Power Optimized VLSI Architecture of Distributed Arithmetic Based Block LMS Adaptive Filter

Gangadharaiah S. L[1], C. K Narayanappa[2], Divya M.N[3], Navaneet S[4] and Dushyant N[5]

[1]VTU Research Centre, Department of Electronics and Communication, M. S. Ramaiah Institute of Technology, Visvesveraya Technological University, Belagavi -590018, India; gdhar@msrit.edu
[2]Department of Medical Electronics, M. S. Ramaiah Institute of Technology, Visvesveraya Technological University, Belagavi -590018, India; c_k_narayanappa@msrit.edu
[3]School of Electronics and Communication Engg, REVA University, Kattigenahalli, Bangalore, India; divya.mnl@reva.edu.in
[4]Department of Electronics and Communication, M. S. Ramaiah Institute of Technology, Visvesveraya Technological University, Belagavi -590018, India; navaneets@gmail.com
[5]Department of Electronics and Communication, M. S. Ramaiah Institute of Technology, Visvesveraya Technological University, Belagavi -590018, India; ndushyant46@gmail.com

*__Correspondence:__ Gangadharaiah S. L; email: gdhar75@gmail.com;

ABSTRACT- In this paper, we are presenting a power-efficient Distributed Arithmetic (DA) based Block Least Mean Square (BLMS) Adaptive Digital Filter (ADF). The proposed DA BLMS architecture proposes a shared area-efficient Multiplier Accumulate Block that calculates both the partial filter products and the weight increment terms in the same module. It also uses Multiplexers (MUX) and Demultiplexers (DEMUX) which passes only L out of N inputs, where N and L are the filter length and chosen block size respectively, into the MAC thus helping in achieving the DA functionality along with reduced power consumption. Also, efficient truncation of the obtained error and weight update terms is performed by being able to select the non-zero-bit part of the signal to be fed back. The entire architecture is driven by a single slow clock which reduces the power consumption of the device further. On comparing with the best existing DA BLMS Structures, the proposed architecture uses 15% lesser power, 14% lesser EPS according to ASIC Synthesis, and for a filter length of N=16 and a block size of L=4 respectively.

Keywords: Adaptive Filter; Block LMS, Least Mean square.

ARTICLE INFORMATION

Author(s): Gangadharaiah S. L, C. K Narayanappa, Divya M. N, Navaneet S and Dushyant N.;
Received: 08/06/2023; **Accepted**: 14/07/2023; **Published**: 23/09/2023;
e-ISSN: 2347-470X;
Paper Id: IJEER 0806-01;
Citation: 10.37391/IJEER.110320
Webpage-link:
https://ijeer.forexjournal.co.in/archive/volume-11/ijeer-110320.html

Publisher's Note: FOREX Publication stays neutral with regard to Jurisdictional claims in Published maps and institutional affiliations.

1. INTRODUCTION

An Adaptive Filter is a digital device which has a linear filter that has a variable transfer function and a method of adjusting those parameters according to the required optimization algorithm. It is a closed loop system that updates the filter coefficients based on negative feedback called the error signal. Adaptive filters are useful in real-time utilities such as echo cancellation, Electroencephalograms (EEGs), Fetus Detection [1], etc.

In this paper, we present the Block Least Mean Square (BLMS) as the optimization algorithm for the adaptive filter. Just like other algorithms it is essentially a derivative of the LMS Adaptive filter but was chosen for its computational efficiency, i.e., even while the convergence performance is the same as that of LMS, it produces a throughput L times that of the LMS [2].

Several schemes have been proposed to implement the DA-BLMS Adaptive Filter in VLSI. Jiang [3] proposed approximate distributed arithmetic circuits where the Radix 8 booth encoder was used in order to reduce the number of partial products and a Wallace tree adder to accumulate those partial products thus reducing area complexity. [4-9] used an Offset Binary Coding (OBC) to the input and weight increment terms which are updated to the LUT. This eliminates the use of multiplication, and this reduces area complexity in the respective designs. But even though the area is lesser, the device does not possess any enable signals which means that the device will be ON for all the iterations which mean that for higher iterations the power consumed would be high.

Another method was proposed in [10] which included the calculation of inputs and weights bit serially for a particular iteration. This approach will then make use of only 1-bit hardware throughout the architecture which reduces the area complexity. But it uses L processing elements (PE) to calculate the N partial filter products and the weight increment terms and it could increase with an increase in filter length. In the proposed approach we have replaced this by using a single Multiply and accumulate (MAC) unit and the overall hardware used will almost remain the same irrespective of its filter length. Even though the sampling rate of the MAC is slower as compared to the LUT-based DA architectures as proposed in [11-13] it is only enabled whenever it is required by a slow rate clock which significantly reduces the power consumption of the device. Another approach was presented in [14-17] which involved using Carry save accumulation (CSA) with a high-frequency clock instead of a MAC unit and used a slower clock for the other modules for power reduction. This asynchronous clocking system can lead to clock skew and besides, the device would also use extra power to operate two clocks one of them being a high-frequency clock. This will hence slightly increase the power consumption. The architecture we propose instead uses a single synchronized slow clock to operate the entire device which reduces its power consumption.

The key contributions to this paper are:

1. Replacing the conventional LUT-based DA architecture with a single shared MAC made of Vedic multipliers and a ripple carry adder (RCA) which consumes lesser area than the conventional Multiply Accumulate design.

The LUT (Look up table) was used to store the input vectors x(n-k) to be passed to a parallel architecture of multiplexers to compute the partial products of the input and weight increment terms which would be added to the previous weight in order to be updated. This approach used to pass the input vectors from the LUT to the several multiplexers at the same time and hence it would calculate the partial products and provide with all the bits parallelly at the same time thus reducing area and latency since the output would be available in a single clock cycle. But since the LUT does not have an enable signal the partial product generators will have a significantly high-power consumption since the block is running even when it is not being used until the new input vectors are received at the after-error correction for the next iteration. The proposed MAC structure may take more latency and reduced speed since we are using L clock cycles to compute the partial products with the MAC but the power is significantly reduced since we are enabling the MAC block only when we receive the corrected inputs from the adaptive filter feedback.

2. Using a synchronized slow clock to operate the entire device and compromises timing but provides significant power reduction throughout the device.

3. Using basic Mux and Demux blocks with select signals that can manage the N input signals to send only L inputs at a time in L clock cycles. This ensures enabling of the MAC unit and other blocks such as error computation and weight update blocks only when it is required for calculation and will not be on throughout the iteration process. It also helps in maintaining the same hardware irrespective of the order of the filter and hence reducing power consumption for any N order filter.

Time is significantly larger as we have used a slow clock to keep the inputs synchronized between the mux and demux as we are passing inputs one by one. Since mux and demux were also introduced for power savings it was a tradeoff between power and speed. The design can be upgraded for speed by changing some of the logical instances used for computations but VLSI mostly has faced problems mainly with Area and power which increases the cost of chip-making and the machines themselves. Hence, we decided to go for the power efficient approach.

The convergence will be better but also longer than expected. And taking all the accurate bits instead of the approximation would mean using adders and multipliers with more bits which will increase both Area and power consumption of the device significantly larger than it is along with the speed. Hence, we decided to consider the approximation to get it to the closer convergence with an efficient design than to be accurate with an inefficient design.

2. LITERATURE REVIEW

The Block LMS Filter implements an adaptive least mean-square (LMS) filter, where the adaptation of the filter weights occurs once for a block of samples.

Consider an Nth order Filter with block size L. Then according to Block LMS algorithm an input vector.

[r(kL) r(kL-1) r(kL-2) ……. r(kL-N+1)]$ will be divided vectors of block size L such as [r(kL) r(kL-1) ….. r(kL-L+1)] and these vectors will be calculated block by block until all N inputs are calculated and their outputs are obtained. For this let’s take an input matrix of L X N into where the block vectors act as column vectors.

$$R_k = \begin{pmatrix} r(kL) & r(kL-1) & .. & r(kL-N+1) \\ r(kL-1) & r(kL-2) & .. & r(kL-N) \\ .. & .. & .. & .. \\ r(kL-L+1) & r(kL-N) & .. & r(kL-L-N+2) \end{pmatrix} \quad (1)$$

Now, this input vector can be used for the calculation of filter output, error and weight update vectors as shown, The BLMS algorithm follows the weight update mechanism same as that of the LMS algorithm which is given as:

$$\mathrm{w}_{k+1} = \mathrm{w}_k + \Delta\mathrm{w}_k \quad (2)$$

Where the weight increment term Δw_k is given by

$$\Delta\mathrm{w}_k = \mu . X_k^T . \mathrm{e}_k \quad (3)$$

Where μ is the adaptive step size or the learning rate, X_k is the input matrix consisting of the block vectors column wise and ek is the error vector obtained.

The error vector is computed from the filter output and desired filter output as

$$\mathrm{e}_k = \mathrm{d}_k - \mathrm{y}_k \quad (4)$$

Where dk is the desired filter output and yk is the obtained filter output. This comparison result is the error vector which will be fed back to the input side to calculate the weight increment term for the second iteration.

2.1 Calculation of Filter output Y_k according to BLMS algorithm

According to the BLMS algorithm, as mentioned before, the input matrix of L X N will be broken down into M square matrices L X L where N=LM. The corresponding matrices will be selected based on the iterations, *i.e.* If S^j_k is one of the decomposed matrices and j ‘is the iteration number, then the matrix is given as

$$x_k{}^{j} = \begin{pmatrix} r(j'L) & r(j'L-1) & .. & r(j'L-N+1) \\ r(j'L-1) & r(j'L-2) & .. & r(j'L-N) \\ .. & .. & .. & .. \\ r(j'L-L+1) & r(j'L-N) & .. & r(j'L-L-N+2) \end{pmatrix} \quad (5)$$

Where j‘= k-j and k is the original number of iterations of the adaptive filter and j’ is the iteration that deals with the selection and computation of the decomposed block matrices and 0<=j<=M-1.

Also, the weight vector wk can also be decomposed into vectors of size L and is given as

$$c_k^j = (\mathrm{w_k}(jL)\ \ \mathrm{w_k}(jL+1)\ .\ .\ \mathrm{w_k}(jL+L-1)) \quad (6)$$

Now we can calculate the partial filter products from these 2 matrices which is given as

$$u(i,j) = s_k^{ij} . c_k^j \quad (7)$$

Where s^{ij}_k are the $(i+1)^{th}$ row vectors of the matrix S^j_k multiplied with the decomposed block weight matrix c^j_k where 0<=i<=L which is the variable that selects each row of the decomposed matrix S^j_k to be multiplied with the corresponding decomposed weight vector c^j_k. This multiplication for the decomposed input matrix with the decomposed weight vector will give rise to M partial filter products and hence each filter output y_k can be found out by

$$y(kL-i) = \sum_{j=0}^{M-1} u(i,j) \quad (8)$$

2.2 Calculation of Weight increment term Δwk according to the BLMS algorithm

Just like calculating the output the weight vector can also be calculated by decomposing the original vector of size N into M vectors of size L each as

$$\Delta c_k^j = (\Delta \mathrm{w_k}(jL)\ \Delta \mathrm{w_k}(jL+1)\ .\ .\ \Delta \mathrm{w_k}(jL+L-1)) \quad (9)$$

Where each element of Δc^j_k is given by

$$\Delta \mathrm{w_k}(\mathrm{jL}+\mathrm{i}) = \mu v(i,j) \quad (10)$$

Where μ is the adaptive step size and $v(i,j)$ is the partial products of s^{ij}_k *equation (7)* multiplied with e_k for each iteration, *i.e. v(i,j)* is given as

$$v(i,j) = s_k^{ij} . e_k^j \quad (11)$$

This weight increment term calculated within the block will be added with the previous weight vectors to produce the updated weights to be updated in the filter transfer function for the next iteration.

For the subtractor we made use of the existing RCA itself to perform the subtraction as well and for the multiplier we re-used the 8-bit Vedic multiplier. Will be mentioned in the paper.

2.3 Architecture

The proposed DA-BLMS architecture of the adaptive filter consists of a Multiply and Accumulate (MAC) unit, 16:1 MUX and DMUX, Error Computation block, Weight update Block and a decision device as shown in the figure.

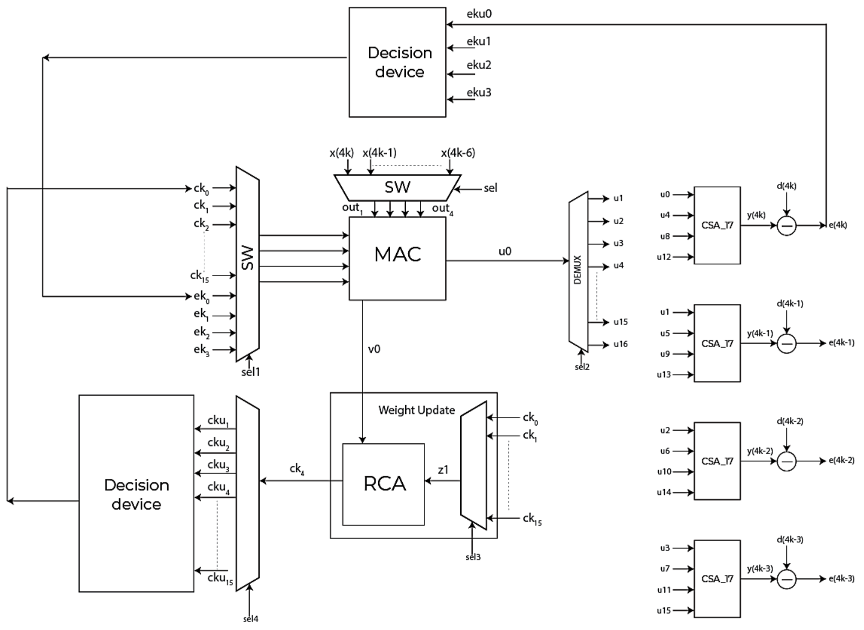

Figure 1: Proposed DA-BLMS Architecture for the Block Size L=4

The above *figure 1* shows the proposed architecture of the DA-BLMS adaptive filter for a filter length of N=16 and L=4. The block filter takes in 2L-2 inputs according to *equation (5)* and 16 filter coefficients for an N=16 filter which is called weights. These inputs are taken 4 at a time with a help of selection devices SW and SW2 for S^j_k and c^j_k / e^j_k as input vectors respectively according to *equations (5) and (6)*. These 4 inputs that arrive from SW and SW2 respectively are provided as inputs to the Multiply and Accumulate (MAC) block which performs the matrix multiplication of the inputs that occur at that instant of time.

The BLMS algorithm broke down the input matrix to blocks and provided for block level outputs which can be integrated later to form the actual output of the adaptive filter. Because of this break down, we would be getting L times the throughput of an LMS filter which is ideal of parallelization if there are more adaptive filters in the picture. This break down also helps in using lesser blocks and lesser inputs which would make computation for the device easier and would occupy lesser power because of the lesser complexity of the logic blocks to be implemented.

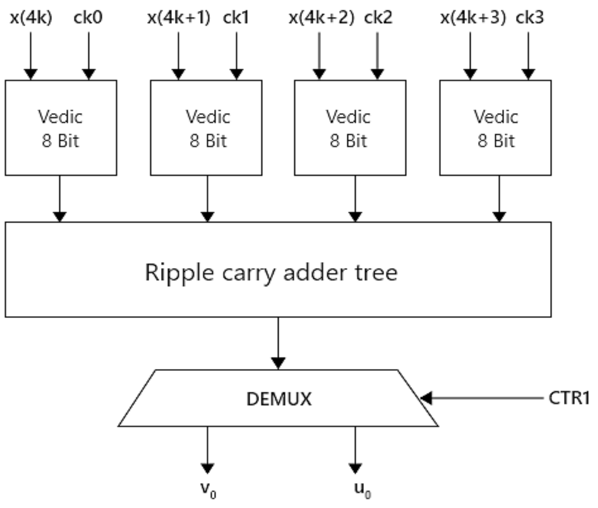

Figure 2: Internal Architecture of the proposed Multiply and Accumulate (MAC) block

The Vedic multiplier provided faster computation of multiplication by usage of adders as well and reusing the outputs of the 4X4 Vedic multipliers. Since the multiplication in logic blocks takes the most amount of time and, we need to iterate the inputs through the computations more than once, we thought about reducing the time complexity of the multiplier by using Vedic multiplication which would in turn reduce the power consumption of the MAC block so that we do not have to use it for a long time.

As shown in figure 2 MAC consists of 4 low area Vedic multipliers [18-21] and one 16- bit Ripple Carry Adder (RCA). This was again chosen because even though we would need 3 RCAs to add 4 operands it occupies lesser area than one Carry Save Adder (CSA) which can add 4 inputs at once. To further simplify the area consumption, we implemented a shared MAC structure that gives either the partial filter product *u(i,j)* or the weight increment term *v(i,j)* at a time depending upon which inputs are coming into the MAC unit, i.e. if the weight vector (c^j_k) are the input to the MAC then it means that MAC calculation is taking place according to *equation (7)* which gives *u(i,j)* as the output of the MAC and if the error vector is the input to the MAC then the MAC calculation takes place according to *equation (11)* which gives the output as *v(i,j)*. To implement this, we have also integrated a DMUX within the MAC whose input will be the MAC output with a select signal as CTR1 which will determine whether the output must be provided to '*u*' or '*v*' from *fig. 1*.

To suit the functionality of the BLMS Adaptive Filter, the MAC first provides the output to 'u' , the partial filter output. Now since we have only used one MAC in the entire architecture, we would get only a single output at a time. But according to (7), since 0<i<M-1 and 0<j<L-1 , we would be requiring N=LM , i.e 16 combinations of partial filter outputs in this case. So hence we use a 1:16 DMUX1 which takes the particular partial filter output as input and provides it to one of the 16 outputs according to the iteration. i.e suppose i=0 and j=0 means the partial filter output for that particular iteration is u(0,0) and hence this MAC output is given to output u1 of DMUX1. Similarly, u (1,0) is given to u2 and so on till all the 16 combinations appear at the output of DMUX1 in 16 clock cycles respectively.

Now the filter output vector y(kL-i) can be found by using *equation (8)* which will add a total of 16 partial filter products obtained in DMUX1 to get the L=4 filter outputs. Here we use a carry save adder for addition since the design was created such that it could add 4 inputs at once which meant that only one adder is required to produce an output instead of a ripple carry adder which uses only 2 inputs for addition in which case, we would require 3 RCAs to produce the output. Once the filter output is obtained by fixing the desired output vector and then comparing it with the obtained filter output vector element by element. This will give the error vector according to *equation (3)*.

But this error vector obtained is of 15 bits which must be passed as input to the Selection element SW2 which takes only B'=8-bit input. So hence it is passed to the input through a decision device for truncation of the LSB 8 bits. The problem with direct truncation of the LSB bits is that for smaller input values, their MSB 8 bits would be zero. So hence if it is passed directly as feedback it will result in a zero input which will eventually continue to remain in the same state infinite number of times without any updation of weights or decrement of the error vector elements. Hence we pass it through a decision device which is essentially a MUX which takes In the bit parts of the error signal *i.e.* $e_k[7:0]$ and $e_k[15:8]$ as the inputs and passes the MSB by default. Once the MSB is 0, then it passes the LSB to the input side. This way of truncation will make sure the Filter does not enter an idle state where it is stuck in one iteration.

Once the error signal appears at the input the Selection device SW2 passes the error as the input to the MAC unit to calculate the weight increment term. Once the MAC calculation is complete the CTR1 value which was at *1* previously to pass the partial filter outputs will now be *0* since this time the partial products of the weight increment term *(v(i,j))* is being calculated according to [11]. Once *v(i,j)* is calculated it is multiplied with the adaptive step size μ. The convergence of the algorithm depends upon μ, *i.e.*, it determines how fast the error signal is going to converge to *0*. It usually varies from *0* to *1* and as μ is closer to *1* it converges at a faster rate and as μ is closer to *0* it converges very slowly. The optimal μ value usually chosen is around 0.6 to 0.8.

The multiplication of $\mu v(i,j)$ is nothing but the weight increment term of that particular iteration. This is now directly provided as input to the 16-bit Ripple Carry Adder (RCA) and the other input to the RCA is the previous/ old weights. Since there are 16 weight terms and only one weight increment term coming in at a time, the old weight associated with the increment term is passed as input to the RCA where the addition takes place according to (1). This is done with the help of a 16:1 MUX and hence we will obtain all the updated weights in 16 clock cycles respectively. This weight updated terms, just as the error vector, needs to be truncated and hence the same decision device used for the error vector will also be used for the weight update coefficient vector and all weights will be truncated in another 16 clock cycles. So therefore, a total of 64 clock cycles are required to complete one update iteration of the DA-BLMS Adaptive Filter but at the same time the respective modules operate only when they are activated, i.e., the modules are operated only when the respective inputs arrive at that particular module and is idle at other times which helps in significant power consumption of the device and can greatly increase its battery life.

3. RESULTS

The proposed DA-BLMS Adaptive Filter design was coded in Verilog HDL for the filter of order N=16 and the block size of L=4 with the input of size B'=8 and was synthesized and compiled using the Libero®SoC Design Suite v11.9 with the ProASIC3®L A3P1000L low-power FPGA. The RTL Schematic of the Synthesized Design and FPGA Synthesis Results are as shown in *fig. 3* and *table 1* respectively.

3.1 Figures, Tables and Schemes

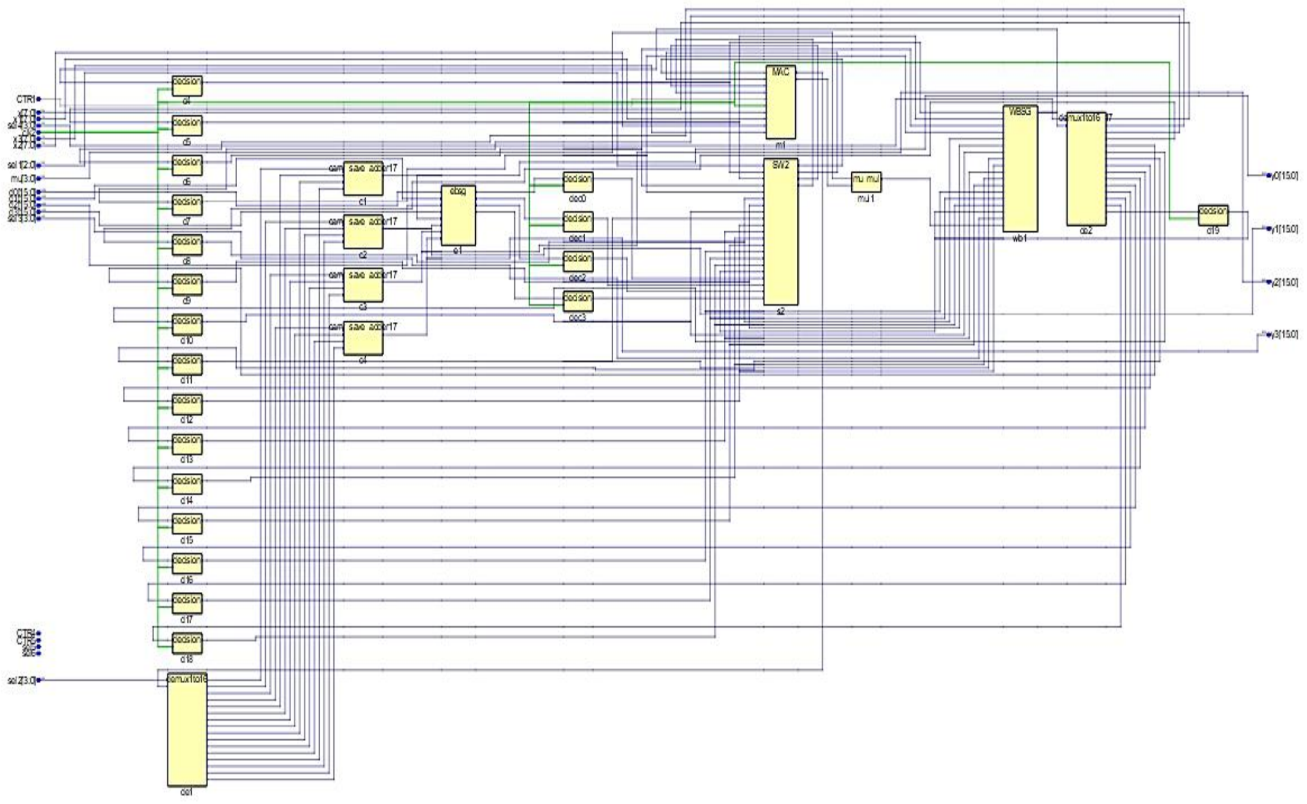

Figure 3: RTL Schematic of the Proposed DA-BLMS Adaptive Filter

Table 1: FPGA Synthesis results of the proposed DA-BLMS Architecture on ProASIC3®L for the filter length N=16 and block size L=4

Number of Slice LUTs	3507
Number of Bonded IOs	207
Power Consumption	1.064mW
Minimum clock period (MCP)	30.93ns

To validate the above FPGA Synthesis results including the Resource Utilization, Power consumption and speed, we compare the proposed architecture with the existing DA-BLMS Adaptive Filter Architectures as shown in *table 2* [22-29].

Table 2. Comparison of Power and Delay of Proposed Architecture with Synthesis Results of the existing

architectures of [10-12],[19],[6] for the filter length N=16 and block size L=4

Structures	Power in mW	MCP (ns)
DA-BLMS (Meher and Mohanty) [13]10	1.4302	25.8
Allred *et al* [16]11	1.5249	24.05
DA-LMS (Park and Meher) [20]12	9.41	3.2
Pipelined LMS (Khan and Shaik) [3]19	13.49	4.29
DA-LMS (Khan and Shaik) [5]6	19.23	4.56
Proposed	1.067	30.39

Figure 4 below shows the power consumption of proposed architecture compared to existing architectures.

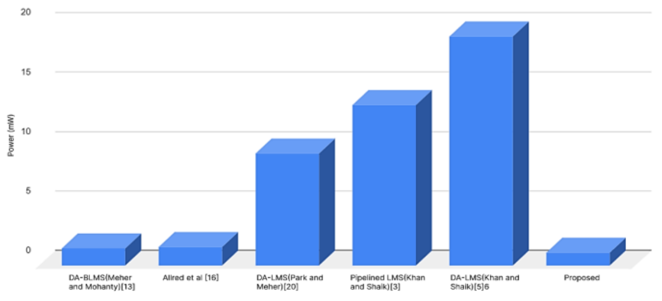

Figure 4. Power comparison with existing architectures

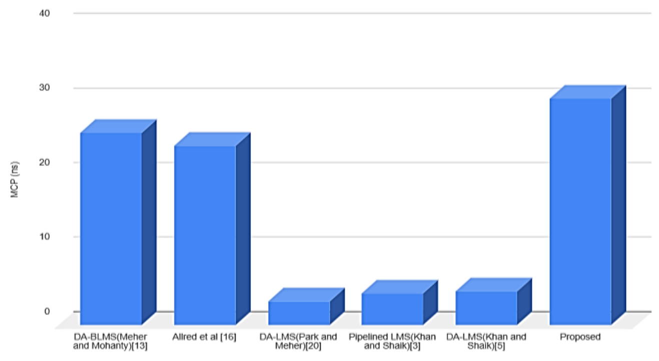

Figure 5. Delay comparison with existing architectures

4. DISCUSSION

As shown in Table 2, The proposed architecture was compared with the recent and best existing DA based Adaptive Filter architectures to validate the statement and arguments proposed in the Abstract and Introduction of the paper respectively. The Minimum Clock Period (MCP) of the proposed architecture is slightly higher than that of [10] and [11] and very high as compared to [12],[19] and [6]. This is because of Results also showed that the proposed design consumed 69.88\% lesser Slice Registers, 86.84\% lesser power and 87.94\% lesser EPS as compared to the best existing DA based VLSI architectures.

And for the improvements, we could be passing more inputs than just one at a time to the mux instead of just passing once at a time. With this we can maintain the current power level since we are still using mux and demux to activate the computational blocks but also it will allow us to use a higher frequency clock to support the speed which can reduce time latency as well significantly but there will be a tradeoff with the area as we will be using more parallel mux and demux to get the output.

5. CONCLUSION

Based on the above calculations and graphs, we can conclude that although the delay of the proposed architecture is comparatively higher, the power consumption is the least.

REFERENCES

[1] S. Haykin and B. Widrow, Least-Mean-Square Adaptive Filters, NJ, Hoboken:Wiley-Interscience, 2003.

[2] G. NagaJyothi and S. SriDevi, "Distributed arithmetic architectures for FIR filters-A comparative review," 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), 2017, pp. 2684-2690, doi: 10.1109/WiSPNET.2017.8300250.

[3] H. Jiang, L. Liu, P.P. Jonker, D.G. Elliott, F. Lombardi, J. Han, A high-performance and energy-efficient FIR adaptive filter using approximate distributed arithmetic circuits. IEEE Trans. Circuits Syst. I Regul. Pap. 66(1), 313–326 (2019).

[4] M. T. Khan and R. A. Shaik, "Optimal Complexity Architectures for Pipelined Distributed Arithmetic-Based LMS Adaptive Filter," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 2, pp. 630-642, Feb. 2019, doi: 10.1109/TCSI.2018.2867291.

[5] T.R. B, G. S. L and N. C K, "FPGA based Optimized LMS Adaptive Filter using Distributed Arithmetic," 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), 2018, pp. 1863-1867, doi: 10.1109/RTEICT42901.2018.9012288.

[6] Khan, Mohd. Tasleem and S. R. Ahamed. “A New High Performance VLSI Architecture for LMS Adaptive Filter Using Distributed Arithmetic.” 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (2017): 219-224.

[7] Prakash, M. S. and R. Shaik. “Low-Area and High-Throughput Architecture for an Adaptive Filter Using Distributed Arithmetic.” IEEE Transactions on Circuits and Systems II: Express Briefs 60 (2013): 781-785.

[8] S.Akhter, S. Kumar and D. Bareja, "Design and Analysis of Distributed Arithmetic based FIR Filter," 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), 2018, pp. 721-726, doi: 10.1109/ICACCCN.2018.8748322.

[9] M. Surya Prakash and R. Shaik, "High performance architecture for LMS based adaptive filter using distributed arithmetic", Proc. ICICA, vol. 24, pp. 18-22, 2012-Mar.

[10] B. K. Mohanty and P. K. Meher, "A High-Performance Energy-Efficient Architecture for FIR Adaptive Filter Based on New Distributed Arithmetic Formulation of Block LMS Algorithm," in IEEE Transactions on Signal Processing, vol. 61, no. 4, pp. 921-932, Feb.15, 2013, doi: 10.1109/TSP.2012.2226453.

[11] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, Jul. 2005, “LMS adaptive filters using distributed arithmetic for high throughput,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 52, no. 7, pp. 1327–1337

[12] P. K. Meher and S. Y. Park, "High-throughput pipelined realization of adaptive FIR filter based on distributed arithmetic", VLSI Symp. Tech. Dig., pp. 428-433, 2011-Oct.

[13] Mohd. Tasleem Khan, Jitendra Kumar, Shaik Rafi Ahamed, Juhi Faridi, "Partial-LUT Designs for Low-Complexity Realization of DA-Based BLMS Adaptive Filter", Circuits and Systems II: Express Briefs IEEE Transactions on, vol. 68, no. 4, pp. 1188-1192, 2021.

[14] Satpute, Nilesh S. et al. "LMS Algorithm and Distributed Arithmetic Based Adaptive FIR Filter with Low Area Complexity." (2015).

[15] Sangeetha and A. Kumar. "FPGA Implementation for Optimized Adaptive Filter Based on Distributed Arithmetic." (2016).

[16] K. Jebin Roy, R. Ramya, 2014, Adaptive FIR Filter based on Distributed Arithmetic and LMS Algorithm for Low-Area and Low-Power, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 03, Issue 02 (February 2014)

[17] Sang Yoon Park, June 2013, "Low power, High Throughput, and Low-Area Adaptive FIR Filter Based on Distributed Arithmetic,"IEEE Transactions on circuits and systems-II: Express Briefs,Vol.60

[18] Premananda B.S., Samarth S. Pai, Shashank B , Shashank S. Bhat "Design and Implementation of 8-Bit Vedic Multiplier" Vol. 2, Issue 12, December 2013.

[19] MT Khan, RA Shaik, SP Matcha "Improved convergent distributed arithmetic based low complexity pipelined least-mean-square filter" IET Circuits, Devices & Systems 12 (6), 792-801

[20] Prakash, S. and R. Shaik. "High Performance Architecture for LMS Based Adaptive Filter Using Distributed Arithmetic."

[21] Aishwarya C PG scholar and Mr. Vijaybhaskar R Assist. Prof. Anna University, RegionalCentre, Coimbatore, "Enhanced Pipelined Architecture for Adaptive FIR Filter Based on Distributed Arithmetic," International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:2682 Vol.23, No23, March 2014.

[22] R. Guo and L. S. DeBrunner, Sep. 2011, "Two high-performance adaptive filter implementation schemes using distributed arithmetic," IEEE Trans. Circuits Syst. II, Exp. Brief s, vol. 58, no. 9, pp. 600–604

[23] C.S. Vinitha, R.K. Sharma, "Area and Energy-efficient Approximate Distributive Arithmetic architecture for LMS Adaptive FIR Filter", Emerging Technology (INCET) 2020 International Conference for, pp. 1-5, 2020.

[24] Basant Kumar Mohanty, Sujit Kumar Patel, "Efficient very large-scale integration architecture for variable length block least mean square adaptive filter", Signal Processing IET, vol. 9, no. 8, pp. 605-610, 2015.

[25] Khan M.T., Ahamed S.R. (2017) VLSI Implementation of Throughput Efficient Distributed Arithmetic Based LMS Adaptive Filter. In: Kaushik B., Dasgupta S., Singh V. (eds) VLSI Design and Test. VDAT 2017. Communications in Computer and Information Science, vol 711. Springer, Singapore. https://doi.org/10.1007/978-981-10-7470-7_3

[26] C. S. Vinitha and R. K. Sharma, "Area and Energy-efficient Approximate Distributive Arithmetic architecture for LMS Adaptive FIR Filter," 2020 International Conference for Emerging Technology (INCET), 2020, pp. 1-5, doi: 10.1109/INCET49848.2020.9154125.

[27] Uma A., Kalpana P., Naveen Kumar T. (2018) Design of DA-Based FIR Filter Architectures Using LUT Reduction Techniques. In: Nath V. (eds) Proceedings of the International Conference on Microelectronics, Computing & Communication Systems. Lecture Notes in Electrical Engineering, vol 453. Springer, Singapore. https://doi.org/10.1007/978-981-10-5565-2_20

[28] A. Tiwari, P. Kumar, and M. Tiwari, "High throughput adaptive block FIR filter using distributed arithmetic," 2016 1st India International Conference on Information Processing (IICIP), 2016, pp. 1-6, doi: 10.1109/IICIP.2016.7975385.

[29] T. Pitchaiah, D. Lakshmi M and P. V. Sree Devi, "FPGA implementation of low area and delay efficient Adaptive Filter using Distributed Arithmetic," 2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014), 2014, pp. 1-5, doi: 10.1109/ICAETR.2014.7012922.