IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 3, 2021, accepted May 17, 2021, date of publication May 24, 2021, date of current version June 3, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3083282

A Novel Multiplier-Less LMS Adaptive Filter Design Based on Offset Binary Coded Distributed Arithmetic

SHAWEZ AHMAD, SAJID GUL KHAWAJA, NAEEM AMJAD, AND MUHAMMAD USMAN
College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Islamabad 46000, Pakistan
Corresponding author: Sajid Gul Khawaja (sajid.gul@ceme.nust.pk)

ABSTRACT Multiply-Accumulate (MAC) operation is the backbone of Least Mean Squares (LMS) digital adaptive filters. Implementing LMS on hardware platform as a Fully Dedicated Architecture (FDA) multiplier becomes bottleneck for higher order filters, prompting high area, cost and power requirements and hence renders the design unsuited for practical implementation. In this paper, we have proposed a composite design that makes use of Distributed Arithmetic (DA) to replace the bottleneck multiplier with memory units that store Partial Products (PPs) to emulate multiplication. The depth of these memory units tends to exponentially grow as the filter order rises. To manage that, we have used Half Memory algorithm (HM) and Offset Binary Coding (OBC) to refine the structure of PPs such that the memory size is reduced at least by a factor of 4 for the same filter order. The proposed design improves system's Throughput, Critical Path Delay, Power Consumption and FPGA Resource Utilization. However, it introduces Latency in both the output and update segments of the LMS algorithm. To provide an option between resource utilization and latency, we have suggested a mechanism to halve the originally produced latency by the Parallel Processing of input bit steam w.r.t even and odd bits. Moreover, we have also proposed a method that reduces the latency of update module at the slight expense of other design attributes. The fundamental structure of the proposed design is flexible owing to the dynamic memory structure as well as the option to choose between latency and resource minimization. Simulations have been carried out in Xilinx Vivado and conclusions have been drawn by comparing both FDA and DA based designs. Results for a 16-tap filter indicate a remarkable improvement in Throughput, Area Utilization and Power Consumption by 18%, 5% and 3.5% respectively at the expense of 4× escalated latency. The Half Latency method allowed the latency to drop 2× but with slightly elevated power and area attributes.

INDEX TERMS Adaptive filter, distributed arithmetic (DA), half memory algorithm (HM), least mean square (LMS), offset binary coding (OBC).

I. INTRODUCTION

Adaptive Finite Impulse Response (FIR) filters find many applications in DSP systems like system identification and equalization [1], noise-echo cancellation [2] and adaptive feedback cancellation [3] etc. Adaptive filters come in many forms and the choice of a particular type and order of filter is solely dependent upon the application under consideration. The working mechanism of an adaptive filter is composed of 2 modules, feedforward or FIR output module and feedback or weight-update module. For each input, the output module computes the FIR output of the filter and the update module calculates the new weights for the next input. In this way, the weights keep on updating until the error becomes zero, which means that the system has converged and the weights cease to update. The error term is the algebraic difference of output and desired signal. Output is the weighted sum of current and preceding input samples.

Typically, adaptive filters are a sub-part of larger systems e.g. communication systems such as Software-Defined Radios (SDR) [4], cognitive radios [5], biomedical systems [6] etc. These systems are gradually shifting towards re-configurable platforms such as Field-Programmable Gate Arrays (FPGAs) for hardware implementation [7], [8] due

The associate editor coordinating the review of this manuscript and approving it for publication was Felix Albu.

to the fact that FPGAs provide much faster floating point operations, sophisticated analog resources, hard memory controllers, ample amount of embedded resources, short time-to-market, high computational speed and flexibility [9].

While implementing such systems, and particularly adaptive filters, the hardware architecture can face area and timing constraints if designed using traditional Fully Dedicated Architecture (FDA). As the name suggests, FDA implies a one-to-one mapping of each arithmetic operation with a dedicated hardware unit. The required number of units is directly proportional to the order of the filter. For higher order filters, FDA implementation becomes unfeasible due to the limited number of multipliers in a DSP system. Moreover, the dedicated architecture demands higher area, budget and power requirements. This compels us to search for alternative designs for LMS adaptive filters. On hardware, multipliers are much more resource-hungry as compared to adders which motivates us to search for multiplier-less designs.

Distributed Arithmetic (DA) is a digital design algorithm that allows us to implement multiplication without actually using a physical multiplier. The basic mechanism of DA is to replace a multiplier by a Lookup-Table (LUT), which holds the pre-calculated Partial Products (PPs). In case of adaptive filters, the contents of LUTs are a function of tap weights, which we need to keep updating for each input. Hence, the contents of LUTs will be replaced by the newly calculated PPs. This can be a complicated task due to the limited access ports of memory as well as area and cost constraints associated with memory size. Therefore, we examine algorithms like Half Memory (HM), Offset Binary Coding (OBC) and Half Latency (HL) to compensate for these obstructions. In this article, we propose a novel design aiming to improve the area, cost, critical path delay, power consumption, resource utilization and throughput of an LMS adaptive filter at the expense of latency. Cost and area advantages of DA based multiplier-less design are compared in [10] with multiplier-based designs.

The rest of the paper is structured in the following manner: Section II discusses the relevant literature, Section III sheds light on the background knowledge required, Section IV elaborates the proposed designs, Section V presents the results based on simulations and Section VI concludes the paper.

II. LITERATURE REVIEW

The resource optimization advantage of DA intrigued the researchers and several multiplier-less LMS designs were proposed. In [11], Cowan *et al.* presented a conventional memory based DA architecture for output calculation. They used an accumulator based combinational approach for weight update process. In [12], Allred *et al.* adopted a memory based shift-accumulate approach to implement LMS algorithm and used an auxiliary LUT for the update process. They showed their design to have a high throughput when compared to the traditional architecture. The conventional DA based design allows a resolution of just a single cycle irrespective of filter order, hence a latency equal to the width of input bits B is introduced into the system. Tiwari *et al.* [13] used a block structure to compute the partial products in a single cycle by employing multiple memory units, allowing the latency to drop to a single cycle. Extending the approach of divided LUT method, Zhou and Shi [14] proposed a pipelined architecture by placing pipeline registers at adder and memory outputs. Their simulation results showed a high speed and low hardware requirement. Tasleem *et al.* [15] further improved the speed and resource consumption by using an adder tree to accumulate the results of all memory units. They used multiplexed memory units for both output and weight update operations of the filter. Updating the memory, however, is a computationally complex process as it consumes further clock cycles. Prakash and Shaik [16] proposed a parallel architecture for weight update mechanism by using a register-adder tree mechanism instead of a memory unit, thus requiring less memory resources and reducing the time consumed by update mechanism.

DA based designs show remarkable performance improvement for small number of filter taps. However, as the filter order grows, the memory size also starts to exponentially grow resulting in reduced efficiency. Researchers found out that the memory growth can be considerably reduced by encoding the input in a different manner. In [17], Prakash *et al.* used offset binary coding scheme to represent the binary input as ± 1. The resulting DA design was able to reduce the memory size by half with minimum resource overhead in the form of adders and combinational circuit. They used dedicated memories for output and update modules. In [18], Tasleem *et al.* kept the newest and discarded the oldest input samples to update the memory. They managed to reduce the memory size by $1.2\times$. A three-stage pipelined design was by proposed by Prabakaran and Yada [19] using offset binary coding with an adaptation delay of two. They used a CSA for partial product computation instead of the conventional shift-accumulate approach, thus reducing the delay and increasing the throughput. In contrast to offset binary based encoding, Vinitha and Sharma [20] proposed to use Canonic Signed Digit (CSD) encoding scheme to represent the input and hence managed to develop an LUT-less DA design. They used a Wallace tree adder to further reduce the complexity of partial product computation. In [21], Tsunekawa *et al.* proposed to use a 2's complement representation for input encoding, improving the convergence characteristics of the filter. Guo and DeBrunner [22] proposed an unconventional memory addressing scheme by using coefficients to access the memory instead of input samples and the memory contents were generated as a combination of input samples instead of coefficients. This allowed a parallel computation of output with the new weights thus reducing the latency. They managed to reduce the memory usage by $1.45\times$. They further improved their design [23] by reducing the memory usage by $2\times$ and eliminating excessive adders in the circuit. Takahashi *et al.* [24] proposed to split the memory into sub-memories using half memory algorithm with OBC based DA design to further reduce the memory size.

In general, three basic methods exist for practical implementation of LMS adaptive filters. The simplest one involves dedicated multipliers and adders for each MAC operation which is quite costly in terms of area, cost and critical path. The second method involves the replacement of bottleneck multipliers with memory unit(s) using simple DA with bit-serial approach. This method suffers from exponential memory growth for higher order filters. The third approach is to encode the binary inputs with 2's complement, CSD or the more common OBC scheme to reduce the memory size by half. Design choice also involves the LUT-less approach which replaces the memory units with multiplexers, hence more combinational logic is involved [25]. The major problem is that whenever we move towards the implementation of higher order filters, either latency or memory shows expensive results. Ideally, we want to find a balance between these two parameters.

In this paper, we present a basic scheme to improve the existing OBC based DA implementation of LMS filter. Our design is inspired by [24] and uses half memory algorithm to dynamically reduce the memory size. This means that the designer can control memory size by adjusting the number of memory units and access bits w.r.t filter taps. We derive the mathematical expressions for the proposed design in light of the LMS algorithm. We also present a variant to the basic scheme by proposing the parallel processing of input bits, thus reducing the latency to half at the expense of minimum resource overhead. Our design achieves at least 4× reduced memory size.

III. BACKGROUND

In order to understand the proposed architecture we need to lay some groundwork. This section briefly discusses the relevant algorithms that are associated with the proposed design.

A. LMS ALGORITHM

LMS algorithm was proposed by Widrow-Hoff in 1959. It employs a special case of gradient descent algorithm i.e. steepest descent to minimize the objective function. According to the LMS algorithm for adaptive filters, filter output is the weighted sum of the current and the previous input samples. Furthermore, the weight-update mechanism is based on the multiplication of computed error with the corresponding input samples. In simpler words, we say that MAC operation is the spine of LMS algorithm.

LMS algorithm is regulated by Equations 1, 2 and 3 for output y, error e and weight h computation respectively:

$$y[n] = \sum_{k=0}^{N-1} h_k \cdot x[n-k] \tag{1}$$

$$e[n] = d[n] - y[n] \tag{2}$$

$$h_k[n+1] = h_k[n] + (\mu \cdot e[n] \cdot x[n-k]) \tag{3}$$

where x represents the input signal, N represents the number of filter taps, n belongs to $0 \cdots N-1$, d denotes the desired/target output and μ represents the step-size or filter convergence coefficient. The characteristics and implications of the step size and delayed coefficient adaptation are studied in [26] with regards to the convergence speed and system stability. Equation 1 regulates the output of the filter by the accumulation of the products of filter weights with corresponding filter inputs. Equation 2 is used to calculate the error as the difference between the current and desired outputs. Equation 3 shows the weight-update process which is a function of error and the current and previous inputs. The resulting terms are moved by the step-size and added to the corresponding previous weights to acquire the updated weights.

Algorithm 1 LMS Algorithm

```
Initialize N, d and μ
k ← 1 to N
h_k ← random
while 1 do
    n ← current iteration
    Shift x[n − k] to x[n − k + 1]        for all k
    x[n] ← Input
    y[n] ← h_k[n] × x[n − k]              for all k
    e[n] ← y[n] − d
    h_k[n + 1] ← h_k[n] + (μ × e[n] × x[n − k])
                                          for all k
    y[n] → Output
end while
```

The pseudo code for LMS algorithm is as follows:

B. FULLY DEDICATED ARCHITECTURE

On hardware, when we intend to implement an algorithm whose equations are well defined, the basic human instinct is to assign a computational unit to each corresponding operation. This one-to-one mapping of arithmetic operations on hardware is called Fully Dedicated Architecture. For LMS adaptive filter, the Equations 1, 2 and 3 define the algorithm. These equations clearly show that the filter can be implemented by simple MAC operations. Since FDA prompts a single computational unit against a single operator, all the computations are performed in a single cycle. For instance, if we are required to implement a 4-tap filter (N = 4), we need four multipliers and three adders for output calculation, one subtractor for error computation and five multipliers and four adders for calculation of new weights. The output is generated in the same cycle as the input is provided.

Figure 1 shows the generic block diagram for FDA implementation of LMS filter. There are two main modules for output calculation and weight adaptation respectively and a smaller unit for error computation. Figure 1 also shows the flow of the algorithm, which is directed from output calculation to error computation to weight adaptation. Figure 2 shows the circuit diagram of each module, in accordance with LMS equations. Figure 2 (a) shows the implementation

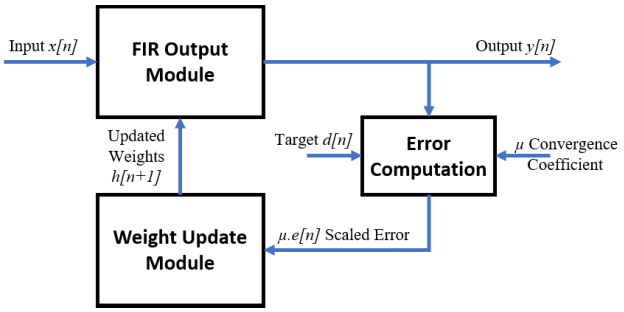


FIGURE 1. Generic block diagram of fully dedicated architecture for LMS adaptive filter.

of output module for a 4-tap filter, which includes dedicated multipliers and adders along-with registers to store previous inputs. Figure 2 (b) shows the error computation unit, which is always the same regardless of filter order. Finally, Figure 2 (c) shows the weight update module, which also includes dedicated multiplication and addition units.

The main advantage of FDA is the availability of output within a single clock cycle i.e. no output latency irrespective of the filter order. Moreover, the design is fairly simple and straight-forward to implement. However, the higher the filter order gets, the more the number of computational units required and the more the consumption of area and power. Moreover, an FPGA/DSP system has a limited number of resources available, hence we become short of adders and multipliers. Another drawback of FDA implementation is the rise in Critical Path Delay (CPD) of the system with a rise in filter order. Therefore, FDA implementation of LMS adaptive filter is not feasible for real-time systems, particularly for higher order filters.

C. DISTRIBUTED ARITHMETIC ALGORITHM

In FDA design, multiplier is the most resource-hungry computational unit owing to its complex design. As multiplier is the bottleneck of FDA, we seek an alternative to it and hence shift towards a multiplier-less design. Distributed Arithmetic (DA) makes use of the fact that multiplication is simply the shifted accumulation of Partial Products (PPs). Hence, it proposes the multiplier to be replaced by a memory/look-up-table (LUT) which holds all the possible Partial Products (PPs). In each cycle, one of these PPs is retrieved from the LUT based on a single bit of input and fed to a shift accumulator. In this manner, all input bits are processed one-by-one to finally generate the multiplication result.

In case of digital filters, each multiplier is associated with a single weight and a single current/previous input, with a total of N multipliers being used for output computation. Hence, to adjust all multiplication units within a single LUT, the PPs are stored as a linear combination of the filter weights. This prompts to use one bit from each of the current and the $N-1$ previous inputs to access the LUT. The output of LUT is fed to a shift accumulator. This process is repeated for all bits of the input one by one and hence an output latency equivalent to the number of input bits is introduced into the system. This delivers us the FIR output of the filter.

Similarly, multipliers of the update module are also replaced with an LUT. However, in this case the PPs are stored as a combination of input samples instead of tap weights. Equation 3 dictates how these PPs are generated. The update module introduces a latency directly proportional to the number of filter taps. From Equation 2, it is obvious that the update module does not start working until the output is generated from the output module. Therefore, the total latency of the system is equivalent to the sum of individual latency of both modules.

The memory is finite and its depth is determined by:

$$Memory\_Depth\_DA = 2^N \tag{4}$$

For instance, if we intend to implement a 4-tap filter, the memory-depth would be $2^4 = 16$ and a four bits wide address would be required to access this memory.

Equation 4 shows that the memory depth is exponentially proportional to the filter order N. Higher filter order also induces a higher latency into the update module. Therefore, high latency and exponential memory growth are the major concerning points of standalone DA based filters as they take a toll on system's area, resources and latency despite improving the CPD. This pushes us to search for other solutions to address these drawbacks.

D. HALF MEMORY BASED DA ALGORITHM

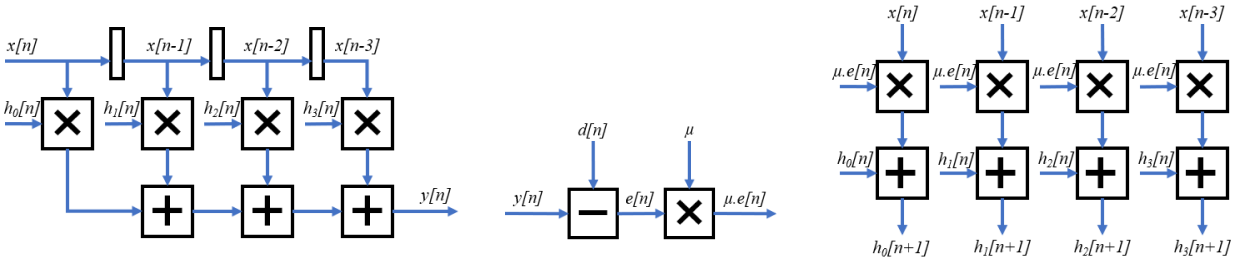
To address the exponential memory growth of DA based design, we examine the Half Memory (HM) algorithm, which states that the contents of a single memory can be divided among 2 or more sub-memories, the combined size of whose is less than the original. Each sub-memory is accessed by its own unique address. During a single iteration, all sub-memories are addressed once and their results are accumulated such that they return the same result as that of original memory. Hence at the expense of more adders, we can reduce the memory size at least by half.

The name half memory is ambiguous as it suggests the division of original memory into just two sub-memories. Rather, the HM-DA method allows us to divide the memory into multiple sub-memories. This is regulated by Equation 5:

$$R = N/M \tag{5}$$

where M represents the number of desired memory units and R is the number of address bits required per memory.

For a 4-tap filter with $M = 2$ and hence $R = 2$, we require two sub-memories with four locations each. Both memories are accessed by a 2-bit address. However, if we were to change M to four, we would require four sub-memories with two locations each. All memories would require a single address bit for access.



(a) FIR Output Module (b) Error Computation (c) Weight Update Module

FIGURE 2. Module-level circuit diagram of fully dedicated architecture for a 4-tap LMS adaptive filter.

E. OFFSET BINARY BASED DA ALGORITHM

In order to further improve the exponential memory growth of DA, we look into another algorithm by the name of Offset Binary Coding (OBC) scheme. Similar to 2's complement representation of a binary number, this method also proposes a new representation of binary numbers. OBC is regulated by the following function:

$$OBC = \begin{cases} +1, \text{ when 2's complement representation is 1} \\ -1, \text{ when 2's complement representation is 0} \end{cases}$$

This means that all ones in a bit stream are represented as $+1$, while all zeros are represented as -1. Following this concept, the PPs stored in the LUT are modified in such a way that the first half of them become the exact negative of the remaining second half. This allows us to drop one of these halves, thus reducing the memory by two and the access bits by one. The dropped PPs are still accessible to us as the dropped access bit allows us to determine if to directly pass the memory contents or take their negative first. Hence, we are able to fully execute the DA algorithm while only keeping half of its original memory size.

The difference between HM-DA and OBC-DA is that the latter does not provide a mechanism to split the memory into sub-memories. Nor does it allow depletion of memory by a factor of more than 2. HM-DA is dynamic in this regard.

IV. PROPOSED ARCHITECTURES

In this paper, we combine the techniques studied in the previous section to propose efficient real-time implementation of LMS adaptive filter. We propose 2 different variants to our scheme; Offset Binary Coding & Half Memory based Distributed Arithmetic algorithm (OBC-HM-DA, occasionally referred to as Type-I) which is oriented towards the minimization of resource utilization. And Offset Binary Coding & Half Memory with Half Latency based Distributed Arithmetic algorithm (OBC-HM-HL-DA, occasionally referred to as Type-II) which is focused more towards the latency reduction. For convenience, first we will discuss the output module for both methods and then the update module separately.

A. TYPE I: OBC-HM-DA DESIGN

Having examined the details of prerequisite mechanisms, naturally the first instinct is to merge HM and OBC algorithms into the DA based design to minimize resource utilization and improve design and timing attributes. For this purpose, we first derive the equations for DA based LMS algorithm. Applying a bit-serial approach to input sample $x[n]$, it becomes:

$$x[n] = -x_{B-1}[n] + \sum_{b=0}^{B-2} x_b[n] \cdot 2^{b-(B-1)} \tag{6}$$

Then applying sample-wise operation to h_k, Equation 1 becomes:

$$y[n] = \sum_{k=0}^{N-1} h_k \Bigg[-x_{B-1}[n-k] + \sum_{b=0}^{B-2} x_b[n-k] \cdot 2^{b-(B-1)} \Bigg] \tag{7}$$

Equation 2 remains the same while Equation 3 takes the following form:

$$h_k[n+1] = h_k[n] + \mu \cdot e[n] \cdot \Bigg[-x_{B-1}[n-k] + \sum_{b=1}^{B-2} x_b[n-k] \cdot 2^{b-(B-1)} \Bigg] \tag{8}$$

where B is the number of input bits and x_{B-1} is the MSB of the input. Equations 7 and 8 represent the output and update methods respectively for DA based LMS algorithm.

Equation 7 is used to derive the PPs to be stored in memory for DA based filter. The depth of this memory is defined by Equation 4. For a 4-tap filter, a memory with sixteen locations is required. Each location is accessed by a 4-bit wide binary address. Table 1 shows the contents against each address for the aforementioned filter. The left column in the table shows the bit-serial addressing scheme such that a_0 is the LSB. The table demonstrates that the PPs are a linear combination of filter weights. Equation 8 regulates the weight-update mechanism of the adaptive filter, which will be discussed later in detail.

TABLE 1. Memory contents of a 4-tap DA based LMS adaptive filter.

$a_3a_2a_1a_0$	Contents
0 0 0 0	0
0 0 0 1	h_0
0 0 1 0	h_1
0 0 1 1	$h_1 + h_0$
0 1 0 0	h_2
0 1 0 1	$h_2 + h_0$
0 1 1 0	$h_2 + h_1$
0 1 1 1	$h_2 + h_1 + h_0$
1 0 0 0	h_3
1 0 0 1	$h_3 + h_0$
1 0 1 0	$h_3 + h_1$
1 0 1 1	$h_3 + h_1 + h_0$
1 1 0 0	$h_3 + h_2$
1 1 0 1	$h_3 + h_2 + h_0$
1 1 1 0	$h_3 + h_2 + h_1$
1 1 1 1	$h_3 + h_2 + h_1 + h_0$

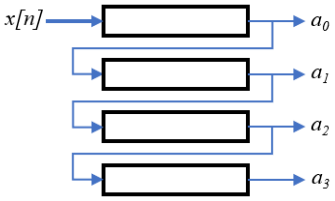


FIGURE 3. Circuit diagram of shift-register bank for distributed arithmetic based LMS adaptive filter @ $N, B = 4$.

In order to access the memory, single bits from each of the current and previous inputs are merged together to formulate the address. To accommodate the bit-serial processing of $x[n-k]$, they are stored in unique shift-registers, such that the first register is directly connected to the input and each register is connected to its following register. This hierarchy of shift-registers is shown in Figure 3 for a 4-bit, 4-tap adaptive filter. The LSB of each register (a_N) is used to access the memory. The bit corresponding to the original input (a_0) is regarded as the LSB of the access bits.

In every clock cycle, a PP is retrieved from the memory using the access bits and fed to the shift-accumulator and then the shift-registers are bit-serially shifted once to the right. The reason for using a shift-accumulator corresponds to the multiplication algorithm itself, which implies that every next PP is shifted once w.r.t its preceding PP. Once all the bits in a shift-register are processed, the result of the shift-accumulator is stored as the output $y[n]$ of the filter. The structure of the shift-accumulator is shown in Figure 4. The term *initial_value* is associated with the PP stored at the first location in the memory.

The next step is to integrate the HM algorithm into DA. HM algorithm does not entirely alter the shape of DA architecture, rather it splits up a larger singular memory into multiple smaller memories in a manner that reduces the overall size of the memory. Hence, the DA equations need to be slightly modified to adjust the splitting element in HM-DA. Equations 7 and 8 are thus

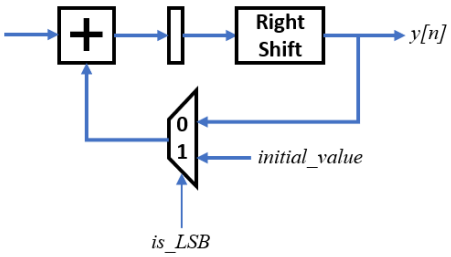


FIGURE 4. Circuit diagram of shift-accumulator for distributed arithmetic based LMS adaptive filter.

modified as:

$$y_m[n] = \sum_{k=0}^{N-1} h_{k_m}\Big[-x_{B-1_m}[n-k] + \sum_{b=0}^{B-2} x_{b_m}[n-k]\cdot 2^{b-(B-1)}\Big] \tag{9}$$

$$h_{k_m}[n+1] = h_{k_m}[n] + \mu\cdot e[n]\cdot\Big[-x_{B-1_m}[n-k] + \sum_{b=1}^{B-2} x_{b_m}[n-k]\cdot 2^{b-(B-1)}\Big] \tag{10}$$

where $m = 0, 1, \ldots M-1$. The PPs are now generated w.r.t Equation 10 and the weights are updated using Equation 10. The memory depth is again defined by Equation 4, however, the perception of N is changed to N_m. So, Equation 4 takes the following form:

$$Memory\_Depth\_\text{HM-DA} = 2^{N_m} \tag{11}$$

Equation 10 shows that in HM-DA, each memory is associated with a unique set of tap-weights. For instance, for a filter with $N = 4$ and $M = 2$, we use two memory units such that the first two taps are associated with the first memory and the remaining two with the second memory. Hence, $N_m = 2$ is used in Equation 11 for both memories. This renders the depth of each memory to be $2^2 = 4$. Therefore, the total memory locations are reduced from 16 to 8 by merging the HM algorithm with the DA based design.

The memory access method does not require any additional resources. The only modification made is that the shift-register bank is divided into the same number of sub-banks as the sub-memories. In this way, a unique set of shift-registers is used to access each memory. The organization of shift-registers is exactly the same as shown in Figure 3. The contents from each memory are then accumulated which generates the complete PP that was required. This PP is then fed to the shift-accumulator to generate the output.

Finally, we incorporate the OBC algorithm into HM-DA, formulating the OBC-HM-DA design. To represent 0 as -1, 2's complement of $x[n]$ is shown below:

$$-x[n] = \overline{x[n]} + 1$$

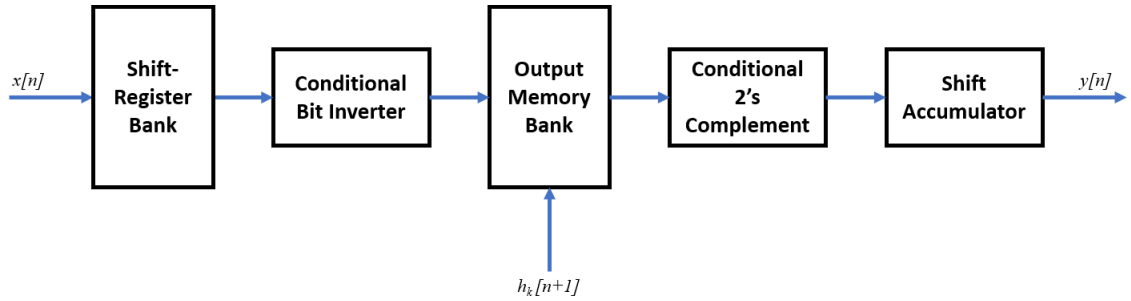


FIGURE 5. Generic block diagram of the output module of the proposed architecture for LMS Adaptive Filter.

Taking 2's complement of Equation 6, it becomes:

$$-x[n] = -\overline{x_{B-1}[n]} + \left[\sum_{b=0}^{B-2} \overline{x_b[n]} \cdot 2^{b-(B-1)}\right] + 1 \quad (12)$$

Now according to the OBC scheme:

$$x = \frac{1}{2}\left[x_i - (-x_i)\right] \quad (13)$$

where $x_i - \overline{x_i} = OBC_i$. Putting Equation 6 and 12 in Equation 13:

$$x[n] = \frac{1}{2}\Bigg[\Bigg\{-x_{B-1}[n] + \sum_{b=0}^{B-2} x_b[n] \cdot 2^{b-(B-1)}\Bigg\} + \Bigg\{-\overline{x_{B-1}[n]} + \sum_{b=0}^{B-2} \overline{x_b[n]} \cdot 2^{b-(B-1)} + 1\Bigg\}\Bigg] \quad (14)$$

Simplifying Equation 14:

$$x[n] = \frac{1}{2}\Bigg[-x_{B-1}[n] - \overline{x_{B-1}[n]} + \sum_{b=0}^{B-2}(x_b[n] - \overline{x_b[n]}) \cdot 2^{b-(B-1)} - 1\Bigg] \quad (15)$$

Using notation $x_i - \overline{x_i} = OBC_i$ in Equation 15:

$$x[n] = \frac{1}{2}\Bigg[-OBC_{B-1} + \left(\sum_{b=0}^{B-2} OBC_b \cdot 2^{b-(B-1)}\right) - 1\Bigg] \quad (16)$$

Now placing $x[n]$ found in Equation 16 into Equations 9 and 10, we get the output and update equations as follows:

$$y_m[n] = \frac{1}{2}\sum_{k=0}^{N-1} h_{k_m}[n]\Bigg[-OBC_{B-1_m} + \left(\sum_{b=0}^{B-2} OBC_{b_m} * 2^{b-(B-1)}\right) - initial\_value\Bigg] \quad (17)$$

$$h_{k_m}[n+1] = h_{k_m}[n] + \mu \cdot e[n] \cdot \frac{1}{2}\Bigg[-OBC_{B-1_m} + \left(\sum_{b=0}^{B-2} OBC_{b_m} * 2^{b-(B-1)}\right) - 1\Bigg] \quad (18)$$

TABLE 2. Contents of memory1 for a 4-tap OBC-HM-DA LMS adaptive filter.

$\mathbf{a}_0$	Contents
0	$-\frac{h_1+h_0}{2}$
1	$-\frac{h_1-h_0}{2}$

TABLE 3. Contents of memory2 for a 4-tap OBC-HM-DA LMS adaptive filter.

$\mathbf{a}_0$	Contents
0	$-\frac{h_3+h_2}{2}$
1	$-\frac{h_3-h_2}{2}$

where *initial_value* is associated with the sum of the contents stored at the first location in the all the memory units placed in the design.

The PPs are generated w.r.t Equation 17 and the memory depth is now defined by:

$$Memory\_Depth = (2^{N_m})/2 \quad (19)$$

Memory contents for a Type-I filter design with $N = 4$ and $M = 2$ are shown in Tables 2 and 3. The memories have a total of 2 locations each, which is justified by Equation 19. The analysis of the contents of these memories show that the PPs are still a linear combination of filter weights with an additional division term which is justified by Equation 17. It is important to note that this division is by a constant factor, which does not require an actual divider on hardware, rather a simple right shift does the job. When we compare the simple DA based memory contents shown in Table 1 with Type-I based contents shown in Tables 2 and 3, we observe that the latter has reduced the memory locations from 16 to a total of just 4, i.e a remarkable 4× reduced memory space. Moreover, the width of the memory access bits is also reduced from 4 to just 1.

Figure 5 shows the generic block diagram for the output module of Type-I design. The input $x[n]$ is fed to the shift-register bank, whose architecture is shown in Figure 3. It is important to note that the MSB of input must always be

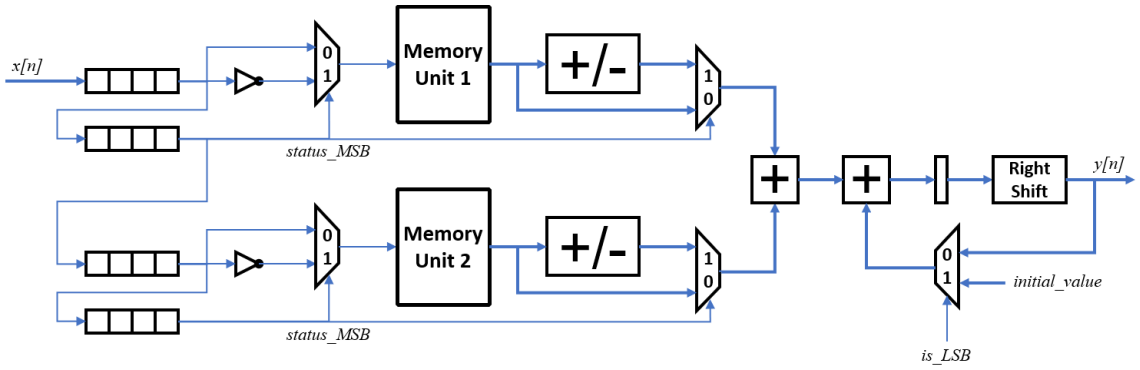


FIGURE 6. Circuit diagram of OBC-HM based distributed arithmetic architecture for a 4-tap LMS adaptive filter.

TABLE 4. No. of memory units vs memory locations.

M	Locations
1	8
2	4
4	4

inverted, as justified by the negative sign in Equation 17. The number of registers in the bank is equal to the number of filter taps N. The width of each register is equal to the number of input bits B. The LSBs of these registers are used to access the memory units. But they are first fed to a conditional bit inverter, which decides to either invert the address or not, in order to cater for the dropped PPs as explained in the background section. The output of the bit inverter unit is used to access the memory units. The contents retrieved from the memory are fed to a conditional 2's complement unit, which is associated with the conditional bit inverter block. And finally, the contents are fed to the shift-accumulator to generate the output $y[n]$. The structure of shift-accumulator is shown in Figure 4.

In order to better understand the Type-I design, we intend to explain it with an example. Consider a 4-bit, 4-tap ($B = 4$, $N = 4$) filter as shown in Figure 6, which presents the circuit diagram of the filter based on Type-I architecture. The following steps show the hardware setup of the output module:

- We first select a suitable number of sub-memories. The choice of the number of memories is regulated by B in powers of 2. Table 4 shows that a single memory $M = 1$ is not be appropriate as the memory size becomes 16 and could still be reduced. If we use four memories $M = 4$, the memory becomes redundant as the same size can be achieved using lesser units. Hence, it is suitable to use two memories $M = 2$, which provides the minimum number of locations in minimum number of units.
- Based on the values of N and B, a total of 4 shift-registers of 4 bits each are required. Based on the value of M, these registers are grouped into two banks of two registers each, as shown in Figure 6.
- In a shift-register bank, the output of first register is used to access the corresponding memory unit while the output of the second register *status_MSB* acts as a flag to activate the conditional bit inverter and conditional 2's complement units shown in Figure 5.
- Tables 2 and 3 show the contents of Memory 1 and Memory 2 respectively.
- *status_MSB* flag is fed as a control signal to the inverter placed against the address bit.
- Memory output is fed to a 2's complement unit which is also controlled by *status_MSB*.
- For $M = 2$, a single adder is required to accumulate the contents of both memories, to form the desired PP, which is then fed to the shift-accumulator.
- The final output is received after B number of cycles, which is 4 in this case.

To summarize, OBC-HM-DA replaces the dedicated multipliers and adders of FDA with memory unit(s) and shift-accumulator respectively, considerably reducing system resource utilization. The reduced number of components in the critical path allows the design to work at a much higher frequency. The memory size is reduced from 2^N to $M \times 2^{R-1}$ when compared to traditional DA based design, with no difference in latency. On the other hand, a latency of B clock cycles is introduced when compared to the FDA implementation.

B. TYPE II: OBC-HM-HL-DA DESIGN

The main purpose of Type-I architecture is to improve the filter design and timing attributes such as CPD, power consumption, throughput and area utilization. However, doing so introduces a certain latency into the system. In real-time situations one may require a design that is not as efficient as Type-I design but consumes lesser clock cycles to produce the result. We can say that such a design will be a trade-off between FDA and OBC-HM-DA architectures. In light of this argument, we propose a Type-II design which aims to halve the latency of the Type-I system with slightly reduced efficiency. For this purpose, we propose parallel processing of even and odd bits of input samples which automatically reduces the latency to half. It is important to note that processing more than two bits simultaneously results in excessive resource utilization

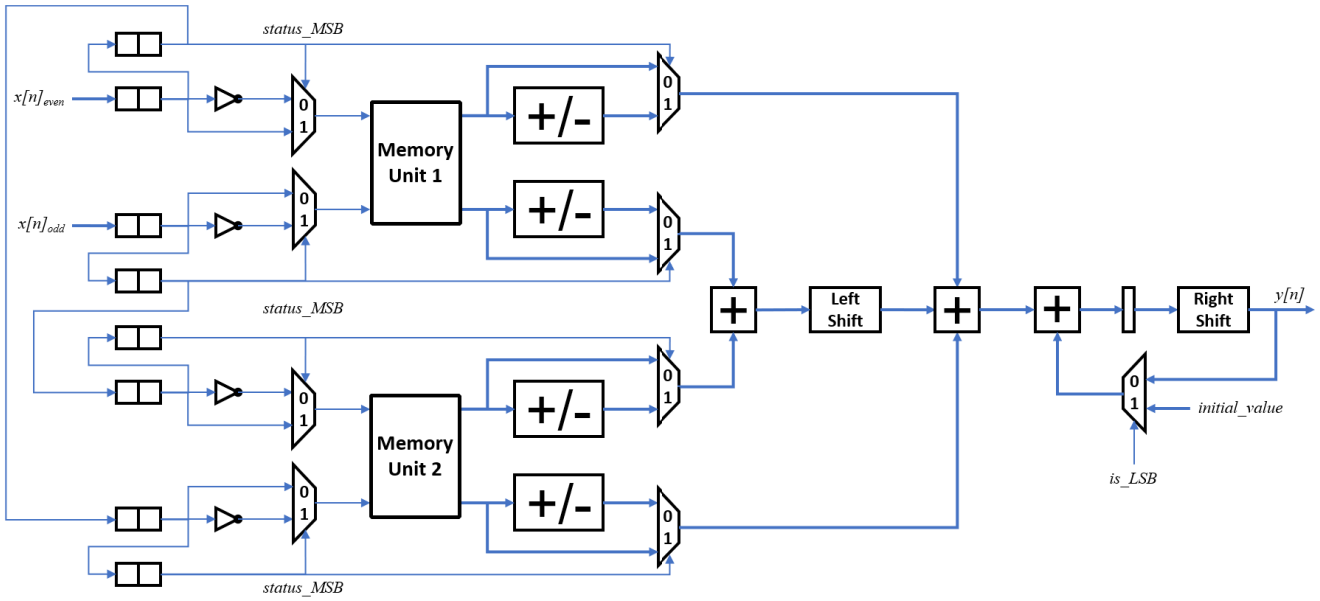


FIGURE 7. Circuit diagram of OBC-HM-HL based distributed arithmetic architecture for a 4-tap LMS adaptive filter.

as opposed to even/odd parallel processing which requires minimum additional resources.

The basic functionality of the system remains the same. Figure 7 shows the block diagram of the output module. Since N and B remain the same, the overall space allotted to the shift-registers also remains the same. However, the configuration of shift-registers is changed as shown in Figure 7. Dual-access memory is used to retrieve PPs simultaneously for both even and odd bits. Hence, a single memory unit produces two unique outputs in a single clock cycle, one w.r.t the even bit and the other w.r.t the odd bit. The bit inversion and 2's complement setup is exactly the same as Type-I. Additional adders are required to accumulate both the inter-memory and intra-memory contents to formulate the desired PP. It is important point to note that in traditional multiplication algorithm, the odd bit (LSB) has lesser weight 2^0 as compared to the even bit 2^1. Therefore, to accommodate parallel processing, a shift operation is performed against the even bits to adjust its weight. Finally, a shift-accumulator is placed to perform the addition operation of the filter. Subsequently, the shift operation inside the shift-accumulator is applied by a factor of 2 to adjust the parallel bits.

To summarize, OBC-HM-HL-DA is a modification to the OBC-HM-DA design which aims to reduce the latency B of the former to $B/2$. This is done by the parallel processing of input bits at the cost of minimum resource overhead. It is important to note that even with the slightly increased resource utilization, Type-II design is still more efficient when compared to the traditional FDA design.

C. WEIGHT UPDATE MECHANISM

The preceding section mathematically establishes the basic algorithm of the proposed architecture. Considering the same

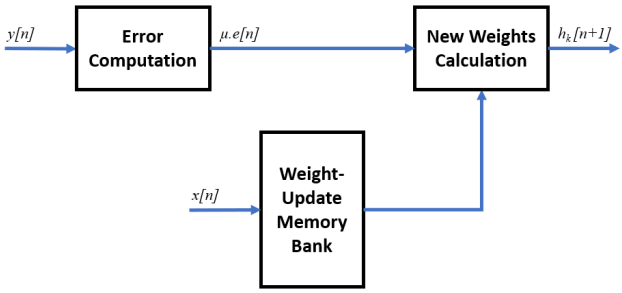


FIGURE 8. Block diagram of the weight update module of the proposed architecture.

scheme, we propose the weight update module of the adaptive filter. According to the LMS algorithm, new weights are calculated after an output is produced as shown in Figure 1. The error computation mechanism (Equation 2) is based on a single MAC operation, so we propose to keep its dedicated design. Similarly, the designer can also choose to retain the dedicated architecture for weight update module, thus keeping the overall latency to just B cycles. Alternatively, we can modify the weight update module (Equation 3) to eliminate the N dedicated multipliers via DA algorithm as shown in Figure 8.

The mathematical application of the proposed architecture on weight update module yields Equation 18. An important point to note is that the left side of Equation 18 corresponds to a singular weight. However, the output module requires the PPs to be saved as a linear combination of filter weights. This means that additional hardware will be required. To avoid this trouble, we modify Equation 18 as follows:

$$g_{k_m}[n+1] = g_{k_m}[n] + \left[\mu \cdot e[n] \cdot \frac{1}{2} \cdot \mathbf{A} \cdot \mathbf{x}\right] \tag{20}$$

TABLE 5. Contents of memory 1 of the update module of a 4-tap OBC-HM-DA adaptive filter.

$\mathbf{a}_0$	Contents
0	$x[n]$
1	$x[n-1]$

TABLE 6. Contents of memory 2 of the update module of a 4-tap OBC-HM-DA adaptive filter.

$\mathbf{a}_0$	Contents
0	$x[n-2]$
1	$x[n-3]$

where g represents the linear combination of weights instead of a single weight and,

$$\mathbf{A} = \begin{bmatrix} -1 & -1 \\ +1 & -1 \\ -1 & +1 \\ +1 & +1 \end{bmatrix}$$

and

$$\mathbf{x} = \begin{bmatrix} x[n] \\ x[n-1] \end{bmatrix}$$

Tables 5 and 6 show the memory contents of the update module for a filter with $N = 4$, $B = 4$ and $M = 2$. Table 5 is associated with Memory 1 of output Module and hence deals with the first 2 weights only. Similarly, Table 6 deals with the last 2 weights only and is used to update Memory 2 of the output module. $\mathbf{A}$ and $\mathbf{x}$ are derived from Equation 11 and they regulate the accessing scheme of the update module memories.

V. RESULTS AND DISCUSSION

In this section, we compare and discuss the simulation results of proposed designs with FDA implementation. We review their performance in terms of area utilization, power consumption, timing and throughput.

A. FILTER SPECIFICATIONS AND IMPLEMENTATION DETAILS

All designs were implemented on Virtex-7 FPGA board using Xilinx Vivado Design Suite 2018. Simulation results and timing diagrams were generated along with the synthesis reports. Vivado synthesis tool has the ability to transform the coded RTL into gate-level representation and generate analysis reports like timing analysis, resource utilization and power consumption. This section compares the area, critical path delay, GMACs, latency, power consumption, resource utilization and throughput of the designs.

Three different implementations were carried out for filter lengths of 4, 8 and 16 for FDA, Type-I and Type-II designs. The convergence coefficient μ was fixed at 0.5 and the desired/target output was set to be -1. Moreover, all inputs and tap-weights were 4-bits wide while the output was 8-bits wide. Random test inputs were applied to generate the outputs. All inputs and filter weights were represented in $Q_{1.3}$ fixed point format. Initial weights were set as shown in Figure 9.

B. OUTPUT AND TIMING DIAGRAM

Simulation results for four clock cycles are shown in Figure 9 for verification. Observe that the output of the proposed designs is the same as that of the FDA design, hence the output module of the proposed designs are verified. Figure 9 also shows the updated values of weights in each cycle, which also emerge to be the same for all methods, hence the update module is also verified. This establishes that the proposed designs follow the working principle of the LMS algorithm to the full extent.

Another important design aspect is the occurrence of output w.r.t clock. Figure 10 shows the timing diagram for all 3 designs. The clock starts from 0*ns*, the red bar represents a waiting state while the black bar represents the output. The crossover shows the occurrence of a new event/output. According to the Timing Diagram, system is initially in a waiting state, which does not necessarily mean that the system is waiting for an input, rather it indicates the processing phase. The first FDA output appears in the second clock cycle and preserved until next output has been calculated. All outputs appear in the same fashion. The reason is that in the first cycle, the system is processing the first input and output is available in that cycle, hence the red bar. The FDA mechanism allows all the processing in a single cycle, therefore, the output for first input is readily available in the second cycle and therefore, a latency of 1 clock cycle. Similarly, the second input is processed in the second cycle and its result is displayed in the next cycle and so on. The three finite black bars show the outputs w.r.t the first three inputs. The last output is preserved for an indefinite period of time as just four inputs are used for verification and the system becomes stable at the last output.

Now for the Type-I architecture, a single input bit is processed in a single cycle instead of the whole bit stream, hence the first output appears after four cycles (since 4-bit input). For the Type-II half latency based architecture, the name is suggestive enough to tell us that the overall latency of the system will be reduced to half of that of Type-I design which becomes 2 clock cycles in this case.

C. TIMING AND DESIGN ANALYSIS

The graph of Figure 11 shows the throughput comparison for all designs. A clear trend in the graph shows that throughput for all filters of same taps displays the highest peak at Type-I and the lowest peak at FDA. Thus, Type-I provides the best throughput results for any filter order while Type-II, though not as efficient as the former, still shows a better throughput performance than FDA.

Table 7 shows the timing and design attributes including CPD, latency and the number of logic levels for the 16-tap filter. Analysis of this table shows that Type-I design provides the shortest CPD but the highest latency, while on the other

x[n]	h0	h1	h2	h3	y (FDA)	y (Type-I)	y (Type-II)
0.875	0.125	-0.75	0.5	-0.875	0.125	0.125	0.125
-1	-0.375	-0.75	0.5	-0.875	-0.25	-0.25	-0.25
0	-0.75	-1	0.5	-0.875	0.875	0.875	0.875
0.5	-0.75	-0.5	0.125	-0.875	-1	-1	-1

FIGURE 9. Simulation results of the 4-tap filter for FDA, OBC-HM-DA & OBC-HM-HL-DA designs.

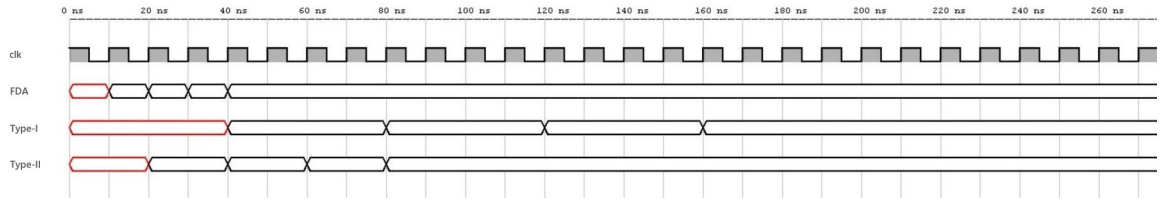


FIGURE 10. Clock based timing diagram of the 4-tap filter for all 3 designs.

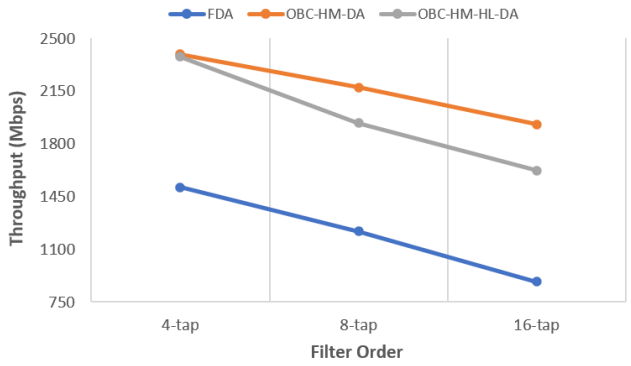


FIGURE 11. Throughput comparison of all designs for filters of length 4, 8 and 16.

TABLE 7. Timing and design attributes of 16-tap filter.

Design	FDA	Type-I	Type-II
CPD (ns)	9.044	4.148	4.924
Latency	1	4	2
Logic Levels	11	7	7

hand, FDA is the exact opposite. Meanwhile, Type-II acts as a compromise between the two with attributes that lie in between. Also, the proposed designs provide 1.57× lower logic delay in terms of logic levels of the design. Hence we conclude that the proposed designs provide an efficient timing-latency trade-off.

D. POWER CONSUMPTION

Power Consumption is an important performance metric in the evaluation of a digital design. It directly affects the cost, energy and temperature of the device. Hence, the lower the power consumption, the higher the system stability and the

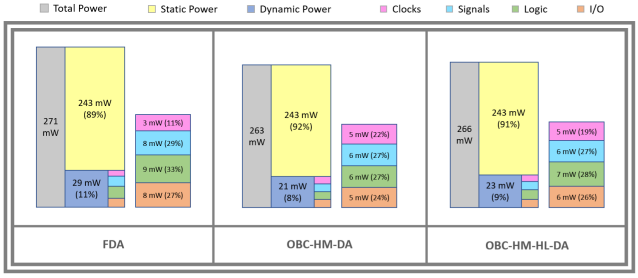


FIGURE 12. A breakdown of power consumption for 16-tap filter.

lower the cost of the system. Basically, there are two types of power linked to a digital system; dynamic power which is associated with the switching activity of the components, and static power which appears due to the leakage current of the components. In terms of a digital system, dynamic power is further categorized into consumption w.r.t clocks, signals, logic and I/O. Figure 12 reports a breakdown of these powers w.r.t each design for the 16-tap filter.

Figure 13 shows a comparison of the total power consumed by all 3 designs for 4, 8 and 16-tap implementations. The general trend shows that FDA consumes the most power for any filter length, whereas Type-I consumes the least. Note that the slope of FDA curve considerably rises as we go from 4-tap to 16-tap, whereas, Type-I design shows just a marginal rise in slope. This means that as the filter order rises, the power consumption of FDA increases rather exponentially while that of Type-I and Type-II remains fairly linear.

E. RESOURCE UTILIZATION

The foundation of this research was laid on the fact that multipliers are limited in number in an FPGA/DSP system. Hence minimizing resource utilization was a driving factor behind our research. Figure 14 gives an account of overall

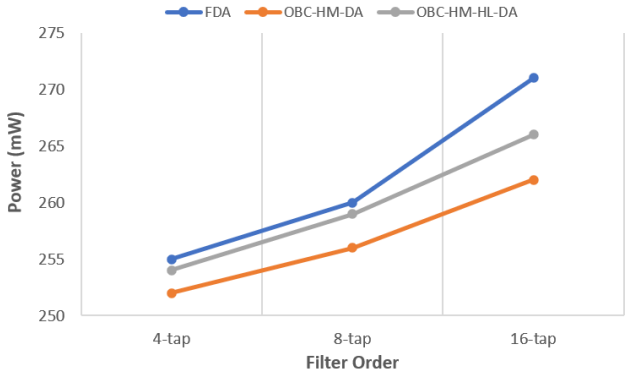


FIGURE 13. Comparison of total power consumed by each design for 4, 8 and 16-tap implementations.

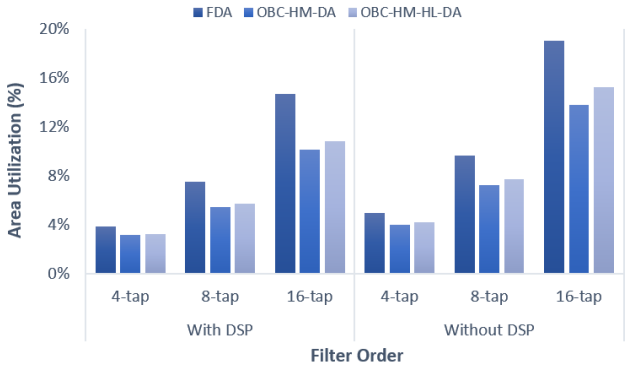


FIGURE 14. Area utilization of all designs for filters of 4, 8 and 16 taps.

TABLE 8. Number of resources utilized for 16-tap implementation.

	With DSP			Without DSP		
Design	**FDA**	**Type-I**	**Type-II**	**FDA**	**Type-I**	**Type-II**
Slice Registers	256	228	226	256	249	247
Slice LUTs	513	283	330	799	448	530
LUT-FF pairs	102	76	75	97	131	139
Bonded IOBs	14	16	16	14	16	16

resource utilization of all designs, according to which the proposed designs require lesser area as compared to FDA. Another trend deduced from the bar graph is that the utilization efficiency of the proposed designs becomes more and more significant as the filter order rises.

A breakdown of resource utilization of 16-tap filter is shown in Table 8. Information regarding the number of slice registers, number of slice LUTs, number of fully utilized LUT-FF pairs and number of bonded IOBs is displayed in this table, regarding both with DSP and without DSP. These stats establish Type-I to have the lowest area requirement for any filter length as compared to the other designs.

F. FDA VS PROPOSED DESIGNS

In this subsection, we compare the overall performance of proposed methods against FDA in light of the graph of Figure 15. The proposed designs show a 2.18× and 1.84× reduced CPD as compared to FDA allowing the system to operate at higher frequency. Meanwhile, the same stats are

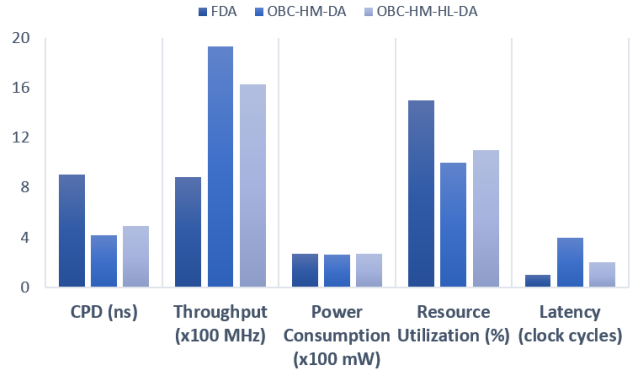


FIGURE 15. Performance comparison of all designs w.r.t 16-tap filter.

observed for throughput but in an increasing fashion which shows that the proposed designs effectively increase the system throughput.

Power Consumption of the proposed designs are scaled down by about 3.5% and 1.9% respectively when compared to FDA. Recall the graph of Figure 13 in the power consumption section, we established that as filter order increases the consumption of FDA shows more of an exponential rise as compared to the linear rise of the proposed designs. Hence for higher order filters, power consumption stats show even more promising results for the proposed designs.

Next, the graph shows that the proposed designs achieve a 5% and 4% alleviation in FPGA resource utilization respectively as compared to FDA. The breakdown of area utilization w.r.t sources, as shown in Table 8, shows about a 2× reduced utilization of slice LUTs for the proposed designs while the rest of the components display rather close results.

Finally, in Figure 15, latency results are displayed which show that FDA, Type-I and Type-II designs require a latency of 1, 4 and 2 clock cycles respectively to compute one result. Hence, where FDA lacks in other design attributes, it provides the best latency and where Type-I achieves the best performance in other design attributes, it does so at a cost of 4× elevated latency. Meanwhile, Type-II acts as a compromise between the other two architectures.

We also compared the performance of proposed designs with FDA in terms of Giga Multiply-Accumulate per Second (GMACS) as presented in Table 9. The results show that the DSP performance improves by 2.18× and 1.84× for Type-I and Type-II designs respectively.

G. INTER-COMPARISON OF 4, 8 AND 16-TAP IMPLEMENTATIONS

We also perform an inter-comparison of 4, 8 and 16-tap filters as shown in Figure 16. Note that as we move from 4 to 8-tap filter, the CPD stats show a 1.24× increased delay for FDA. Similarly, as we move from 8 to 16-tap, the delay further increases to 1.38×. However, in case of Type-I design, we observe an increase of 1.10× and 1.12× in the delay. Similarly, Type-II design shows an elevation of about 1.2× in both cases. In light of these stats, we deduce that the

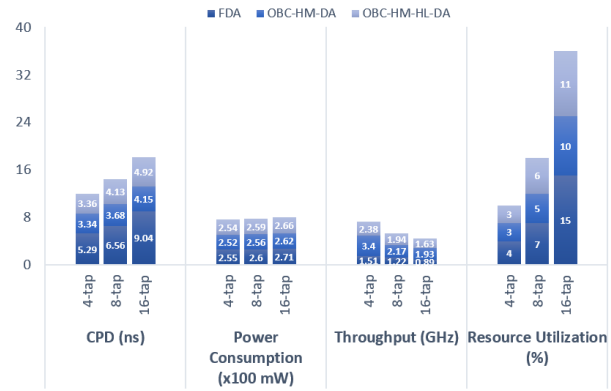


FIGURE 16. Comparison of various performance metrics of all designs for 4, 8 and 16-tap filters.

TABLE 9. DSP performance comparison based on GMACS.

Design	GMACS
FDA	1.77
Type-I	3.86
Type-II	3.25

proposed designs show a more linear and lenient increase in CPD as compared to FDA. Hence they provide better timing performance as we move towards higher order filters.

When we compare the power consumption stats in a similar manner, we find a 1.9% and 4.2% increase in FDA as we progress through the two graph. Similarly, Type-I shows a 1.5% and 2.3% increase and Type-II shows a 1.9% and 2.6% rise in power consumption. Now that moving from 4 to 8-tap, the increment factor is almost the same for all three filters. However, at moving from 8 to 16-tap, the proposed designs show a $1.62 - 1.83\times$ lesser elevation in power consumption as compared to FDA.

These stats show a linearly stable behavior of proposed designs as compared to FDA which appears to exhibit an exponential performance-drop as we move towards higher order filters. The improved power and timing performance of Type-I and Type-II designs also significantly reduce the system cost. This establishes them as stable, balanced and cost-effective designs.

H. TYPE-I VS TYPE-II

To compare the proposed designs among themselves, we again refer to the graph of Figure 15. Based on that, we work out that Type-I design possesses about $1.2\times$ lower CPD as compared to Type-II for the 16-tap filter. This means a $1.2\times$ increase in throughput of the former as compared to that of the latter. With regards to power consumption, the former requires 1.5% lesser power in contrast to the latter. However, there is not much daylight between the two designs in terms of resource utilization as Type-I uses just 1% lesser area as compared to Type-I. The reason for this is that the latter does not require additional memory or registers, just a small combinational logic allows to process the input in a parallel fashion.

TABLE 10. Resource utilization of OBC-HM-DA as compared to other designs.

Design	Platform	Area Utilization
ADF1 [12]	Stratix	915
ADF2 [15]	Cyclone III	845
ADF3 [17]	Spartan	712
ADF4 [23]	Stratix II	945
Type-I (proposed)	Virtex-7	620

TABLE 11. Power consumption and number of slices of ADF5 and proposed designs for a 16-tap filter.

Design	Power Consumption (mW)	No. of Slices
ADF5 [20]	234	189
Type-I	262	131
Type-II	266	139

The last performance metric answers any doubts regarding the benefits of Type-II design. Regardless the filter length, it always provides a latency exactly half of Type-I design. Timing diagram of Figure 10 also shows a clock based development of latency. Hence, Type-II design improves system latency at the cost of minimum resource overhead.

So, the bottom-line is that Type-I (OBC-HM-DA) provides the most stable and power efficient design as compared to other designs. Moreover, it also provides the best timing statistics and the least resource utilization. These properties lead to lower area and cost requirements for the device. However, it displays more latency which might not be desirable at times, for which we proposed the Type-II (OBC-HM-HL-DA) design, which acts as a compromise between Type-I and FDA, i.e. it is not as competent as the former in design attributes but provides better latency, and is still more efficient than FDA in terms of area, cost, power and timing stability.

I. PROPOSED DESIGNS VS PREVIOUSLY PROPOSED DESIGNS

In this section, we compare the results of Type-I design with existing schemes proposed in [12], [15], [17] and [23]. For simplicity, we refer these as ADF1, ADF2, ADF3 and ADF4 respectively. These designs have used unique FPGA platforms for implementation. Since each FPGA board has its own specifications, a fair comparison is not possible. Therefore we enlist the platform used by each design to compare their area utilization as shown in Table 10.

We also compare our designs to a Virtex-7 implemented (same as us) design proposed in [20], referred as ADF5. Table 11 shows a comparison of power consumption and slice utilization of ADF5 and OBC-HM-DA. According to these stats, ADF5 provides a $1.11\times$ and $1.13\times$ better power utilization than Type-I and Type-II respectively. On the other hand, the proposed designs require a $1.45\times$ and $1.36\times$ lesser area when compared to ADF5. Thus we have a possible trade-off between area and power utilization. These comparisons show

that the proposed designs surpass the previously proposed designs in terms of area utilization.

VI. CONCLUSION

LMS is a sophisticated algorithm for digital adaptive filters which makes use of MAC operations for output and weight calculation. Despite the algorithm's simplicity, its dedicated real-time implementation is costly in terms of limited resources, especially multiplier which is the most expensive unit in the design. This paper presents a time-multiplexed design for multiplier-less implementation of LMS algorithm using distributed arithmetic, half memory algorithm, offset binary coding and parallel processing. Based on the design requirements, two variants of the fundamental architecture have been proposed:

- OBC-HM-DA (Type-I), to improve the timing and design attributes such as delay, throughput, power and resource utilization at the expense of latency.
- OBC-HM-HL-DA (Type-II), to halve the latency induced by Type-I with slightly reduced performance for other design attributes.

Vivado was used to implement and synthesize these designs as well as the traditional dedicated, fully parallel architecture for reference. Throughput of Type-I and Type-II architectures showed an improvement of 2.18× and 1.84× respectively against FDA for a 16-tap filter. Similarly, power consumption was reduced by 3.5% and 1.9%, and area utilization by 5% and 4% respectively. Latency being the trade-off, turned out to be 4 and 2 clock cycles respectively against a single clock cycle of FDA. Inter-comparison of 4, 8 and 16-tap filter lengths showed a linear utilization and consumption approach of the proposed designs in contrast to an exponential approach of FDA. These statistical comparisons establish OBC-HM-DA to be a balanced, stable and resource-efficient design while OBC-HM-HL-DA being the lesser version of the former but with improved latency.

REFERENCES

[1] A. J. Humaidi, I. Kasim Ibraheem, and A. R. Ajel, "A novel adaptive LMS algorithm with genetic search capabilities for system identification of adaptive FIR and IIR filters," *Information*, vol. 10, no. 5, p. 176, May 2019.
[2] E. P. Jayakumar and P. S. Sathidevi, "An integrated acoustic echo and noise cancellation system using cross-band adaptive filters and wavelet thresholding of multitaper spectrum," *Appl. Acoust.*, vol. 141, pp. 9–18, Dec. 2018.
[3] M. T. Akhtar, F. Albu, and A. Nishihara, "Acoustic feedback cancellation in hearing aids using dual adaptive filtering and gain-controlled probe signal," *Biomed. Signal Process. Control*, vol. 52, pp. 1–13, Jul. 2019.
[4] R. Martinek and J. Zidek, "The real implementation of ANFIS channel equalizer on the system of software-defined radio," *IETE J. Res.*, vol. 60, no. 2, pp. 183–193, Mar. 2014.
[5] C.-H. Lee and W. Wolf, "Evaluation of functional architectures for cognitive radio systems," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Mar. 2008, pp. 5392–5395.
[6] A. Garcés Correa, L. L. Orosco, P. Diez, and E. Laciar Leber, "Adaptive filtering for epileptic event detection in the EEG," *J. Med. Biol. Eng.*, vol. 39, no. 6, pp. 912–918, Dec. 2019.
[7] T. Kazaz, M. Kulin, and M. Hadzialic, "Design and implementation of SDR based QPSK modulator on FPGA," in *Proc. 36th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2013, pp. 513–518.
[8] C. Venkatesan, P. Karthigaikumar, and R. Varatharajan, "FPGA implementation of modified error normalized LMS adaptive filter for ECG noise removal," *Cluster Comput.*, vol. 22, no. S5, pp. 12233–12241, Sep. 2019.
[9] J. J. Rodríguez-Andina, M. D. Valdes-Pena, and M. J. Moure, "Advanced features and industrial applications of FPGAs—A review," *IEEE Trans. Ind. Informat.*, vol. 11, no. 4, pp. 853–864, Aug. 2015.
[10] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Mag.*, vol. 6, no. 3, pp. 4–19, Jul. 1989.
[11] C. Cowan, S. Smith, and J. Elliott, "A digital adaptive filter using a memory-accumulator architecture: Theory and realization," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 31, no. 3, pp. 541–549, Jun. 1983.
[12] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, "LMS adaptive filters using distributed arithmetic for high throughput," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 7, pp. 1327–1337, Jul. 2005.
[13] A. Tiwari, P. Kumar, and M. Tiwari, "High throughput adaptive block FIR filter using distributed arithmetic," in *Proc. 1st India Int. Conf. Inf. Process. (IICIP)*, Aug. 2016, pp. 1–6.
[14] Y. Zhou and P. Shi, "Distributed arithmetic for FIR filter implementation on FPGA," in *Proc. Int. Conf. Multimedia Technol.*, Jul. 2011, pp. 294–297.
[15] M. T. Khan, S. R. Ahamed, and F. Brewer, "Low complexity and critical path based VLSI architecture for LMS adaptive filter using distributed arithmetic," in *Proc. 30th Int. Conf. VLSI Design 16th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2017, pp. 127–132.
[16] M. Surya Prakash and R. A. Shaik, "High performance architecture for LMS based adaptive filter using distributed arithmetic," Tech. Rep., 2012.
[17] M. S. Prakash and R. A. Shaik, "Low-area and high-throughput architecture for an adaptive filter using distributed arithmetic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 11, pp. 781–785, Nov. 2013.
[18] M. T. Khan and S. R. Ahamed, "Area and power efficient VLSI architecture of distributed arithmetic based LMS adaptive filter," in *Proc. 31st Int. Conf. VLSI Design 17th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2018, pp. 283–288.
[19] H. B. Kundhu Prabakaran and A. Yada, "High throughput parallelized realization of adaptive FIR filter based on distributive arithmetic using offset binary coding," in *Proc. 10th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2019, pp. 1–6.
[20] C. S. Vinitha and R. K. Sharma, "Area and energy-efficient approximate distributive arithmetic architecture for LMS adaptive FIR filter," in *Proc. Int. Conf. Emerg. Technol. (INCET)*, Jun. 2020, pp. 1–5.
[21] Y. Tsunekawa, K. Takahashi, S. Toyoda, and M. Miura, "High-performance VLSI architecture of multiplierless LMS adaptive filters using distributed arithmetic," *Electron. Commun. Jpn. (III: Fundam. Electron. Sci.)*, vol. 84, no. 5, pp. 1–12, 2001.
[22] R. Guo and L. S. DeBrunner, "A novel adaptive filter implementation scheme using distributed arithmetic," in *Proc. Conf. Rec. 45th Asilomar Conf. Signals, Syst. Comput. (ASILOMAR)*, Nov. 2011, pp. 160–164.
[23] R. Guo and L. S. DeBrunner, "Two high-performance adaptive filter implementation schemes using distributed arithmetic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 9, pp. 600–604, Sep. 2011.
[24] K. Takahashi, Y. Tsunekawa, S. Toyoda, and M. Miura, "High-performance architecture of lms adaptive filter using distributed arithmetic based on half-memory algorithm," *IEICE Trans. Fundam.*, vol. 84, pp. 777–787, Feb. 2001.
[25] G. NagaJyothi and S. SriDevi, "Distributed arithmetic architectures for FIR filters–A comparative review," in *Proc. Int. Conf. Wireless Commun., Signal Process. Netw. (WiSPNET)*, Mar. 2017, pp. 2684–2690.
[26] 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.

SHAWEZ AHMAD received the B.S. degree in electrical engineering from the National University of Computer and Emerging Sciences (NUCES-FAST), Lahore, in 2018. He is currently pursuing the M.S. degree in computer engineering from the College of Electrical and Mechanical Engineering (CE&ME), National University of Sciences and Technology (NUST), Islamabad. His research interests include approximate computing, digital system design, and machine learning.

SAJID GUL KHAWAJA received the B.S., M.S., and Ph.D. degrees in computer engineering from the College of Electrical and Mechanical Engineering (CE&ME), National University of Sciences and Technology (NUST), Islamabad. He is currently an Assistant Professor with the Department of Computer and Software Engineering, CE&ME, NUST. His research interests include embedded systems, image processing, and machine learning.

NAEEM AMJAD received the B.S. degree in electrical engineering from COMSATS University, Islamabad, in 2014. He is currently pursuing the M.S. degree in computer engineering from the College of Electrical and Mechanical Engineering (CE&ME), National University of Sciences and Technology (NUST), Islamabad. He has worked as a Senior Design Engineer with the Center for Advanced Research in Engineering (CARE), Islamabad. His research interests include *ad hoc* wireless sensor networks, digital design, and software defined radios.

MUHAMMAD USMAN received the B.S. degree (Hons.) in electrical computer engineering from COMSATS University, Abbottabad, in 2014. He is currently pursuing the M.S. degree in computer engineering from the College of Electrical and Mechanical Engineering (CE&ME), National University of Sciences and Technology (NUST), Islamabad. He has been working as a Senior Designer with the Center for Advanced Research in Engineering (CARE), Islamabad, since 2015. His research interests include digital signal processing, deep learning, and software defined radio.