IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 3 July 2023, accepted 5 August 2023, date of publication 10 August 2023, date of current version 16 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3304234

TOPICAL REVIEW

A Systematic Review on Distributed Arithmetic-Based Hardware Implementation of Adaptive Digital Filters

SANZHAR YERGALIYEV, (Student Member, IEEE), AND MUHAMMAD TAHIR AKHTAR, (Senior Member, IEEE)

Department of Electrical and Computer Engineering, School of Engineering and Digital Sciences, Nazarbayev University, 010000 Astana, Kazakhstan

Corresponding author: Muhammad Tahir Akhtar (muhammad.akhtar@nu.edu.kz; akhtar@ieee.org)

This work was supported in part by the Nazarbayev University Collaborative Research Grants Program under Grant 021220CRP0222.

ABSTRACT Adaptive Digital Filters (ADFs) are computationally demanding Digital Signal Processing (DSP) systems with applications in diverse areas signal processing, such as active noise control, channel equalization, system identification. The most popular implementation structure for ADFs is FIR transversal filter which can be efficiently implemented without multipliers by Distributed Arithmetic (DA) method. DA is used to efficiently implement digital FIR filter with a precomputed weight DA combinations in a memory element. However, due to the operation of ADFs the memory content has to be recomputed at each iteration; the advantage of DA method disappears. The objective of this review is to cover the research gap on DA-ADF implementations. This article reviews the main features and contributions of DA-ADF designs present in literature. The DA-ADF designs have been grouped into three categories: non-pipelined, pipelined and block processing. Pipelining and block processing increase the throughput of the ADF. The reviewed designs implement only Least Mean Square (LMS) type adaptive algorithms due to the simplicity of implementation. Latest and most efficient designs tend to employ conjugate Offset Binary Coding (OBC) DA structure.

INDEX TERMS Adaptive digital filters (ADFs), distributed arithmetic (DA), least mean square (LMS).

I. INTRODUCTION

Adaptive Digital Filters (ADFs) are powerful self-designing systems used in a wide variety of digital signal processing (DSP) applications [1], [2]. ADFs are notably applied in active power filters [3], bearing prognosis [4], hearing aids [5], [6], active noise control [7], cerebellum modeling [8], 5G technology [9] etc. Finite Impulse Response (FIR) is the most commonly used adaptive filter structure due to its stability property. It consists of an FIR filter with adjustable coefficients and a module that updates these coefficients according to an adaptive algorithm. The Least Mean Square (LMS) adaptive algorithm is one of the most employed for hardware implementations due to its simplicity.

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

A. MOTIVATION

Most of the contemporary portable DSP systems require high performance while keeping the design small and low-power. The fully dedicated architecture implementation, where every arithmetic operation is realized with dedicated hardware unit, of direct form ADF with LMS algorithm would require $2N$ multipliers and $2N - 1$ additions, where N is the FIR filter length. Hence, fully dedicated architecture implementation demands large power consumption and is costly in terms of hardware. This cost is unbearable for high order filter implementation as hardware is proportional to the order of the filter and portable devices that are constrained by power and area limits. A serial implementation of the ADFs with one multiply and accumulate unit is possible to mitigate issues with area and power. However, the throughput of the system is traded off and the maximum speed of the system becomes limited. A parallelized implementation with several

multiplexed multipliers or processing units is possible [10] at the cost of higher power, area, and hardware consumption as the multipliers are hardware and power demanding circuits.

Distributed Arithmetic (DA) is a computationally efficient bit-serial inner product computation method [11] that is used in implementation of digital filtering process without a need for hardware multipliers. Combinations of coefficients of the filter can be precomputed and stored in the memory element, usually Look-Up-Tables (LUTs), which are accessed by an address formed with the bits of the input samples from Least Significant Bit (LSB) to Most Significant Bit (MSB). The outputs of LUT are then shifted and added to compute the output of the digital filter. Major disadvantage of DA method is that the number of partial products to be stored in the LUTs grows exponentially for large filter orders. Another problem arises while implementing ADFs with DA method. For ADFs the precomputation of filter coefficients are not possible as at every iteration the filter weights should be updated. This problem almost entirely eliminates advantages of DA method. Thus, an efficient designs are needed to overcome this limitations.

B. DISTRIBUTED ARITHMETIC ADF LITERATURE REVIEW

First study of DA has been done by Croisier et al. [12] in 1973 and the most commonly recognized description of DA method has been proposed in 1974 [13] by A. Peled and B. Liu. It has been applied to second-order Infinite Impulse Response (IIR) filter that can be combined to realize higher order IIR filters. The initial attempts to employ DA for LMS ADFs have been reported in [10], [11], [12], [13], [14], and [15]. As the most notable obstacle to use DA for ADFs is that the filter coefficients had to be recomputed at every iteration early works tried to reduce hardware requirements. Basic feasibility of DA-ADF using prototype hardware implementation of 8 point transversal filter has been proven in [14] and [15]. To ease the LMS update process an assumptions of zero mean, white input signal and Offset Binary Coded (OBC) samples were made. OBC encodes binary data with ± 1. In contrast, regular DA is derived assuming 2's complement data representation and is referred to as Two's Complement DA (TC-DA). As a result, the update reduces to shifting and adding error sample to the contents of the memory. In [16] and [17] it was shown that large filters can be broken down into several smaller filters which eases the implementation. These, technique of implementing high-order filters with a combination of smaller filters is widely used among other researchers as LUT decomposition method. Also, study of nonlinear adaptive filters in context of echo cancelers were conducted [18], [19]. The complexity reduction in these architectures were achieved by utilizing symmetries of OBC and Volterra kernels together with the previously discussed assumption of zero-mean and white input signal. Later, the same simplified update method presented in [14] and [15] but assuming 2's complement representation of the samples has been derived in [20].

The work of Allred et. al [21] got much attention, where they proposed an efficient update procedure using two LUTs: filtering LUT and auxiliary LUT. Update of auxiliary LUTs is done by reusing the upper half of the previous LUT contents and adding the most current sample to compute the other half. The filtering LUT is updated by scaling and adding the contents of auxiliary LUT to the corresponding addresses according to the LMS weight update rule. This update method does not put any assumptions on the input signal statistics. In 2006, a conjugate DA structure for ADFs was proposed [22]. Compared to traditional or direct DA structure conjugate DA uses the weights of ADF to form an addresses to the filtering LUT avoiding the need for the auxiliary LUT, effectively halving the amount of memory needed and increasing the throughput. Later, two designs for TC and OBC encoding that used the conjugate DA method have been independently proposed in [23]. The update of the LUT for the TC-DA ADF is done as in [21], however, for the OBC-DA ADF a different update has been used. For OBC-DA design a term computed with the most current and the oldest samples is subtracted from the LUT content at even addresses and added to content at odd addresses to compute the second and first half of the LUT for next iteration, respectively. Conjugate DA structure have been also applied to implement Sliding Block DA (SB-DA) ADF [24] and Modified Sliding Block DA (MSB-DA) ADF [25]. The SB-DA ADF utilizes the fact that the collection of input samples processed by the filter changes slowly, by one sample per iteration. By employing the LUT decomposition done in [16] and [17] only one of the smaller LUTs that contain the oldest sample has to change and one extra LUT is dedicated to hold newest sample. To execute such an operation with LUT reusing the weights should be circularly shifted. This notion of LUT reusing is also widely applied in block processing DA-ADF implementations.

The block processing designs have been studied in [43], [44], [45], [46], [47], [48], and [49]. Given the block length of L, block digital filters process L input vectors and compute L outputs in one iteration. Thus, for the higher hardware cost L times higher throughput can be achieved. The DA method has been employed in block ADFs in [45], where DA is applied to efficiently mechanize the Fast Fourier Transform in the implementation of Fast Block LMS. The first DA formulation of Block LMS (BLMS) based on conjugate TC-DA structure has been done in [46]. A similar concept to SB-DA have been used to update contents of LUTs. A collection of LUTs, called a processing element, with oldest samples is overwritten by the combinations of the newest samples for each iteration called inter-iteration LUT reuse. Rest of the processing elements with intermediate samples remain the same. The LUT sharing is realized by implementing register array based LUT. This design has been optimized in [47] by reusing LUT contents during one iteration called intra-iteration LUT sharing. As the successive LUTs in one PE have three overlapping samples half of the previous LUT contents can be shared with the next LUT. Thus, reducing the memory complexity. An OBC-DA implementation have been proposed in [48].

The initial design [46] have been improved with even-odd LUT decomposition in [49]. Intra-iteration have been adapted for OBC-DA design in [50] and further optimized in [51].

Another class of DA-ADF designs are pipelined architectures [35], [36], [37], [38], [39], [40], [41], [42]. Pipelining is used to reduce the critical-path and increase the throughput of the system. One of the first DA-ADF pipelined design has been proposed in [37] based on TC direct DA structure. Later, a TC conjugate DA structure pipelined ADF design has been proposed in [38]. The differences with non-pipelined DA designs are delay elements that separate the processing stages, register based LUTs, and shift accumulate unit. Delay is introduced between filtering, error computation and weight update processes in [37] and [38]. Register based LUT computes the DA combinations in parallel and online. Thus, the number of registers needed increases exponentially. To reduce the number of registers needed a multiplexed LUT and AND cell LUT have been proposed in [40]. An improved performance DA-ADF have been proposed in [41] that used AND cell LUT as base unit LUT and introduced delay elements in the adder tree. A convergence enhanced DA-ADF based on convex combination of two adaptive filters [42] has been presented in [43]. Two combined ADFs have been replaced with one DA-ADF unit with two multiplexed step sizes. A 2-Dimensional DA-ADF based on pipelined DA-ADF architecture have been proposed in [44].

To the best knowledge of authors, no comprehensive literature review is available on the almost three decade long DA-ADF research. The intent of this article is to provide better understanding of the DA-ADF implementations by reviewing the main features of the designs. The present work indeed summarizes the most significant works in one place for the reference of interested. The articles have been selected based on the chronological order of significant improvement of the DA-ADF structure. The works on FPGA implementation have been omitted as they are FPGA architecture specific. Furthermore, papers with very incremental improvements have also not been considered in the current review. The rest of the paper is organized as follows. Section II is devoted for DA method, its OBC variation, and DA digital filter implementation. Section III reviews LMS, Delayed LMS (DLMS), and BLMS. Section IV presents the non-pipelined, pipelined, and block processing DA-ADF designs. Section V concludes this paper. For the sake of convenience, a list of abbreviations used in the paper in given in Table 1.

TABLE 1. List of abbreviations used in the paper.

ADF	Adaptive Digital Filter
FIR	Finite Impulse Response
DSP	Digital Signal Processing
LMS	Least Mean Square
DA	Distributed Arithmetic
LUT	Look-Up-Table
LSB	Least Significant Bit
MSB	Most Significant Bit
IIR	Infinite Impulse Response
OBC	Offset Binary Coding
TC-DA	Two's Complement DA
SB-DA	Sliding Block DA
MSB-DA	Modified Sliding Block DA
BLMS	Block LMS
DLMS	Delayed LMS
AxC	Approximate Computing
PPG	Partial Product Generator
CLA	Carry Lookahead Adder
CSA	Carry Save Adder
RLS	Recursive Least Square

II. DISTRIBUTED ARITHMETIC METHOD

A. TWO's COMPLEMENT DISTRIBUTED ARITHMETIC

DA is a method of implementing the inner product of two vectors in a bit-serial manner [11]. As the output of FIR filter can be computed as inner product of input sample vector and weight vector DA can be applied to implement digital FIR filter. To illustrate the inner product computation of TC-DA, consider:

$$y[n] = \mathbf{w}^T \mathbf{x}[n] = \sum_{k=0}^{N-1} w_k x[n-k], \tag{1}$$

where $\mathbf{w} = [w_0\ w_1\ \cdots\ w_{N-1}]^T$ is a vector of fixed coefficients, and $\mathbf{x}[n] = [x[n]\ x[n-1]\ \cdots\ x[n-N+1]]^T$ is a vector of input data samples. Assuming input data samples are in 2's complement form with B bits, then $x_k = x[n-k]$ can be expressed as:

$$x_k = -x_{k0} + \sum_{j=1}^{B-1} x_{kj} 2^{-j}, \tag{2}$$

where x_{kj} is the jth bit of the input data word such that $x_{kj} \in [0, 1]$ and the x_{k0} is a sign bit. Replacing x_k in (1) by its representation in (2):

$$y = \sum_{k=0}^{N-1} w_k \left[-x_{k0} + \sum_{j=1}^{B-1} x_{kj} 2^{-j} \right], \tag{3}$$

where $y = y[n]$ to ease the notation. By distributing the terms and exchanging the order of the summations, gives:

$$y = -\sum_{k=0}^{N-1} w_k x_{k0} + \sum_{j=1}^{B-1} \left[\sum_{k=0}^{N-1} w_k x_{kj} \right] 2^{-j}, \tag{4}$$

where the term $\sum_{k=0}^{N-1} w_k x_{kj}$ can be precomputed as it has only 2^N possible values given w_k's because $x_{kj} \in [0, 1]$. For example, for $N = 2$ there are only 2^2 partial products: $[0, w_1, w_2, w_1 + w_2]$. Partial products can stored in a LUT and accessed with an address formed by the bits of the input samples from one bit position. According to (4) these partial products should be then shift added to compute the result of an inner product or the output of a digital filter. Fourth order DA FIR filter structure with the description of the content of the memory is depicted in Fig. 1. The bits of the input samples form an address that is then used to look up corresponding combination of weight from the memory. As the addresses

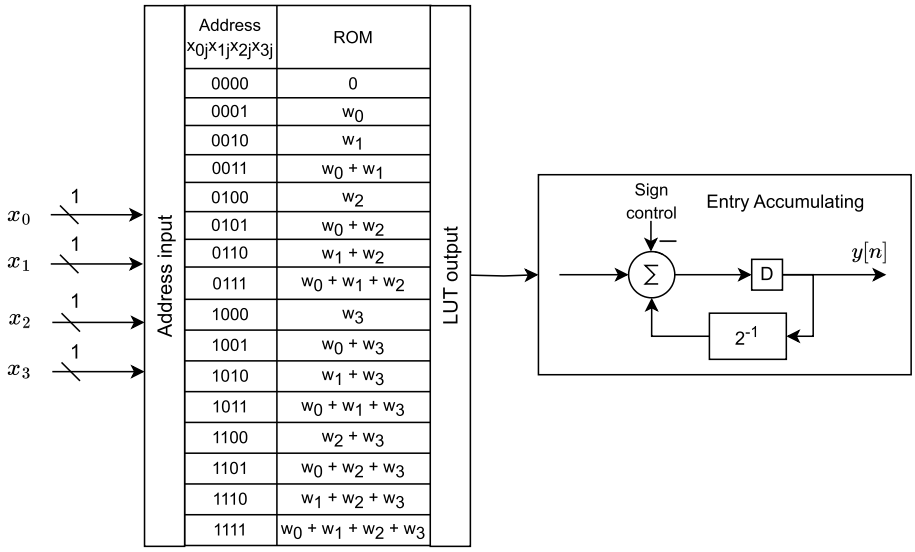

FIGURE 1. Block diagram of length 4 TC-DA FIR filter.

formed from LSBs to MSBs of the input samples the output of the memory should be right shifted accordingly to account for the bit position. When the MSBs arrive the sign control signal becomes high and changes addition to subtraction. Because of TC number representation the MSB is the sign bit. The output sample computation takes several clock cycles equal to the word length of the input samples. So that MSBs arrive last to form an address and negate the output of the memory.

Immediate observations are that the memory required grows exponentially with the filter order 2^N and due to the serial mechanization 1 bit-at-a-time DA filter may seem slow. However, if the word length of the input signal equal to the number of weights, the time required to compute output sample with DA is equal to the approach where one multiplier is used and samples are fed serially [11]. The exponential memory problem can be partially mitigated by using OBC.

B. OFFSET BINARY CODING DISTRIBUTED ARITHMETIC

By using OBC the memory requirement can be reduced by half due to the arising symmetries [11]. OBC assumes bits to be either $+1$ or -1. So the bit values are symmetric around 0. Consider x_k to be:

$$x_k = \frac{1}{2}(x_k - (-x_k)), \tag{5}$$

where $-x_k$ is 2's complement of x_k written as:

$$-x_k = -\bar{x}_{k0} + \sum_{j=1}^{B-1} \bar{x}_{kj} 2^{-j} + 2^{-(B-1)}, \tag{6}$$

where $\bar{x}_{kj}$ is the bit inverted x_{kj}. Substituting (2) and (6) in (5):

$$x_k = \frac{1}{2}(-(x_{k0} - \bar{x}_{k0}) + \sum_{j=1}^{B-1}(x_{kj} - \bar{x}_{kj})2^{-j} - 2^{-(B-1)}). \tag{7}$$

Consider the following substitution:

$$c_{kj} = \begin{cases} x_{kj} - \bar{x}_{kj}, & j \neq 0, \\ -(x_{k0} - \bar{x}_{k0}), & j = 0, \end{cases} \tag{8}$$

where $c_{kj} \in [-1, 1]$. Rewriting (7) with this substitution:

$$x_k = \frac{1}{2}\left(\sum_{j=0}^{B-1} c_{kj} 2^{-j} - 2^{-(B-1)}\right). \tag{9}$$

Substituting (9) in (1) the output sample can be computed as:

$$\begin{aligned} y &= \frac{1}{2}\sum_{k=0}^{N-1} w_k \left[\sum_{j=0}^{B-1} c_{kj} 2^{-j} - 2^{-(B-1)}\right], \\ &= \sum_{j=0}^{B-1}\left(\sum_{k=0}^{N-1} \frac{1}{2} w_k c_{kj}\right) 2^{-j} - \left(\sum_{k=0}^{N-1} \frac{1}{2} w_k\right) 2^{-(B-1)}, \end{aligned} \tag{10}$$

where $\sum_{k=0}^{N-1} \frac{1}{2} w_k$ represents the initial condition that is added at the LSB cycle or first cycle when LSBs of the inputs samples are used as address. The term $\sum_{k=0}^{N-1} \frac{1}{2} w_k c_{kj}$ can also be precomputed resulting in 2^N combinations. However, due to the emerging symmetries the number of combinations can be reduced to 2^{N-1}. The OBC-DA memory content and its OBC reduction are presented in Fig. 2. The upper half of the memory is the sign reversed lower half. Regarding the address x_{0j} bit decides whether the upper or lower half is accessed. Thus, bits from the most recent sample x_0 can be excluded from the address formation and the overall structure is modified as illustrated in Fig. 3. The XOR gates at the input operate like a 1's complement inverter of the address controlled by bits of x_0. The XOR gate at the sign control of the adder can be explained by extending the truth table to include the MSB sign control signal and then reducing it size by abusing the symmetries [11].

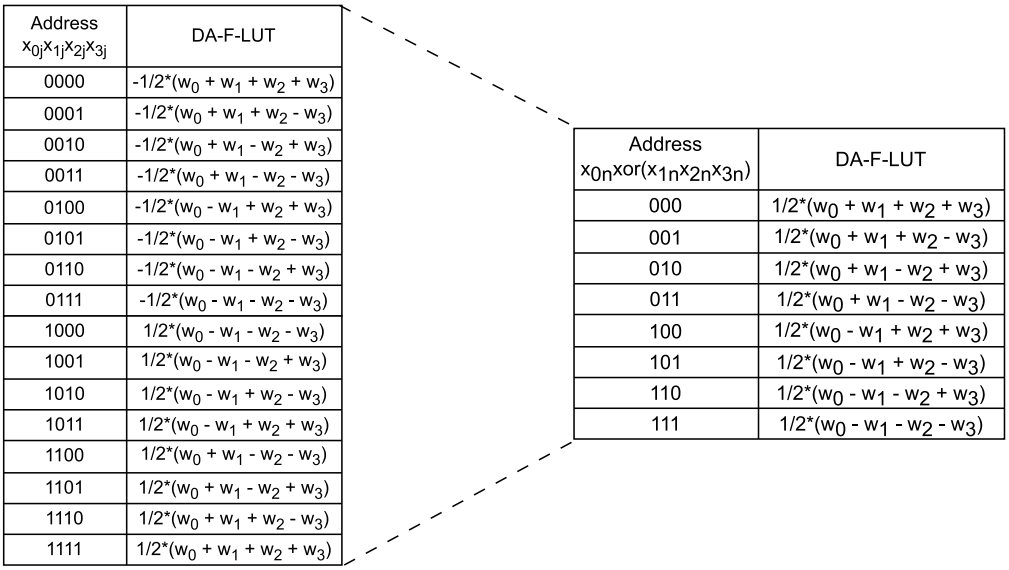

FIGURE 2. Length 4 OBC-DA FIR filter memory content reduction illustration.

C. CONJUGATE DISTRIBUTED ARITHMETIC

Conjugate DA has been first proposed in [22]. This method assumes the weights of the filter to be treated as addresses of the filter. Assuming weights to be in 2's complement form with B bits, then w_k can be expressed as:

$$w_k = -w_{k0} + \sum_{j=1}^{B-1} w_{kj} 2^{-j}, \tag{11}$$

where w_{kj} are bits of the filter weights. Performing the same derivation as for conventional DA, the output sample can be computed as:

$$y = -\sum_{k=0}^{N-1} x_k w_{k0} + \sum_{j=1}^{B-1} \left[\sum_{k=0}^{N-1} x_k w_{kj} \right] 2^{-j}. \tag{12}$$

After performing the same argumentation as for traditional DA, conjugate DA essentially has the role of input samples and weights switched. Thus, the content of filtering LUT has to be recomputed as the new input sample arrive. Although, this result has little meaning in the context of FIR digital filter implementation, it has considerable advantage for DA-ADF implementations.

III. ADAPTIVE ALGORITHMS

Fig. 4 depicts the general transversal FIR ADF block diagram where $x[n]$ is the input signal, $y[n]$ is an output of the programmable digital FIR filter, $d[n]$ is the desired response, and $e[n]$ is the error signal. General structure of transversal FIR ADF consists of two main computationally intensive blocks: programmable digital FIR filter and adaptive algorithm. Adaptive algorithm updates the weights of the programmable filter based on the input and error signals.

A. LMS ADAPTIVE ALGORITHM

In LMS output, $y[n]$ of the programmable digital FIR filter is first computed as:

$$y[n] = \mathbf{w}^T[n]\mathbf{x}[n] = \sum_{k=0}^{N-1} w_k[n]x[n-k], \tag{13}$$

where $\mathbf{w}[n] = [w_0[n] \; w_1[n] \; \cdots \; w_{N-1}[n]]^T$ is the time varying weight vector. Then the error signal, $e[n]$, is computed:

$$e[n] = d[n] - y[n], \tag{14}$$

which is used to change or update the weight of the programmable digital FIR filter as:

$$w_k[n+1] = w_k[n] + \mu e[n]x[n-k], \tag{15}$$

where $w_k[n]$ is k^{th} digital filter weight at time n and μ is the fixed step-size or learning rate. The vector form of the LMS update rule is given as follows:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu e[n]\mathbf{x}[n]. \tag{16}$$

B. DLMS ADAPTIVE ALGORITHM

Delayed LMS (DLMS) [36] algorithm has an adaptation delay that can be formulated as follows:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu e[n-m]\mathbf{x}[n-m], \tag{17}$$

where $e[n-m]$ and $\mathbf{x}[n-m]$ m-sample delayed error signal and input signal vector, respectively. The error can be delayed due to the pipelining. Input signals are delayed accordingly to match the delay of the error sample. The block diagram of DLMS is illustrated in Fig. 5. The pipelining delay or adaptation delay is lumped to m delay elements from the error computation and input signal sides.

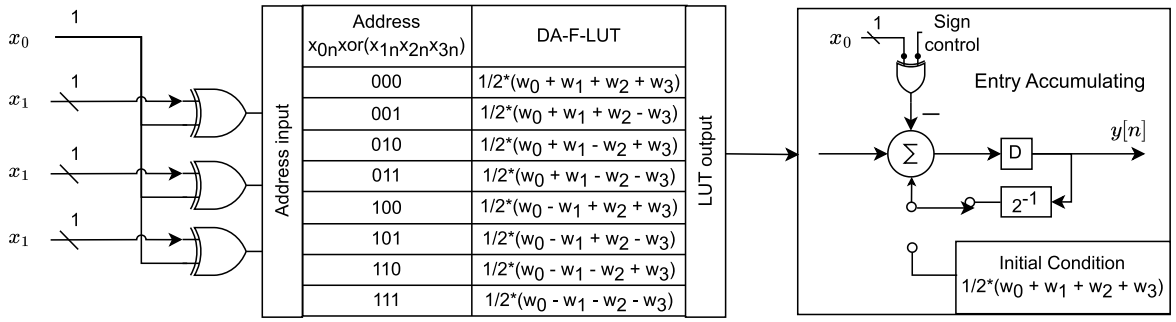

FIGURE 3. Block diagram of length 4 OBC-DA FIR filter.

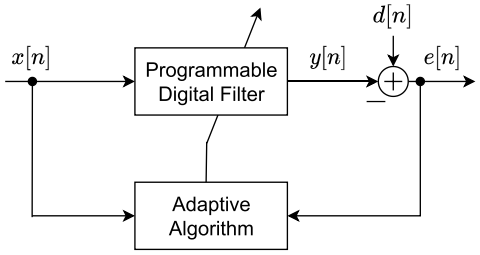

FIGURE 4. Block diagram of transversal FIR generic adaptive digital filter.

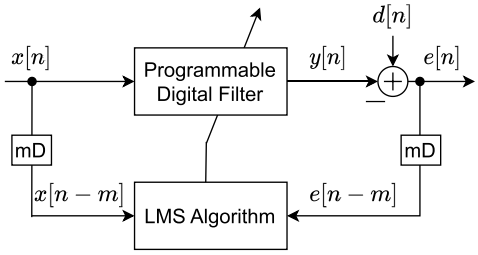

FIGURE 5. Block diagram of DLMS ADF.

C. BLMS ADAPTIVE ALGORITHM

The output of the BLMS algorithm is computed as:

$$\mathbf{y}[n] = \mathbf{X}[n]\mathbf{w}[n], \tag{18}$$

where $\mathbf{y}[n] = [y[nL]\ y[nL-1]\ \cdots\ y[nL-L+1]]^T$ is a vector of convolution sums of weights, index n denotes the block iteration, $\mathbf{X}[n]$ is the data matrix being given as:

$$\mathbf{X}[n] = \begin{bmatrix} x[nL] & x[nL-1] & \cdots & x[nL-N+1] \\ x[nL-1] & x[nL-2] & \cdots & x[nL-N] \\ \vdots & \vdots & \ddots & \vdots \\ x[nL-L+1] & x[nL-L] & \cdots & x[nL-L-N-2] \end{bmatrix}, \tag{19}$$

and the corresponding error vector is computed as:

$$\mathbf{e}[n] = \mathbf{d}[n] - \mathbf{y}[n], \tag{20}$$

where $\mathbf{e}[n] = [e[nL]\ e[nL-1]\ \cdots\ e[nL-L+1]]^T$ and $\mathbf{d}[n] = [d[nL]\ d[nL-1]\ \cdots\ d[nL-L+1]]^T$. The weight update equation becomes:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \Delta\mathbf{w}[n], \tag{21}$$

where $\Delta\mathbf{w}[n] = [\Delta w_0[n]\ \Delta w_1[n]\ \cdots\ \Delta w_{N-1}[n]]^T$ is computed as follows:

$$\Delta\mathbf{w}[n] = \mu\mathbf{X}^T[n]\mathbf{e}[n]. \tag{22}$$

IV. REVIEW OF DA-ADF DESIGNS

A. NON-PIPELINED DA ADFs

1) DA LMS UPDATE METHODS

Most of the non-pipelined designs employ LMS adaptive algorithm. The output of the programmable digital FIR filter can be computed as per TC-DA, OBC-DA, and conjugate DA methods in (4), (10), and (12), respectively. However, based on the chosen DA implementation the LMS update mechanism may differ. The operation of adaptive filters requires ongoing update of the weights, thus the whole DA LUT with weights should be recomputed at each iteration.

To ease the LMS update and reduce the computational complexity the following simplified update have been proposed for TC-DA [20] and OBC-DA [14], [15]:

$$\mathbf{p}[n+1] = \mathbf{p}[n] + \gamma\mu Ne[n]\mathbf{s}, \tag{23}$$

where $\gamma = 0.5$ for TC-DA and $\gamma = 2$ for OBC-DA, $\mathbf{p}[n]$ is a vector of DA partial products defined as $\mathbf{p}[n] = [p_0[n]\ p_1[n]\cdots p_{B-1}[n]]^T$, $\mathbf{s} = [-2^0\ 2^{-1}\ \cdots\ 2^{-(B-1)}]^T$ for TC-DA, and $\mathbf{s} = [-2^{-1}\ 2^{-2}\ \cdots\ 2^{-B}]^T$ for OBC-DA. A DA partial product is the output of the LUT that is related to some address formed by individual bits of the input sample. These DA partial products are essentially contents of DA LUT, i.e. some combination of filter weights corresponding to some address. The update mechanism is: first, to compute the output by DA operation, second, to update the few entries of the DA LUT by adding the scaled, or bit shifted if the μ and N are power's of two, error signal to the DA partial product and storing the result in the corresponding DA LUT

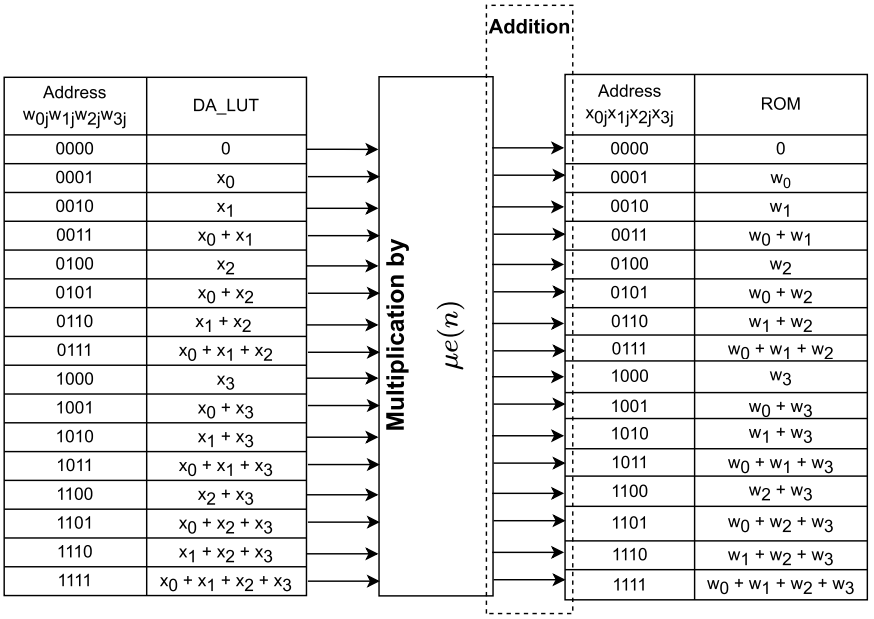

FIGURE 6. Filtering LUT update mechanism [21].

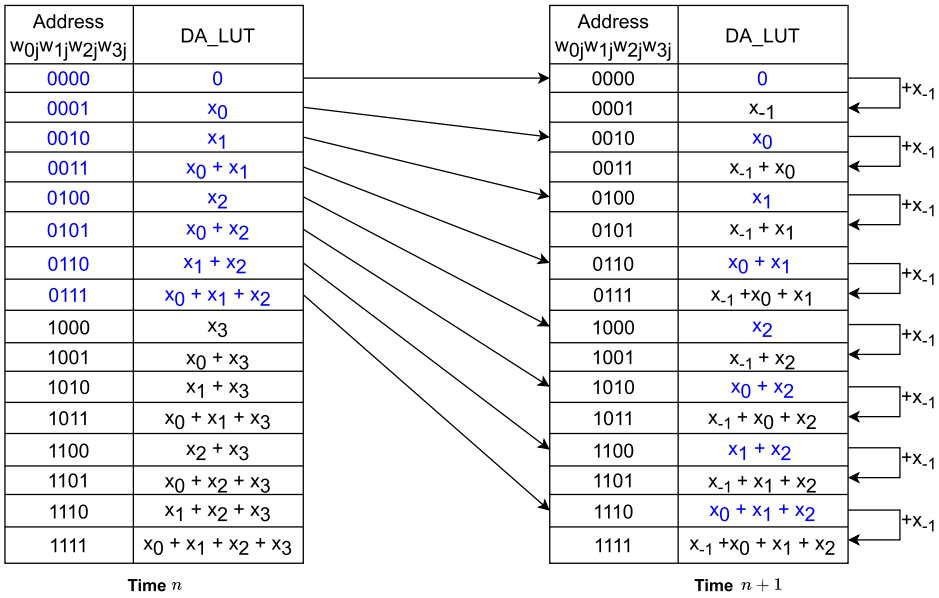

FIGURE 7. Auxiliary LUT update mechanism [21].

location. As the DA partial product is the result of a look up operation with some address, the update of the partial product should be performed on the same entry. Thus, storing the corresponding input addresses and DA partial products is essential. The simplified update is derived with an assumption that input signal is zero mean and white. However, such an assumption may not be applicable in many real world applications.

LMS update mechanism that does not require any assumptions on input signal distribution have been proposed in [21]. The update method requires two LUTs. One LUT, called filtering LUT, contains the TC-DA combinations of filter weights same as in Fig. 1. The other auxiliary LUT contains the TC-DA combinations of input samples that correspond to weight combinations. The basic principle of the weight or filtering DA LUT update is illustrated in Fig. 6. Due to the creation of an auxiliary LUT it contents can be used to directly update the filtering LUT according to the LMS weight update rule shown in (15). This LMS update method can be formulated as:

$$DA\_F\_LUT_{(r)}[n+1] = DA\_F\_LUT_{(r)}[n] + \mu e[n] DA\_A\_LUT_{(r)}[n], \quad (24)$$

where the $DA\_F\_LUT_{(r)}[n]$ and $DA\_A\_LUT_{(r)}[n]$ are the rth entries of the DA filtering LUT and DA auxiliary

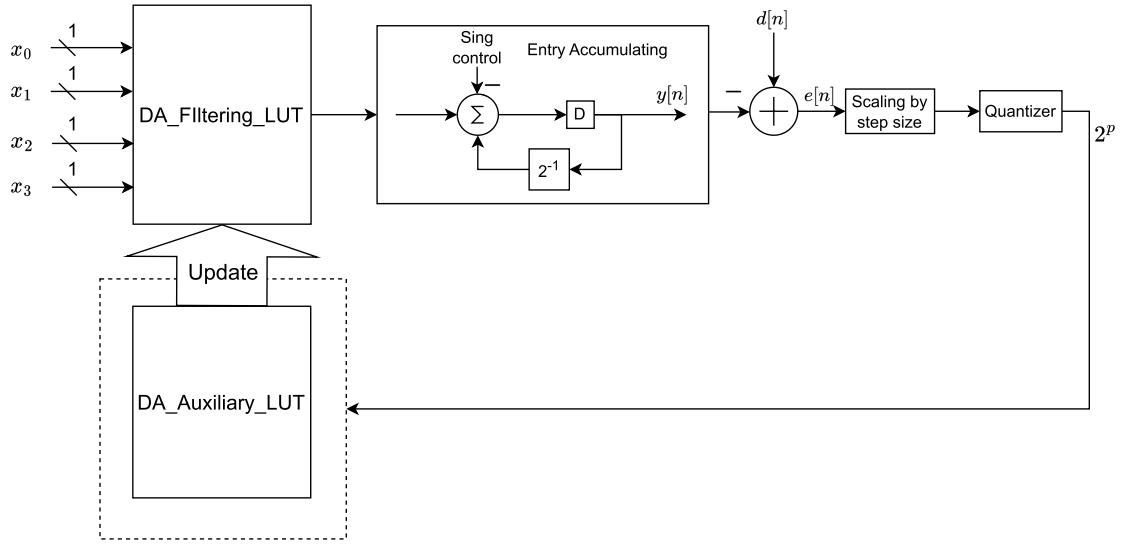

FIGURE 8. Block diagram of DA-ADF by Allred et. al. [21].

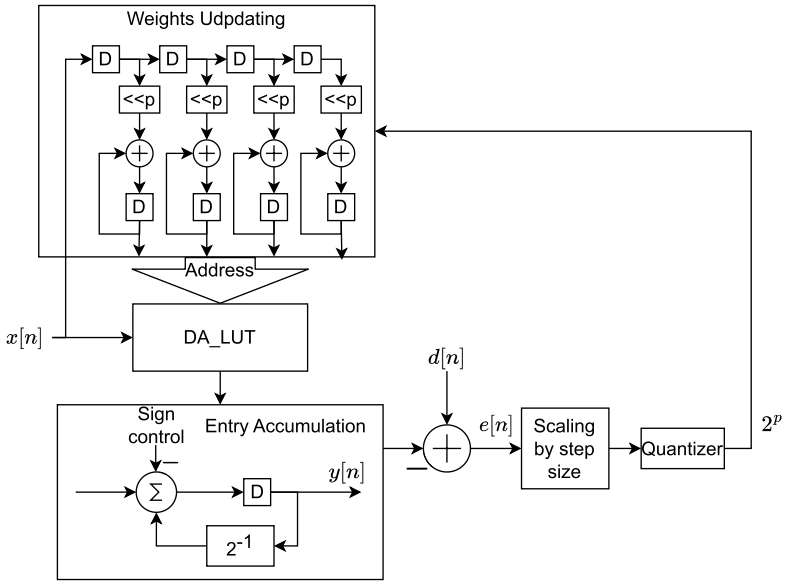

FIGURE 9. Block diagram of DA-ADF by Guo and DeBrunner [23].

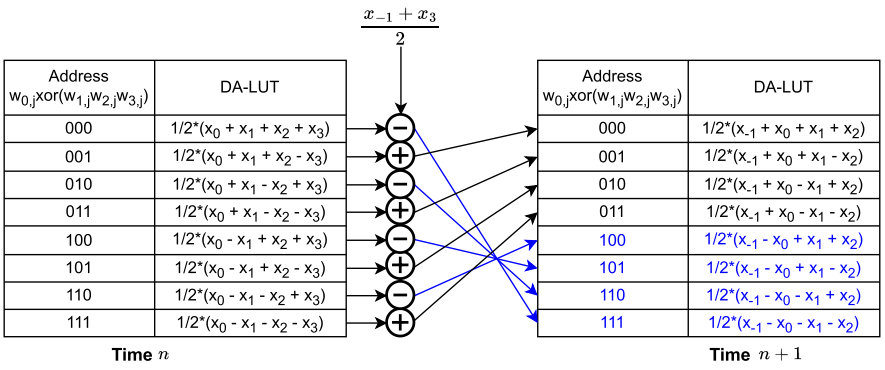

FIGURE 10. The update mechanism for OBC conjugate DA [23].

LUT, respectively. To maintain a valid DA auxiliary LUT an efficient LUT update is performed on the auxiliary LUT as illustrated in Fig. 7. The upper half is directly used in the updated auxiliary LUT as the even address contents. The odd addresses are then generated by adding the newest sample $x_{-1} = x(n+1)$ to the previous

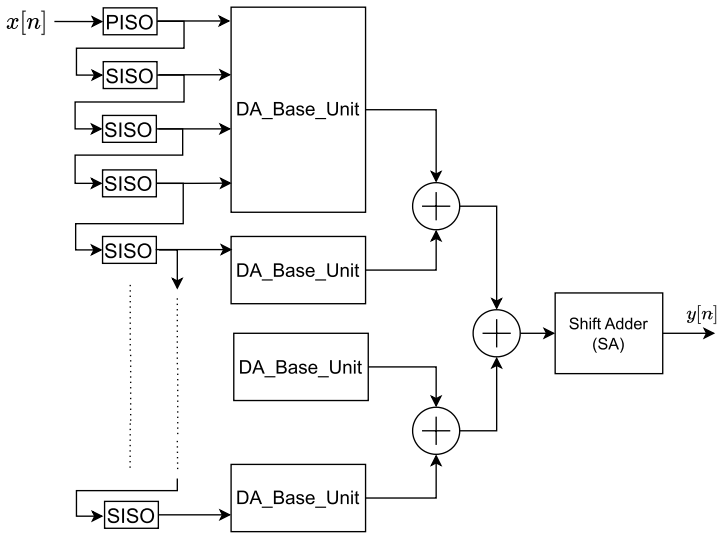

FIGURE 11. Illustration of length 16 DA filter with LUT decomposition.

Address $w_{-1j}w_{0j}$	Updated DA_LUT
00	0
01	x_0
10	0
11	x_0

Address $w_{1j}w_{2j}$	DA_LUT1
00	0
01	x_2
10	x_1
11	$x_1 + x_2$

Address $w_{3j}w_{4j}$	DA_LUT2
00	0
01	x_4
10	x_3
11	$x_3 + x_4$

Time n

Address $w_{0j}w_{1j}$	DA_LUT1
00	0
01	x_{-1}
10	x_0
11	$x_0 + x_{-1}$

Address $w_{2j}w_{3j}$	DA_LUT2
00	0
01	x_2
10	x_1
11	$x_1 + x_2$

Address $w_{4j}w_{-1j}$	Updated DA_LUT
00	0
01	0
10	0
11	0

Time $n + 1$

Address $w_{1j}w_{2j}$	DA_LUT1
00	0
01	x_{-1}
10	x_0
11	$x_0 + x_{-1}$

Address $w_{3j}w_{4j}$	DA_LUT2
00	0
01	x_2
10	x_1
11	$x_1 + x_2$

Address $w_{-1j}w_{0j}$	Updated DA_LUT
00	0
01	x_{-2}
10	0
11	x_{-2}

Time $n + 2$

FIGURE 12. Example of the SB-DA working principle for length 4 ADF.

even address. It can be observed that the contents of auxiliary LUT at time $n + 1$ are exactly the needed DA combinations.

The structure of the DA-ADF proposed by Allred et. al. [21] is illustrated in Fig. 8. The filtering process and auxiliary LUT update process are concurrent. After the output sample is computed the filtering LUT update begins. After the computation of the error sample it is scaled by the step-size μ that is usually in the order of $O(1/N)$ and the filter length is usually in power's of two. Scaling by the number that is a power of two is realized as left (for positive powers) or right (for negative powers) shifting of bits. The scaled error sample $e[n]$ is then quantized to be in the form of a power of two with the same idea. Thus, to update the filtering LUT the contents of auxiliary LUT are scaled by $\mu e[n]$ and added to the corresponding addresses of filtering LUT.

It is noticed that this design requires two equally sized LUTs. Thus, twice as much memory is needed to implement DA-ADF compared to DA digital filter. However, using the conjugate DA method [22] DA-ADF implementation without auxiliary LUT can be realized. The first proposition to use conjugate DA structure has been done in [22], which was later independently proposed by other researchers in [23]. The design of conjugate DA-ADF is shown in Fig. 9. Comparing with the design in [21], the auxiliary LUT becomes filtering LUT, which is more generally referred to as DA LUT in conjugate DA structures. The weight update mechanism now closely resembles the LMS update rule in (15) as the weights are updated directly. The weights are stored in registers, that are represented as delay elements. The updated weight w_{k+1} is computed as

$$w_{k+1}[n] = w_k[n] + Quantize(\mu e[n])x[n-k], \qquad (25)$$

where $Quantize(\mu e[n]) = 2^p$ to reduce the scaled error sample to a power of two to replace multiplication by bit shifting operation. The update of DA LUT is done exactly with the efficient update method of auxiliary LUT update seen in Fig. 7. Authors of [23] have proposed a second design using OBC to further reduce the memory required. Most of the design remains the same, but the update mechanism is different when the DA LUT contains OBC-DA combinations. The update of OBC-DA LUT for conjugate DA is illustrated in Fig. 10. The mean of the newest sample and the oldest sample is used to update the contents. Even contents of OBC-DA LUT are subtracted from this term and stored at the upper half of the updated LUT. Odd contents are added

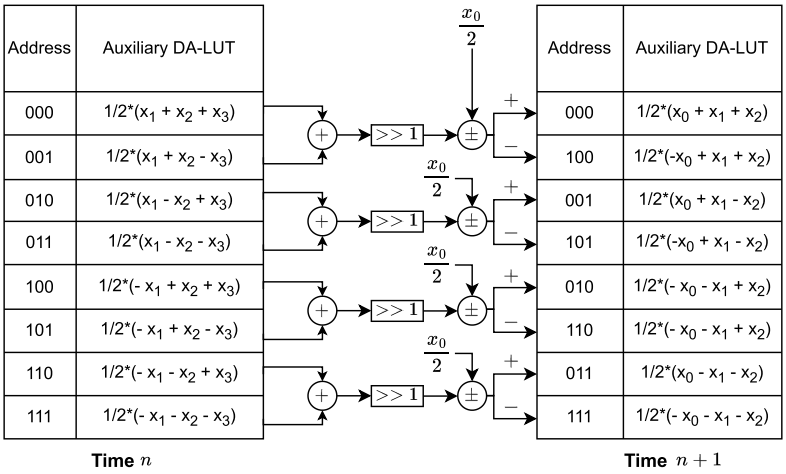

FIGURE 13. Direct DA-ADF OBC update mechanism [26].

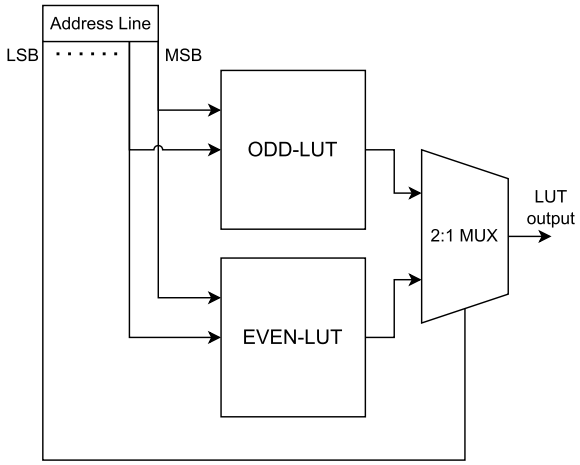

FIGURE 14. Illustration of even-odd LUT decomposition.

with the same term and fill the lower half of the updated LUT. It is important to note that this LUT updates are done serially, which means that for every clock cycle only one address is generated and only one LUT content is updated. Thus, LUT update process may take a considerable amount of time when higher order filters are implemented.

Presented structures and LUT contents are mainly for ADF with filter length of 4. To design a higher order system mostly a base filtering unit is defined, for example length 4 DA filter. By connecting the outputs of the base units via an adder tree an efficient implementation can be achieved. As the direct realization of 16 length filter would require 2^{16} words to be stored and an array of 4 base units of length 4 would require $4 \times 2^4 = 2^2 \times 2^4 = 2^6$ memory locations. An example of length 16 DA filter implemented with base unit is shown in Fig. 11. Parallel-Input-Serial-Output (PISO) register receives the current input sample and outputs it bit by bit, which is essentially needed to form an address. Serial-Input-Serial-Output (SISO) register synchronously passes the input sample bits one by one. This series of PISO and SISO registers essentially implements the delay line.

2) SLIDING BLOCK DA-ADF

Another work that used the efficiency of conjugate DA has been proposed in [24]. In the SB-DA the property of slowly changing input vector was utilized. As only the oldest input sample is discarded and the newest sample is added. DA blocks that contain combinations of intermediate samples may remain unchanged. Only the addresses should be circularly shifted accordingly to get a valid output. The SB-DA implies subdivision of the DA LUTs into several smaller LUTs and one extra used for update. An extension of the SB-DA from TC-DA to OBC-DA was presented in [25]. The SB-DA, despite its name, works on sample by sample basis. But the mechanism of reusal of LUTs with combinations of intermediate samples is widely used in DA based BLMS ADF implementations to efficiently update LUTs.

The basic idea of the SB-DA for length 4 filter is presented in Fig. 12. Two smaller DA LUTs with one extra update LUT is needed for the SB-DA. The main idea is that for several iteration (two for example) the contents of two DA LUTs do not change. Only the weights that are used to access the contents left shift every iteration. The DA LUT with the oldest sample is initialized to zeros and is assigned to be an update LUT. After, it accepts newest sample at each iteration until one of the DA LUTs with the oldest sample will be emptied. MSB-DA [25] has the same main idea of achieving efficiency, but DA LUTs contain OBC-DA combinations. Regarding the overall structure, the outputs of several DA LUTs are combined by an adder tree as shown Fig. 11. Also, an extra logic is needed to rotate the weights, so that appropriate weight bits access the corresponding DA LUT. The error computation and weight update is essentially done the same way as in [23].

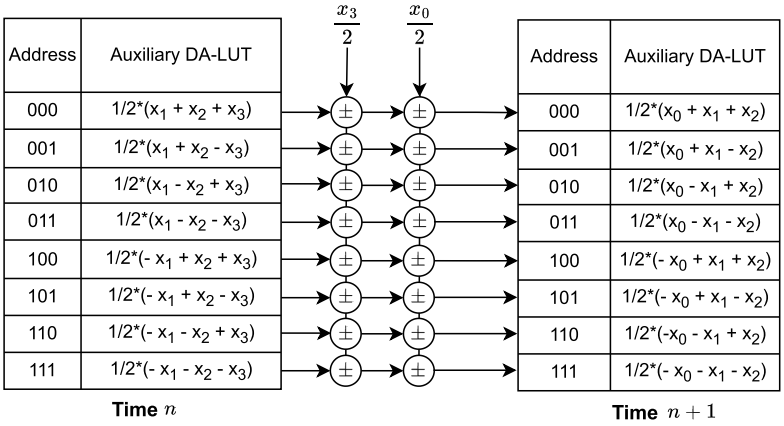

FIGURE 15. OBC-DA auxiliary LUT update without address rotation [28].

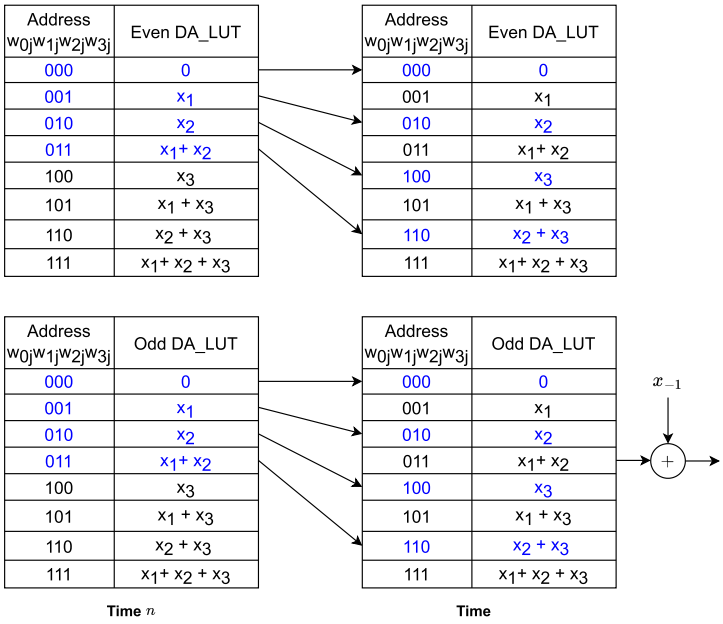

FIGURE 16. Conjugate TC-DA even-odd LUT update [30].

3) DIRECT DA IMPLEMENTATIONS

The direct or traditional DA-ADF structure of [21] have been further improved in [26], [27], [28], and [29]. An OBC-DA implementation of the design in [21] has been proposed in [26]. The update mechanism for the OBC auxiliary LUT is presented in Fig. 13.

The average of even and subsequent odd address, contents results in the elimination of the oldest sample. The block with $\gg 1$ represents the right shift, which is essentially a division by two. A scaled version of the current sample, stored in external register, is subtracted and added for odd and even addresses, respectively. After the update the new sample is stored into the external register. However, this update method has a drawback which is related to the address rotation. As it can be noticed from Fig. 13, the contents of updated auxiliary LUT are at different locations. This means that some amount of address rotation and extra circuitry is needed. Another proposition in [26] is to implement an even-odd LUT decomposition as illustrated in Fig. 14. The LSB of the address is the bit that actually decides if the address is odd or even. Thus, it was used to switch between the outputs of both LUT as a control bit to the 2:1 multiplexer. In general, LUT decompositions help to reduce the LUT size and reduce the access time when high order filters are implemented and large size LUTs are used.

The issue with address rotation has been addressed in [28]. The rest of the design staying the same, i.e. direct OBC-DA with even-odd LUT decomposition, only the auxiliary LUT update has been modified. The modified OBC-DA auxiliary LUT update method is shown in Fig. 15. Now, each

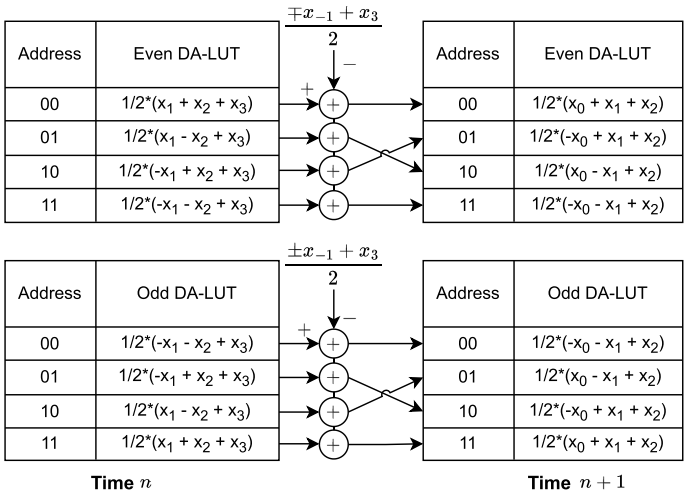

FIGURE 17. Conjugate OBC-DA even-odd LUT update [30].

LUT entry is directly updated by removing the oldest sample and introducing the current sample. Keeping in mind that the newest sample x_{-1} is stored in a separate register after the update. Still, the OBC-DA LUT update mechanism requires several adder and subtract units, which increases the complexity and critical path. A TC-DA design as an improvement of original [21] design has been implemented with the even-odd LUT decomposition in [27]. TC-DA auxiliary LUT is updated with the efficient update method illustrated in Fig. 7. The OBC-DA auxiliary LUT update proposed in [23] and illustrated in Fig. 10 have been used for the direct DA structure without LUT decomposition in [29] due to the apparent reduction of area and power consumption.

4) CONJUGATE DA IMPLEMENTATIONS

A more recent work on non-pipelined DA-ADF proposed two conjugate DA designs for TC and OBC-DA with even-odd LUT decomposition [30]. The even and odd LUT contents are identical. This property can be comprehended if the efficient LUT update mechanism without decomposition is considered [21]. By observing the update method shown in Fig. 7, the even and odd addresses are almost identical with one difference being the addition of the current sample to the odd addresses. As shown in Fig. 16, such even-odd LUT decomposition allows addition of new sample to be done at the output of the odd DA LUT.

An interesting feature to be noticed is the same efficient TC LUT update being applied to even-odd decomposed LUT. The OBC-DA update is shown in Fig. 17. Again, the idea is to remove the oldest sample. The odd OBC-DA LUT contents are negated so that a oldest sample value is positive for easy removal. The sign of the contents in OBC-DA are flexible as the MSB of the address controls the addition or subtraction at the output of the LUT, it can be easily inverted or adjusted. The main advantage of both of these TC and OBC designs is that the even and odd LUTs are updated concurrently. Thus, not only the access time is reduced but, also the clock cycles required for LUT update.

5) APPROXIMATE DA IMPLEMENTATION

Another implementation method applies a new design paradigm of Approximate Computing (AxC) for energy and hardware efficient implementations [31]. This method is mainly applied in inherently error resilient and data-rich applications such as signal processing, machine learning, image processing and computer vision. AxC offers a trade-off situation for accuracy and hardware resources. In order to implement approximate ADF approximate circuits are used such as approximate adders, multipliers, and dividers [32].

Approximate DA-ADF implementation has been presented in [33]. Fig. 18 depicts architecture of the error computation module. This architecture uses Booth encoder [34] and Partial Product Generation (PPG) for multiplication without the partial product accumulation. Partial product accumulation is done via a Wallace tree [35] which adds up all partial products from all the taps. Wallace tree outputs two parts of the sum which is then needs to be summed by another Wallace tree or a Carry Lookahead Adder (CLA). Weight update module, illustrated in Fig. 19, has: Booth encoder and PPGs, and a smaller Wallace tree and a CLA for partial product accumulation at each tap. Essentially, due to the method of partial product accumulation this design can be thought of as distributed.

B. PIPELINED DA ADFs

Pipelining is mainly applied to reduce the critical-path and achieve high throughput as dividing the process in pipelining stages allows concurrent processing. Throughput being the number of samples processed in a second and critical-path

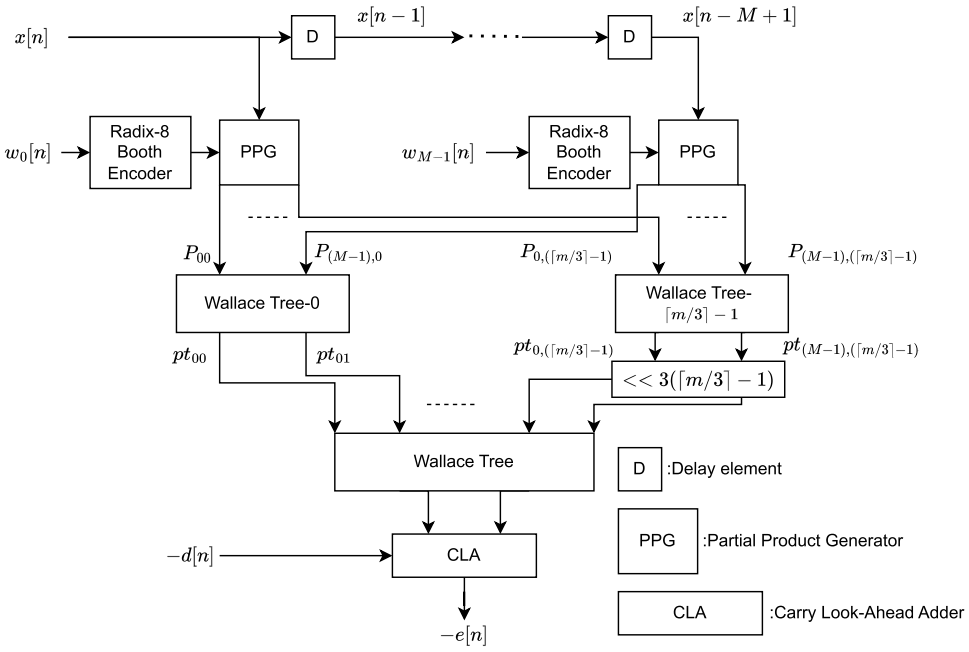

FIGURE 18. Block diagram of approximate DA-ADF error computation module [33].

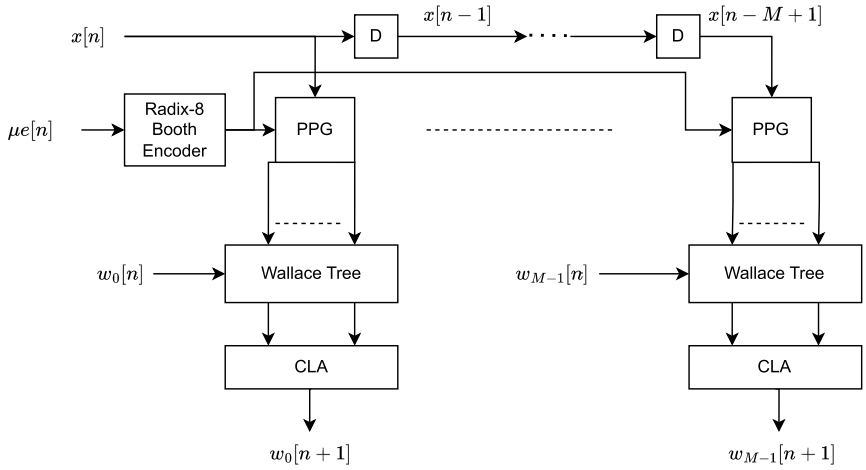

FIGURE 19. Block diagram of approximate DA-ADF weight update module [33].

being the longest signal path through combinational logic. However, due to the recursive nature of adaptive algorithms pipelining actually degrades performance. For the LMS algorithm performance is only slightly degraded for low adaptation delays [36]. First DA-ADF pipelined design has been proposed in [37] based on TC direct DA structure. However, later an improved TC conjugate DA structure pipelined ADF design has been proposed [38]. Two pipelining delay elements have been applied to separate the filtering, error computation, and weight update stages. Thus, the performance of the ADF are not severely degraded. To increase the throughput following main modifications has been applied: register based LUT and Carry Save Adder (CSA) based accumulation. An example of register based LUT for length 3 filter is demonstrated in Fig. 20. The outputs on the right are precisely TC-DA combinations. An array of delay elements, which are realized via registers, store the DA combination computed previously. With this design the update of the filtering LUT can be done in one clock cycle compared to the non-pipelined designs. However, the complexity of the LUT increases.

The accumulation of the DA partial products are done via a CSA to reduce the critical path of a ripple carry adder, which is usually used for addition. The carry propagation is

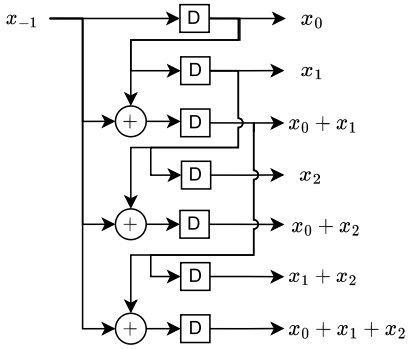

FIGURE 20. Register based LUT for a lenth 3 DA filter.

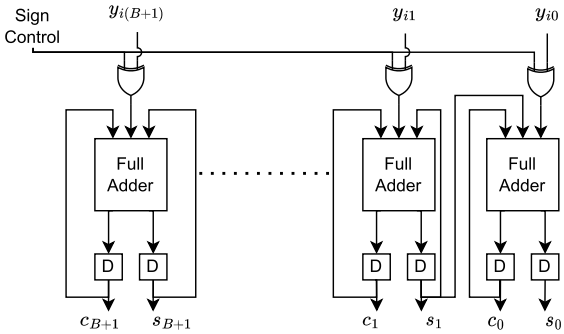

FIGURE 21. Carry save adder accumulator.

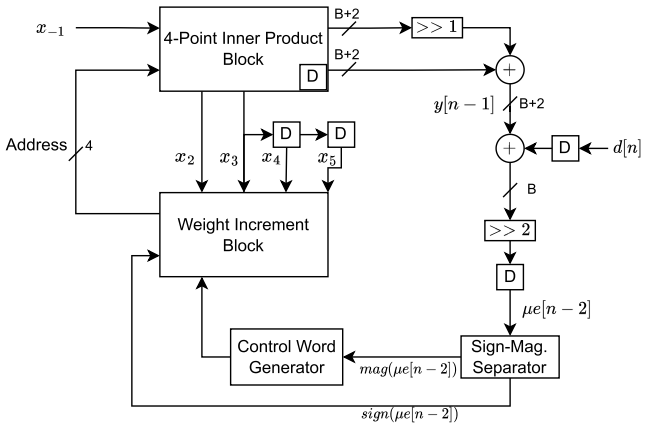

FIGURE 22. Block diagram of length 4 pipelined DA-ADF [38].

essentially postponed for later stage. The structure of CSA is illustrated in Fig. 21, where y_{ij} is some partial product bit j. These partial product bits pass through XOR gates, when MSBs arrive, the sign bit becomes high and the XOR gates behave like switchable NOT gates inverting the incoming most significant partial product. Thus, obtaining one's complement. To obtain two's complement 1 is fed as an initial carry to the least significant adder of the summer outside of the accumulation block.

Fig. 22 illustrates the overall structure of the conjugate pipelined DA-ADF from [38]. Due to the conjugate DA structure only one filtering LUT is used. Two output signals from the filtering or inner product block are due to the CSA based accumulation. Little delay element inside the inner

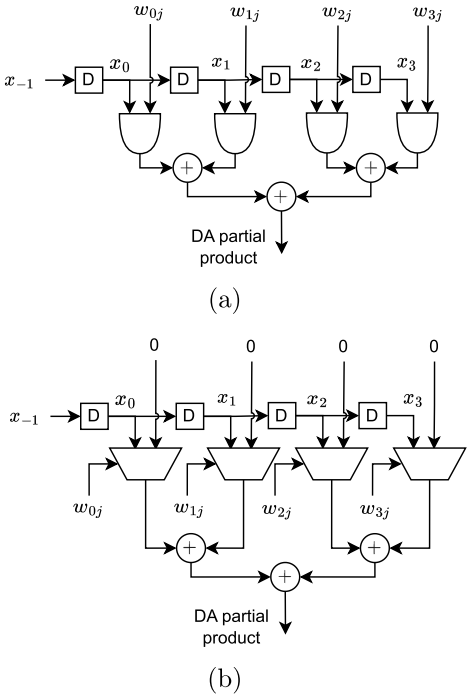

FIGURE 23. (a) Multiplexed, and (b) AND cell TC-DA LUTs [40].

product block indicate that at the output a delay element is present. This delay element or register separates the accumulation stage from the error computation stage. The final output of the accumulation is delayed output sample $y[n-1]$. Thus, the desired response should also be delayed accordingly. The result of this comparison is then right shifted two times. Which means that the step size is chosen to be $\mu = 1/N = 1/4$. Two right shift in binary number representation perform division by 4. After scaling another delay element is inserted to separate error computation from the weight increment or weight update block. The oldest input sample is further delayed to match the delay of the error sample as per the update rule of DLMS. The weight update block is done the similar way as conjugate TC-DA design [23] shown in Fig. 9.

The register-based LUT introduces several adders and a considerable amount of registers. Thus, pipelined DA-ADF designs with multiplexed LUT and AND cell LUT have been proposed in [40]. Both multiplexing and AND operations are used to select a TC-DA combination of input samples which are then summed by an adder tree to output the correct DA partial product. Length 4 multiplexed LUT and AND cell LUT are illustrated in Fig. 23, where w_{ij} is j^{th} bit of i^{th} weight. In both cases, if weight bit is zero, corresponding input sample will not be present in the final TC-DA combination. This LUT design uses less adders and less registers. However, the critical path is increased due to the adder tree.

Three pipelined OBC-DA ADF designs have been proposed in [39]. These design mainly differ in the OBC-DA LUT structure which are demonstrated in Fig. 24. First

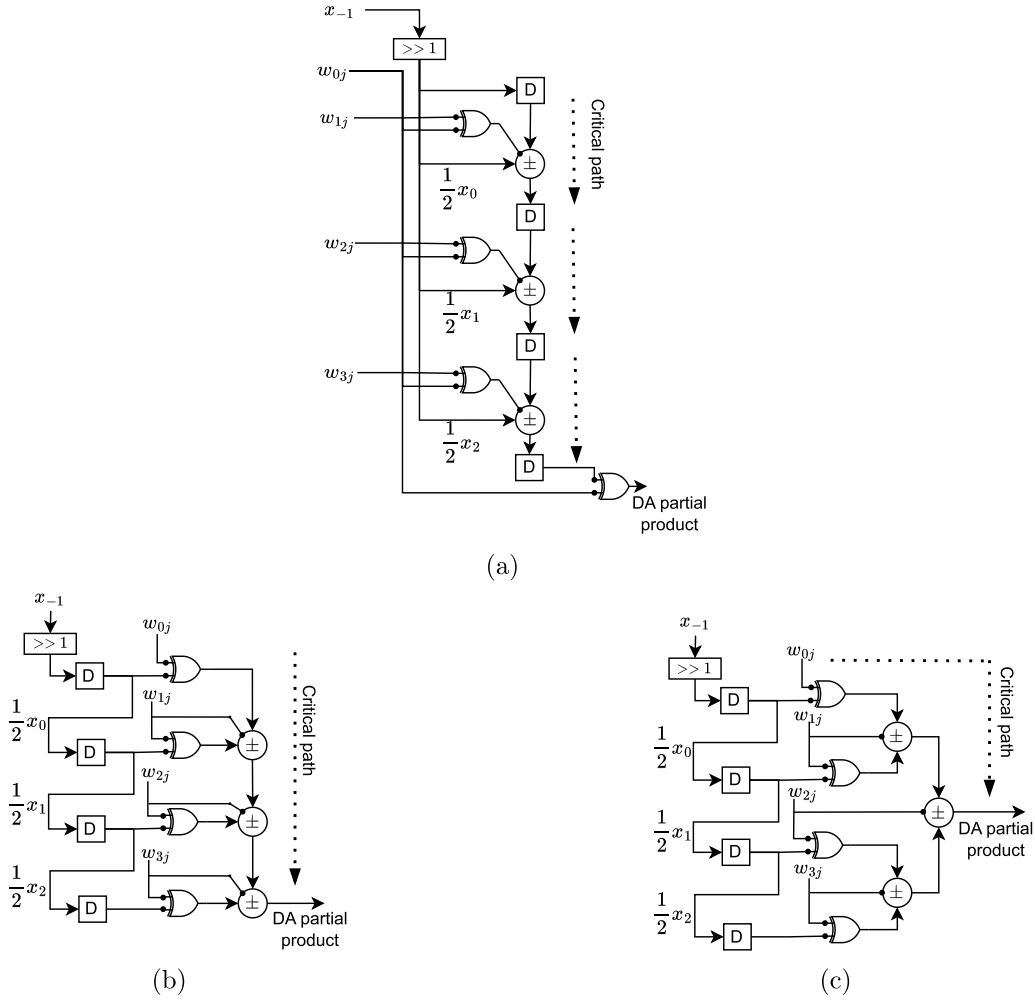

FIGURE 24. (a) OBC-DA LUT Type 1, (b) OBC-DA LUT Type 2, and (c) OBC-DA LUT Type 3 conjugate OBC-DA LUT designs [39].

design, actually resembles the transpose FIR filter structure. The main feature of this structure is that the critical path is minimized. However, the timing issues occur as this design cannot be driven by iteration clock and bit clock simultaneously. Second design can be operated with one clock. It has less registers, and less XOR gates at bit level, but he critical path increases linearly with filter order. Third design employ an adder tree which reduces the critical path as it will increase logarithmic with filter order. In addition to OBC-DA LUT designs, a radix-4 OBC encoding have been proposed in [39] to reduce the number of partial products. Due to higher radix encoding the CSA accumulator have been improved with minus-minus-plus redundant adder to reduce the critical path.

C. BLOCK PROCESSING DA ADFs

1) DA FORMULATION OF BLMS

Matrix-vector multiplications in the BLMS algorithm should be converted to vector multiplications to implement it with DA method. First, the input block matrix $\mathbf{X}[n]$ of size $(L \times N)$ is decomposed into $M = N/L$ square matrices $\mathbf{A}_j[n]$ of size $(L \times L)$ that are defined as:

$$\mathbf{A}_j[n] = \begin{bmatrix} x[j'L] & x[j'L-1] & \cdots & x[j'L-L+1] \\ x[j'L-1] & x[j'L-2] & \cdots & x[j'L-L] \\ \vdots & \vdots & \ddots & \vdots \\ x[j'L-L+1] & x[j'L-L] & \cdots & x[j'L-2L-2] \end{bmatrix}, \tag{26}$$

where $j' = n - j$. The weight and weight increment vectors should be also both decomposed into M smaller vectors of length L as follows:

$$\mathbf{f}_j[n] = [w_{jL}[n]\ w_{jL+1}[n]\ \cdots\ w_{jL+L-1}[n]]^T, \tag{27}$$

$$\Delta\mathbf{f}_j[n] = [\Delta w_{jL}[n]\ \Delta w_{jL+1}[n]\ \cdots\ \Delta w_{jL+L-1}[n]]^T. \tag{28}$$

Both vectors $\mathbf{f}_j[n]$ and $\Delta\mathbf{f}_j[n]$ are of length L, and $j = 0, \cdots M-1$. Then, the filter output and weight increment can be computed as follows according to (18) and (22):

$$\mathbf{y}[n] = \sum_{j=0}^{M-1} \mathbf{A}_j[n]\mathbf{f}_j[n], \tag{29}$$

$$\Delta\mathbf{f}_j[n] = \mu\mathbf{A}_j[n]\mathbf{e}[n]. \tag{30}$$

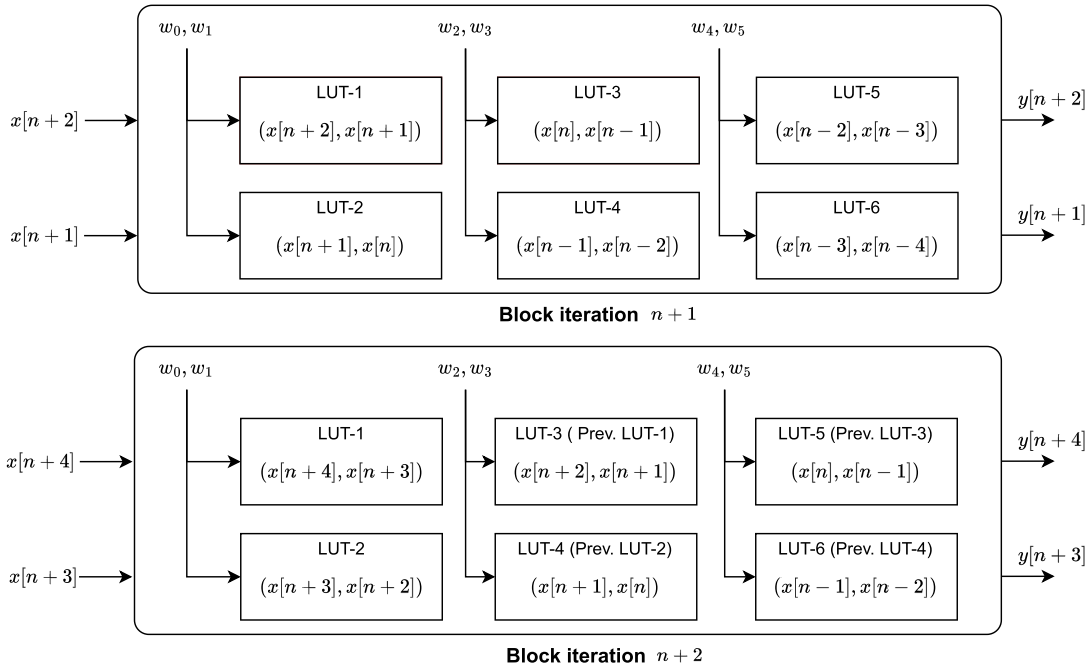

FIGURE 25. Block processing DA-ADF working principle illustration [46].

In (30), the transpose is omitted because matrix $\mathbf{A}_j[n]$ is symmetrical by definition in (26). The common square matrix $\mathbf{A}_j[n]$ can be decomposed into row vectors as:

$$\mathbf{a}_{ij}[n] = [x[j'L - i]\, x[j'L - 1 - i]\, \cdots\, x[j'L - L + 1 - i]], \tag{31}$$

where $i = 0, \cdots L-1$, and $\mathbf{a}_{ij}[n]$ is the $(i+1)$th row vector of the matrix $\mathbf{A}_j[n]$. Filter output sample and weight increment element can be computed as follows:

$$y[nL - i] = \sum_{j=0}^{M-1} \mathbf{a}_{ij}[n]\mathbf{f}_j[n], \tag{32}$$

$$\Delta w_{jL+i}[n] = \mu \mathbf{a}_{ij}[n]\mathbf{e}[n]. \tag{33}$$

As a result (32) and (33) involve vector multiplication that can be implemented with DA method. Conjugate DA structure fits BLMS DA the most due to common row vector $\mathbf{a}_{ij}[n]$. Thus, storing the DA combinations of the input block samples in LUTs and access them by addresses formed by weights and error samples is the most efficient approach.

2) BLOCK PROCESSING DA ADFs

Initial DA BLMS design have been based on conjugate TC-DA structure due to the two matrix-vector multiplications (33), (32) involving the same input sample block [46]. Hence, the same collection of LUTs containing the DA combinations of input sample can be used for filtering and weight increment computation processes. An example of BLMS DA-ADF with filter length $N = 6$ and block length $L = 2$ is illustrated in Fig. 25. After every new block of data and block iteration LUTs are essentially right shifted. The right most LUT with the oldest samples is discarded and a LUT with the newest samples is formed in the left most side. Physically transport LUT contents is an inefficient process. Instead it is better to rotate the weights that are actually static in Fig. 25. Thus, the oldest LUT is always overwritten with newest sample, while other LUTs remain the same. This technique is called inter-iteration LUT reuse.

Yet another, technique applied in BLMS DA is LUT sharing. From Fig. 25, one sample overlap can be observed between subsequent LUTs. The meaning is that some of the contents are identical and can be shared to reduce hardware. To be able to tap specific LUT contents only the register based LUT shown in Fig. 20. So called, intra-iteration LUT sharing has been first proposed in [47]. An OBC BLMS DA implementation of this scheme have been proposed in [48]. By applying recursive OBC scheme the LUT sizes have been reduced by a factor of 4. Thus, reducing the LUT access time, area, and power. However, as the recursive OBC increases the critical path delay another design have been proposed in [49]. Initial OBC LUT is divided into two smaller LUTs based on the address reducing the LUT access time. In [50] the notion of intra-iteration LUT sharing has been applied to OBC BLMS DA architecture in form of two composite-LUT designs that support LUT sharing. In [51] the previous design has been further optimized. The redundancies of upper and lower LUT decomposition have been exploited further in context of intra-iteration LUT sharing.

TABLE 2. Summary of DA-ADF designs.

Non-pipelined desings			
Article	DA type	DA Structure	Key features
[21]	TC-DA	Direct DA	Efficient auxiliary LUT update mechanism, Fig. 7, 8.
[22]	TC-DA	Conjugate DA	Single filtering LUT design.
[23]	TC-DA and OBC-DA	Conjugate DA	OBC-DA extension of single LUT design, Fig. 9. OBC LUT update mechanism illustrated in Fig. 10.
[24], [25]	TC-DA and OBC-DA	SB-DA	Sub-filtering LUT update with inter-iteration LUT reuse and weight rotation, Fig. 12.
[26]	OBC-DA	Direct DA	OBC extension of [21]. The recent sample stored in external register, Fig. 13. Introduction of even-odd LUT decomposition, Fig. 14.
[27]	TC-DA	Direct DA	Improvement of [21] with even-odd LUT decomposition.
[28]	OBC-DA	Direct DA	Resolved the issue with address rotation in [26] with new OBC LUT update, Fig. 15.
[29]	OBC-DA	Direct DA	Implementation of [26] with OBC LUT update of [23] and without even-odd LUT decomposition.
[30]	TC-DA and OBC-DA	Conjugate DA	Even-odd LUT decomposition applied to conjugate DA structure, Fig. 16, 17.
[33]	-	-	Approximate ADF implementation that cannot be thought of as classical DA-ADF implementation, Fig. 18, 19.
Pipelined desings			
Article	DA type	DA Structure	Key features
[37]	TC-DA	Direct DA	Introduction of pipelining, register-based LUTs, and CSA accumulator, Fig. 20, 21.
[38]	TC-DA	Conjugate DA	Restructuring of [37], Fig. 22.
[39]	OBC-DA	Conjugate DA	Three OBC LUT design have been proposed, Fig. 24. The CSA accumulation have been improved to handle radix-4 OBC encoding.
[40]	TC-DA	Conjugate DA	Multiplexed and AND cell LUTs have been proposed to reduce the register count of the register-based LUT, Fig. 23.
[41]	TC-DA	Conjugate DA	AND cell LUT design with pipelining integrated into the adder tree.
[43]	TC-DA	Conjugate DA	Implementation of convex combined ADF with one DA-ADF unit.
Block processing desings			
Article	DA type	DA Structure	Key features
[46]	TC-DA	Conjugate DA	DA BLMS formulation. Introduction of inter-iteration LUT reuse.
[47]	TC-DA	Conjugate DA	Introduction of intra-iteration LUT sharing.
[48]	OBC-DA	Conjugate DA	Implementation of [46] with recursive OBC-DA scheme .
[49]	OBC-DA	Conjugate DA	Improvement of [48] with even-odd LUT decomposition.
[50]	OBC-DA	Conjugate DA	Intra-iteration LUT sharing applied in context of OBC-DA design.
[51]	OBC-DA	Conjugate DA	Intra-iteration LUT sharing have been further optimized by decomposing LUTs into upper and lower halves.

V. CONCLUSION

In this paper, a review of most important DA-ADF designs has been performed. Three main types of designs have been identified: non-pipelined, pipelined, and block processing. Discussed designs have been summarized into several key features. Non-pipelined designs are the most simplistic and

block processing designs are the most complex to implement. A trend has been observed that each design type converges to conjugate OBC-DA. They do not need any auxiliary LUT and the memory requirement is further reduced by OBC technique. In general, it can be said that first direct TC-DA structures are proposed and then improved by either applying conjugate DA or OBC-DA. Except for the discussed approximate DA-ADF [33], which employed AxC to efficiently implement ADF. Most of the designs only considers LMS-type adaptive algorithms for implementation due to its simplicity and satisfactory performance. No DA-ADF designs has been found that discusses implementation of more advanced adaptive algorithms like Recursive Least Square (RLS).

AxC is a new emerging design philosophy that can benefit hardware-efficient implementations to a high degree. Thus, another possible future work is to apply AxC to already proposed designs to further reduce the hardware cost. Also, AxC can help to mitigate the computational complexities of more advanced adaptive algorithms like RLS and allow a feasible hardware efficient implementation of these algorithms. A systematic summary of designs reviewed in this article is provided in Table 2, which lists key features of various designs for DA-based implementation of ADFs.

Targeting new adaptive algorithms that may exhibit useful symmetries in DA implementation framework is also another feasible direction of future research. One particular example are tensor adaptive algorithms [52]. Due to the assumption of linear separability of the parameter space, estimation of which is the goal of adaptive filtering, some symmetries may arise which can be exploited in DA implementation.

REFERENCES

[1] S. Haykin, *Adaptive Filtering Theory*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[2] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*. Chichester, U.K.: Wiley, 1998.

[3] R. R. Pereira, C. H. da Silva, L. E. B. da Silva, G. Lambert-Torres, and J. O. P. Pinto, "New strategies for application of adaptive filters in active power filters," *IEEE Trans. Ind. Appl.*, vol. 47, no. 3, pp. 1136–1141, May 2011, doi: 10.1109/TIA.2011.2125931.

[4] Q. Li, M. J. Zuo, and S. Y. Liang, "Prognosis of bearing degeneration using adaptive quaternion least mean biquadrate under framework of hypercomplex data," *IEEE Sensors J.*, vol. 20, no. 5, pp. 2659–2670, Mar. 2020, doi: 10.1109/JSEN.2019.2954054.

[5] 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, doi: 10.1016/j.bspc.2019.03.012.

[6] M. T. Akhtar, F. Albu, and A. Nishihara, "Prediction error method (PEM)-based howling cancellation in hearing aids: Can we do better?" *IEEE Access*, vol. 11, pp. 337–364, 2023, doi: 10.1109/ACCESS.2022.3232334.

[7] M. T. Akhtar, "Developing a new filtered-X recursive least squares adaptive algorithm based on a robust objective function for impulsive active noise control systems," *Appl. Sci.*, vol. 13, no. 4, p. 2715, Feb. 2023, doi: 10.3390/app13042715.

[8] M. Fujita, "Adaptive filter model of the cerebellum," *Biol. Cybern.*, vol. 45, no. 3, pp. 195–206, Oct. 1982.

[9] A. Gebhard, O. Lang, M. Lunglmayr, C. Motz, R. S. Kanumalli, C. Auer, T. Paireder, M. Wagner, H. Pretl, and M. Huemer, "A robust nonlinear RLS type adaptive filter for second-order-intermodulation distortion cancellation in FDD LTE and 5G direct conversion transceivers," *IEEE Trans. Microw. Theory Techn.*, vol. 67, no. 5, pp. 1946–1961, May 2019.

[10] D. Allred, V. Krishnan, W. Huang, and D. Anderson, "Implementation of and LMS adaptive filter on an FPGA employing multiplexed multiplier architecture," in *Proc. Asilomar Conf. Signals Syst. Comput.*, Nov. 2003, pp. 918–921.

[11] 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.

[12] A. Croisier, D. Esteban, M. Levilion, and V. Rizo, "Digital filter for PCM encoded signals," U.S. Patent 3 777 130, Dec. 4, 1973.

[13] A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-22, no. 6, pp. 456–462, Dec. 1974.

[14] C. F. N. Cowan and J. Mavor, "New digital-adaptive filter implementation using distributed-arithmetic techniques," *IEE Proc. F, Commun., Radar Signal Process.*, vol. 128, no. 4, pp. 225–230, Aug. 1981.

[15] 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. ASSP-31, no. 3, pp. 541–549, Jun. 1983.

[16] C. H. Wei and J. J. Lou, "Multimemory block structure for implementing a digital adaptive filter using distributed arithmetic," *IEE Proc. G-Electron. Circuits Syst.*, vol. 133, no. 1, pp. 19–26, Feb. 1986.

[17] Y. Chiu and C. Wei, "On the realization of multimemory block structure digital adaptive filter using distributed arithmetic," *J. Chin. Inst. Eng.*, vol. 10, no. 1, pp. 115–122, Jan. 1987.

[18] G. Sicuranza and G. Ramponi, "Adaptive nonlinear digital filters using distributed arithmetic," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 3, pp. 518–526, Jun. 1986.

[19] M. J. Smith, C. F. N. Cowan, and P. F. Adams, "Nonlinear echo cancellers based on transpose distributed arithmetic," *IEEE Trans. Circuits Syst.*, vol. 35, no. 1, pp. 6–18, Jan. 1988.

[20] 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.

[21] 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.

[22] W. Huang, V. Krishnan, and D. V. Anderson, "Conjugate distributed arithmetic adaptive FIR filters and their hardware implementation," in *Proc. 49th IEEE Int. Midwest Symp. Circuits Syst. (MWSCAS)*, vol. 2, Aug. 2006, pp. 295–299.

[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] D. L. Jones, "Efficient computation of time-varying and adaptive filters," *IEEE Trans. Signal Process.*, vol. 41, no. 3, pp. 1077–1086, Mar. 1993.

[25] W. Huang and D. V. Anderson, "Modified sliding-block distributed arithmetic with offset binary coding for adaptive filters," *J. Signal Process. Syst.*, vol. 63, no. 1, pp. 153–163, Apr. 2011.

[26] 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, doi: 10.1109/TCSII.2013.2281747.

[27] 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)*, Hyderabad, India, Jan. 2017, pp. 127–132, doi: 10.1109/VLSID.2017.16.

[28] M. T. Khan and S. R. Ahamed, "A new high performance VLSI architecture for LMS adaptive filter using distributed arithmetic," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Bochum, Germany, Jul. 2017, pp. 219–224, doi: 10.1109/ISVLSI.2017.46.

[29] 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)*, Pune, India, Jan. 2018, pp. 283–288, doi: 10.1109/VLSID.2018.77.

[30] M. T. Khan, M. A. Alhartomi, S. Alzahrani, R. A. Shaik, and R. Alsulami, "Two distributed arithmetic based high throughput architectures of non-pipelined LMS adaptive filters," *IEEE Access*, vol. 10, pp. 76693–76706, 2022, doi: 10.1109/ACCESS.2022.3192619.

[31] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. Test Symp. (ETS)*, Avignon, France, May 2013, pp. 1–6, doi: 10.1109/ETS.2013.6569370.

[32] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proc. IEEE*, vol. 108, no. 12, pp. 2108–2135, Dec. 2020, doi: 10.1109/JPROC.2020.3006451.

[33] H. Jiang, L. Liu, P. P. Jonker, D. G. Elliott, F. Lombardi, and J. Han, "A high-performance and energy-efficient FIR adaptive filter using approximate distributed arithmetic circuits," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 1, pp. 313–326, Jan. 2019, doi: 10.1109/TCSI.2018.2856513.

[34] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 booth multipliers for low-power and high-performance operation," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2638–2644, Aug. 2016, doi: 10.1109/TC.2015.2493547.

[35] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964, doi: 10.1109/PGEC.1964.263830.

[36] 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, doi: 10.1109/29.31293.

[37] P. K. Meher and S. Y. Park, "High-throughput pipelined realization of adaptive FIR filter based on distributed arithmetic," in *Proc. IEEE/IFIP 19th Int. Conf. VLSI Syst.-Chip*, Hong Kong, Oct. 2011, pp. 428–433, doi: 10.1109/VLSISoC.2011.6081621.

[38] S. Y. Park and P. K. Meher, "Low-power, high-throughput, and low-area adaptive FIR filter based on distributed arithmetic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 6, pp. 346–350, Jun. 2013, doi: 10.1109/TCSII.2013.2251968.

[39] M. T. Khan and R. A. Shaik, "Optimal complexity architectures for pipelined distributed arithmetic-based LMS adaptive filter," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 2, pp. 630–642, Feb. 2019, doi: 10.1109/TCSI.2018.2867291.

[40] M. T. Khan and S. R. Ahamed, "VLSI realization of low complexity pipelined LMS filter using distributed arithmetic," in *Proc. IEEE Region Conf. (TENCON)*, Penang, Malaysia, Nov. 2017, pp. 433–438, doi: 10.1109/TENCON.2017.8227903.

[41] M. T. Khan and R. A. Shaik, "High-performance VLSI architecture of DLMS adaptive filter for fast-convergence and low-MSE," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 4, pp. 2106–2110, Apr. 2022, doi: 10.1109/TCSII.2022.3141687.

[42] J. Arenas-Garcia, A. R. Figueiras-Vidal, and A. H. Sayed, "Mean-square performance of a convex combination of two adaptive filters," *IEEE Trans. Signal Process.*, vol. 54, no. 3, pp. 1078–1090, Mar. 2006, doi: 10.1109/TSP.2005.863126.

[43] M. T. Khan, R. A. Shaik, and S. P. Matcha, "Improved convergent distributed arithmetic based low complexity pipelined least-mean-square filter," *IET Circuits, Devices Syst.*, vol. 12, no. 6, pp. 792–801, Nov. 2018.

[44] P. C. Shrivastava, P. Kumar, M. Tiwari, and A. Dhawan, "Efficient architecture for the realization of 2-D adaptive FIR filter using distributed arithmetic," *Circuits, Syst., Signal Process.*, vol. 40, no. 3, pp. 1458–1478, Mar. 2021, doi: 10.1007/s00034-020-01539-y.

[45] S. Baghel and R. Shaik, "FPGA implementation of fast block LMS adaptive filter using distributed arithmetic for high throughput," in *Proc. Int. Conf. Commun. Signal Process.*, Kerala, India, Feb. 2011, pp. 443–447, doi: 10.1109/ICCSP.2011.5739356.

[46] 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," *IEEE Trans. Signal Process.*, vol. 61, no. 4, pp. 921–932, Feb. 2013, doi: 10.1109/TSP.2012.2226453.

[47] B. K. Mohanty, P. K. Meher, and S. K. Patel, "LUT optimization for distributed arithmetic-based block least mean square adaptive filter," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 5, pp. 1926–1935, May 2016, doi: 10.1109/TVLSI.2015.2472964.

[48] M. T. Khan and R. A. Shaik, "Analysis and implementation of block least mean square adaptive filter using offset binary coding," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, May 2018, pp. 1–5, doi: 10.1109/ISCAS.2018.8350946.

[49] M. T. Khan and R. A. Shaik, "High-performance hardware design of block LMS adaptive noise canceller for in-ear headphones," *IEEE Consum. Electron. Mag.*, vol. 9, no. 3, pp. 105–113, May 2020, doi: 10.1109/MCE.2020.2976418.

[50] M. T. Khan, J. Kumar, S. R. Ahamed, and J. Faridi, "Partial-LUT designs for low-complexity realization of DA-based BLMS adaptive filter," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 4, pp. 1188–1192, Apr. 2021, doi: 10.1109/TCSII.2020.3035693.

[51] M. T. Khan, R. A. Shaik, and M. A. Alhartomi, "An efficient scheme for acoustic echo canceller implementation using offset binary coding," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–14, 2022, doi: 10.1109/TIM.2021.3132087.

[52] L.-M. Dogariu, C.-L. Stanciu, C. Elisei-Iliescu, C. Paleologu, J. Benesty, and S. Ciochină, "Tensor-based adaptive filtering algorithms," *Symmetry*, vol. 13, no. 3, p. 481, Mar. 2021, doi: 10.3390/sym13030481.

SANZHAR YERGALIYEV (Student Member, IEEE) was born in Sary-Ozek, Kazakhstan, in 2001. He received the B.Eng. degree in electrical and computer engineering from the School of Engineering and Digital Sciences, Nazarbayev University, Astana, in 2023. Since 2021, he has been a Research Assistant with the Applications of Signal Processing Laboratory (ASP-LAB), Nazarbayev University. His research interests include the study of adaptive filtering theory and the hardware-efficient implementation of adaptive digital filters. The current manuscript is based on his final-year Capstone Project.

MUHAMMAD TAHIR AKHTAR (Senior Member, IEEE) received the B.Sc. degree in electrical electronics and communication engineering from the University of Engineering and Technology Taxila, Taxila, Pakistan, in 1997, the M.Sc. degree in systems engineering from Quaid-i-Azam University, Islamabad, Pakistan, in 1999, and the Ph.D. degree in electronic engineering from Tohoku University, Sendai, Japan, in 2004.

From 2004 to 2005, he was a COE Postdoctoral Fellow of Tohoku University. From 2006 to 2008, he was an Assistant Professor with United Arab Emirates University. From December 2008 to February 2009, he was a Visiting Researcher with the Institute of Sound and Vibration Research (ISVR), University of Southampton, U.K. From 2008 to 2014, he was an Assistant Professor with The University of Electro-Communications, Tokyo, Japan, and a Special Visiting Researcher with the Tokyo Institute of Technology, Tokyo. From November 2010 to March 2011, he was with the Institute for Neural Computations (INC), University of California at San Diego. From 2014 to 2017, he was an Associate Professor with COMSATS University Islamabad, Pakistan. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, School of Engineering and Digital Sciences, Nazarbayev University, Astana, Kazakhstan. He has published about 110 papers in peer-reviewed international journals and conference proceedings. His research interests include adaptive signal processing, active noise control, blind source separation, and biomedical signal processing.

Dr. Akhtar is a member of the IEEE Signal Processing Society and the IEEE Industrial Electronic Society. He was awarded the Certificate of Best Presenter at the IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEEE IEMCON 2019), Vancouver, Canada, the Best Student Paper Award at the IEEE 2004 Midwest Symposium on Circuits and Systems (IEEE MWSCAS 2004), Hiroshima, Japan, and a Student Paper Award (with Marko Kanadi) at 2010 RISP International Workshop on Nonlinear Circuits, Communications, and Signal Processing. He was on the editorial board of *Advances in Mechanical Engineering*, and served as a Co-Editor for the newsletter of the Asia–Pacific Signal and Information Processing Association (APSIPA) (2011–2013). Since 2023, he has been a Full Member of Sigma Xi.