nm)	8	238	19	4.8	343	75	14.29	5.89	32.488
	16	485	32	7.3	1429	74	14.22	6.71	48.211
	32	968	58	15	3663	74	14.79	6.98	60.491
Virtex-4 (90 nm)	8	238	19	5.5	377	75	14.47	6.44	34.832
	16	917	32	9.8	1414	74	14.42	6.74	36.26
	32	943	32	18.4	9889	74	15.03	6.9	44.816

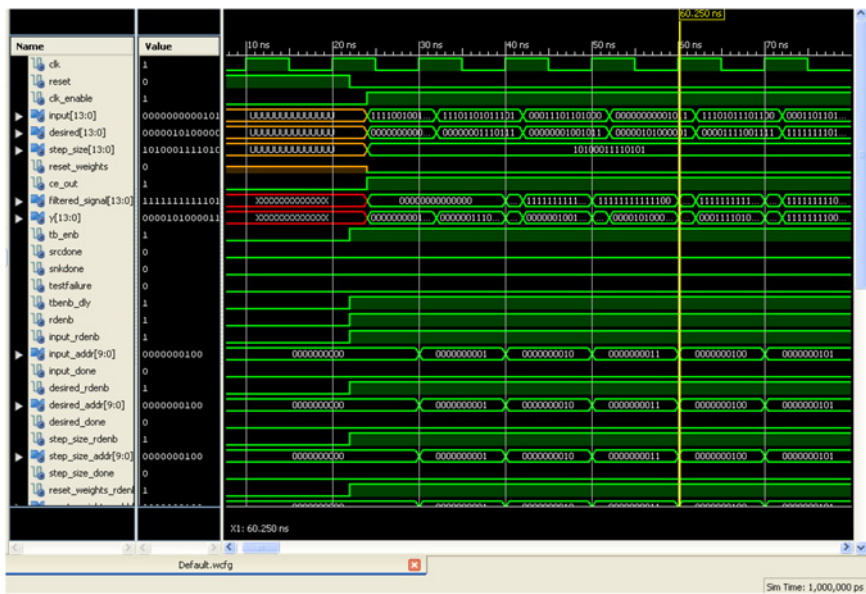


Fig. 3 Simulation for order $N = 8$

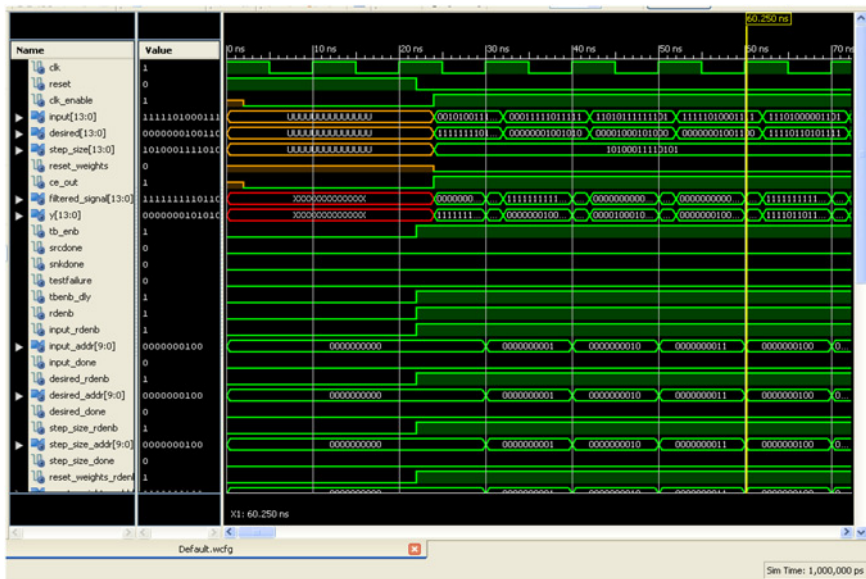


Fig. 4 Simulation for order $N = 16$

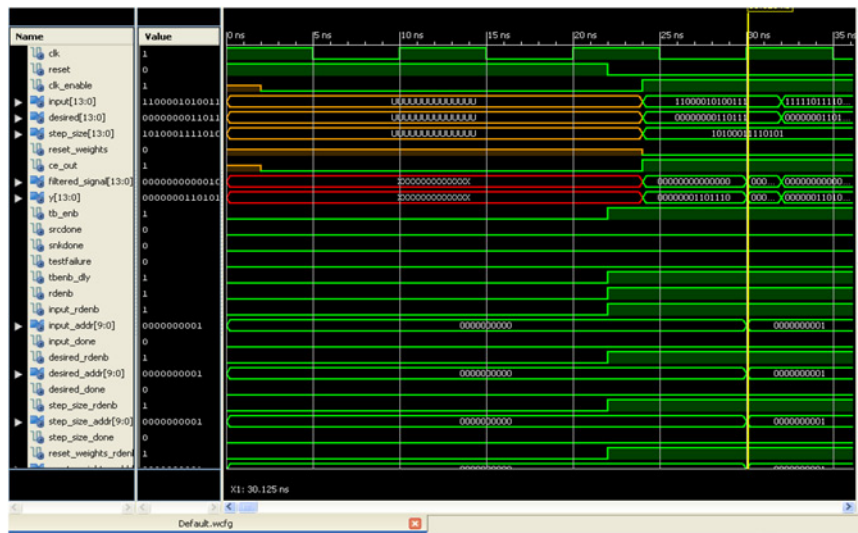


Fig. 5 Simulation for order $N = 32$

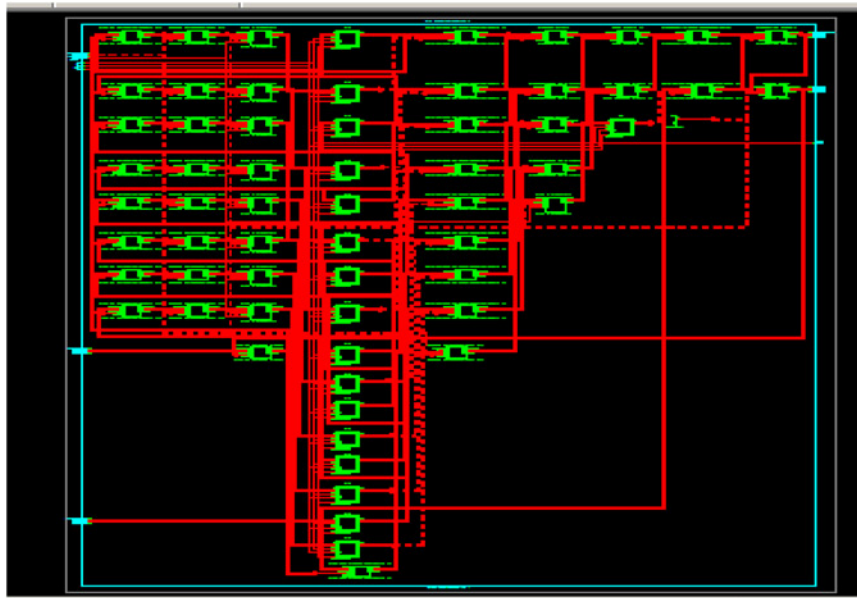


Fig. 6 RTL architecture for $N = 8$

FPGAs [14–16] selected here, the path delay is increased along with the power consumption. Among all the simulations, the Artix family has the lowest combinational delay. Figs. 3–5 show the simulated model and generated waveforms for different orders, which show that as order increases, the consumed power and the critical path also increase. This power is consumed in the LUTs, because as the order of the filter increases, the LUTs also increase.

The adaptation delay also increases as the order increases. In the same fashion, register transistor logic (RTL) schematics of simulated models for different orders are shown in Fig. 6–8.

It can be seen from the RTL structures of the different orders that as the order increases, the LUTs, digital signal processors (DSPs) slice registers and CPD also increase. Furthermore, the synthesis results analysis is shown in the graphs for different constraints. The number of LUTs and corresponding power consumption is shown in Table 1. To reduce the power consumption, the LUTs are combined within the processor. This process of combining is also known as fine-grained pipelining. These RTLs are the optimised structures which are achieved as the result of register optimisation and LUT combination.

From Fig. 9, it can be seen that in every case the power consumption increases as the higher-order filters are taken. Among the three families, it can be seen that the Artix-7 family consumes lowest power.

Similarly, from Fig. 10 it can be seen that maximum LUTs are used by the Virtex-4 family; so correspondingly, the power consumption is greater. From Table 1 it can be seen that the path delay is maximum in Virtex-4. Therefore, the performance analysis of the implemented algorithm shows that the Artix family is better in comparison to others under the stated constraints.

4 LMS direct form architecture

As it can be seen from the conventional architecture in Fig. 1 of the LMS adaptive filter, there are two main blocks, error computation and coefficients-update. If the computation of these blocks occurs in the same period, then less adaptation delayed implementation is possible. Since the pipelining is not used in this implementation, the computation of both the blocks is not possible in the same period. Therefore, an available option is multiplexing of the

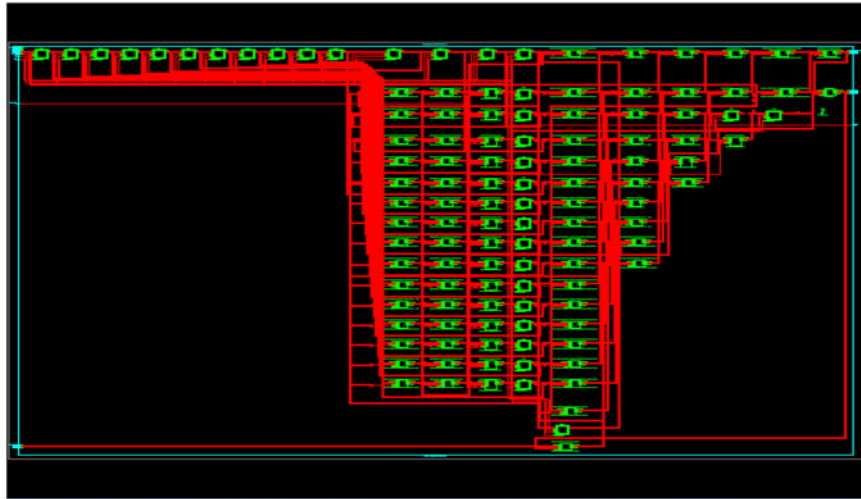


Fig. 7 RTL architecture for $N = 16$

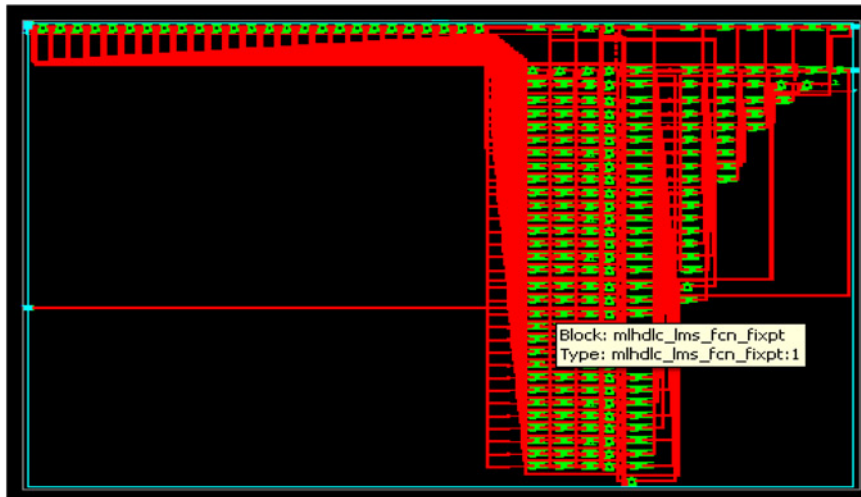


Fig. 8 RTL architecture for $N = 32$

above two tasks using the same hardware components (such as multipliers, adders, registers etc.). Both the tasks should be performed in the same period on a one-by-one basis which enhances the performance of the architecture. The delay can be put either after the adder or after calculation of μe_n .

In the prior half cycle, each clock estimation of μe_n is done and in the latter half of the cycle these estimated weights are given to the multipliers by use of multiplexers for estimation of $x(n)\mu e_n$. In the design implementation, shown in Fig. 11, it is better to utilise combinational circuits such as multiplexers at the place of multipliers.

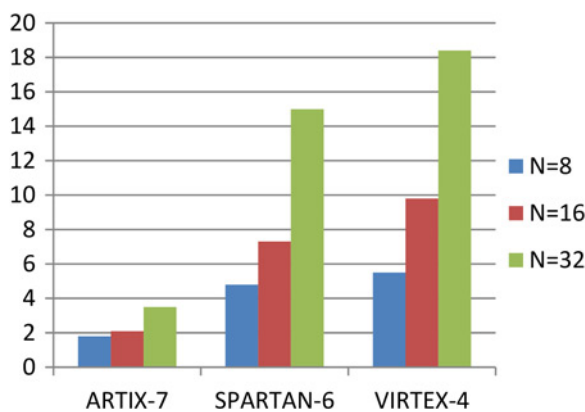


Fig. 9 Power consumed in different CMOS families

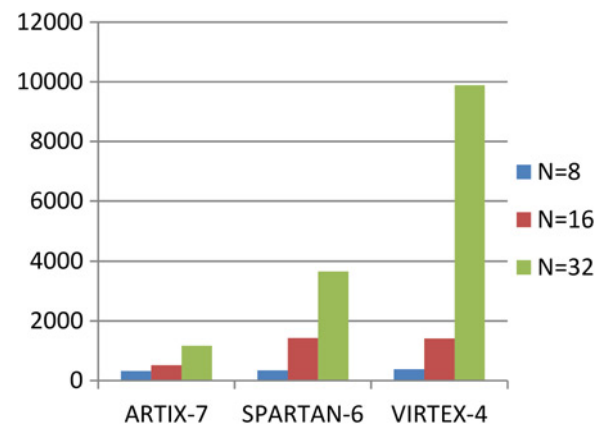


Fig. 10 Slice LUTs for different CMOS families

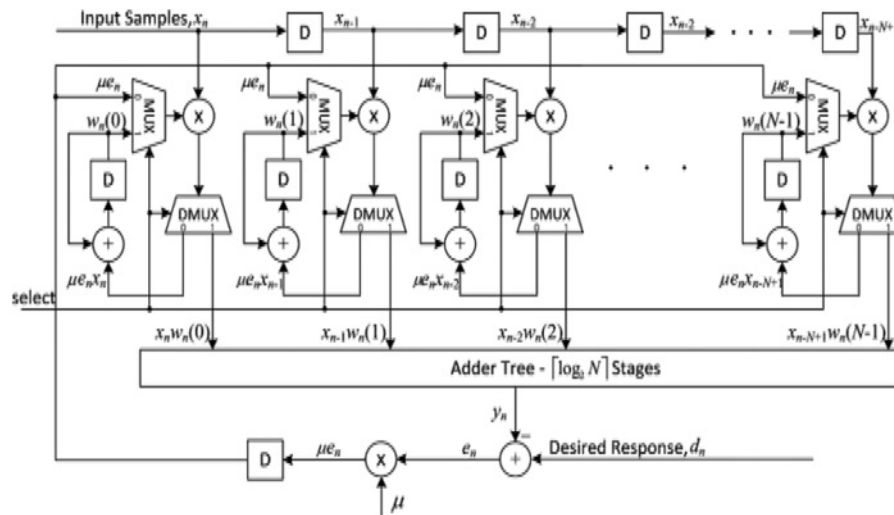


Fig. 11 LMS architecture

The basic reason behind this is that the area required for the multiplier is more when compared with the combinational logics.

Final addition is done by the use of the binary tree to achieve the final output as $y(n)$. The reason for utilising the binary tree adder is that it requires $\log_2 N$ stages, whereas the normal adder requires N stages to obtain the output.

5 Conclusion

In this paper, synthesised simulation work is carried out over different CMOS families and detailed simulated results with generated waveforms and RTL schematics are presented. To make the architecture faster, critical path analysis is restructured with different assumptions for starting arithmetic operation. At the design level for utilising the area, efficient resource sharing is implemented. From the simulated results of different CMOS families, Artix-7 consumes less hardware resources and proves its efficiency in terms of high-speed and low-power implementation of the LMS adaptive filter. It can also be concluded that as the order increases ($N=8, 16$ and 32), the resource requirements such as power consumption, IOBs, LUTs etc. also increase.

6 Future scope

There are different applications of the LMS adaptive filters in the real world where high speed is required. Some of the applications include noise cancellation, channel equalisation and system identification. In future, we can use this LMS direct form FIR filter in different applications for different families, so it can be decided which FPGA family gives the better results under some specific constraints. Furthermore, if we can use this architecture on the application-specific FPGAs, then complexity, adaptation delay, power consumption etc. can be reduced.

7 References

- [1] Widrow B., Stearns S.D.: 'Adaptive signal processing' (Prentice-Hall, Englewood Cliffs, NJ, USA, 1985)
- [2] Haykin S., Widrow B.: 'Least-mean-square adaptive filters' (Wiley-Inter Science, Hoboken, NJ, USA, 2003)
- [3] Yi Y., Woods R., Ting R.L.-K., Cowan C.F.N.: 'High speed FPGA-based implementations of delayed-LMS filters', *J. Very Large Scale Integr. (VLSI) Signal Process.*, 2005, **39**, (1–2), pp. 113–131
- [4] Paarhi Keshav K.: 'VLSI digital signal processing systems design and implementations' (John Wiley & Sons, New York, NY, 2003)
- [5] Long G., Ling F., Proakis J.G.: 'The LMS algorithm with delayed coefficient adaptation', *IEEE Trans. Acoust. Speech Signal Process.*, 1989, **37**, (9), pp. 1397–1405
- [6] Meyer M.D., Agrawal D.P.: 'A modular pipelined implementation of a delayed LMS transversal adaptive filters'. Proc. IEEE Int. Symp. Circuits and Systems, May 1990, pp. 1943–1946
- [7] Meher P.K., Park S.Y.: 'Low adaptation-delay adaptive filter part-II: an optimized architecture'. Proc. IEEE Int. Midwest Symp. Circuits and Systems, August 2011
- [8] Meher P.K., Park S.Y.: 'Area-delay-power efficient fixed-point LMS adaptive filter with low adaptation-delay', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 2014, **22**, (2), pp. 362–371
- [9] Park S.Y., Meher P.K.: 'Low-power, high-throughput, and low area adaptive FIR filter based on distributed arithmetic', *IEEE Trans. Circuits Syst. II, Express Briefs*, 2013, **60**, (6), pp. 346–350
- [10] Meher P.K., Park S.Y.: 'Critical-path analysis and low-complexity implementation of the LMS adaptive algorithm', *IEEE Trans. Circuits Syst. I, Regul. Pap.*, 2014, **61**, (3), pp. 778–788
- [11] Prakash S., Kumar T.G.R., Subramani H.: 'An FRGA implementation of the LMS adaptive filter for active vibration control'. *Int. J. Research in Eng. and Tech.*, 2013, **2**, (10), pp. 1–10
- [12] Meher P.K., Maheshwari M.: 'A high-speed FIR adaptive filter architecture using a modified delayed LMS algorithm'. Proc. IEEE Int. Symp. Circuits Systems, May 2011, pp. 121–124
- [13] Vanus J., Styskala V.: 'Application of optimal settings of the LMS adaptive filter for speech signal processing'. Proc. IEEE Int. Multiconf. Computer Science and Information Technology, October 2010, pp. 767–774
- [14] Spartan-6 user-guide. Available at http://www.xilinx.com/support/documentation/user_guide/ug380.pdf
- [15] Vitex-4 user guide. Available at http://www.xilinx.com/support/documentation/user_guide/ug070.pdf
- [16] Artix-7 Low Power FPGA user guide. Available at http://www.xilinx.com/support/documentation/user_guide/