Algorithm and Architecture Design of Random Fourier Features-based Kernel Adaptive Filters

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

***Abstract*—Numerous real-life systems exhibit complex nonlinear input-output relationships. Kernel adaptive filters, a popular class of nonlinear adaptive filters, can efficiently model these nonlinear input-output relationships. Their growing network structure, however, poses considerable challenges in terms of their hardware implementation, making them inefficient for real-time applications. Random Fourier features (RFF) facilitate the development of kernel adaptive filters with a fixed network structure. For the first time, this paper attempts to implement the RFF-based kernel least mean square (RFF-KLMS) algorithm on hardware. To this end, we propose several reformulations of the feature functions (FFs) that are computationally expensive in their native form so that they can be implemented in real-time VLSI. Specifically, we reformulate inner product evaluation, cosine, and exponential functions that appear in the implementation of FFs. With these reformulations, the proposed delayed RFF-KLMS (DRFF-KLMS) is then synthesized using 45-nm CMOS technology with 16-bit fixed-point representations. According to the synthesis results, pipelined DRFF-KLMS architectures require minimal hardware increase over the state-of-the-art conventional delayed LMS architecture while significantly improving estimation performance for the nonlinear model. Our results suggest that the cosine feature function-based DRFF-KLMS is appropriate for applications requiring high accuracy, whereas the exponential function-based DRFF-KLMS may be well suited for resource-constrained applications.**

***Index Terms*—VLSI architectures, nonlinear adaptive filters, kernel LMS, random Fourier features, cosine implementation, nearest power-of-two quantization.**

I. Introduction

THE benefit of adaptive filters for processing information in dynamic environments is well established. Linear adaptive filters, including least-mean-squares (LMS), affine projection algorithm (APA), and recursive least-squares (RLS) [1], assume that inputs and outputs are linearly related. Nonlinear models are handy tools to describe the behavior of highly complex systems. For example, multiple-input multiple-output (MIMO) communication systems have nonlinear components, such as power amplifiers that exhibit distortion owing to nonlinear effects [2]. Consequently, in a MIMO system using high-speed communication, equalizers that reconstruct the actual signal using adaptive filters can be modeled better using nonlinear adaptive filtering to guard against the inter-symbol interference (ISI). Similarly, many engineering problems can be modeled and solved effectively in nonlinear domains, including biomedical signal processing [3], time-series prediction [4], and channel equalization [5].

In contrast to conventional linear adaptive filters, Volterra filters [6], spline filters [7], and kernel methods [8] can model these nonlinear relationships effectively. Among these, kernel methods that operate in reproducing kernel Hilbert space (RKHS) have gained popularity due to their mathematical simplicity and universal approximation capabilities [9]. In modeling the nonlinear input-output relationships, kernel methods use a Mercer kernel to convert the original input space into an infinite-dimensional RKHS, where the inner product operation is efficiently performed using the kernel trick. The kernel tricks operate in an implicit, high-dimensional space without necessitating the computation of the coordinates of the input data on the feature plane, but rather by calculating the inner products of the input pairs of data in the dual space [10]. Kernel least mean square (KLMS) [11], kernel affine projection algorithm (KAPA) [12], kernel recursive least squares (KRLS) [13] are well-known kernel adaptive filtering algorithms (KAFs).

Although KAFs are effective at modeling nonlinear input-output relationships, they suffer from a linearly growing network structure, popularly known as the *growing dimensionality problem* in KAFs [14]. The problem of growing dimensionality leads to a substantial computational cost and large storage requirements. Several sparsification methods [15], such as novelty criterion, coherence criterion [8], and quantification method [16], [17] were proposed as a solution to the growing dimensionality problem. Under these methods, the network length is restricted by learning a finite-dimensional dictionary. While learning the dictionary, these methods employ thresholding and carefully weed out 'uninformative' sample inputs. Though these sparsification methods are efficient in modeling nonlinear relationships, the dictionary must be retrained whenever the underlying system experiences changes.

To deal with the growing dimensionality problem in KAFs, a comprehensive solution has been proposed utilizing random Fourier features (RFF) [18]. RFF maps the input data into a finite-dimensional space (supposed to be larger than the input regressor dimension) in which the inner products provide an

This work was partly supported by the Research Council of Norway and the Science and Engineering Research Board (SERB), Department of Science and Technology, Government of India 'Startup Research Grant' (SRG/2020/000858).

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

Ramesh Sambangi is with the Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur 721302, India (e-mail: sambangi.ramesh86@iitkgp.ac.in).

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

acceptable approximation of the kernel function evaluations. The nonlinear input-output relationship can be modeled using transformed input data together with a set of fixed-size linear parameters. As opposed to KLMS, RFF-based KLMS (RFF-KLMS) maintains a fixed network structure and does not require sparsification, so its computational load is significantly reduced, and suitable for various real-life applications [19]–[25]. While implementing the RFF-KLMS, various feature functions (FFs) have been used in the literature. More details on these FFs can be found at [19], [20]. Due to their inherent complexity, however, these FFs may be prohibited for implementation in hardware in their native form.

Due to the iterative nature of the adaptive filtering algorithms, their implementation involves high complexity and memory consumption. Consequently, the power and throughput requirements for adaptive filters cannot be met by software solutions or digital signal processing processors, especially in real-time applications, such as the rapidly growing fifth-generation (5G) and sixth-generation (6G) communication technologies [26], the internet-of-things (IoT), wearable biomedical devices [27], etc. An application-specific integrated circuit (ASIC) or field-programmable gate array (FPGA) provides a low-power alternative and benefits from the fixed-point operation. For example, the 5G new-radio (NR) global standard has higher throughput, lower latency, and new spectrum bands compared to LTE-advanced of fourth-generation (4G) communication technology [28]. The adaptive decision-feedback equalizer (ADFE) is typically used to get rid of the inter-symbol interface (ISI) in the received signal [26]. In FR1 spectrum [29], 5G NR operates in 450 MHz to 6 GHz, and provides a throughput of 1.80 to 10.85 Gbps with time-division duplex (TDD). For ADFE implementation to meet these throughput requirements, RFF-KLMS is a better choice than conventional KLMS since it has fixed-scale network structures with stable complexity. However, in the literature, most research efforts have focused on developing high-performance adaptive filter algorithms with linear properties [30], [31]. Less research is known about implementing nonlinear adaptive filters on hardware [32]–[34]. To best of our knowledge, this paper is the first attempts to enable a hardware-friendly implementation of RFF-KLMS with a fixed network structure, and that can be used as generic intellectual property (IP) in real-time applications.

Our main contributions are:

- We propose several reformulations for feature functions (FFs) evaluation such as cosine implementation through angle mapping and Taylor series expansion, selective application of logarithmic number system (LNS) to exponential and division operations, nearest power of two quantization for inner product evaluation. We then propose a pipelined VLSI architecture for the resultant algorithm termed as delayed RFF-KLMS (DRFF-KLMS) algorithm.
- To examine the impact of these reformulations on the estimation performance, we conduct numerical simulations and bit-width analysis. We provide a comparative study on the performance of the reformulated DRFF-KLMS algorithm against the original RFF-KLMS algorithm. This study reveals that the reformulated DRFF-KLMS is as efficient as the original RFF-KLMS algorithm.
- We present ASIC synthesis results of the proposed DRFF-KLMS architecture with various FFs using 45-nm CMOS technology. Considering the tradeoffs between performance and complexity, we present a comparative study of the DRFF-KLMS with different feature functions. The synthesis results demonstrate that despite the performance improvement the DRFF-KLMS architecture can be implemented with a minimal area and power overhead over the the existing pipelined delayed LMS (DLMS) architectures.

The remainder of this paper is organized as follows. Section II presents a detailed discussion on KLMS and RFF-KLMS. The performance of RFF-KLMS with different FFs is examined in Section III. Section IV presents a VLSI architecture for the DRFF-KLMS. ASIC synthesis results of the proposed DRFF-KLMS and comparison with other existing architectures are presented in Section V. Finally, Section VI concludes the paper.

II. Kernel Adaptive Filters

Conventional adaptive filter methods, such as LMS, RLS, and their variants, assume a linear relationship between the input u_n and the desired output d_n in applications like system identification and regression. In such cases, the relation between u_n and d_n can be described as

$$d_n = \mathbf{u}_n^{\mathrm{T}}\, \mathbf{h}^{\star} + \nu_n, \tag{1}$$

where $\mathbf{h}^{*} \in \mathbb{R}^L$ is an optimal parameter vector to be estimated, $\mathbf{u}_n = [u_n, u_{n-1}, \ldots, u_{n-L+1}]^{\mathrm{T}}$ is the input signal vector and ν_n is zero-mean observation noise with variance σ_ν^2. Here, T represent transpose operator. However, in many engineering problems in real-life applications, we frequently encounter nonlinear models whose input-output relationships are described as

$$d_n = f(\mathbf{u}_n) + \nu_n, \tag{2}$$

The function $f : \mathbb{R}^L \to \mathbb{R}$ is a continuous nonlinear function. Linear adaptive filters cannot model these sophisticated input-output relationships [32]–[34]. In order to estimate $f(\cdot)$ based on the data pairs $\{\mathbf{u}_i, d_i\}_{i=1:n}$, various methods have been proposed in the literature. Among these, kernel adaptive filters, namely nonlinear adaptive filters operating in the reproducing kernel Hilbert space (RKHS), have attracted considerable attention considering their mathematical simplicity.

Kernel methods evaluate the nonlinear function $f(\cdot)$ by mapping the input regressors $\{\mathbf{u}_i\}_{i=1:n}$ onto a high-dimensional feature space as $\boldsymbol{\phi}(\mathbf{u}_i)$, in which the inner products can be calculated using kernels [8], [35]. A continuous, symmetric, and positive-definite kernel function $\kappa(\cdot,\cdot) : \mathbb{R}^L \times \mathbb{R}^L \to \mathbb{R}$, satisfies the following Mercer's condition [8]:

$$\kappa(\mathbf{u}_i, \mathbf{u}_n) = \boldsymbol{\phi}^{\mathrm{T}}(\mathbf{u}_i)\boldsymbol{\phi}(\mathbf{u}_n). \tag{3}$$

Without knowing the mapping $\boldsymbol{\phi}(\cdot)$, inner products in higher dimensional space can be obtained via kernel function evalu-

ation. According to [8], a kernel is a reproducing kernel if it fulfills the following requirements:

$$\kappa(\mathbf{u}_i, \mathbf{u}_n) = \langle \kappa(\cdot, \mathbf{u}_i), \kappa(\cdot, \mathbf{u}_n) \rangle_{\mathcal{H}}, \tag{4}$$

where $\mathcal{H}$ is the RKHS in which the reproducing kernel is defined and $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ represents the corresponding inner product. In (4), $\kappa(\cdot, \mathbf{u}_i)$ is a representer evaluation at $\mathbf{x}_i$. This paper focuses exclusively on the Gaussian kernel, a well-known Mercer kernel due to its universal approximation capabilities [8], [16]. Considering the data pairs $\{\mathbf{u}_i, d_i\}_{i=1}^{n-1} \bigcup \{\mathbf{u}_n\}$, from the representer theorem [8], the evaluation of the nonlinear model output d_n, denoted by $\hat{d}_n$, is given by

$$\hat{d}_n = \sum_{i=1}^{n-1} \alpha_i \, \kappa(\mathbf{u}_n, \mathbf{u}_i). \tag{5}$$

One can observe how the model order grows with time n by looking at (5). Growing model order makes the kernel methods unsuitable for applications that require real-time updates [36]. There are various sparsification methods that can be used to learn a fixed-size dictionary while dealing with this issue. The methods decide whether or not to include a candidate regressor in the dictionary by using a similarity metric [36] between the candidate and the current dictionary. Due to the fact that dictionary training must be repeated whenever the underlying system changes, these methods are unsuitable for time-varying environments. Additionally, they are not flexible with the distributed network settings [36].

A. KLMS using RFF (RFF-KLMS)

A flexible alternative solution to sparsification methods can be obtained using RFF [18]. The shift-invariant kernel function evaluation can be approximated with an inner-product in the P-dimensional RFF space. Thus, the nonlinear filtering problem becomes a linear one in large but finite-dimensional RFF space. Consider $\mathbf{x}_n$ as the mapping of $\mathbf{u}_n$ into the P-dimensional RFF space $\mathbb{R}^P$. Then, the kernel function evaluation can be approximated as $\kappa(\mathbf{u}_i, \mathbf{u}_n) \approx \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_n$; as a result, $\hat{d}_n$ can be alternatively expressed as

$$\hat{d}_n = \sum_{i=1}^{n-1} \alpha_i \, \kappa(\mathbf{u}_i, \mathbf{u}_n) \approx \Big(\sum_{i=1}^{n-1} \alpha_i \, \mathbf{x}_i \Big)^{\mathrm{T}} \mathbf{x}_n = \mathbf{w}^{\mathrm{T}} \mathbf{x}_n. \tag{6}$$

In the above, $\mathbf{w} \in \mathcal{R}^P$ is a linear representation of nonlinear function $f(\cdot)$ in the P-dimensional RFF space. The kernel function can be well approximated as P increases. By solving the following optimization problem, the linear representation of $f(\cdot)$ in the RFF space, i.e., $\mathbf{w}$, can be estimated:

$$\min_{\mathbf{w} \in \mathcal{R}^P} \mathrm{E}[(d_n - \mathbf{x}_n^{\mathrm{T}} \mathbf{w})^2]. \tag{7}$$

The following RFF-based KLMS update rule results from stochastic gradient descent iterations:

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mu e_n \mathbf{x}_n \tag{8}$$

with $e_n = d_n - \hat{d}_n$, where $\hat{d}_n = \mathbf{x}_n^{\mathrm{T}} \mathbf{w}_n$ and μ is the adaptation step size. The range of μ is dependent on the maximum eigenvalue of the mapped data correlation matrix, i.e., $0 < \mu < 1/\lambda_{\max}(\mathbf{R})$, where $\mathbf{R} = \mathrm{E}[\mathbf{x}_n \mathbf{x}_n^{\mathrm{T}}]$ [22]. The input $\mathbf{u}_n$ can be mapped into the P-dimensional RFF space $\mathbb{R}^P$ using cosine, exponential and Gaussian FFs.

1) Cosine feature function: From the cosine feature function, the mapped vector $\mathbf{x}_n$ can be computed as follows [14]:

$$\mathbf{x}_n = (P/2)^{-\frac{1}{2}} [\cos(\mathbf{v}_1^{\mathrm{T}} \mathbf{u}_n + b_1), \ldots, \cos(\mathbf{v}_P^{\mathrm{T}} \mathbf{u}_n + b_P)]^{\mathrm{T}}, \tag{9}$$

where the phase terms $\{b_i\}_{i=1:P}$ are drawn from a uniform distribution on the interval $[0, 2\pi]$. Vectors $\{\mathbf{v}_i\}_{i=1:P}$ are drawn from the probability density function $p(\mathbf{v})$ such that

$$k(\mathbf{u}_n - \mathbf{u}_i) = \int p(\mathbf{v}) \exp(\mathrm{j}\mathbf{v}^{\mathrm{T}}(\mathbf{u}_n - \mathbf{u}_i)) \mathrm{d}\mathbf{v}, \tag{10}$$

where $\mathrm{j}^2 = -1$. In other words, the Fourier transform of $k(\mathbf{u}_n - \mathbf{u}_i)$ is given by $p(\mathbf{v})$.

2) Exponential feature function: Using the exponential feature function, the mapped vector $\mathbf{x}_n$ can be generated as [20]:

$$\mathbf{x}_n = \Big[\exp(-(\mathbf{v}_1^{\mathrm{T}} \mathbf{u}_n + b_1)), \ldots, \exp(-(\mathbf{v}_P^{\mathrm{T}} \mathbf{u}_n + b_P)) \Big]^{\mathrm{T}}, \tag{11}$$

where the phase terms $\{b_i\}_{i=1:P}$ are drawn from a uniform distribution and vectors $\{\mathbf{v}_i\}_{i=1:P}$ are the same as discussed in the Section II-A1.

3) Gaussian feature function: Using the Gaussian feature function, the mapped vector $\mathbf{x}_n$ can be obtained as [19]:

$$\mathbf{x}_n = \Big[\exp(-\frac{\|\mathbf{u}_n - \mathbf{v}_1\|^2}{2\, b_1^2}), \ldots, \exp(-\frac{\|\mathbf{u}_n - \mathbf{v}_P\|^2}{2\, b_P^2}) \Big]^{\mathrm{T}}, \tag{12}$$

where the phase terms $\{b_i\}_{i=1:P}$ are drawn from a uniform distribution and location parameters $\{\mathbf{v}_i\}_{i=1:P}$ are generated from the same probability density function as the input.

III. Study on the Performance of RFF-KLMS for Different FFs

This section investigates the performance of the RFF-KLMS algorithm using cosine, exponential, and Gaussian FFs. To this end, we conducted a series of experiments in the context of nonlinear system identification. In all experiments, both the input signal $\mathbf{u}_n$ and the observation noise ν_n were considered to be zero-mean Gaussian random processes with input signal variance $\sigma_u^2 = 1$ and noise variance $\sigma_\nu^2 = 0.03$, respectively. The mean square error (MSE), defined by $\mathrm{MSE} = \mathrm{E}[e_n^2]$, is considered as a performance metric. The first experiment tested the RFF-KLMS in identifying and tracking the following nonlinear systems:

$$\begin{aligned} f_1(\mathbf{u}_n) &= 0.9 - 0.5 \exp(-u_{1,n}^2) u_{2,n} + 0.2 \sqrt{\sin(\pi u_{2,n})}, \\ f_2(\mathbf{u}_n) &= 0.9 - 0.5 \exp(-u_{2,n}^2) u_{1,n} + 0.2 \sqrt{\sin(\pi u_{1,n})}. \end{aligned} \tag{13}$$

An abrupt change was considered halfway through the iterations, i.e., after 4000 samples, the underlying nonlinear system changes from $f_1(\mathbf{u}_n)$ to $f_2(\mathbf{u}_n)$. The dimensionality of the RFF space was set to $P = 32$. The learning curves, i.e., MSE in dB vs. iteration index (n), are displayed in Fig. 1. All simulations in this section were performed using MATLAB. The step sizes μ were adjusted so that the learning

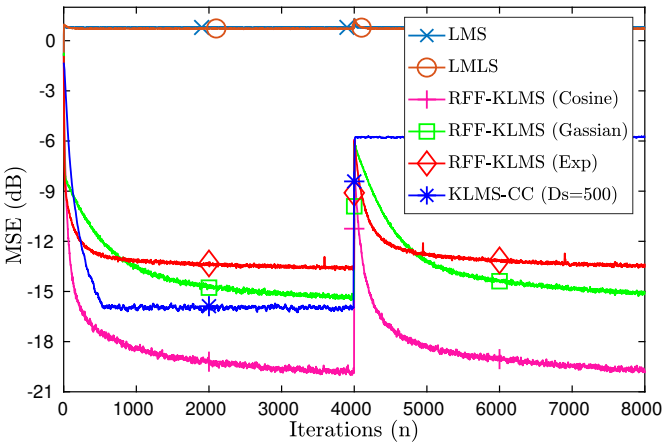

Fig. 1. Learning curves (MSE vs iterations (n)) of RFF-KLMS for different FFs when $P = 32$. Also included the learning curve of conventional LMS, least mean logarithmic square (LMLS) [30], and KLMS with coherence criterion (KLMS-CC) [8] with a dictionary size (Ds) of 500.

curves attain the same initial convergence rate or steady-state MSE. For comparative assessment, the learning curve of conventional LMS, least mean logarithmic square (LMLS) [30], and KLMS with coherence criterion (KLMS-CC) [8] with a dictionary size of 500 is plotted in Fig. 1. From Fig. 1, we see that RFF-KLMS with different FFs successfully identifies and tracks the nonlinear systems while the conventional LMS and LMLS fails to do so. KLMS-CC has a growing dimensionality problem with time n, requiring a large dictionary size for decent steady-state values. In addition, KLMS-CC with a limited dictionary size exhibits poor tracking performance since the dictionary is not updated when the system to be identified changes. Furthermore, RFF with cosine FF attained lowest steady-state MSE (i.e., -19dB), while exponential and Gaussian FFs based RFF-KLMS reached -13dB and -15dB, respectively. Finally, we can see that the RFF-KLMS with any FF performs identification and tracking tasks consistently regardless of the initial conditions.

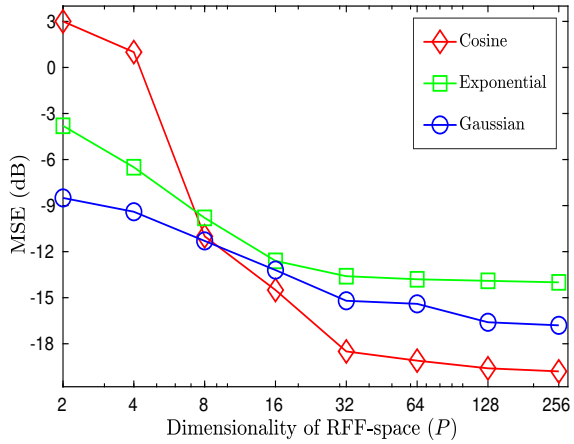

Fig. 2. Steady-state MSE vs. the dimensionality of the RFF space (P) for various FFs.

Next, to examine the effect of P (i.e., the dimensionality of the RFF space) on the performance of RFF-KLMS with various FFs, we repeated the above simulation exercise for different values of P, where $P \in \{2, 4, 8, 16, 32, 64, 128, 256\}$. The rest of the settings remain unchanged. The corresponding steady-state MSE (in dB) vs. P is shown in Fig. 2. From Fig. 2, we can see that the cosine feature function based RFF-KLMS performs poorly compared to other variants for very small values of P (e.g., 2 and 4). As the P value increases (i.e., $P \geq 8$), it achieves lower steady-state MSE than other variants. Beyond $P = 64$, as P increases further, we observe a minor decrease in the steady-state MSE of all variants of RFF-KLMS. Therefore, in the remainder of this paper, the dimensionality of the RFF-space is set to $P = 64$ for a better trade-off between steady-state MSE and computational complexity.

Finally, to examine the effect of kernel bandwidth σ on the performance of RFF-KLMS for both cosine and exponential FFs, we carried out the system identification exercise for $f_1(\cdot)$ with different values of σ. The resulting learning curves are presented in Fig. 3 for $P = 64$. We can see that the optimal σ values are 1 and 3.5 for RFF-KLMS with cosine and exponential FFs, respectively. The learning curve of the exponential feature function diverges when the value of σ is less than 3.5.

A. Bit-Width Consideration of the Proposed DRFF-KLMS Fixed-point Implementation

The DRFF-KLMS algorithm with reformulated FFs was simulated for the same system identification problem mentioned in section III using 16-bit, 12-bit, and 8-bit fixed-point representations. The corresponding learning curves are presented Fig. 4. For comparative assessment, the original floating-point DRFF-KLMS was also simulated. From Fig. 4, it is apparent that the 16-bit fixed-point representation exhibits approximately the same steady-state performance as the DRFF-KLMS floating-point algorithm. Due to its stochastic and iterative nature, the reformulated DRFF-KLMS is tolerant to all proposed approximations. In addition, DRFF-KLMS performance deteriorates with reduced bit-width. Therefore, a 16-bit architecture is preferable for the VLSI implementation of DRFF-KLMS.

IV. VLSI Architecture for Proposed Delayed RFF-KLMS Algorithm

This section details the proposed VLSI architecture for RFF-KLMS with different FFs. We propose several reformulations to RFF-KLMS FFs that result in minimal algorithmic performance loss with significant computational complexity reduction. In the following, we explain the proposed reformulations.

A. RFF-KLMS Architecture

The VLSI architecture of RFF-KLMS consists of two main blocks, namely the feature function block and the LMS-type adaptive filter block. Features function block generates mapped data in a P-dimensional RFF-space from an input vector $\mathbf{u}_n$. The LMS-type adaptive filter block then updates

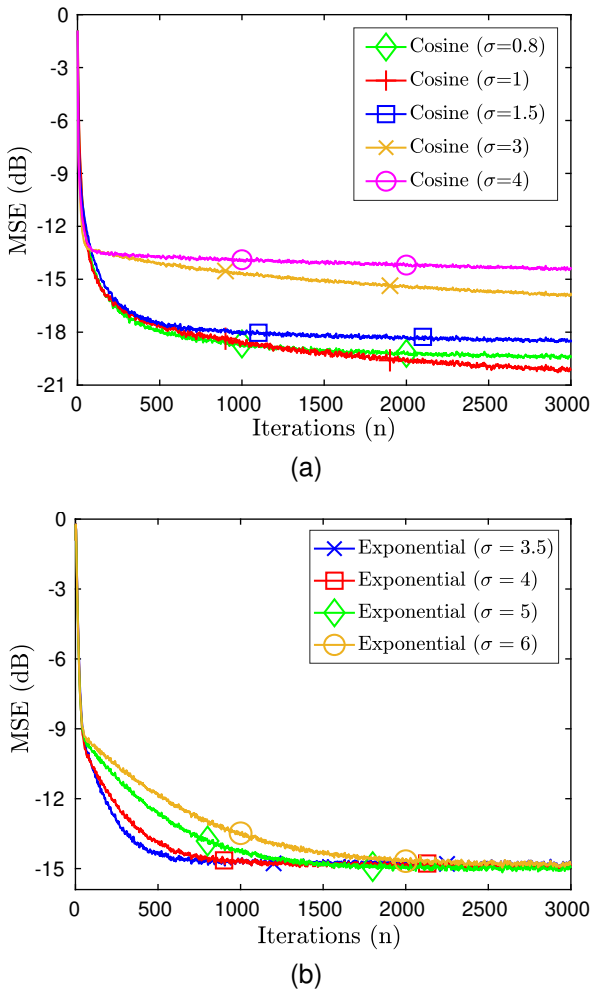

Fig. 3. Learning curves (MSE vs iterations (n)) of RFF-KLMS for different values of kernel bandwidth (σ). Dimensionality of the RFF space (P) is 64.

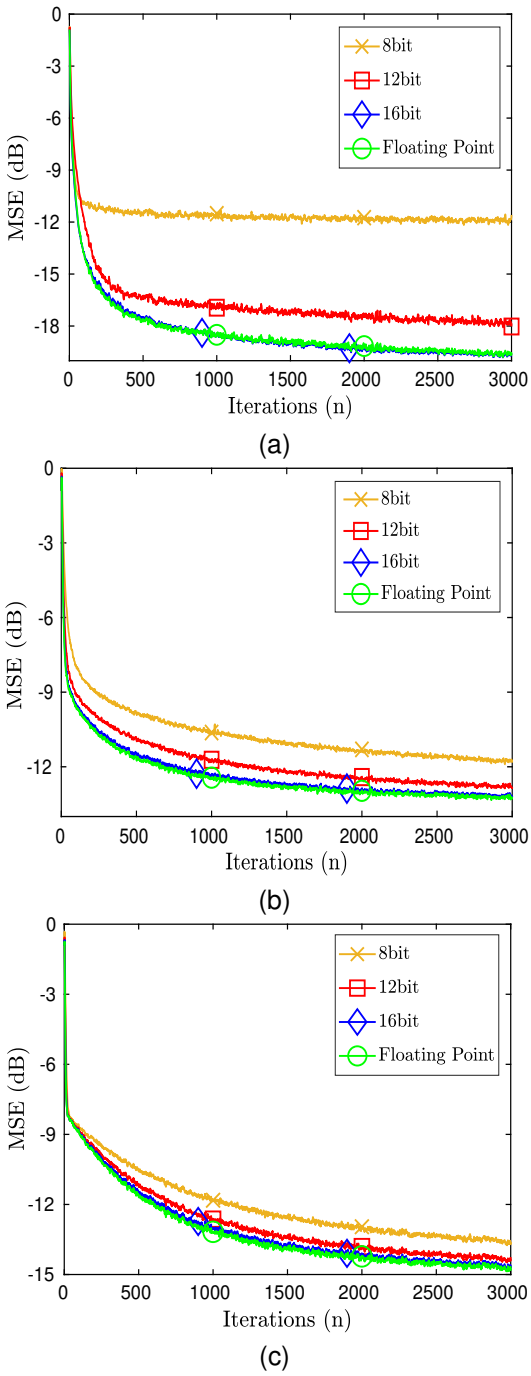

Fig. 4. Learning curves (MSE vs iterations (n)) of DRFF-KLMS floating point and fixed-point representations for various FFs: (a). Cosine. (b). Exponential. (c). Gaussian.

the filter weights using the mapped data. An adaptive filter block of the LMS type can be implemented using any LMS architecture available in the literature. It is important to note that the LMS architecture can be transformed into a RFF-KLMS by adding a feature function block prior to that. However, in real-time VLSI implementations of high-throughput adaptive filter blocks, the LMS algorithm cannot be directly mapped into hardware [30], [37]. After each iteration, the LMS algorithm evaluates the error e_n and updates the weight vector $\mathbf{w}_n$. This feedback loop restricts pipelining of the architecture. Therefore, re-timing techniques [38]–[40] must be applied to the VLSI architecture to formulate a DRFF-KLMS algorithm. For the resultant DRFF-KLMS algorithm, the weight update equation is

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \ e_{n-M}\mathbf{x}_{n-M}, \tag{14}$$

where M is the delay in adaptation. As the number of adaptation delays increases, however, the convergence rate of DRFF-KLMS decreases accordingly. Therefore, the adaptation delay should be kept low to maintain a good performance-throughput trade-off.

The VLSI architecture of the proposed DRFF-KLMS algorithm is shown in Fig. 5. The architecture essentially implements (14). We can see that the proposed architecture has two blocks, i.e., the feature function block and the adaptive filter block. Considering that the exponential, cosine, and Gaussian FFs involve different reformulations, we provide a detailed analysis and corresponding architectural diagram for each separately. Throughout the architecture, the bit widths of all the intermediate signals are shown in $Q_{n.m}$ format (where n and m denote the number of integers and fractional bits, respectively). Note that the value of j is calculated using expression $\mu = 2^{-j}$. In order to prevent overflows,

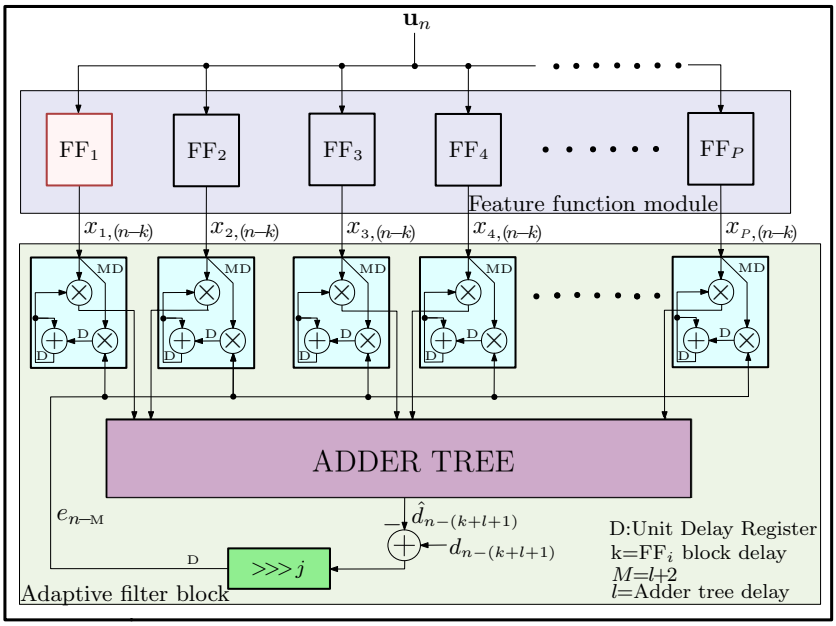

Fig. 5. Proposed VLSI architecture of DRFF-KLMS

the bit widths are determined by the MATLAB floating-point to fixed-point conversion methods. In the following sections, we describe the proposed reformulation for efficient hardware development of feature function modules.

B. Feature Function Module

The feature function module maps the input vector $\mathbf{u}_n$ into P-dimensional RFF-space using either of the FFs discussed in Section II-A. The feature function module contains P number of sub-blocks, where ith sub-block takes $\mathbf{u}_n$ as the input and produces ith element of the mapped data vector $\mathbf{x}_n$, i.e., $x_{i,n}$ for $1 \leq i \leq P$. A straightforward implementation of FFs in hardware is inefficient due to vector multiplications, cosine/exponential operations in the case of cosine/exponential FFs, and square, exponential, and division operations in the case of Gaussian FFs. Therefore, we propose certain approximations and reformulations to achieve efficient hardware implementations of FFs with minimal loss of accuracy. The choice of feature function depends on the application at hand. Following is a detailed description of how the feature function module is implemented on hardware.

1) Inner product calculation: The input mapping process given in (9) and (11), begins with the evaluation of the inner product, i.e., $(\mathbf{v}_i^{\mathrm{T}}\mathbf{u}_n)$ for $i = 1, 2, \ldots P$. Here, $\{\mathbf{v}_i\}_{i=1:P}$ is a random weight vector drawn from the probability density function $p(\mathbf{v})$. Due to the large number of multipliers required, direct evaluation of inner product is costly in VLSI. Using the fact that the elements of the vector $\mathbf{v}_i^{\mathrm{T}}$ are fixed both in cosine and exponential FFs, we can employ multipliers-free vector multiplication methods [41]–[43] for a hardware friendly implementation of feature function module.

Multiplier-free multiplication using distributed arithmetic (DA) has the advantage of minimizing the area of implementation. DA removes the need for conventional multipliers by distributing multiply operations across shifters, read-only memories (ROMs), and adders [44], [45]. The vector $\mathbf{v}_i$ in inner product ($p_i = \mathbf{v}_i^{\mathrm{T}}\mathbf{u}_n$) is a constant. Therefore, by considering the contents of vector $\mathbf{v}_i = [v_{i,1}, v_{i,2}, v_{i,3}, \ldots, v_{i,L}]^{\mathrm{T}}$ and input signal vector $\mathbf{u}_n = [u_n, u_{n-1}, u_{n-2}, \ldots, u_{n-L+1}]^{\mathrm{T}}$, this inner product can be implemented using DA. The inner product of these two vectors can be alternatively expressed as

$$\begin{aligned} p_i &= \sum_{k=1}^{L} v_{ik} u_{n-k+1} \\ &= v_{i1}u_n + v_{i2}u_{n-1} + v_{i3}u_{n-2} + \cdots + v_{iL}u_{n-L+1} \end{aligned} \tag{15}$$

Here, $v_{ik} = -v_{ik}^0 + \sum_{j=1}^{B-1} v_{ik}^j 2^{-j}$ (where v_{ik}^0 is the sign bit) and $u_n = -u_n^0 + \sum_{j=1}^{B-1} u_n^j 2^{-j}$ (where u_n^0 is the sign bit) are fractional numbers represented using B-bit signed fixed-point notation. Using B-1 bits, we represent fractional part, and the remaining bit (MSB) represents the sign of the number. In the fixed-point notation with the terms rearranged according to the weights of the bits, (15) can be written as follows:

$$\begin{aligned} p_i = &- \left(v_{i1}u_n^0 + v_{i2}u_{n-1}^0 + v_{i3}u_{n-2}^0 + \cdots + v_{iL}u_{n-L+1}^0\right) \\ &+ \left(v_{i1}u_n^1 + v_{i2}u_{n-1}^1 + v_{i3}u_{n-2}^1 + \cdots + v_{iL}u_{n-L+1}^1\right) 2^{-1} \\ &+ \left(v_{i1}u_n^2 + v_{i2}u_{n-1}^2 + v_{i3}u_{n-2}^2 + \cdots + v_{iL}u_{n-L+1}^2\right) 2^{-2} \\ &+ \cdots + \left(v_{i1}u_n^{B-1} + v_{i2}u_{n-1}^{B-1} + \cdots + v_{iL}u_{n-L+1}^{B-1}\right) 2^{-(B-1)} \end{aligned} \tag{16}$$

The partial inner-products (PPs) in (16) is denoted by,

$$f_i\left(u_n^t, u_{n-1}^t, \ldots, , u_{n-L+1}^t\right) = \sum_{j=1}^{L} v_{ij} u_{n-j+1}^t \tag{17}$$

Equation (16) can be represented using (17) as

$$\begin{aligned} p_i = &-f_i\left(u_n^0, u_{n-1}^0, \ldots, u_{n-L+1}^0\right) \\ &+ f_i\left(u_n^1, u_{n-1}^1, \ldots, u_{n-L+1}^1\right) 2^{-1} \\ &+ f_i\left(u_n^2, u_{n-1}^2, \ldots, u_{n-L+1}^2\right) 2^{-2} + \cdots \\ &+ f_i\left(u_n^{B-1}, u_{n-1}^{B-1}, \ldots, u_{n-L+1}^{B-1}\right) 2^{-(B-1)} \end{aligned} \tag{18}$$

The function, say $f_i(.)$ has 2^L distinct values $(0, v_{i1}, v_{i2}, v_{i1} + v_{i2}, v_{i3}, \ldots)$ for all possible value combinations of $u_n^t, u_{n-1}^t, u_{n-2}^t, \ldots, u_{n-L+1}^t$. Instead of calculating (17) at run-time, we can pre-compute and store these values in a read only memory (ROM). We can implement (18) by supplying tth bit values of all elements $(u_n^t, u_{n-1}^t, u_{n-2}^t, \ldots, u_{n-L+1}^t)$ as address inputs to the ROM. In every cycle, the accumulator content is added with the fetched value from the ROM and shifted to the right to accommodate weights at different bit locations. The evaluation is complete when the ROM contents (corresponding to MSBs address) are subtracted from the accumulator contents. So, we need B clock cycles to finish this computation. Hereafter, these implementations are referred to as bit-serial DA designs, as they involve serial operations. The serial shifting in bit-serial DA slows down the computation, making it unsuitable for applications which require high performance.

To overcome the limitations of bit-serial DA, approximate bit-parallel DA architecture has been proposed in [46]. In bit-parallel DA designs, all the PPs are processed in parallel using B LUTs. The final result is obtain by performing addition operation on these PPs using shift-adder-tree (SAT) [47]. The total area of bit-parallel DA design is dominated by the LUTs and adder-width of the SAT. To reduce the area and energy by maintaining better mean error distance (MED), authors in [48], truncate m LSBs of PPs as well as on the m LSBs of the inputs. Truncation on m LSBs of PPs reduces the adder-width by m in SAT and reduces the corresponding LUT width by m. In addition to it, truncation on m LSBs of coefficients reduces the number of LUTs to $B-m$, and number of adders required in SAT. To perform truncation on LSBs of PPs, authors in [48] propose two weight dependent truncation approaches called, row truncation (DA_RT) and column truncation (DA_CT) approach. In DA_RT, m LSBs of PPs and inputs are truncated. Similarly, in DA_CT, m LSBs of the inputs are truncated and number of LSBs of PPs to be truncated calculated using [48]. The area of DA_RT and DA_CT dominated by LUTs, makes it unsuitable for low-power applications.

Meanwhile, powers-of-two quantization reduces the time needed to evaluate inner products by replacing the multiplication step with a simple shifted operation [41]. So, we quantize the vector $\{\mathbf{v}_i\}_{i=1:P}$ to the nearest power of 2 using the Algorithm 1. In Algorithm 1, $\{\mathbf{v}_i'\}_{i=1:P}$ represents the quantize output of the vector $\{\mathbf{v}_i\}_{i=1:P}$. After that, we calculate the inner product $(\mathbf{v}_i^{\mathrm{T}}\mathbf{u}_n)$ using wired shift operation followed by the addition operation.

To verify the effect of nearest powers-of-two quantization (NPTQ) on the performance of DRFF-KLMS, we carried out the same simulation exercise presented in Section III-A. The corresponding learning curves are shown in Fig. 4. Fig. 4 confirms that the performance loss is negligible due to power-of-two quantization. We have implemented the inner product using the above approaches in Verilog and synthesized using cadence 45-nm CMOS library. As in [44], we implement the inner product using bit-serial DA. Approximate parallel DA using row and column truncation approach implemented using [48]. For all the designs, we set 1 bit for the integer part and 15 bits for the fractional part. Note that bit-serial DA synthesis results are presented at 131.02 MHz clock frequency. Because bit-serial DA requires two sources of clock [49]. The actual operation of the bit-serial DA is performed using bit clock clk_b while the design calculates the output at every sample clock clk_s. The values of clk_s is B times slower than clk_b. In the case of clk_b, the maximum frequency reached is 2096.43MHz, therefore clk_s is 131.02MHz. Thus, for bit-serial DA, the sample clock frequency is listed in Table I. A comparison of the synthesis results is presented in Table I. From these synthesis results, we see that the powers-of-two quantization is 8 times faster and requires 80% less area compared to the DA approach. We also observe that the NPTQ reduces area by 89%, 86%, and 84% compared to DA_RT for $m = 2, 3, \text{and } 4$ respectively. Similarly, we observe that the NPTQ reduces area by 80%, 79%, and 79% compared to DA_CT for $m = 2, 3, \text{and } 4$ respectively. Importantly, the dynamic power consumption of the bit-serial DA, DA_RT (for $m = 4$), and DA_CT (for $m = 4$) are 81.4%, 81.6%, and 74.8% more than that of the NPTQ respectively. Taking these findings into account, we continue our architectural design with the NPTQ.

Algorithm 1 Nearest power-of-two quantization method

Input: $\{\mathbf{v}_i\}_{i=1:P}$, L, P
Output: $\{\mathbf{v}_i'\}_{i=1:P}$

```
f_i, e_i, g_i = 0_L, ∀i ∈ {1, 2, ..., P};
for i ← 1 to P do
    g_i = sign(v_i);
end for
for i ← 1 to P do
    f_i = 2^⌊log2|g_i|⌋ ;
    e_i = 2^⌈log2|g_i|⌉;
end for
for i ← 1 to P do
    for j ← 1 to L do
        if |v_{i,j} − f_{i,j}| < |v_{i,j} − e_{i,j}| then
            v'_{i,j} = f_{i,j};
        else
            v'_{i,j} = e_{i,j};
        end if
    end for
end for
```

After adding the phase term b_i to the inner product $\mathbf{v}_i^{\mathrm{T}}\mathbf{u}_n$, the result will be fed to the cosine or exponential function to generate the mapped data $x_{i,n-k}$. In the following, we present an efficient implementations of these functions.

2) Cosine feature function: Many efficient fixed-point implementations of trigonometric functions are available in the literature. Of these, CORDIC-based implementations [50] are

TABLE I
SYNTHESIS RESULTS OF INNER PRODUCT IMPLEMENTATION USING DA AND NEAREST POWERS-OF-TWO QUANTIZATION METHODS.

Design	m	Clock freq. (MHz)	Total Area (μm^2)	LP[a] (μW)	DP[b] (mW)
Bit-serial DA [44]	-	131.02	2357	0.332	7.69
DA_RT [48]	2	1052.63	4322	0.492	10.98
	3	1052.63	3591	0.408	9.26
	4	1052.63	2994	0.339	7.79
DA_CT [48]	2	1052.63	2349	0.247	6.02
	3	1052.63	2287	0.236	5.83
	4	1052.63	2248	0.231	5.69
NPTQ[c]	-	1052.63	472	0.056	1.43

Leakage Power[a], Dynamic Power[b], nearest powers-of-two quantization[c]

popular and hardware efficient. However, CORDIC algorithms utilize considerable space and require a higher number of clock cycles to evaluate trigonometric functions, making them inefficient for area-efficient high-speed applications. As a solution, we implement the cosine function using approximated Taylor series.

Let $a_i = \mathbf{v}_i^{\mathrm{T}}\mathbf{u}_n + b_i$. Then, Taylor series expansion of cosine(a_i) is

$$\text{cosine}(a_i) = 1 - \frac{a_i^2}{2!} + \frac{a_i^4}{4!} - \cdots \tag{19}$$

For efficient implementation of (19) on hardware, we consider only the first three terms. The error due to this approximation is less if the value of $a_i \in [-\pi/2, \pi/2]$. So, we transform the given value of a_i to lie in the range $-\pi/2$ to $\pi/2$ with appropriate sign correction. The transformation of the value of a_i and its sign correction is summarized in Algorithm 2. In Algorithm 2, t_i represents the mapping of a_i into the range $-\pi/2$ to $\pi/2$.

Algorithm 2 Angle Mapping

Input: a_i
Output: t_i, $sign$
1: **if** $sign(a_i)$ **then**
2: $a_i = |a_i|$
3: **end if**
4: **if** $0 \le a_i \le \pi/2$ **then**
5: $t_i = a_i$
6: $sign$ = 1
7: **else if** $\pi/2 \le a_i \le 3\pi/2$ **then**
8: $t_i = |\pi - a_i|$
9: $sign$ = -1
10: **else**
11: $t_i = |2\pi - a_i|$
12: $sign$ = 1
13: **end if**

Taylor series expansion of cosine(a_i) given in (19) contains division, square and fourth power terms. Fixed-point implementations of these terms are complex and require more clock cycles. Usage of logarithmic number system (LNS) makes these implementations simpler and hardware friendly [40]. The LNS transforms the division operation into subtraction, and square and fourth power operations turn into shifting operations. Note that LNS does increase the complexity of additions and subtractions. For this reason, LNS was used only in the feature function block which contains many exponentiation, division and multiplication operations. Using LNS, Taylor series approximation-based cosine function implementation is described as follows:

$$R = \log_2\left(\frac{t_i^2}{2}\right) = 2\log_2(t_i) - 1 \tag{20a}$$

$$Q = \log_2\left(\frac{t_i^4}{4!}\right) = 4\log_2(t_i) - \log_2(4!) \tag{20b}$$

$$\text{cosine}(a_i) = sign \times (1 - 2^R + 2^Q) \tag{20c}$$

We use Mitchell's scheme [30], [51] for the fixed-point implementation of logarithm and antilogarithm in (20). To examine the accuracy of the proposed Taylor series approximation-based cosine function evaluation, we compared it with the floating-point cosine function as shown in Fig. 6. The cosine function is implemented using Verilog 16-bit fixed-point representation.In addition, we set 4 bits for the integer part and 12 bits for the fractional part. Fig. 6 compares the cosine values computed from the Verilog simulations and the actual values computed from the MATLAB floating-point simulations. From Fig. 6, we see that the proposed Taylor series approximation-based cosine function implementation produces negligible error compared to the original cosine function implementation.

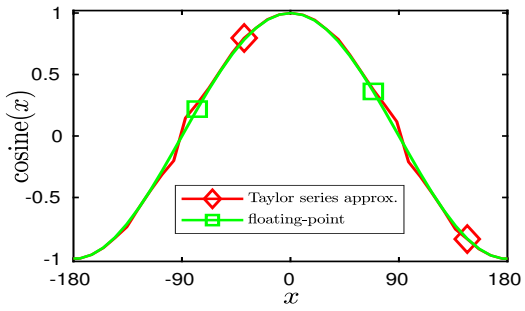

Fig. 6. Comparison of cosine function implemented using Taylor series (2-terms) with actual cosine values.

TABLE II
SYNTHESIS RESULTS OF CORDIC AND TAYLOR SERIES APPROXIMATION-BASED COSINE FUNCTION IMPLEMENTATIONS FOR CLOCK FREQUENCY 1052.63 MHZ.

Design	Total Area (μm^2)	Leakage Power (μW)	Dynamic Power (mW)
TSA[a]	2059	0.18	0.77
CORDIC [53]	16322	1.88	10.50

[a] Taylor series approx.

Synthesis results of cosine function implementation using the proposed Taylor series approximation are summarized in

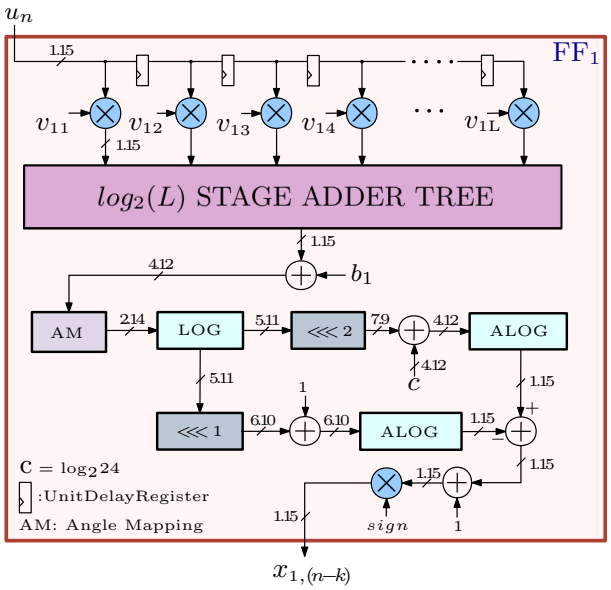

Fig. 7. Cosine feature function architecture for subblock (FF1).

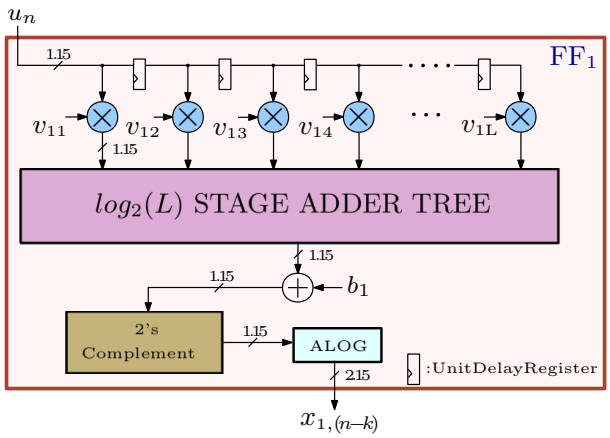

Fig. 8. Exponential feature function architecture for subblock (FF1).

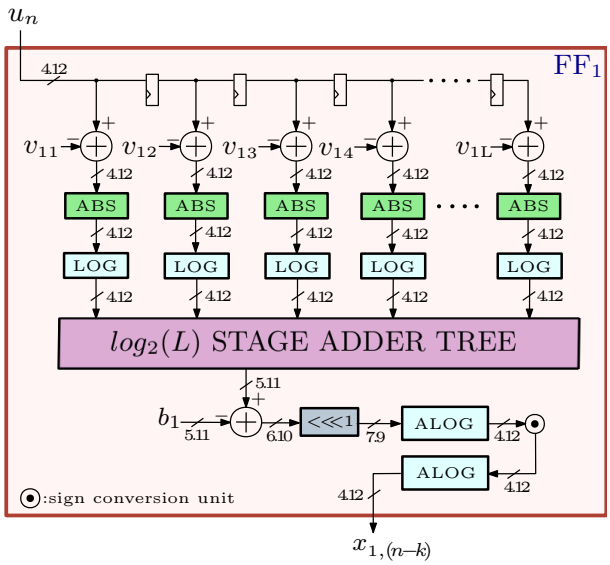

Fig. 9. Gaussian feature function architecture for subblock (FF1).

Table II. For comparative assessment, we also implemented the cosine function using the CORDIC [52], [53]. In this implementation, we set 4 bits for the integer part and 12 bits for the fractional part. The Taylor series approximation approach and CORDIC were implemented in Verilog and synthesized using Cadence 45-nm CMOS library. The CORDIC algorithm requires 16 pipeline stages, while the proposed approach requires only one pipeline stage for achieving the clock frequency of 1052.63 MHz. The synthesis results reveal that the CORDIC-based cosine function implementation occupies 7.9 times more area and consumes 13.6 times more power than the proposed Taylor series approximation-based implementation. Therefore, we use Taylor series approximation-based cosine function implementation while designing the architecture for the cosine feature function. Integrating all these reformulations together, the reformulated cosine feature function is then implemented using Mitchell's scheme, as shown in Fig. 7.

3) Exponential feature function: Realization of the exponential feature function (11) involves inner product followed by exponential operation. The vector multiplication is the same as in the cosine feature function and can be implemented using the nearest powers-of-two quantization. To reduce the implementation cost of exponential operation in hardware, we replace the exponential with 2^x function as in [30]. The reformulated exponential feature function is then implemented using Mitchell's scheme, as shown in Fig. 8.

4) Gaussian feature function: Realization of Gaussian feature function (12) involves l_2-norm, exponential and division operations. A straightforward VLSI implementation of these operations occupies a large chip area. Therefore, we propose certain reformulations in the following for efficient real-time VLSI implementation of the Gaussian feature function.

To reduce the implementation cost of the exponential function presented in (12), we replace it with the 2^x function, which is easy to implement in hardware, as discussed in the case of the exponential feature function. To further reduce the hardware complexity, we employ the LNS and transform the l_2-norm, exponential and division operations into addition, shifting, logarithm, and antilogarithm that are comparatively easy to implement in hardware. With these reformulations, the Gaussian feature function becomes

$$\mathbf{x}_n = \left[2^{-2^{2\,(\mathbf{s}_1-\mathbf{c}_1)}}, 2^{-2^{2\,(\mathbf{s}_2-\mathbf{c}_2)}}, \ldots, 2^{-2^{2\,(\mathbf{s}_P-\mathbf{c}_P)}}\right]^{\mathrm{T}}, \quad (21)$$

with

$$s_i = \sum_{j=1}^{L} \log_2|\mathbf{u}_n - \mathbf{v}_i|]^{\mathrm{T}}, \text{and } c_i = L\ \log_2(\sqrt{2}\, b_i), \quad (22)$$

where $(1 \leq i \leq P)$. The feature function block for the Gaussian feature function employing the above-proposed reformulations is shown in Fig. 9. From the architecture, one can easily see that the reformulated feature function subblock containing LNS, addition, and shifting operations is hardware-friendly compared to its original form.

To examine the VLSI implementation aspects, we implemented all the above-discussed reformulated designs of cosine, exponential, and Gaussian FFs in Verilog and synthesized

TABLE III
SYNTHESIS RESULTS OF A FEATURE FUNCTION(FF) MODULE IMPLEMENTED USING DIFFERENT FFS AT CLOCK FREQUENCY 1052.63 MHZ.

Feature function	Total Area (μm^2)	Leakage Power (μW)	Dynamic Power (mW)
Cosine	3642	0.42	8.01
Exponential	1212	0.14	3.15
Gaussian	3550	0.39	7.78

using Cadence 45-nm technology libraries. The bit-widths for integer and fractional parts of all the intermediate signals in the designs are specified in their corresponding architectural designs, i.e., in Figs. 7, 8, and 9. Note that k is the pipeline depth of the architecture and will be determined by the critical path.

Table III shows the hardware requirement of the sub-block module for different FFs. So the additional hardware requirement for DRFF-KLMS to be implemented compared to delayed least mean square (DLMS) architecture is $P\times$ the hardware requirement of the sub-block module. An important feature of the proposed algorithm is that the logic specific to DRFF-KLMS can be integrated into any LMS architecture described in the literature. Based on the synthesis results presented in Table IV, we see that the exponential FF design requires a minimal area and power but exhibits higher steady-state MSE than its counterparts, as shown in Fig. 1. On the other hand, cosine FF requires a larger area and power but lower steady-state MSE than its counterparts, as shown in Fig. 1. In order to select the appropriate FF, it is necessary to weigh the trade-offs between MSE value and VLSI implementation aspects (such as area and power) relevant to the application.

Please note that LNS is employed selectively in the proposed design. It is only used in the feature function block that contains many exponentiation, division, and multiplication operations. ABS and LOG units in the feature function architectures convert the operands from 2's complement to LNS, while ALOG and sign conversion units convert the results back to 2's complement representation.

V. ASIC SYNTHESIS RESULTS

The proposed DRFF-KLMS architectures were implemented in Verilog HDL and simulated using the Cadence NCSim simulator. We implemented all the operations in the DRFF-KLMS architecture, including LOG and ALOG, in fixed-point MATLAB, and we used the results as our golden reference values. MATLAB's golden reference output sets were compared to Verilog simulation results for various random input sets. Following verification, the designs were synthesized using the Cadence Genus tool using 45nm CMOS technology. The designs were all represented in a 16-bit fixed-point representation. Fig. 6 and Fig. 9, for instance, explicitly mention the bit widths of the FFs in their respective architectures. A set of 100000 input samples was used for Verilog simulation, and the results were verified against MATLAB outputs. After verification, the designs were synthesized with the Cadence Genus tool in 45-nm CMOS technology. The DRFF-KLMS uses cosine, exponential, and Gaussian FFs designed separately. Each of these DRFF-KLMS variants was synthesized with filter orders of 16, 32, and 64 to verify the scalability of the proposed architectures. Results of the synthesis are tabulated in Table IV. A prior architecture for DRFF-KLMS is not available, so the results are compared to state-of-the-art DLMS [37] architectures. The DLMS architectures from [37] were also coded in Verilog and synthesized using the same 45-nm libraries for a fair comparison. Since DLMS is one of the lowest complexity adaptive filtering algorithms that can be implemented in hardware, we chose it for comparing DRFF-KLMS synthesis results. In both DRFF-KLMS and DLMS, we have set M=3 so that the critical path of their designs is T_{mult} with a slack value of 0. Table III compares synthesis results of cosine module implemented with CORDIC and the Taylor series approximation approach. Tables III and IV represent dynamic power values that were extracted from post-synthesis power reports by annotating switching activity interchange format (SAIF) files which are generated from gate-level timing simulations with 100000 random inputs generated from a Gaussian distribution. In comparison with the DLMS architecture, implementing the RFF module contributes to additional area and power consumption. Even though there is an increase in area and power overhead for DRFF-KLMS implementations over the DLMS, the improvement in steady-state MSE and convergence rate achieved by DRFF-KLMS over DLMS (which did not converge in the task of nonlinear system identification as shown in Fig. 1) calls for DRFF-KLMS VLSI implementation.

The synthesis results of DRFF-KLMS architectures with different FFs for various filter orders are also given in Table IV. From Table IV, we see that compared to DLMS synthesis results the DRFF-KLMS using cosine feature function has an increase of 31.44%, 31.33%, 51.79% in area and 31.92%, 33.93%, 57.82% in dynamic power for 16, 32, and 64 tap sizes, respectively. Whereas in the case of DRFF-KLMS using the Gaussian feature function, there is an increase in area by 20.35%, 11.56% and 14.05% and increases in dynamic power by 13.59%, 14.26% and 12.21% for 16, 32, and 64 tap sizes, respectively when we compare with DLMS. In the case of exponential feature function-based DRFF-KLMS, there is an increase in area by 5.93%, 3.03%, 19.16% and an increase of 17.18%, 16.01%, 35.74% for 16, 32, and 64 tap sizes, respectively.

Proposed DRFF-KLMS using cosine feature function has 27.84%, 29.96%, and 51.05% area increase, and 25.92%, 30.40%, and 55.86% dynamic power increase compared to DLMLS for 16, 34, and 64 taps, respectively. Whereas in the case of DRFF-KLMS using exponential feature function has 3.02%, 1.95%, and 18.58% area increase, and 11.85%, 12.95%, and 34.06% dynamic power increase compared to DLMLS for 16, 34, and 64 taps, respectively. In the case of DRFF-KLMS using Gaussian feature function has 17.05%, 10.39%, and 13.49% area increase, and 8.42%, 11.25%, and 10.82% dynamic power increase compared to DLMLS for 16, 34, and 64 taps, respectively.

TABLE IV
SYNTHESIS RESULTS OF DRFF-KLMS VARIANTS, DLMS AND DLMLS WITH DIFFERENT FILTER ORDER USING CADENCE 45-NM CMOS LIBRARY FOR CLOCK FREQUENCY 1.052GHZ

Algorithm	**RFF-Space Dimension** (P)	**MSE** (dB)	**Cell Area** $(k\text{GE})^a$	**Total Area** (μm^2)	**Leakage Power** (μW)	**Dynamic Power** $(m\text{W})$
DRFF-KLMS (Cosine)	16	-14.8	39.29	176397	22.57	458.18
	32	-18.5	78.86	353508	45.63	917.76
	64	-19.3	174.12	772667	102.68	2042.74
DRFF-KLMS (Exponential)	16	-12.8	32.65	142160	19.11	406.99
	32	-13.3	63.58	277324	37.04	794.98
	64	-14.1	139.89	606557	82.25	1756.98
DRFF-KLMS (Gaussian)	16	-12.4	36.18	161516	21.01	394.50
	32	-15.7	67.90	300290	39.70	782.96
	64	-15.9	178.62	580530	76.02	1452.46
DLMS [37]	16	0.55	30.48	134202	18.44	347.30
	32	0.5	61.34	269174	36.58	685.24
	64	0.38	116.26	509016	68.21	1294.34
DLMLS [30]	16	0.51	31.27	137981	18.88	363.87
	32	0.38	62.21	272004	36.87	703.78
	64	0.28	117.10	511513	69.87	1310.56

[a] One Gate Equivalent (GE) corresponds to the size of a two input NAND gate of size 3.25 μm^2

As discussed earlier, the additional logic used to project the input into RFF-space contribute to the area and power overhead over the DLMS architecture. When compared to the various feature functions-based DRFF-KLMS designs, cosine feature function-based implementation consumes more area and power. Gaussian feature function-based implementation requires slightly reduced area and power consumption than the cosine feature function-based implementation. Exponential feature function-based implementation only has a marginal increase in area and dynamic power over DLMS and DLMLS implementation. However, DRFF-KLMS using the cosine feature function converges to a better steady-state MSE value than their counterparts. Among Gaussian and exponential FFs, the Gaussian feature function converges to a better steady-state value over the exponential feature function (ref. Fig. 2). Hence, the FFs for VLSI implementation can be selected based on the requirements of the application. The DRFF-KLMS with cosine feature function is the best option if the application demands stringent steady-state MSE values. On the other hand, if area and power are critical for the application, it is best to select DRFF-KLMS with exponential feature function. The DRFF-KLMS with Gaussian feature function has intermediate area and power requirements.

VI. CONCLUSION

Kernel adaptive filters exhibit superior performance when identifying nonlinear systems compared to conventional adaptive filtering algorithms that assume a linear relationship between input and output signals. However, kernel adaptive filters are computationally expensive, and their native form is unsuitable for VLSI implementations. In this paper, we proposed a VLSI architecture that implements one of the more well-known kernel adaptive filters, namely the DRFF-KLMS algorithm. This can be accomplished with high-speed and area-efficient methods of implementing inner products and cosine functions. The simulation results demonstrated that the proposed DRFF-KLMS algorithm performed well with various FFs. Using synthesis results, we compared DRFF-KLMS to delayed LMS architecture in terms of hardware complexity. The proposed work offers a promising avenue for designing efficient VLSI architectures for nonlinear adaptive filtering in complex systems.

REFERENCES

[1] A. H. Sayed, *Adaptive filters*, 2011, John Wiley & Sons.

[2] C. A. R. Fernandes, "Nonlinear MIMO communication systems: channel estimation and information recovery using Volterra models," *Univ. of Nice Sophia Antipolis*, France, 2009.

[3] J. -H. Kim, F. Bießmann and S. -W. Lee, "Reconstruction of hand movements from EEG signals based on non-linear regression," *Int. Winter Workshop on Brain-Computer Interface*, 2014, pp. 1-3.

[4] S. Wei and H. Qun, "Research on network data forecast system based on non-linear time series," *Int. Conf. on Test and Meas.*, 2009, pp. 251-254.

[5] T. Ogunfunmi and T. Drullinger, "Equalization of non-linear channels using a Volterra-based non-linear adaptive filter," *IEEE 54th Int. Midwest Symp. on Circuits and Syst. (MWSCAS)*, 2011, pp. 1-4.

[6] O. Tokunbo, *Adaptive nonlinear system identification*, The Volterra and Wiener model approaches, 2007.

[7] M. Scarpiniti, D. Comminiello, R. Parisi and A. Uncini, "Nonlinear spline adaptive filtering," *Signal Process.*, Vol. 93, No. 4, p. 772-783, 2013.

[8] W. Liu, J. C. Principe, and S. Haykin, *Kernel adaptive filtering: a comprehensive introduction*, vol. 57, 2011, John Wiley & Sons.

[9] J. -W. Xu, A. R. C. Paiva, I. Park and J. C. Principe, "A reproducing kernel Hilbert space framework for information-theoretic learning," *IEEE Trans. Signal Process.*, vol. 56, no. 12, pp. 5891-5902, Dec. 2008.

[10] N. Kwak , "Nonlinear projection trick in kernel methods: an alternative to the kernel trick," *IEEE Trans. Neural Netw. Learn. Syst.*, Dec. 2013, 24(12):2113-9.

[11] W. Liu, P. P. Pokharel and J. C. Principe, "The kernel least-mean-square algorithm," *IEEE Trans. Signal Process.*, vol. 56, no. 2, pp. 543-554, Feb. 2008.

[12] F. Albu, D. Coltuc, M. Rotaru and K. Nishikawa, "An efficient implementation of the kernel affine projection algorithm," *8th Int. Symp. Image and Signal Process. and Anal. (ISPA)*, 2013, pp. 349-353.

[13] Y. Engel, S. Mannor and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2275-2285, Aug. 2004.

[14] A. Singh, N. Ahuja and P. Moulin, "Online learning with kernels: overcoming the growing sum problem," *IEEE Int. Workshop on Machine Learn. for Signal Process.*, 2012, pp. 1-6.

[15] J. Zhao, H. Zhang, G. Wang and J. A. Zhang, "Projected kernel least mean p -power algorithm: convergence analyses and modifications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 10, pp. 3498-3511, Oct. 2020.

[16] B. Chen, S. Zhao, P. Zhu and J. C. Principe, "Quantized kernel least mean square algorithm," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 1, pp. 22-32, Jan. 2012.

[17] S. Nan, L. Sun, B. Chen, Z. Lin and K. Toh, "Density-dependent quantized least squares support vector machine for large data sets," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 1, pp. 94-106, Jan. 2017.

[18] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *Proc. 20th Int. Conf. Neural Inf. Process. Syst. (NIPS'07)*, Curran Associates Inc., Red Hook, NY, USA, 1177–1184.

[19] J. Dong, Y. Zheng and B. Chen, "A unified framework of random feature KLMS algorithms and convergence analysis," *Int. Joint Conf. Neural Netw.*, 2018, pp. 1-8.

[20] P. Bouboulis, S. Chouvardas and S. Theodoridis, "Online distributed learning over networks in RKH spaces using random Fourier features," *IEEE Trans. Signal Process.*, vol. 66, no. 7, pp. 1920-1932, 1 Apr., 2018.

[21] S. Wang, L. Dang, B. Chen, S. Duan, L. Wang and C. K. Tse, "Random Fourier filters under maximum correntropy criterion," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 10, pp. 3390-3403, Oct. 2018.

[22] V. C. Gogineni, V. R. M. Elias, W. A. Martins and S.Werner, "Graph diffusion kernel LMS using random Fourier features," in *Proc. Asilomar Conf. Signals, Syst., and Comput.*, Nov. 2020, pp. 1528-1532.

[23] V. C. Gogineni, V. Naumova, S.Werner and Y-F. Huang, "Graph kernel recursive least-squares algorithms," in *Proc. IEEE Int. Conf. Asia Pacific Signal and Info. Process. Assoc.*, Tokyo, 2021, pp. 2072-2076.

[24] T. Deb, D. Ray and N. V. George, "A reduced complexity random Fourier filter based nonlinear multichannel narrowband active noise control system," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 1, pp. 516-520, Jan. 2021.

[25] Y. Zhu, H. Zhao, X. He, Z. Shu and B. Chen, "Cascaded random Fourier filter for robust nonlinear active noise control," *IEEE/ACM Trans. Audio, Speech, and Lang. Process.*, vol. 30, pp. 2188-2200, 2022.

[26] M. T. Khan and R. A. Shaik, "High-throughput and improved-convergent design of pipelined adaptive DFE for 5G communication," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 2, pp. 652-656, Feb. 2021.

[27] M. N. Sasikanth, S. Gambhira and M. Sharad, "Design optimization of DSP for wearable biomedical device," *IEEE Int. Symp. Nanoelectronic and Inf. Syst. (iNIS)*, 2017, pp. 1-5.

[28] S. Ahmadi, *5G NR: architecture, technology, implementation, and operation of 3GPP new radio standards*, 2019, Academic Press.

[29] R. Dilli, "Analysis of 5G wireless systems in FR1 and FR2 frequency bands," *2nd Int. Conf. Innovative Mech. Ind. App. (ICIMIA)*, 2020, pp. 767-772.

[30] 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, Sept. 2017.

[31] L. -D. Van and W. -S. Feng, "An efficient systolic architecture for the DLMS adaptive filter and its applications," *IEEE Trans. Circuits Syst. II, Analog and Digit. Signal Process.*, vol. 48, no. 4, pp. 359-366, Apr. 2001.

[32] X. Ren, P. Ren, B. Chen, T. Min and N. Zheng, "Hardware implementation of KLMS algorithm using FPGA," *Int. Joint Conf. Neural Netw. (IJCNN)*, 2014, pp. 2276-2281.

[33] N. J. Fraser, J. Lee, D. J. M. Moss, J. Faraone, S. Tridgell, C. T. Jin and P. H. W. Leong, "FPGA implementations of kernel normalised least mean squares processors," *ACM Trans. Reconfigurable Technol. Syst.*, no. 26, Dec. 2017.

[34] Y. Pang, S. Wang, Y. Peng, N. J. Fraser and P. H. W. Leong, "A low latency kernel recursive least squares processor using FPGA technology," *Int. Conf. Field-Programmable Tech. (FPT)*, 2013, pp. 144-151.

[35] L. Wang, A. Gehre, M. M. Bronstein and J. Solomon, "Kernel functional maps," *Comput. Graph. Forum*, 2018, 37: 27-36.

[36] W. Gao, J. Chen, C. Richard and J. Huang, "Online dictionary learning for kernel LMS," *IEEE Trans. Signal Process.*, vol. 62, no. 11, pp. 2765-2777, Jun., 2014.

[37] 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.

[38] T. C. Denk and K. K. Parhi, "Exhaustive scheduling and retiming of digital signal processing systems," *IEEE Trans. Circuits Syst. II, Analog and Digit. Signal Process.*, vol. 45, no. 7, pp. 821-838, Jul. 1998.

[39] K. K. Parhi, *VLSI digital signal processing systems: design and implementation*, John Wiley & Sons, 2007.

[40] D. Alex, V. C. Gogineni, S. Mula and S. Werner, "Novel VLSI architecture for fractional-order correntropy adaptive filtering algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 7, pp. 893-904, Jul. 2022.

[41] S. Traferro and A. Uncini, "Power-of-two adaptive filters using tabu search," *IEEE Trans. Circuits Syst. II, Analog and Digit. Signal Process.*, vol. 47, no. 6, pp. 566-569, Jun. 2000.

[42] S. H. Mirfarshbafan, S. Taner and C. Studer, "SMUL-FFT: a streaming multiplierless fast Fourier transform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1715-1719, May 2021.

[43] 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.

[44] 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.

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

[46] S. Venkatachalam and S. B. Ko, "Approximate sum-of-products designs based on distributed arithmetic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 8, pp. 1604-1608, Aug. 2018.

[47] P. K. Meher, "New approach to look-up-table design and memory-based realization of FIR digital filter," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 3, pp. 592-603, Mar. 2010.

[48] D. Ray, N. V. George and P. K. Meher, "An analytical framework and approximation strategy for efficient implementation of distributed arithmetic-based inner-product architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 1, pp. 212-224, Jan. 2020.

[49] S. A. Khan, *Digital Design of Signal Processing Systems*, John Wiley & Sons, 2011.

[50] J. E. Volder, "The CORDIC trigonometric computing technique." *IRE Trans. Electron. Comput.*, 1959, 330-334.

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

[52] G. L. Haviland and A. A. Tuszynski, "A CORDIC arithmetic processor chip," *IEEE J. Solid-State Circuits*, vol. 15, no. 1, pp. 4-15, Feb. 1980.

[53] Y. H. Hu, "CORDIC-based VLSI architectures for digital signal processing," *IEEE Signal Process. Mag.*, vol. 9, no. 3, pp. 16-35, Jul. 1992.

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 VIT University, India, in 2008, and the Ph.D. degree in electronics and electrical communication engineering from Indian Institute of Technology Kharagpur, India in 2019. From 2008 to 2011, he was with a couple of MNCs in India. Currently, he is working as a postdoctoral research fellow at the department of electronic systems, NTNU-Norway. His research interests include statistical signal processing, distributed and federated learning, and geometric deep learning. He was a recipient of the ERCIM Alain Bensoussan Fellowship in 2019 and the Best Paper Award at APSIPA ASC-2021, Tokyo, Japan.

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 is currently a Professor at the Department of Electronic Systems, Norwegian University of Science and Technology (NTNU), Director of IoT@NTNU, and Adjunct Professor with Aalto University in Finland. He was a visiting Melchor Professor with the University of Notre Dame during the summer of 2019 and an Adjunct Senior Research Fellow with the Institute for Telecommunications Research, University of South Australia, from 2014 to 2020. He held an Academy Research Fellowship, funded by the Academy of Finland, from 2009 to 2014. His research interests include adaptive and statistical signal processing, wireless communications, and security and privacy in cyber-physical systems. He is a member of the editorial boards for the EURASIP Journal of Signal Processing and the IEEE Transactions on Signal and Information Processing over Networks.