Novel VLSI Architecture for Fractional-Order Correntropy Adaptive Filtering Algorithm

Daney Alex, Vinay Chakravarthi Gogineni, *Member, IEEE*, Subrahmanyam Mula, *Member, IEEE*, and Stefan Werner, *Senior Member, IEEE*

***Abstract*— Conventional adaptive filters, which assume Gaussian distribution for signal and noise, exhibit significant performance degradation when operating in non-Gaussian environments. Recently proposed fractional-order adaptive filters (FoAFs) address this concern by assuming that the signal and noise are symmetric α-stable random processes. However, the literature does not include any VLSI architectures for these algorithms. Toward that end, this article develops hardware-efficient architecture for fractional-order correntropy adaptive filter (FoCAF). We first reformulate the FoCAF for its efficient real-time VLSI implementation and then demonstrate that these reformulations cause negligible performance degradation under the 16-bit fixed-point implementation. Using this reformulated algorithm, we design an FoCAF architecture. Furthermore, we analyze the critical path of the design to select the appropriate level of pipelining based on the sampling rate of the application. According to the critical-path analysis, the FoCAF design is pipelined using retiming techniques to obtain delayed FoCAF (DFoCAF), which is then synthesized using 45-nm CMOS technology. Synthesis results reveal that DFoCAF architecture requires a minimal increase in hardware over the prominent least mean square (LMS) filter architecture and achieves a significant increase in the performance in symmetric α-stable environments where LMS fails to converge.**

***Index Terms*— α-Stable signals, adaptive filters, fractional-order correntropy criterion, logarithmic number system, VLSI architectures.**

I. Introduction

IN REAL-WORLD applications, adaptive signal processing algorithms are commonly used to learn the underlying unknown system from a limited number of observations [1]. In order to achieve mathematical tractability and computational efficiency, these algorithms often consider a Gaussian statistical model for signal and noise [2]. The Gaussian assumption on input and noise signals, however, is not the best choice in a large number of modern applications, such as seismic activity, climatology and weather, ocean wave variability, acoustic emissions from cracks growing in engineering materials under stress [3], underwater acoustics [4], wideband communications [5], financial data modeling [6], and neuroimage processing [7], in which the signals exhibit sharp spikes. These signals can be effectively modeled by symmetric α-stable (SαS) distributions, having heavier tails than those of Gaussian distributions [8]. Since SαS signals do not have finite second- or high-order moments, the performance of adaptive filters based on minimizing second- or high-order moments of error deteriorates when operating in such an environment [9].

Adaptive filters based on the maximum correntropy criterion (MCC) [10]–[15] demonstrate better performance in a non-Gaussian noise environment compared to conventional adaptive filters. However, MCC adaptive filters cannot replicate this performance when both signal and noise are SαS random processes since the cost function of MCC is defined over second- or high-order moments of the error [16]. To address this issue, fractional-order adaptive filters (FoAFs) that minimize fractional-order errors using fractional-order calculus were proposed in [17]. Although the FoAFs show improved performance over conventional adaptive filters, the FoAFs are sensitive to the characteristic exponent α. Moreover, residual jitters may still be present in their steady-state estimates [16], which is undesirable in real-time implementations. Recently, fractional-order correntropy adaptive filters (FoCAF) were proposed to solve these problems by blending the concepts of fractional-order calculus and a correntropy-type similarity measure [9], [16]. The FoCAF has been demonstrated to be an effective solution for tracking dynamic systems in SαS signal and noise environments [16].

Adaptive filtering algorithms are iterative in nature; thus, their implementation involves a high degree of computational complexity and memory access. Software solutions for adaptive filters cannot meet the power and throughput requirements, especially in real-time applications such as channel estimation [18], where performance is directly influenced by the speed and precision of channel estimation. Thus, tailored VLSI architectures are necessary for such applications to implement adaptive filters in real time [19]. Numerous efforts were made in the literature toward high-performance architectures for adaptive filters and their variants when the

Manuscript received November 15, 2021; revised February 5, 2022; accepted March 31, 2022. This work was supported in part by the Science and Engineering Research Board (SERB), Department of Science and Technology, Government of India, under Startup Research Grant SRG/2020/000858; and in part by the Research Council of Norway. *(Daney Alex and Vinay Chakravarthi Gogineni contributed equally to this work.) (Corresponding author: Subrahmanyam Mula.)*

Daney Alex and Subrahmanyam Mula are with the Department of Electrical Engineering, IIT Palakkad, Kozhippara 678557, India (e-mail: 122003001@smail.iitpkd.ac.in; svmula@iitpkd.ac.in).

Vinay Chakravarthi Gogineni and Stefan Werner are with the Department of Electronic Systems, Norwegian University of Science and Technology (NTNU), 7491 Trondheim, Norway (e-mail: vinay.gogineni@ntnu.no; stefan.werner@ntnu.no).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2022.3169010.

Digital Object Identifier 10.1109/TVLSI.2022.3169010

noise alone is modeled as non-Gaussian [20], [21]. To the best of our knowledge, no architectures have been developed for adaptive filters when input and noise are both modeled as non-Gaussian signals. Although FoCAF exhibits good performance in non-Gaussian signal and noise environments, direct implementation of FoCAF in hardware is highly challenging. Weighted-sum operations and fractional-order operations result in a long critical path. This long critical path limits the throughput in applications such as underwater acoustic channel estimation [2], [22]–[24], where the required operating frequency is in the megahertz and gigahertz range. In order to address this issue, this article proposes several reformulations that minimize the algorithm complexity in a VLSI implementation. The main contributions of this work are summarized as follows.

1) We propose reformulations for the FoCAF algorithm to make it suitable for VLSI implementation, which includes a hardware-friendly scheme for exponentiation functions with base $\in (0, 1]$ using the Maclaurin series and we design a pipelined VLSI architecture for the reformulated FoCAF.
2) A detailed critical-path analysis of the FoCAF architecture is presented. Based on this analysis, we propose an optimal architecture considering the area-delay product as the metric.
3) The proposed delayed FoCAF (DFoCAF) architecture is synthesized for ASIC using 45-nm CMOS technology and the synthesis results are compared with the state-of-the-art pipelined DLMS architecture, to show that the area and power overhead are minimal despite the performance improvement.

The remaining article is organized as follows. Section II briefly reviews SαS signals and FoCAF algorithm. Section III presents the implementation of the exponentiation function, the reformulation of the FoCAF algorithm, and the VLSI architecture design of DFoCAF. Section IV discusses the application-specific integrated circuit (ASIC) synthesis results. Finally, Section V concludes this article.

II. Preliminaries

Consider the system identification problem shown in Fig. 1. Here, the unknown system is modeled by an L-tap coefficient vector $\mathbf{w}_{\text{opt}}$. At time index n, the system takes u_n as input and produces the desired output $d_n = \mathbf{w}_{\text{opt}}^{\text{T}}\mathbf{u}_n + v_n$, where $\mathbf{u}_n = [u_n, u_{n-1}, \ldots, u_{n-L+1})]^{\text{T}}$ is the input signal vector and v_n is the unknown observation noise. The input signal and observation noise are assumed to be zero-mean SαS random processes. The goal of system identification setup is to estimate the unknown system $\mathbf{w}_{\text{opt}}$, given u_n and d_n.

A. SαS Signals

An important class of non-Gaussian phenomena [25] is associated with occasional bursts or sharp spikes present in their realizations. In particular, the SαS distribution can model input signal and noise for such phenomena. These distributions have heavier tails compared to the Gaussian distribution. In the density function of SαS random processes, the characteristic

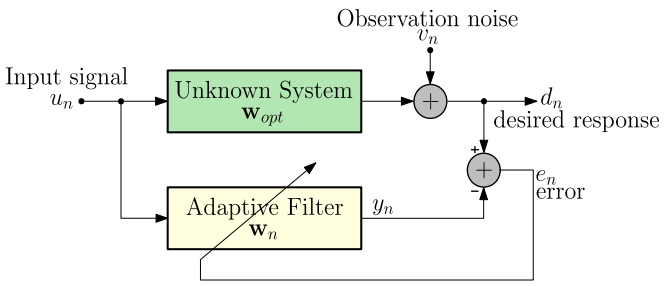


Fig. 1. System identification.

exponent $\alpha \in (0, 2]$ regulates the tail heaviness [16]. For $\alpha = 2$, the density function is Gaussian, and as the value of α decreases, the density function exhibits heavier tails. Except for the Gaussian case, SαS random processes only pose finite statistical moments of orders strictly less than α. Therefore, adaptive filtering algorithms whose derivation relies on a second-order moment of the error measure experience significant performance degradation when both signal and noise are modeled as SαS random processes. As for solutions to filtering, it is implicitly assumed that $\alpha \in (1, 2]$, so conditional expectations may be established. Thus, this article only considers real-valued SαS random processes with $\alpha \in (1, 2]$, without loss of generality. More details can be found in [12], [25], and [26].

B. Fractional-Order Correntropy Adaptive Filter

Conventional adaptive filters that minimize the second-order moment of an error measure exhibit considerable performance degradation in SαS environments due to the absence of high-order moments in the SαS distribution. To solve this problem, a class of adaptive filters based on the concepts of fractional-order calculus has been proposed [17], [27], [28]. FoAFs [9] minimize the fractional-order error measure using fractional-order calculus. The objective function of FoAF in the case of system identification is given by

$$J = \mathrm{E}\Big[e_n\, e_n^{\langle \alpha'-1\rangle}\Big] \tag{1}$$

where $e_n = d_n - y_n$ is the estimated error with $y_n = \mathbf{u}_n^{\text{T}}\mathbf{w}_n$ and $\mathbf{w}_n \in \mathbb{R}^L$ is the estimate of $\mathbf{w}_{\text{opt}}$ at time instance n and the parameter $\alpha' \in (1, \alpha)$. Here, $e_n^{\langle \alpha'-1\rangle}$ is the fractional-order error, given by $|e_n|^{(\alpha'-1)}\,\text{sign}(e_n)$, where sign$(\cdot)$ and $|\cdot|$ denote sign and absolute values of their arguments, respectively. The weight update equation of FoAF is

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu\, e_n\, \mathbf{u}_n^{\langle \alpha'-1\rangle} \tag{2}$$

where μ is the step size. In (2), $\mathbf{u}_n^{\langle \alpha'-1\rangle}$ is the fractional-order input, which is the elementwise implementation of the function $|[\mathbf{u}_n]_i|^{(\alpha'-1)}\,\text{sign}([\mathbf{u}_n]_i)$. The fractional-order scaling of the input signal in the update equation makes it tolerant to jittery SαS input signals.

Although FoAF performs better in α-stable environments, its steady-state estimate may still contain residual jitters. The presence of jitter is due to the impulsive nature of noise, which affects the error in the FoAF update equation. FoCAF solves

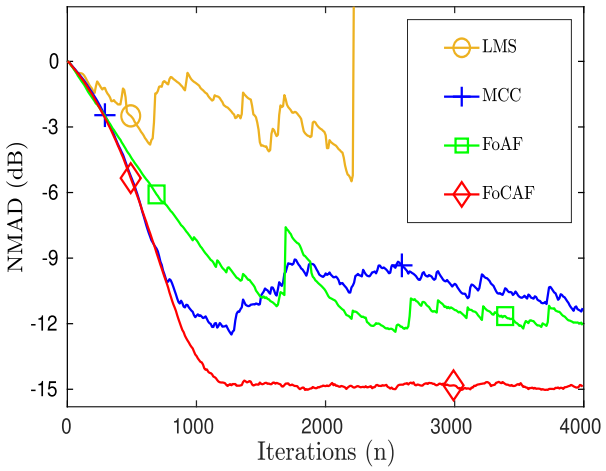


Fig. 2. Learning curves [NMAD versus iterations (n)] of the state-of-the-art algorithms in α-stable environment.

this issue by fusing the concepts of fractional-order calculus and correntropy-type localized similarity measure. The FoCAF iteratively estimates the unknown system by maximizing the following fractional-order correntropy criterion [16]:

$$J = \mathrm{E}\left[\exp\left(-\frac{e_n\, e_n^{\langle\alpha'-1\rangle}}{2\,\beta^2}\right)\right] \tag{3}$$

where $\beta > 0$ regulates the bandwidth of the kernel [16]. The FoCAF weight update equation is given by

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu\, \exp\left(-\sigma\, e_n\, e_n^{\langle\alpha'-1\rangle}\right) e_n\, \mathbf{u}_n^{\langle\alpha'-1\rangle} \tag{4}$$

where $\sigma = (1/2\,\beta^2)$.

C. Performance Study of FoCAF

In this section, the performance of the FoCAF algorithm is demonstrated in an α-stable environment. For this, the FoCAF is simulated to identify a randomly chosen 16-tap linear system. Input signal and observation noise are zero-mean SαS signals with variance 0.2 and 0.1, respectively. The characteristic exponent α of input signal and observation noise are set to 1.6 and 1.5, respectively. In addition to FoCAF, the state-of-the-art approaches, such as least mean square (LMS), MCC, and FoAF algorithms are also simulated for identifying the same system. The normalized mean absolute deviation (NMAD) given by $\mathbb{E}[((\|\mathbf{w}_{\mathrm{opt}} - \mathbf{w}_n\|_1)/(\|\mathbf{w}_{\mathrm{opt}}\|_1))]$ is considered as a performance metric. The performance of all these algorithms is compared by plotting NMAD in decibel versus iteration index n, by averaging over 100 independent experiments. For both FoAF and FoCAF algorithms, the parameter α' is chosen to be 1.3. The kernel width β of both FoCAF and MCC algorithms is fixed at 0.9. The step size μ of each algorithm is selected so that the initial convergence of all algorithms is the same to have a fair comparison of their steady-state estimation performance. The learning curves obtained from simulations are plotted in Fig. 2.

From Fig. 2, we see that the FoCAF algorithm achieves a better convergence rate and improved steady-state NMAD than MCC and FoAF algorithms. The conventional LMS diverges after certain iterations. Furthermore, FoAF and MCC algorithms exhibit residual jitters in their steady-state estimates. It is important to note that FoCAF is robust against the selection of parameters α' and β. The robust performance of FoCAF in α-stable environments motivated us to develop its VLSI implementation to serve in real-time applications such as underwater acoustic channel estimation and equalization.

III. VLSI Architecture for FoCAF Algorithm

In this section, we present the details of the proposed VLSI architecture for FoCAF. Due to the exponentiation operation and numerous multiplications in the update equation (4), FoCAF in its native form is inefficient for hardware implementation. Hence, we reformulate FoCAF to ensure that it is hardware friendly without losing accuracy. With the fixed-bit implementation, we examine the performance of the reformulated FoCAF. In order to achieve higher clock frequency and throughput, the reformulated algorithm is retimed based on a detailed critical-path analysis. Finally, we design a pipelined VLSI architecture for the reformulated FoCAF.

For the purpose of reformulation, the update equation of FoCAF algorithm in (4) can be written as

$$\mathbf{w}_{n+1} = \mathbf{w}_n + r_n\, \mathbf{x}_n \tag{5}$$

where

$$r_n = \mu\, \exp\left(-\sigma\, e_n\, e_n^{\langle\alpha'-1\rangle}\right) e_n \tag{6}$$

and

$$\mathbf{x}_n = [x_n, x_{n-1}, \ldots, x_{n-L+1}]^{\mathrm{T}}$$

is the fractional-order input vector. Fractional-order input x_n is obtained from u_n as

$$x_n = u_n^{\langle\alpha'-1\rangle}. \tag{7}$$

We then reformulate r_n and x_n to make them suitable for VLSI implementation. We first propose a hardware-friendly method for approximating exponentiation function using a logarithmic number system (LNS), which simplifies this function to a great extent.

A. Hardware Implementation of LNS

Consider a binary number Q with an int integer and fr fractional bits, i.e., $q_{\mathrm{int}-1}\, q_{\mathrm{int}-2} \cdots q_0\; q_{-1}\, q_{-2}\, q_{-\mathrm{fr}}$ and q_t be the leading one bit of Q. Then, the value of Q can be written as $Q = 2^t(1 + j)$, where j is a fraction, with $0 \le j \le 1$. The equation for $\log_2(Q)$ conversion based on Mitchell's scheme [29], [30] is shown as follows:

$$\begin{aligned} t1 &= \mathrm{LOD}(Q) \\ j &= Q[t1 - 1 : 0] \\ t &= t1 - \mathrm{fr} \\ \log_2(Q) &= \{t, j\} \end{aligned} \tag{8}$$

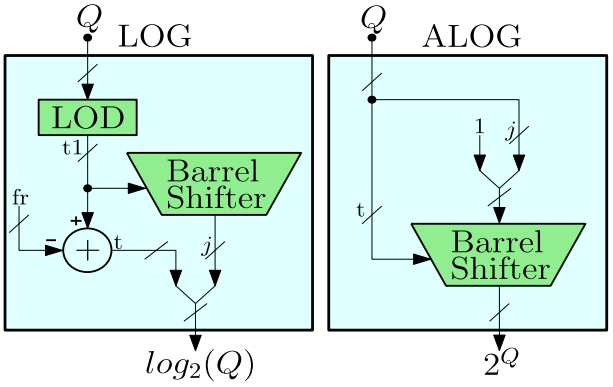


Fig. 3. Architectures for LNS conversions [29].

where leading one detector (LOD) detects the leading one position $t1$ in Q. The equations for $\text{alog}_2(Q)$ based on Mitchell's scheme are shown as follows:

$$\begin{aligned} j1 &= \{1, Q[\text{fr}-1:0]\} \\ t &= 2^{Q[\text{int+fr}-1:\text{fr}]} \\ 2^Q &= t * j1. \end{aligned} \tag{9}$$

The architecture for LNS conversions based on (8) and (9) is shown in Fig. 3. From Fig. 3, we see that the log converter consists of a simple LOD circuit followed by a barrel shifter, while the antilog converter involves the concatenation of 1 and j, followed by a barrel shifter.

B. Hardware Implementation of the Exponentiation Function

This section presents a hardware-friendly method for approximating exponentiation operations of type a^x in (4), where x is the variable and a is the constant. Exponential and logarithmic operations are commonly used in signal processing algorithms. Several architectures based on the Taylor and Maclaurin series have been proposed to implement these operations [31], [32]. However, if the value of a is large, these schemes lead to many of multipliers and adders. FoCAF, on the other hand, allows an exponentiation base to be bound between 0 and 1. This facilitates the implementation of exponentiation in hardware with the Maclaurin series, as described in the following. From the Maclaurin series, we have

$$a^x = 1 + \sum_{m=1}^{\infty} (\text{sign}(x))^m (\text{sign}(\ln a))^m \, |x|^m \frac{|\ln a|^m}{m!}. \tag{10}$$

When the value of $a \in (0, 1]$, a^x can be approximated with first nt terms of the series in (10) and $\ln a$ is always negative. The computation of (10), however, requires many multiplications and power operations that synthesize very poorly for VLSI real-time implementations. Thus, we rewrite (10) using LNS. Considering the first nt terms of the series and including LNS, (10) can be rewritten as

$$a^x = 1 + \sum_{m=1}^{nt} (\text{sign}(x))^m (-1)^m 2^{\left(m \log_2 |x| + \log_2\left(\frac{|\ln a|^m}{m!}\right)\right)}. \tag{11}$$

Note that $\log_2(|\ln a|^m/m!)$ is a constant for a given m. Depending on the value of m where $m = 1, 2, 3, \ldots, nt$, these constants can be stored in registers without calculating them every time. In this case, $m \log_2(x)$ can be realized using addition when the first nt terms of the series are chosen. The value of nt is selected based on required accuracy. Therefore, the reformulation proposed in (11) allows us to implement the exponentiation function a^x with $a \in (0, 1]$ without any multiplications and power operations. Furthermore, with few reformulations, functions of type e^{ax} and x^a can also be implemented based on these Maclaurin approximations as discussed in Section III-C.

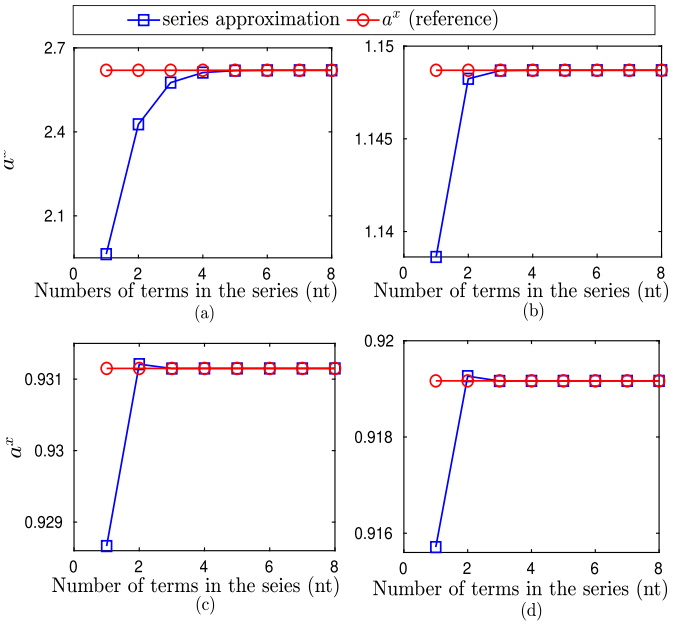


Fig. 4. Approximate implementation of a^x for different values of terms in the series (nt) versus accurate a^x values. (a) $x = -0.3$ and $a = 0.3$. (b) $x = -0.2$ and $a = 0.5$. (c) $x = 0.2$ and $a = 0.7$. (d) $x = 0.8$ and $a = 0.9$.

1) Performance Study of the Proposed Approximations: The performance of the proposed exponentiation calculation scheme in (11) is compared with the actual exponentiation values using numerical simulations. The exponentiation function a^x is simulated for different values of x and a. The function is then implemented using (11) for different numbers of terms in the series nt. The results are then compared with actual values for the same inputs and shown in Fig. 4.

From Fig. 4, one can see that in real-time implementations, exponentiation functions with base values in range $(0, 1]$ can be approximated by using the proposed reformulations in (11). The value of nt is determined by the amount of precision needed for the application.

2) Hardware Implementation Cost of the Proposed Approximations: To quantify the hardware efficiency of the proposed simplifications, we implemented a^x using direct series approximation and also through the LNS approximation. We coded the designs in Verilog and synthesized them using the Cadence Genus synthesis tool with 45-nm CMOS standard cell library. The plot comparing the area-delay products of the proposed LNS approximation and the direct approximations for different numbers of terms in the series nt is shown in Fig. 5.

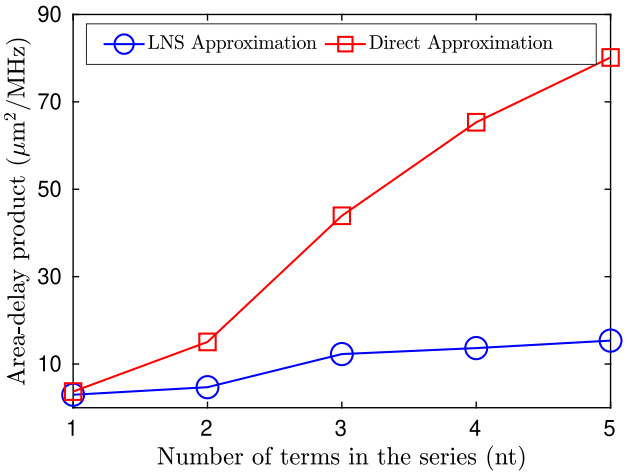


Fig. 5. Approximate implementation of a^x using LNS and direct approximation methods for different number of terms in the series (nt).

It can be seen from Fig. 5 that the proposed approximations incur significantly lower area-delay product compared to the direct series approximation. The intuitive explanation for this is given as follows. In the case of direct series approximation, an increase in nt leads to more number of multipliers, whereas, in the case of LNS approximation, it leads to more ALOG units whose hardware complexity is much less compared to multipliers.

C. FoCAF Algorithm Reformulation

This section presents hardware-friendly implementation of fractional-order input and FoCAF error function based on hardware implementation exponentiation function.

1) Fractional-Order Input: Based on the definition for the fractional-order input in Section II-B, (7) can be rewritten as

$$\begin{aligned} x_n &= \text{sign}(u_n)\left(|u_n|^{(\alpha'-1)}\right) \\ &= \text{sign}(u_n)\left(2^{(\alpha'-2)\log_2|u_n|}\,|u_n|\right) \\ &= \text{sign}(u_n)\left(p^{\log_2|u(n)|}\,|u_n|\right) \end{aligned} \tag{12}$$

where $p = 2^{(\alpha'-2)}$ is a constant. The term $p^{\log_2|u_n|}$ can be approximated using a Maclaurin series, as shown in Section III-B1. Here, $p\,\epsilon\,(0,1)$ since $\alpha'\epsilon(1,\alpha)$, and therefore, the value of $\ln p$ is always negative. Thereby, $p^{\log_2|u_n|}$ is approximated with first three terms of the Maclaurin series as

$$p^{\log_2|u_n|} = 1 + \sum_{m=1}^{3}\left((-1)^m\left(\log_2|u_n|\right)^m\left(\frac{|\ln p|^m}{m!}\right)\right). \tag{13}$$

Substituting (13) into (12) and denoting $(|\ln p|^m/m!)$ as q_m, we have

$$x_n = \text{sign}(u_n)\left(|u_n| + \sum_{m=1}^{3}\left((-1)^m\left(\log_2|u_n|\right)^m q_m\,|u_n|\right)\right). \tag{14}$$

By introducing LNS in (14), we obtain

$$\begin{aligned} x_n = \text{sign}(u_n) \times\Bigg(|u_n| + \sum_{m=1}^{3}\Big(&\left(\text{sign}\left(\log_2|u_n|\right)\right)^m \\ &\times(-1)^m 2^{\left(m\log_2|\log_2|u_n||+k_m+\log_2|u_n|\right)}\Big)\Bigg) \end{aligned} \tag{15}$$

where $k_m = \log_2|q_m|$ is a constant for a given m. Depending on the value of m, where $m = 1, 2, 3, \ldots, nt$, and these constants can be stored in registers without calculating them every time. It is worth noting that (15) is free of multiplications and only contains additions, LNS, and shifting operations. Therefore, the reformulation of (7) to (15) enables hardware-friendly calculation of x_n.

2) FoCAF Error Function: The FoCAF error function is

$$\begin{aligned} f(e_n) &= \exp\left(-\sigma\,|e_n|\,|e_n|^{\langle\alpha'-1\rangle}\right)e_n \\ &= b^{\left(|e_n|^2\,|e_n|^{\langle\alpha'-2\rangle}\right)}\,e_n \end{aligned} \tag{16}$$

where $b = \exp(-\sigma)$ is a positive constant. By introducing LNS in (16), we have

$$\begin{aligned} f(e_n) &= \text{sign}(e_n)\,2^{\left(|e_n|^2|e_n|^{(\alpha'-2)}\log_2 b+\log_2|e_n|\right)} \\ &= \text{sign}(e_n)\,2^{g(e_n)} \end{aligned} \tag{17}$$

where $g(e_n) = (|e_n|^2|e_n|^{(\alpha'-2)}\log_2 b + \log_2|e_n|)$. Furthermore, by employing LNS, the term $|e_n|^{(\alpha'-2)}$ from $g(e_n)$ can be rewritten as $2^{(\alpha'-2)\log_2|e_n|}$. By following the same reformulation procedures as in (12)–(14), we get

$$g(e_n) = z|e_n|^2\left(1+\left(\sum_{m=1}^{3}(-1)^m\left(\log_2|e_n|\right)^m q_m\right)\right)+\log_2|e_n| \tag{18}$$

where $z = \log_2(b)$. Here, z is always negative. The terms in (18) can then be rearranged as

$$\begin{aligned} g(e_n) = -\Bigg(|z||e_n|^2 + \left(\sum_{m=1}^{3}(-1)^m|s_m|\left(\log_2|e_n|\right)^m|e_n|^2\right) \\ -\log_2|e_n|\Bigg) \end{aligned} \tag{19}$$

where $s = z\,q_m$. By incorporating LNS in (19)

$$\begin{aligned} g(e_n) = -\Bigg(2^{(c0+2\log_2|e_n|)} + \Bigg(\sum_{m=1}^{3}\left(\text{sign}\left(\log_2|e_n|\right)\right)^m \\ \times 2^{\left(m\log_2|\log_2|e_n||\right)+c_m+2\log_2|e_n|)}\Bigg) \\ -\log_2|e_n|\Bigg) \end{aligned} \tag{20}$$

where $c_0 = \log_2|z|$ is a constant and $c_m = \log_2|s_m|$ is a constant for a given m. Depending on the value of m where $m = 1, 2, 3, \ldots, nt$, these constants can be stored in registers

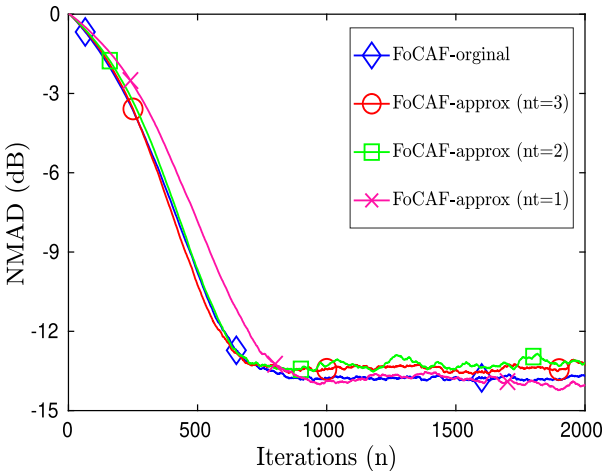


Fig. 6. Learning curves of the original and reformulated FoCAF with different number of terms in series (nt).

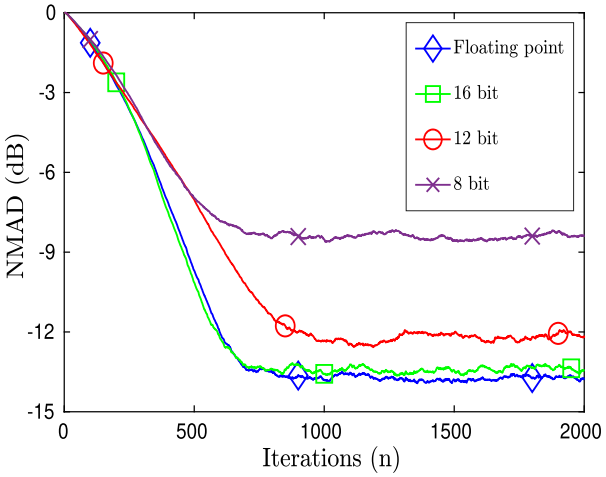


Fig. 7. Learning curves of the reformulated FoCAF algorithm in 16-, 12-, and 8-bit fixed-bit representation.

without calculating them every time. Similar to (15), the reformulated error function in (20) is free of multiplications and only contains additions, LNS, and shifting operations. From (6), (17), and (20), we get

$$r_n = \mu \, \text{sign}(e_n) \, 2^{g(e_n)}. \tag{21}$$

3) Performance of the Reformulated FoCAF: The performance of FoCAF with the proposed reformulations in (15) and (20) is demonstrated using numerical simulations. The reformulated FoCAF with different nt values are simulated for the same system identification problem in Section II-C. In order to compare the performance, simulations are also performed with the original FoCAF, considering the same input parameter and step size values. The learning curves obtained from simulations are plotted in Fig. 6. From Fig. 6, we see that the reformulated FoCAF with $nt = 3$ has similar convergence and steady-state performance as that of the original FoCAF algorithm. The steady-state value and convergence rate show tiny degradation as the value of nt decreases. Thus, the value of nt has to be selected based on the performance-hardware complexity tradeoff.

D. Bit-Width Consideration of the Proposed FoCAF Fixed-Point Implementation

The performance of the original floating-point FoCAF algorithm and reformulated FoCAF algorithm with 16-, 12-, and 8-bit fixed-point representations are compared in Fig. 7 by considering the same system identification problem in Section II-C. From Fig. 7, it is evident that reformulated FoCAF with 16-bit fixed-point representation has approximately the same steady-state performance as that of the original FoCAF algorithm. Reformulated FoCAF is tolerant to logarithmic approximations due to its stochastic and iterative nature. From the figure, we can also observe degradation in the steady-state performance as bit width reduces. Thus, architecture with a bit width of 16 is preferable for VLSI implementation of the reformulated FoCAF.

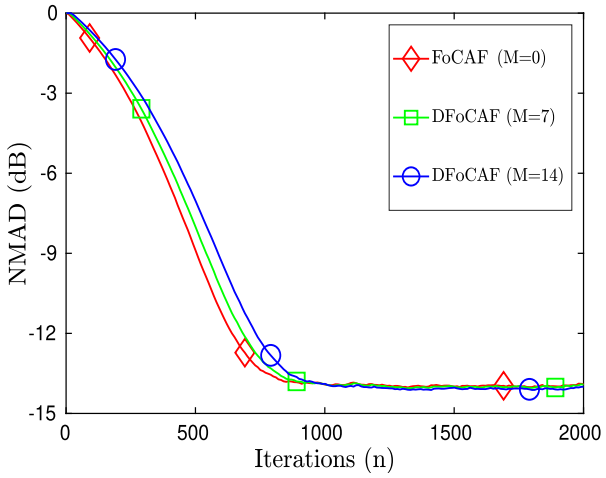


Fig. 8. Learning curves of the FoCAF and DFoCAF algorithms with different values of adaptation delay M.

E. Delayed FoCAF Algorithm

For real-time implementations, even after the simplifications described in Section III-C, the time complexity of the reformulated FoCAF algorithm remains high since r_n must be computed at every iteration, and the weights must be updated before proceeding to the next iteration. The feedback loop that updates the filter weights in each iteration restricts direct pipelining. In order to resolve this issue, FoCAF is modified to a form known as DFoCAF by extending the concept of delayed adaptation [33]. Delayed adaptation assumes that the error gradient $r_n \, x_n$ is not affected much by the delay M (adaptation delay). As long as $M < L$, this is a fair assumption [33]. The update equation of the DFoCAF algorithm is given by

$$\mathbf{w}_{n+1} = \mathbf{w}_n + r_{n-M} \, \mathbf{x}_{n-M} \tag{22}$$

where

$$r_{n-M} = \mu \, \text{sign}(e_{n-M}) \, 2^{g(e_{n-M})}. \tag{23}$$

By introducing the M delay registers in the feedback loop, we can apply retiming [34] to reduce the critical path, thereby

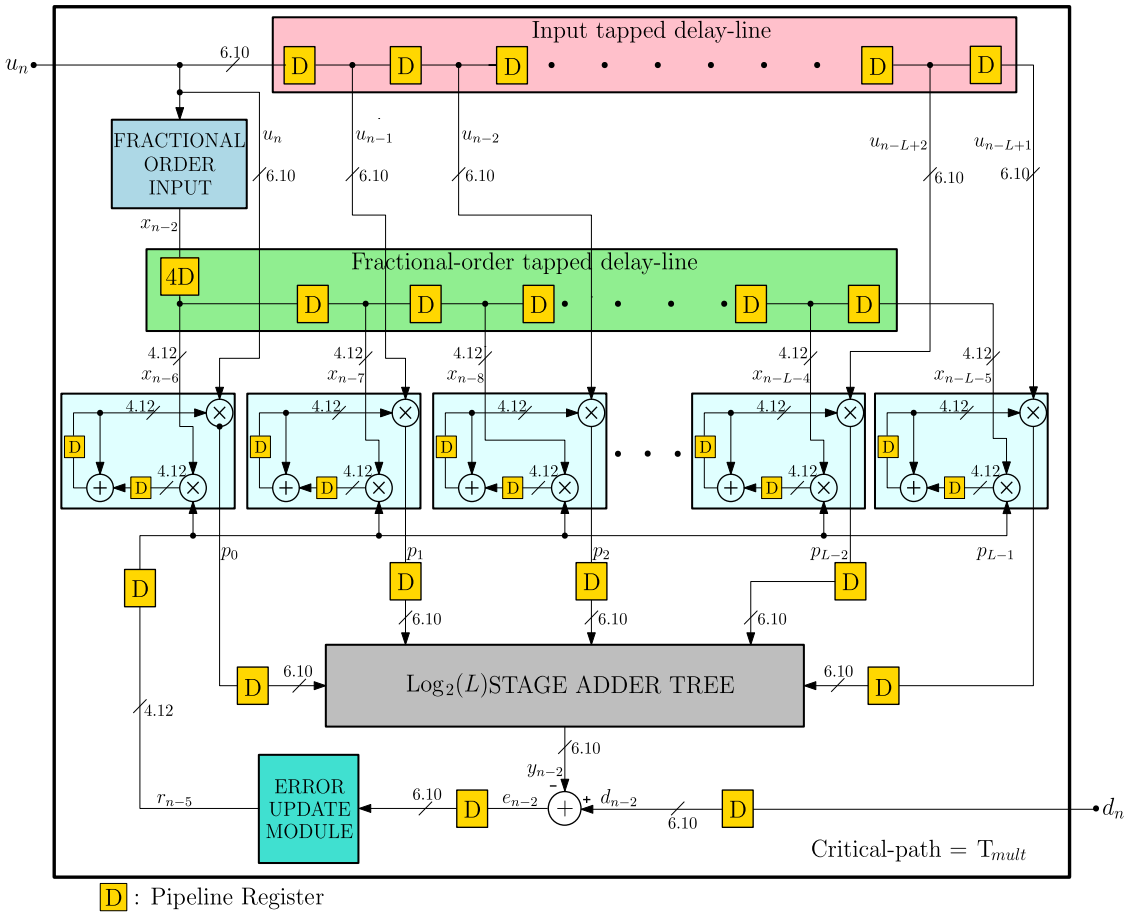


Fig. 9. DFoCAF architecture.

increasing the sampling rate. We now compare the results of DFoCAF and FoCAF to check the extent to which the adaptation delay M affects proportionate adaptation. Learning curves of DFoCAF for the system identification problem in Section II-C are plotted with different values of M and compared with the FoCAF, as shown in Fig. 8. We see that the DFoCAF algorithm has negligible degradation in convergence rate compared to the FoCAF algorithm for small values of M. However, as the adaptation delay increases, we observe slight degradation in the convergence rate and steady-state NMAD.

F. DFoCAF Architecture

The VLSI architecture of the proposed DFoCAF algorithm is shown in Fig. 9. The bit widths of all the intermediate signals in the $Q_{\text{int.fr}}$ format (where int is the number of integer bits and fr is the number of fractional bits) are also shown in the architecture. These bit widths are determined by following the MATLAB floating-point to fixed-point conversion methodology to avoid overflows. The architecture essentially implements (22), which updates the filter coefficients in every iteration. As shown in Fig. 9, the architecture features a fractional-order input module, L tap modules, $\log_2(L)$ stage adder tree, and an error update module. The functionality of each module is explained briefly in the following.

1) Fractional-Order Input Module: The fractional-order input module computes the fractional-order input x_n from the original input u_n, based on (15). The architecture for the fractional-order input module is shown in Fig. 10. Considering that logarithm is only defined for positive numbers, the absolute value of input u_n is first computed in the ABS block. LOG and ALOG blocks implement base-2 logarithm and antilogarithm, respectively. The LOG and ALOG blocks are realized using Mitchell's scheme, which is a very simple scheme requiring only an LOD and barrel shifter. The multiplication with -1 in the architecture is nothing but 2's complement operation. $k1$, $k2$, and $k3$ are constants calculated from (15) and are stored in storage registers. Adders and subtractors are used to accumulate the first three terms of the Maclaurin series. However, as seen from (15), the sign of the first and third terms of the series are assigned with respect to sign($\log_2 |u_n|$). Similarly, (15) also involves the computation of sign(u_n). In order to realize these sign assignments in hardware, a combination of SIGN module, 2′s complement module, and multiplexer module is used. While realizing the sign assignment operation of type sign(a_n)b, the SIGN module determines the sign of a, whereas the multiplexer module assigns the sign to b, by either selecting b directly or $-b$ depending on sign of a. Delays are introduced in the module based on critical-path analysis in Section III-G.

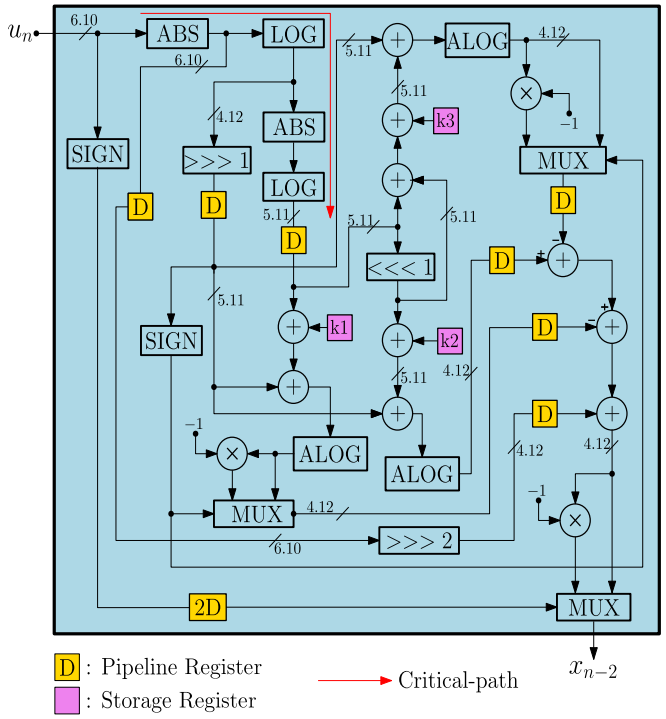


Fig. 10. Fractional-order input module architecture.

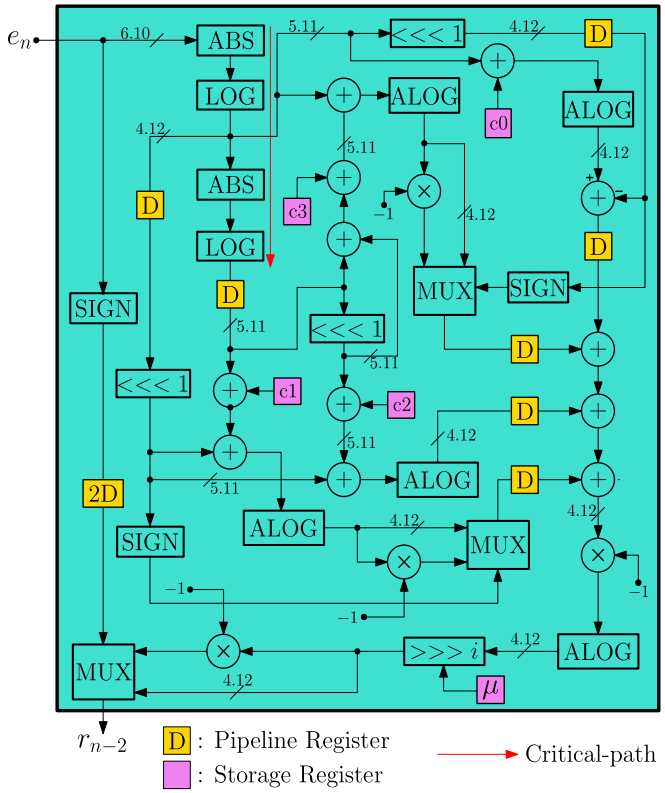


Fig. 11. Error update module architecture.

We can see that all the submodules in the architecture, such as LOG, ALOG, MUX, and 2's complement evaluation, require considerably less hardware complexity than multipliers and exponential operators. From (21), we can see that the FoCAF update equation involves computation of vector $\mathbf{x}_n$ containing elements $\mathrm{x}_n, \mathrm{x}_{n-1}, \ldots, \mathrm{x}_{n-L+1}$. In order to realize $\mathbf{x}_n$, the architecture involves a fractional-order tapped delay line (shown in Fig. 9) with the fractional-order input as the delay-line input. The number of delay registers in the delay line is $L-1$. Therefore, as the filter order increases, the length of the tapped delay line increases linearly.

2) Tap Module: The tap module in Fig. 9 is responsible for calculating the partial products of the filter output and also for realizing the weight update recursion. Each tap module realizes (24) and (25) in hardware

$$w_{n+1,i} = w_{n,i} + r_n\, x_{n-i} \tag{24}$$

where $w_{n,i}$ represents ith filter weight corresponding to filter tap i $(0 \le i \le L-1)$ at time index n

$$p_i = u_{n-i}\, w_{n,i}. \tag{25}$$

Here, u_{n-i} is the input to the ith tap module and p_i is the output from ith tap module. The output of the filter, y_n, is obtained by adding all individual tap module outputs using an adder tree, i.e., $(y_n = \Sigma_{i=0}^{L-1} p_i)$. In order to reduce the propagation delay through the adder tree and to keep T_{mult} as the critical path, a carry save adder tree [21] is utilized in the design.

3) Error Update Module: The error update module computes the error update factor r_n from the error e_n, based on (21). The architecture for the error update module is shown in Fig. 11. Here, $c0$, $c1$, $c2$, and $c3$ are constants that are calculated and stored in storage registers. The architecture is very similar to the fractional-order input module except for the multiplication with step size μ. We consider the step size to be negative power of 2, i.e., of the form $(1/2^k)$ to avoid multiplication. The error update factor is fed to all the taps to realize the weight update equation.

To evaluate the hardware complexity of the proposed architecture, we compared it with the DLMS architecture [33] where DLMS is a high-throughput retimed implementation of the LMS algorithm. It can be observed from Fig. 9 that, compared to the DLMS architecture in [33], the hardware overhead of the DFoCAF architecture is a fractional-order input module, error update module, and fractional-order tapped delay line. Based on the synthesis results in Section IV, we will show that this area overhead of DFoCAF is very small. We see in the literature that several strategies [19], [33] can be utilized to implement variants of LMS, all claiming different benefits. It is important to note that the proposed DFoCAF algorithm can be implemented by integrating a fractional-order input module, an error update module, and a fractional-order tapped delay line appropriately into any of those LMS architectures.

G. Critical-Path Analysis

As discussed in Section III-E, DFoCAF architecture is retimed to improve the throughput. However, retiming has two adverse effects. First, the number of registers increases, which leads to more area and power dissipation. Second, as the adaptation delay increases, the convergence rate degrades, as shown in Fig. 8. Thus, the architecture has to be carefully retimed based on a detailed critical-path analysis. This analysis

TABLE I

SYNTHESIS RESULTS OF FOCAF WITH FILTER ORDER 16 USING CADENCE CMOS LIBRARY FOR VARIOUS ADAPTATION DELAYS M

Algorithm	Adaptation Delays (M)	Max. Clock Freq. (MHz)	Area (μm^2)	Leakage Power (μW)	Dynamic Power (mW)	Area-Delay product (μm^2/MHz)
DFoCAF	0	221	201959	30.04	13.14	913.84
	1	322	144641	19.51	10.48	449.20
	3	439	101197	11.53	4.86	230.52
	4	606	107842	12.71	5.03	176.97
	5	1036	145911	19.19	5.76	140.84
	6	1048	151885	20.41	5.95	145.05

TABLE II

SYNTHESIS RESULTS OF LMS AND DFOCAF WITH DIFFERENT FILTER ORDER USING CADENCE 45-nm CMOS LIBRARY FOR CLOCK FREQUENCY 1.048 GHz

Algorithm	Filter Order (L)	Adaptation Delays (M)	NMAD * (dB)	Area (μm^2)	Leakage Power (μW)	Dynamic Power (mW)
Proposed DFoCAF	16	6	-14.5	151885	20.41	5.95
	32	6	-18	302432	41.48	10.72
	64	6	-23	536858	70.39	22.04
DLMS [28]	16	3	-2	134163	18.44	4.71
	32	3	-3.5	268929	36.58	9.59
	64	3	-5	509016	68.21	19.96

* The given NMAD value for DFoCAF is equivalent to the steady-state value since the algorithm is convergent. For LMS, the given NMAD is equivalent to the minimum obtained value since it is not converging.

TABLE III

THEORETICAL HARDWARE COMPLEXITY OF DFOCAF ARCHITECTURE

Number of units		Adders	Subtractor	Multipliers	LOG blocks	ALOG blocks	Delay Registers	Storage Registers	Multiplexers
Filter order (L)	16	50	4	32	4	8	105	8	6
	32	82	4	64	4	8	187	8	6
	64	146	4	128	4	8	351	8	6

is done by increasing the adaption delays (number of pipeline stages) in the architecture from 0 to 6 and synthesizing the corresponding designs to find the maximum clock frequency. Table I shows the synthesis results of DFoCAF architecture with filter length 16 and different values of adaptation delay M. The designs are coded in Verilog and synthesized using Cadence Genus synthesis tool with a 45-nm CMOS standard cell library. Based on the synthesis results, Fig. 12 shows the histograms for different values of M. These histograms show "number of paths versus slack in ns." In static timing analysis (STA), slack is defined as the difference between data required time and data arrival time (DAT) at the destination register input. Next, we discuss the placement of the pipeline registers for different values of M and the resulting critical path.

1) Zero Adaptation Delay: When the FoCAF adaptive filter is implemented without pipelining, the critical path of corresponding design is $(T_{\text{tap}} + T_{\text{add\_tree}} + T_{\text{sub}} + T_{\text{err\_upd}})$, where T_{tap}, $T_{\text{add\_tree}}$, T_{sub}, and $T_{\text{err\_upd}}$ are the propagation delays of tap module, adder tree, subtraction module, and error updation module, respectively. From Table I, we can observe that the maximum attainable frequency of the FoCAF architecture with no adaption delay is very low because of the long critical path. Also, from histogram 12, we can observe that there are many paths with large positive slack values. Thus, the FoCAF architecture has to be pipelined to increase the maximum attainable frequency.

2) One Adaptation Delay: Here, we add delay registers in between the subtractor that calculates the error e_n and the error update module. The addition of this pipeline register will reduce the critical-path to $(T_{\text{err\_upd}} + T_{\text{mult}})$. From Table I, we can observe that the maximum attainable frequency of the DFoCAF architecture with one adaption delay improved considerably as a result of pipelining. By observing histogram in Fig. 12(b), still, there are many paths with large positive slack values, and thus, the FoCAF architecture can be pipelined further.

3) Three Adaptation Delays: The architecture is further retimed by adding two more pipelined registers. A pipeline register is placed at the output of the tap module and another one at the output of the error update module. The resulting critical path is reduced to $T_{\text{err\_upd}}$ and we can observe further increase in maximum attainable frequency from Table I. Despite the decrease in maximum slack value in the histogram in Fig. 12(c), there remain many paths with positive slacks. This indicates that there is still room for improvement.

4) More Adaptation Delays: In order to further retime the architecture, we need to do fine-grain pipelining of the fractional-order input module and the error update module, i.e., we add pipeline registers at appropriate places inside these modules, as shown in Figs. 10 and 11. Critical path of the architecture with four adaptation delays is reduced to $(2T_{\log}+2T_{\text{abs}}+3T_{\text{adder}}+T_{\text{alog}}+T_{\text{inv}}+T_{\text{mux}})$. Similarly, the critical

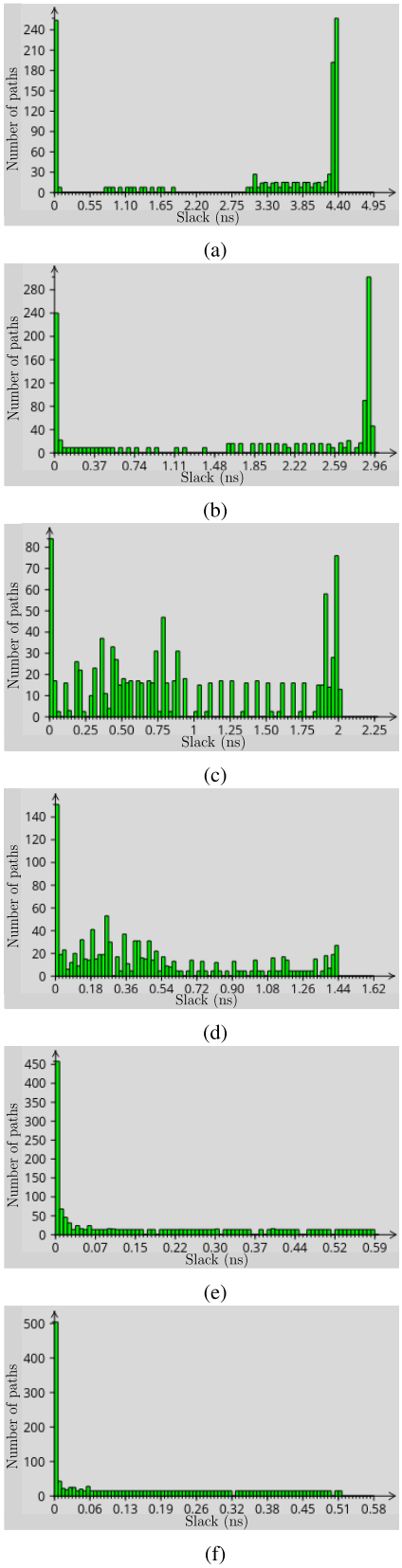


Fig. 12. Histogram with slack value of various paths in DFoCAF design for various adaptation delays M. (a) $M = 1$. (b) $M = 2$. (c) $M = 3$. (d) $M = 4$. (e) $M = 5$. (f) $M = 6$.

paths with five and six adaptation delays are $T_{\text{add\_tree}}$ and T_{mult}, respectively. From synthesis results, we observe that $T_{\text{add\_tree}}$ for a three-stage adder tree is almost the same as T_{mux}. We can also observe from the histogram that with $M = 5$, there are many paths with slack close to zero and we stop the pipelining at this point and take the critical path to be T_{mux}. Moreover, from Table I, we can also observe that the area-delay product is minimum for $M = 5$. Note that, as the order of the filter increases, the number of stages of the adder tree increases. In order to maintain the critical path at T_{mult}, the adder tree has to be pipelined after every third stage.

IV. VLSI Implementation Results

The proposed DFoCAF architecture is implemented in Verilog HDL and simulated using the Cadence NCSim simulator. All the operations in the DFoCAF architecture, including LOG and ALOG, are also implemented in fixed-point MATLAB, and the outputs are taken as the golden reference values. The Verilog simulation results are verified against these golden reference outputs from MATLAB for many sets of random inputs. After verification, the designs are synthesized using the Cadence Genus tool in 45-nm CMOS technology. As discussed in Section III-G, Table I shows the synthesis results for FoCAF architecture for different values of adaptation delay, M. From Table I, we can observe that the best area-delay product is obtained at $M = 5$.

Next, DFoCAF architectures with filter orders of 16, 32, and 64 are considered for synthesis to check the scalability of the proposed architectures. The synthesis results are tabulated in Table II. Since there are no prior architectures available for DFoCAF, the results are compared against state-of-the-art DLMS [29] architectures. Note that the DLMS architectures from [29] are also coded in Verilog and synthesized using the same 45-nm libraries for a fair comparison. Here, DLMS is chosen for comparing the synthesis results of DFoCAF since DLMS is the least complex adaptive filtering algorithm that can be implemented on hardware. DFoCAF and DLMS with $M = 6$ and $M = 3$, respectively, are chosen for comparing the synthesis results so that the critical path of both designs is T_{mult} and the clock frequency is 1.048 GHz. The dynamic power values given in Tables I and II are extracted from postsynthesis power reports by annotating switching activity interchange format (SAIF) files from gate-level timing simulations with random inputs generated from a symmetrical α-stable distribution. From Table II, we see that DFoCAF has 1.13×, 1.12×, and 1.05× increment in area and has 1.26×, 1.11×, and 1.10× increase in the dynamic power for 16, 32, and 64 taps, respectively, when compared with DLMS. Compared to the DLMS architecture, the additional logic to obtain the fractional-order input, the nonlinear error function in DFoCAF, and the delay chain for fractional-order input contribute to the area and power overhead. However, the increase in area and power is very little compared to the improvement achieved in steady-state NMAD and convergence rate by DFoCAF over the DLMS algorithm, which does not even converge in SαS signal and noise environment (see Fig. 2). From Table II, it is worth noting that the area overhead of DFoCAF decreases with filter order and becomes negligible for large order filters.

Table III presents the theoretical hardware complexity of the DFoCAF architecture in terms of a number of each basic

TABLE IV
AREA BREAKDOWN OF 256-TAP DFOCAF SUBBLOCKS

	kGE[1]	%
Logic specific to DFoCAF	16.67	3.29
Logic common to DLMS and DFoCAF	480.38	96.70
Total	506.05	100

[1] One kilo Gate Equivalent (kGE): Area of a two input NAND gate = $3.25\mu\text{m}^2$.

module required per iteration such as adders and multipliers. Table III shows that the increase in the number of basic modules with the increase in filter order is not linear. This is because the fractional-order and error update module are scalar modules, i.e., they are independent of filter order. Hence, for higher filter orders, their area and power overhead decrease, which is evident from Table IV showing the area overhead in percentage between different modules in the design. Here, logic specific to DFoCAF includes fractional-order module, error update module, and fractional-order tapped delay line, as explained in Section III-F. A notable feature of the proposed DFoCAF algorithm is that it can be implemented by integrating the logic specific to DFoCAF with any variant of the LMS architecture described in the literature. We can see from both theoretical complexity analysis as well as synthesis results that the area overhead of DFoCAF decreases with filter order and becomes negligible for high-order filters.

V. CONCLUSION

Fractional-order correntropy adaptive algorithms exhibit robust performance in non-Gaussian signal and noise environments, where the conventional LMS variants fail. In this work, we demonstrated that the hardware implementation cost of the FoCAF algorithm could be brought down significantly through several reformulations without compromising the performance of the algorithm. We also provided a comparative discussion on the hardware complexities of the conventional DLMS and the proposed DFoCAF through ASIC synthesis results and showed that the performance gain of the DFoCAF over DLMS is significant with minimal increase in hardware overhead. In future, we wish to extend the fractional-order correntropy approach to other state-of-the-art adaptive filters to realize them in hardware.

REFERENCES

[1] A. H. Sayed, *Adaptive Filters*. Hoboken, NJ, USA: Wiley, 2011.

[2] T. Kailath, A. H. Sayed, and B. Hassib, *Linear Estimation*. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.

[3] R. D. Pierce, "Application of the positive alpha-stable distribution," in *Proc. IEEE Signal Process. Workshop Higher-Order Statist.*, Jul. 1997, pp. 420–424.

[4] M. Bouvet and S. C. Schwartz, "Comparison of adaptive and robust receivers for signal detection in ambient underwater noise," *IEEE Trans. Acoust., Speech Signal Process.*, vol. 37, no. 5, pp. 621–626, May 1989.

[5] N. Azzaoui and L. Clavier, "Statistical channel model based on α-stable random processes and application to the 60 GHz ultra wide band channel," *IEEE Trans. Commun.*, vol. 58, no. 5, pp. 1457–1467, May 2010.

[6] J. P. Nolan, "Modeling financial data with stable distributions," in *Handbook of Heavy-Tailed Distributions in Finance*, vol. 1, S. T. Rachev, Ed. Amsterdam, The Netherlands: Elsevier, 2003, pp. 105–130.

[7] D. Salas-Gonzalez, J. M. Górriz, J. Ramírez, and E. W. Lang, "Why using the alpha-stable distribution in NeuroImage?" in *Proc. 11th Int. Conf. Signal Process. Multimedia Appl.*, 2014, pp. 297–301.

[8] E. Masry, "Alpha-stable signals and adaptive filtering," *IEEE Trans. Signal Process.*, vol. 48, no. 11, pp. 3011–3016, Nov. 2000.

[9] V. C. Gogineni, S. P. Talebi, S. Werner, and D. P. Mandic, "Fractional-Order correntropy adaptive filters for distributed processing of α-stable signals," *IEEE Signal Process. Lett.*, vol. 27, pp. 1884–1888, 2020.

[10] A. Singh and J. C. Principe, "Using correntropy as a cost function in linear adaptive filters," in *Proc. Int. Joint Conf. Neural Netw.*, 2009, pp. 2950–2955.

[11] K. Xiong, W. Shi, and S. Wang, "Robust multikernel maximum correntropy filters," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 6, pp. 1159–1163, Jun. 2020.

[12] W. Shi, Y. Li, and B. Chen, "A separable maximum correntropy adaptive algorithm," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 11, pp. 2797–2801, Nov. 2020.

[13] W. Liu, P. P. Pokharel, and J. C. Principe, "Correntropy: Properties and applications in non-Gaussian signal processing," *IEEE Trans. Signal Process.*, vol. 55, no. 11, pp. 5286–5298, Nov. 2007.

[14] B. Chen, L. Xing, H. Zhao, N. Zheng, and J. C. Príncipe, "Generalized correntropy for robust adaptive filtering," *IEEE Trans. Signal Process.*, vol. 64, no. 13, pp. 3376–3387, Jul. 2016.

[15] F. Huang, J. Zhang, and S. Zhang, "Adaptive filtering under a variable kernel width maximum correntropy criterion," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 10, pp. 1247–1251, Oct. 2017.

[16] V. C. Gogineni, S. P. Talebi, S. Werner, and D. P. Mandic, "Fractional-order correntropy filters for tracking dynamic systems in α-stable environments," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 12, pp. 3557–3561, Dec. 2020.

[17] G. Jumarie, "On the derivative chain-rules in fractional calculus via fractional difference and their application to systems modelling," *Open Phys.*, vol. 11, no. 6, pp. 617–633, Jan. 2013.

[18] S. H. Mirfarshbafan *et al.*, "Beamspace channel estimation for massive MIMO mmWave systems: Algorithm and VLSI design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 12, pp. 5482–5495, Dec. 2020.

[19] 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.

[20] S. Mula, V. C. Gogineni, and A. S. Dhar, "Robust proportionate adaptive filter architectures under impulsive noise," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1223–1227, May 2019.

[21] V. C. Gogineni and S. Mula, "Improved proportionate-type sparse adaptive filtering under maximum correntropy criterion in impulsive noise environments," *Digit. Signal Process.*, vol. 79, pp. 190–198, Aug. 2018.

[22] W. Chen, L. Qi, and F. Yanjun, "An improved least square channel estimation algorithm for underwater acoustic OFDM systems," in *Proc. 2nd Int. Conf. Future Comput. Commun.*, 2010, pp. V3-577–V3-580.

[23] X.-L. Shi and Y.-X. Yang, "Adaptive sparse channel estimation based on RLS for underwater acoustic OFDM systems," in *Proc. 6th Int. Conf. Instrum. Meas., Comput., Commun. Control (IMCCC)*, Jul. 2016, pp. 266–269.

[24] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Hoboken, NJ, USA: Wiley, 2007.

[25] S. C. Schwartz, J. B. Thomas, and E. J. Wegman, *Topics in Non-Gaussian Signal Processing*. New York, NY, USA: Springer, 1989.

[26] O. Arikan, M. Belge, A. E. Cetin, and E. Erzin, "Adaptive filtering approaches for non-Gaussian stable processes," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, 1995, pp. 1400–1403.

[27] S. P. Talebi, S. Werner, and D. Mandic, "Distributed adaptive filtering of alpha-stable signals," *IEEE Signal Process. Lett.*, vol. 25, no. 10, pp. 1450–1454, 2018.

[28] S. P. Talebi, S. Werner, S. Li, and D. P. Mandic, "Tracking dynamic systems in α-stable environments," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 4853–4857.

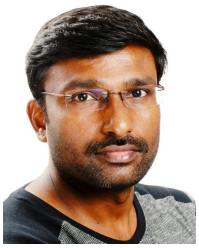
[29] S. Mula, V. C. Gogineni, and A. S. Dhar, "Algorithm and architecture design of adaptive filters with error nonlinearities," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2588–2601, Sep. 2017.

[30] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vol. 11, no. 4, pp. 512–517, Aug. 1962.

[31] S. M. Aroutchelvame and K. Raahemifar, "An efficient algorithm and architecture for natural logarithm using Maclaurin series," in *Proc. 12th IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 2005, pp. 1–4.
[32] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, "Hardware implementation of the exponential function using Taylor series," in *Proc. NORCHIP*, Oct. 2014, pp. 1–4.
[33] P. K. Meher and S. Y. Park, "Critical-path analysis and low-complexity implementation of the LMS adaptive algorithm," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 3, pp. 778–788, Mar. 2014.
[34] T. C. Denk and K. K. Parhi, "Exhaustive scheduling and retiming of digital signal processing systems," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 7, pp. 821–838, Jul. 1998.

Daney Alex received the bachelor's degree in electronics and communication engineering from the Cochin University of Science and Technology, Kochi, India, in 2018. He is currently working toward the M.S. by Research degree at the Department of Electrical Engineering, IIT Palakkad, Kozhippara, India.

His research interests include VLSI signal processing and machine learning architectures, and algorithms and architectures for adaptive filters.



Vinay Chakravarthi Gogineni (Member, IEEE) received the bachelor's degree in electronics and communication engineering from Jawaharlal Nehru Technological University, Andhra Pradesh, India, in 2005, the master's degree in communication engineering from the Vellore Institute of Technology, Vellore, India, in 2008, and the Ph.D. degree in electronics and electrical communication engineering from IIT Kharagpur, Kharagpur, India, in 2019.

From 2008 to 2011, he was with a couple of MNCs in India. He is currently a Postdoctoral Research Fellow with the Department of Electronic Systems, Norwegian University of Science and Technology (NTNU), Trondheim, Norway. His research interests include statistical signal processing, signal processing and machine learning over graphs, and federated learning.

Dr. Gogineni was a recipient of the ERCIM Alain Bensoussan Fellowship in 2019 and the Best Paper Award at APSIPA ASC-2021, Tokyo, Japan.

Subrahmanyam Mula (Member, IEEE) received the B.E. degree in electronics and communication engineering from Andhra University, Visakhapatnam, India, in 2001, the M.Tech. degree in microelectronics and VLSI design from the IIT Kharagpur, Kharagpur, India, in 2003, and the Ph.D. degree from the Department of Electronics and Electrical Communication Engineering, IIT Kharagpur, in 2018.

From 2003 to 2014, he was with Intel, Bengaluru, India, where he was involved in front-end design verification of gigabit Ethernet switches, processors, chipsets, and GPUs. He is currently an Assistant Professor with the Department of Electrical Engineering, IIT Palakkad, Kozhippara, India. His current research interests include digital VLSI circuits and systems, VLSI signal processing, and machine learning architectures.

Stefan Werner (Senior Member, IEEE) received the M.Sc. degree in electrical engineering from the Royal Institute of Technology, Stockholm, Sweden, in 1998, and the D.Sc. degree (Hons.) in electrical engineering from the Signal Processing Laboratory, Helsinki University of Technology, Espoo, Finland, in 2002.

He was a Visiting Melchor Professor with the University of Notre Dame, Notre Dame, IN, USA, in 2019, and held an Academy Research Fellowship, funded by the Academy of Finland, from 2009 to 2014. He is currently a Professor with the Department of Electronic Systems and the Director of IoT@NTNU, Norwegian University of Science and Technology (NTNU), Trondheim, Norway. He is also an Adjunct Professor with Aalto University, Espoo, and an Adjunct Senior Research Fellow with the Institute for Telecommunications Research, University of South Australia, Adelaide. His research interests include adaptive and statistical signal processing, signal processing for communications, and security and privacy in cyber-physical systems.

Prof. Werner is a member of the Editorial Board of the *EURASIP Journal of Signal Processing* and the IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORKS.