A High-Performance and Energy-Efficient FIR Adaptive Filter using Approximate Distributed Arithmetic Circuits

Honglan Jiang, Leibo Liu, Pieter Jonker, Duncan Elliott, Fabrizio Lombardi, *Fellow, IEEE* and Jie Han, *Senior Member, IEEE,*

***Abstract*—In this paper, a fixed-point finite impulse response (FIR) adaptive filter is proposed using approximate distributed arithmetic (DA) circuits. In this design, the radix-8 Booth algorithm is used to reduce the number of partial products in the DA architecture, although no multiplication is explicitly performed. Additionally, the partial products are approximately generated by truncating the input data with an error compensation. To further reduce hardware, an approximate Wallace tree is considered for the accumulation of partial products. As a result, the delay, area and power consumption of the proposed design are significantly reduced. The application of system identification of a 48-tap bandpass filter and a 103-tap high-pass filter shows that the approximate design achieves a similar accuracy as its accurate counterpart. Compared with a state-of-the-art adaptive filter using bit-level pruning in the adder tree (referred to as the delayed least mean square (DLMS) design), it has a lower steady-state mean squared error (MSE) and a smaller normalized misalignment. Synthesis results show that the proposed design attains on average a 55% reduction in energy per operation (EPO) and a $3.2\times$ throughput per area compared with the accurate design. Moreover, the proposed design achieves 45%-64% lower EPO compared with the DLMS design. A saccadic system using the proposed approximate adaptive filter based cerebellar model achieves a similar retinal slip as using an accurate filter. These results are promising for large-scale integration of approximate circuits into high-performance and energy-efficient systems for error-resilient applications.**

***Index Terms*—adaptive filter, approximate arithmetic, distributed arithmetic, radix-8 Booth algorithm, truncation, Wallace tree.**

I. Introduction

The human beings' superior ability to accurately control complex movements, due to the cerebellum, has engaged considerable attention. Many computational models have been proposed to explain and to mimic the cerebellar function for signal processing and motor control applications, including the perceptron-based model [1], [2], the continuous spatio-temporal model [3], the higher-order lead-lag compensator model [4] and the adaptive filter-based model [5]. Among them, the most widely used cerebellar model is based on the adaptive filter [5] due to its low complexity and high structural resemblance to the cerebellum. However, little has been done on implementing the cerebellar model in hardware due to its high complexity.

Adaptive filters are widely used in applications such as image processing, signal prediction/identification and echo suppression [6]. The finite impulse response (FIR) adaptive filter is one of the most pervasively employed adaptive filters; it is composed of an FIR filter with variable coefficients (or weights) and a weight update module. The coefficients are adjusted by an adaptive algorithm. Due to the closed-loop adaptive process and related algorithm, the hardware implementation of a direct form FIR adaptive filter is very complex. Moreover, the high power consumption, large area and long critical path of the weighted sum operation in the linear filter significantly limit the throughput of such a digital signal processing (DSP) system.

In this paper, distributed arithmetic (DA) is combined with the radix-8 Booth algorithm and approximate computing for a high-performance and energy-efficient FIR adaptive filter design. To the best knowledge of the authors, this is the first integrated FIR adaptive filter design using the radix-8 Booth algorithm in a DA architecture. In this design, the computation of weighted sums using multipliers and adders is transformed to a DA architecture with no lookup table (LUT). Thus, no multiplier is used; the partial product generation and accumulation circuits are still required. By using the radix-8 Booth algorithm, the number of partial products is reduced by $2/3$ compared to a conventional DA architecture. Therefore, a significant reduction is achieved in the accumulation circuits. Moreover, an input truncation scheme is proposed to approximately generate the partial products and an approximate recoding adder is used to reduce the critical path, area and power consumption. To further reduce the latency, approximate Wallace trees are used for the accumulation of partial products.

The applications in the system identification and saccadic system show that the proposed approximate FIR adaptive filters incur a very small loss in accuracy compared with the accurate implementation. Synthesis results indicate that the proposed design achieves nearly 55% reduction in energy per operation (EPO) and a $3.2\times$ throughput per area (TPA). Compared with the delayed least mean square (DLMS)-based design of [7], the proposed design requires up to 60% lower EPO with a higher

H. Jiang, D. Elliott and J. Han are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada T6G 1H9. (e-mail: {honglan, elliott, jhan8}@ualberta.ca)

L. Liu is with the Institute of Microelectronics, Tsinghua University, China, Beijing. (e-mail: liulb@tsinghua.edu.cn)

P. Jonker is with the Delft University of Technology, The Netherlands. (e-mail: jonker@tudelft.nl)

F. Lombardi is with the Department of Electrical and Computer Engineering, Northeastern University, Boston, USA. (e-mail: lombardi@ece.neu.edu)

accuracy (i.e., lower mean squared error and misalignment).

This paper is organized as follows. Section II introduces the cerebellar model, the basic principles of DA, the review of FIR adaptive filters, and FIR adaptive filtering. Section III proposes the architecture of the FIR adaptive filter using approximate DA, including error computation and weight update modules. The proposed truncated multiplication and approximate Wallace tree are presented in Section IV. Section V shows the simulation and synthesis results of the adaptive filter design. Additionally, the proposed design is compared with the most efficient existing design in terms of accuracy and hardware overhead. Section VI evaluates the accuracy of the adaptive filter designs in a saccadic system. Section VII concludes the paper.

II. Background

A. Cerebellar Model

Fig. 1 shows a connection network of cerebellar cells [8], where the Purkinje cell (PC), granule cell (GC), Golgi cell (Go), mossy fibre (MF) and climbing fibre (CF) are key elements for the cerebellum. In the adaptive filter based cerebellar model, the GC and Go are combined and simplified to a tap-delay line [9]. The output of the PC is given by

$$z(t) = \sum_{i=0}^{M-1} w_i(t) \cdot x_i(t), \tag{1}$$

where $w_i(t)$ is the synaptic weight between the i^{th} parallel fibre (PF) and the PC, $x_i(t) = u(t - Ti)$ is the delayed input of $u(t)$, T is the constant delay of the Go-GC system, and M is the number of synapses. The synaptic weights are updated by the error signal carried on the CF according to the least mean square (LMS) algorithm. The LMS algorithm is formulated as

$$w_i(t+T) = w_i(t) + \mu \cdot e(t) \cdot x_i(t), i = 0, 1, \cdots, M-1, \tag{2}$$

where μ is the step size, and $e(t) = d(t) - z(t)$ is the error between the desired signal $d(t)$ and the PC output.

B. FIR Adaptive Filter Architecture

Fig. 2 shows the basic structure of an FIR adaptive filter. It consists of an FIR filter with variable weights and a weight update module. The weights of the FIR filter are adjusted by

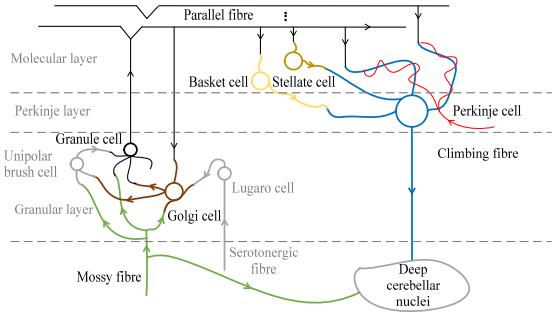


Fig. 1. A connection network of cerebellar cells.

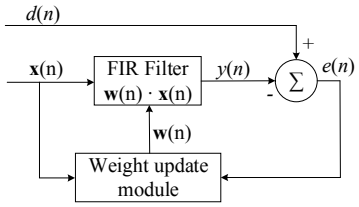


Fig. 2. An FIR adaptive filter [13]. n is the iteration number, $\mathbf{x}(n)$ is the input vector, $y(n)$ is the output signal, $d(n)$ is the interfered desired signal with the undesired noise, $e(n)$ is the error output, and $\mathbf{w}(n)$ is the weight vector.

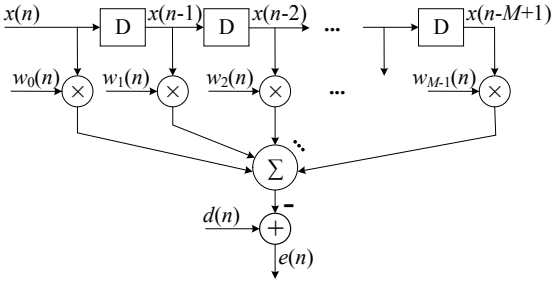


Fig. 3. Error computation module.

the adaptive algorithm through a negative feedback loop. An M-tap FIR filter is implemented by

$$y(n) = \mathbf{w}(n) \cdot \mathbf{x}(n) = \sum_{i=0}^{M-1} w_i(n) \cdot x(n-i), \tag{3}$$

where $\mathbf{w}(n) = [w_0(n), w_1(n), \cdots, w_{M-1}(n)]$ is the weight vector, $\mathbf{x}(n) = [x(n), x(n-1), \cdots, x(n-M+1)]^T$ is the input vector at the n^{th} iteration, and M is the length of $\mathbf{w}(n)$ and the tap of the FIR filter. The weights of the FIR filter are variables with the iteration number n as determined by the adaptive algorithm. They are updated until a set of optimized values are obtained. There are many adaptive algorithms, e.g. the LMS, the normalized LMS, the recursive LMS algorithms [10] and the affine projection algorithm [11]. The selection of an adaptive algorithm is based on a tradeoff between computational complexity and convergence speed. As the LMS algorithm is very simple with a satisfactory convergence [12], it is widely used for hardware implementation and thus it is considered in this paper. The LMS algorithm is formulated as

$$w_i(n+1) = w_i(n) + \mu \cdot e(n) \cdot x(n-i), i = 0, 1, \cdots, M-1, \tag{4}$$

where μ is the step size, and $e(n) = d(n) - y(n)$ is the error signal between the desired signal $d(n)$ (interfered by an undesired noise) and the filter output $y(n)$.

As per Fig. 2, the implementation of an FIR adaptive filter can be divided into the error computation and the weight update modules; they are implemented by delay registers, multipliers and adders (shown in Figs. 3 and 4, respectively). In Fig. 4, the step size μ is set to 2^{-q} (where q is a positive integer); thus the multiplication by μ is realized by a right shift operation.

Still, $2M$ multipliers (with M multipliers for the error computation and M multipliers for the weight update) are required

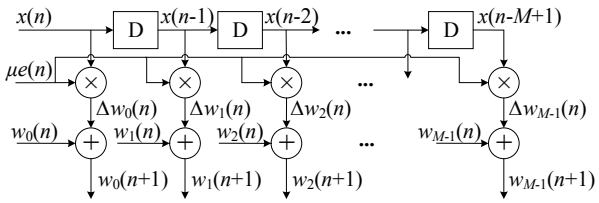


Fig. 4. Weight update module.

for an M-tap FIR adaptive filter. This process consumes a significant amount of power and it also incurs a large area for the required hardware implementation.

C. Distributed Arithmetic

Distributed arithmetic presents an efficient computation structure for DSP. It is widely used in the computation of sum of products or inner products [14]. For example, consider computing the inner product of a M-dimensional vector pair $x = [x_0, x_1, \cdots, x_{M-1}]$ and $y = [y_0, y_1, \cdots, y_{M-1}]$, where M is the number of numbers in each vector pair

$$z = \sum_{i=0}^{M-1} x_i y_i. \tag{5}$$

Assume that $y_i = -y_{i,m-1}2^{m-1} + \sum_{j=0}^{m-2} y_{i,j}2^j$ is a binary number in 2's complement, where m is the bit width of y_i. Then, (5) becomes

$$\begin{aligned} z &= \sum_{i=0}^{M-1} x_i(-y_{i,m-1}2^{m-1} + \sum_{j=0}^{m-2} y_{i,j}2^j) \\ &= -2^{m-1}\sum_{i=0}^{M-1} x_i y_{i,m-1} + \sum_{j=0}^{m-2}(\sum_{i=0}^{M-1} x_i y_{i,j})2^j \end{aligned} \tag{6}$$

As $y_{i,j}$ is either '0' or '1', $\sum_{i=0}^{M-1} x_i y_{i,j}$ has 2^M possible values. Take $M = 3$ as an example, $\sum_{i=0}^{2} x_i y_{i,j}$ can be 0, x_0, x_1, $x_1 + x_0$, x_2, $x_2 + x_0$, $x_2 + x_1$ or $x_2 + x_1 + x_0$. These 2^3 values can be precomputed and stored in an 8-word LUT, and $[y_{0,i}, y_{1,i}, y_{2,i}]$ is used to address the LUT. Finally, a shifted accumulator is required to obtain the final result z for the inner product.

As the length of the vector pair increases, the size of the required LUT grows exponentially if a full LUT based DA is used to compute the inner product, i.e., 2^M-word. Thus, directly using full LUT based DA to compute the inner product is not efficient when M is large. Usually, decomposition techniques are used to decompose the M-dimensional vector pair into K-dimensional vector pairs ($K < M$) [15]. The inner product of a K-dimensional vector pair is implemented based on a full LUT (2^K-word) based DA. Then, the inner product of the M-dimensional vector pair is obtained by accumulating the inner products of the K-dimensional vector pairs. Another way to solve this problem is to compute $\sum_{i=0}^{M-1} x_i y_{i,j}$ on line by accumulating the partial products $x_i y_{i,j}$ for a large M [16]. The partial products can be accumulated in a bit-serial or bit-parallel mode [17]. An adder tree and a scaling accumulator are sufficient for a bit-serial DA. For an m-bit input, however, m processing cycles are required. A parallel DA is significantly faster, but it requires m adder trees and a shifted adder tree to accumulate the partial products, incurring a larger area and higher power dissipation.

D. Review of FIR Adaptive Filter Designs

Several FIR adaptive filter designs based on DA have been proposed to reduce the critical path for high throughput. In the two DA-based FIR adaptive filters presented in [18], weights are used as addresses to access the LUTs storing the sums of the weighted delayed inputs. Two schemes have been proposed for updating the LUTs. Although the memory requirement is reduced by half compared with previous schemes, the size of the LUT increases exponentially with the order of the adaptive filter. Therefore, these designs are not suitable for adaptive filters with high orders. An efficient DA formulation has been presented for the block least mean square (BLMS) algorithm in an FIR adaptive filter [19]. In this design, the LUT is shared between the computations of the filter output and the weight increment; only one column of LUTs is updated in each iteration by shifting the weight-vectors. Thus, figures of merits such as circuit area, power and timing are improved for the LUT updating process. However, the size of the LUT is still L times (where L is the block size of the BLMS algorithm) the size of the LUT in [18] and hence, the area and power dissipations of this design are rather large. Therefore, DA-based FIR adaptive filter designs using LUTs perform well for a low order; however, they are not efficient for adaptive filters of a high order due to the overheads for updating and accessing the LUTs. For high-order designs, DA architecture using decomposition techniques or without using LUTs is more efficient [16].

A novel shared-LUT design has been proposed to implement DA for a reconfigurable FIR filter [20]. In this design, an M-dimensional vector pair is decomposed into L P-dimensional small vector pairs (i.e., $M = LP$). A 2^P-word LUT is shared by the bit slices (consisting of P bits) of different weightage. Totally, L partial product generators, L 2^P-word LUTs, m (as the bit width of inputs) adder trees and a shift-add tree are required to compute the inner product. The contents in the LUTs are updated in parallel. This FIR filter achieves a significant reduction in energy compared with the systolic decomposition of a DA-based design.

A different methodology to improve the throughput of an adaptive filter is to use a pipelined structure. However, the least mean square (LMS) algorithm does not directly support pipelining due to its recursive operation. Therefore, the LMS algorithm is modified into the so-called DLMS [21]. DLMS significantly reduces the critical path delay of an adaptive filter by pipelining, whereas the performance of convergence is degraded significantly due to the adaptation delay [22]. A DLMS FIR adaptive filter with a low adaptation delay has been proposed in [7] by using a novel partial product generator and an optimized balanced pipeline; a bit-level pruning of the adder tree is further employed to reduce the area and power consumption of the implementation. Synthesis and simulation have shown that this scheme consumes less power and requires less area than other DLMS adaptive filter designs. However, a large number of additional latches are used for the

pipelined implementation of a DLMS adaptive filter and hence, overheads in area and power dissipation are incurred compared to an adaptive filter using the LMS algorithm.

Many other techniques have been combined with DA to increase its efficiency. Factor sharing has been employed in a DA architecture to reduce the number of adders [23]. It reduces 44.5% of the adders in a multistandard transform core design. A result-biased circuit for DA has been used in the filter architectures for computing the discrete wavelet transform; it leads to a 20% to 25% reduction in hardware [24].

III. Proposed Adaptive Filter Architecture

For an M-tap direct-form FIR adaptive filter (i.e., an m-bit fixed-point implementation), the critical path delay is the sum of delays in the error computation ($t_M + \lceil log_2(M+1) \rceil \times t_A$) and weight update processes ($t_M + t_A$), where t_M and t_A are the critical path delays of an $m \times m$ multiplier and an m-bit adder, respectively. Therefore, the sample rate of the input signal is limited due to this long latency. An important feature of the proposed adaptive filter using DA is the reduction of the latency to achieve a high throughput with significantly low area and power consumption.

In the adaptive learning process for the weight update, errors in the adaptive filter circuit can be inherently compensated or corrected. Therefore, power and area efficient approximate arithmetic circuits are considered for a fixed-point implementation. Truncation is an efficient method to save power and area for approximate arithmetic circuits at a limited loss of accuracy [25], so it has been extensively used in the design of fixed-width multipliers [26]. Most existing designs are based on the truncation of the partial products to save circuitry for partial product accumulation [27]. All bits of the input operands are required for these multipliers and therefore, memory is not reduced for storage requirements. However, memory consumes a significant amount of power and accounts for a large area in an application involving a large data set. Moreover, efficient data transfers are very important for achieving a high throughput [28].

As per the results in [25], compared to the partial product truncation, truncating the input operands achieves more significant reduction in hardware overhead for adder and multiplier designs. Thus, truncation on the input operands is applied to achieve savings in the partial product generation.

A. Error Computation Module

A weight $w_i(n)$ can be represented in 2's complement as $w_i(n) = -w_i^{m-1}(n)2^{m-1} + \sum_{j=0}^{m-2} w_i^j(n)2^j$, where $w_i^j(n)$ is the j^{th} least significant bit (LSB) of $w_i(n)$ and m is the width of the binary representation. For the ease of analysis, $w_i(n)$ is represented as an integer; it can be easily transformed to a fixed-point format by shifting. By using the radix-8 Booth encoding, as shown in Table I, four bits of $w_i(n)$ are grouped with one overlapping bit. Then, $w_i(n)$ is given by

TABLE I. The radix-8 Booth encoding algorithm

$w_i^{3j+2}(n)$	$w_i^{3j+1}(n)$	$w_i^{3j}(n)$	$w_i^{3j-1}(n)$	$\overline{w}_i^j(n)$
0	0	0	0	0
0	0	0	1	+1
0	0	1	0	+1
0	0	1	1	+2
0	1	0	0	+2
0	1	0	1	+3
0	1	1	0	+3
0	1	1	1	+4
1	0	0	0	-4
1	0	0	1	-3
1	0	1	0	-3
1	0	1	1	-2
1	1	0	0	-2
1	1	0	1	-1
1	1	1	0	-1
1	1	1	1	0

$$w_i(n) = \sum_{j=0}^{\lceil m/3 \rceil - 1} (-2^2 w_i^{3j+2}(n) + 2w_i^{3j+1}(n) + w_i^{3j}(n) + w_i^{3j-1}(n))2^{3j} = \sum_{j=0}^{\lceil m/3 \rceil - 1} \overline{w}_i^j(n)2^{3j}, \tag{7}$$

where $w_i^{-1} = 0$, $\overline{w}_i^j(n) = -2^2 w_i^{3j+2}(n) + 2w_i^{3j+1}(n) + w_i^{3j}(n) + w_i^{3j-1}(n)$, and $\overline{w}_i^j(n) \in \{-4,-3,-2,-1,0,1,2,3,4\}$. Sign extension is used when the width of the encoded input is shorter than $3 \times \lceil m/3 \rceil$.

The filter output $y(n)$ in (3) is then obtained as

$$y(n) = \mathbf{w}(n) \cdot \mathbf{x}(n) = \delta \cdot \overline{\mathbf{w}}(\mathbf{n}) \cdot \mathbf{x}(n), \tag{8}$$

where

$$\overline{\mathbf{w}}(\mathbf{n}) = \begin{bmatrix} \overline{w}_0^0(n) & \overline{w}_1^0(n) & \cdots & \overline{w}_{M-1}^0(n) \\ \overline{w}_0^1(n) & \overline{w}_1^1(n) & \cdots & \overline{w}_{M-1}^1(n) \\ \vdots & \vdots & \cdots & \vdots \\ \overline{w}_0^{\lceil \frac{m}{3} \rceil - 1}(n) & \overline{w}_1^{\lceil \frac{m}{3} \rceil - 1}(n) & \cdots & \overline{w}_{M-1}^{\lceil \frac{m}{3} \rceil - 1}(n) \end{bmatrix}, \tag{9}$$

$\delta = [2^0, 2^3, \cdots, 2^{3\lceil m/3 \rceil - 3}]$, and $\mathbf{x}(n) = [x(n), x(n-1), \cdots, x(n-M+1)]^T$. By computing $\mathbf{pp}(n) = \overline{\mathbf{w}}(\mathbf{n}) \cdot \mathbf{x}(n)$ first through the accumulation of partial products and then $y(n) = \delta \cdot \mathbf{pp}(n)$ by a shift accumulation, a DA architecture is obtained. Let $\mathbf{pp}(n)$ be $[pp_0(n), pp_1(n), \cdots, pp_{\lceil m/3 \rceil - 1}(n)]^T$, then $pp_j(n)$ is given by

$$pp_j(n) = \sum_{i=0}^{M-1} \overline{w}_i^j(n)x(n-i) = \sum_{i=0}^{M-1} PP_{ij}, \tag{10}$$

where $PP_{ij} = \overline{w}_i^j(n)x(n-i)$ is the j^{th} row in the partial product array of $w_i(n)x(n-i)$ using the radix-8 Booth algorithm.

Compared with a conventional DA architecture, the number of partial products in $\mathbf{pp}(n)$ is reduced by roughly $m - \lceil m/3 \rceil \approx \frac{2m}{3}$ due to the use of the radix-8 Booth algorithm. Thus, the required number of accumulations to obtain $y(n)$ is reduced by about $2/3$.

Fig. 5 shows the proposed error computation module using DA. In this design, no LUT is used due to the large size incurred in a high-order filter. Thus, the partial product

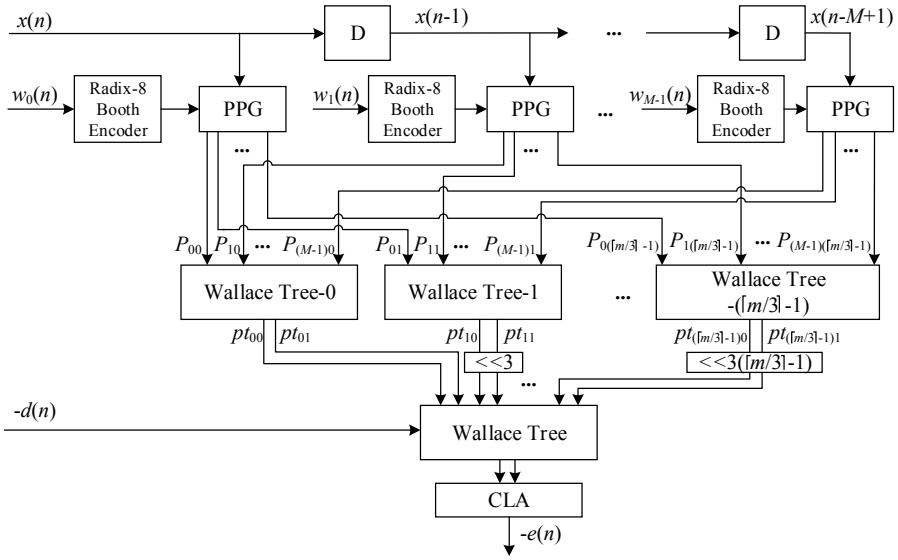


Fig. 5. Proposed error computation scheme using distributed arithmetic. PPG: the partial product generator; CLA: the m-bit carry lookahead adder.

vectors $PP_{i,j}$ are generated online and accumulated. Initially, the inputs $\mathbf{w}(n)$ and $\mathbf{x}(\mathbf{n})$ are truncated and compensated (will be discussed in Section IV-A). Then, the partial product vectors PP_{ij} ($i=0,1,\cdots,M-1$ and $j=0,1,\cdots,\lceil m/3\rceil-1$) in the weighted sum operation for $y(n)$ are generated using the radix-8 Booth encoder, the partial product generator (PPG) and the approximate recoding adder from the approximate radix-8 Booth multiplier (ABM2-R15) in [29]. The Radix-8 Booth encoder is used to encode every 4 bits in the weight $w_i(n)$ (with an overlap of one bit) into one number $\overline{w}_i^j(n)$ (i.e., $0,\pm1,\pm2,\pm3$ and ±4), as per Table I and (7). The partial product generator (PPG) and the approximate recoding adder (to generate $3\times x(n-i)$) are used to produce partial products PP_{ij} as per (10). The partial product vectors are then accumulated by the Wallace trees.

An M-input Wallace tree is used to compute (10) and hence, $\lceil m/3\rceil$ such Wallace trees are required to obtain $\mathbf{pp}(n)$. Let the two intermediate results generated by the j^{th} Wallace tree be pt_{j0} and pt_{j1}, then $pp_j(n)=pt_{j0}+pt_{j1}$. To implement it, a multi-bit carry-propagation adder is needed, which causes a long latency. Thus, the intermediate results pt_{j0} and pt_{j1} are kept for the next stage to eliminate the long latency. In this case, $y(n)=\delta\cdot\mathbf{pp}(n)=[2^0,2^3,\cdots,2^{3\lceil m/3\rceil-3}]\cdot[pt_{00}+pt_{01},pt_{10}+pt_{11},\cdots,pt_{(\lceil m/3\rceil-1)0}+pt_{(\lceil m/3\rceil-1)1}]^T$. Let $\overline{\delta}=[2^0,2^0,2^3,2^3,\cdots,2^{3\lceil m/3\rceil-3},2^{3\lceil m/3\rceil-3}]$ and $\overline{\mathbf{pp}}(n)=[pt_{00},pt_{01},pt_{10},pt_{11},\cdots,pt_{(\lceil m/3\rceil-1)0},pt_{(\lceil m/3\rceil-1)1}]^T$, then $y(n)=\overline{\delta}\cdot\overline{\mathbf{pp}}(n)$. The negative error signal $-e(n)=y(n)-d(n)=[\overline{\delta},1]\cdot\begin{bmatrix}\overline{\mathbf{pp}}(n)\\-d(n)\end{bmatrix}$. This step can be implemented by shifting the intermediate results followed by a Wallace tree, as shown in Fig. 5. Also, $-d(n)$ is the input to the Wallace tree to reduce the long latency of a carry-propagation adder for computing $e(n)$. Thus, a $(2\lceil m/3\rceil+1)$-input Wallace tree is used. Finally, the negative error output is obtained by adding the two output vectors of the Wallace tree using an m-bit carry lookahead adder (CLA).

Specifically, several LSBs of the input signals and the weights are initially truncated and compensated. Then, the partial products are generated by the PPGs as in [29]. The partial product vectors PP_{ij} are obtained by left shifting the multiplicand when the recoded digit number $\overline{w}_i^j(n)$ is $+2$ or $+4$. For a $+3$ value of $\overline{w}_i^j(n)$, a recoding adder is required to generate $3x(n-i)$. In this design, the approximate recoding adder in ABM2-R15 is used to reduce the latency (albeit not shown in Fig. 5). When $\overline{w}_i^j(n)$ is negative, the PP_{ij} is approximately computed by inverting all bits of the partial product vector produced by the corresponding positive $\overline{w}_i^j(n)$. As in ABM2-R15, half of the partial products at the LSB positions is truncated for a fixed-width multiplication output, as shown in Fig. 6. The '1' in the last row is the average error compensation due to partial product truncation. Finally, the approximate Wallace trees proposed in Section IV-B and one accurate CLA are used to implement the accumulation operation.

Compared with the conventional error computation circuit in Fig. 3, the proposed design saves the delay of a final adder in the multiplier due to the DA. Moreover, the use of the Wallace trees in the proposed scheme makes it even faster. Finally, the area and power consumption of the design are significantly reduced due to the approximation in the partial product generation and accumulation.

B. Weight Update Module

For the weight update in the FIR adaptive filter, $\mu e(n)$ is first obtained by right shifting with a truncation error compensation. Let the m-bit negative output in 2's complement from the error computation block take the value $-e(n)=$

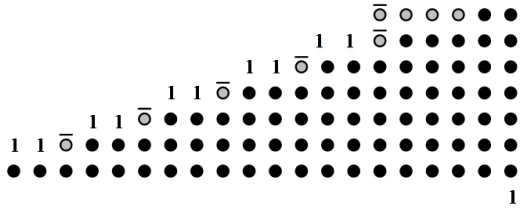


Fig. 6. Partial product tree of an approximate 20×20 radix-8 Booth multiplier with truncation. ●: a partial product; ◍: the sign bit; $\overline{◍}$: the inverted sign bit.

$-e_{m-1}2^{m-1}+\sum_{j=0}^{m-2}e_j2^j$, where e_j is the j^{th} LSB in the output. In this case $e(n)$ is represented as an integer for easier analysis; it can be easily transformed to a fixed-point format by shifting. If the step size μ for the weight update is 2^{-q} and q is a positive integer, $-\mu e(n)=-e_{m-1}2^{m-q-1}+\sum_{j=0}^{m-2}e_j2^{j-q}$ by right shifting $-e(n)$ by q bits. By truncating the q LSBs in the fractional part, $-\mu e(n)\approx -e_{m-1}2^{m-q-1}+\sum_{j=q+1}^{m-2}e_j2^{j-q}+1=(e_{m-1}\cdots e_{q+2}e_{q+1}1)_2$, where the '1' at the LSB position is the error compensation for truncation. $\mu e(n)$ is then obtained by a 2's complement operation, i.e., $\mu e(n)=(\overline{e}_{m-1}\cdots\overline{e}_{q+2}\overline{e}_{q+1}1)_2$, where $\overline{e}_i$ is the inverted value of e_i, $i=q+1,q+2,\cdots,m-1$. After shifting and the 2's complementing operation, $\mu e(n)$ can be represented by $(m-q)$ bits by keeping one sign bit. Therefore, an $(m-q)\times m$ multiplication is sufficient for computing each weight increment $\mu e(n)x(n-i)$. Fig. 7 shows the partial product tree based on an approximate Booth multiplier (ABM-R15) when m and q are 20 and 8, where the partial products at the 19 LSB positions are truncated.

Let $v(n)=\mu e(n)$, and $v(n)=-v_{m-q-1}(n)2^{m-q-1}+\sum_{j=0}^{m-q-2}v_j(n)$ in 2's complement, where $v_j(n)$ is the j^{th} LSB of $v(n)$. As per the radix-8 Booth algorithm, $v(n)$ can be represented as

$$\begin{aligned} v(n)=&\sum_{j=0}^{\lceil (m-q)/3\rceil -1}(-2^2v_{3j+2}(n)+2v_{3j+1}(n)+v_{3j}(n)\\ &+v_{3j-1}(n))2^{3j}=\sum_{j=0}^{\lceil (m-q)/3\rceil -1}\overline{v}_j(n)2^{3j}, \end{aligned} \tag{11}$$

where $\overline{v}_j(n)=-2^2v_{3j+2}(n)+2v_{3j+1}(n)+v_{3j}(n)+v_{3j-1}(n)$ is the radix-8 recoded number in $\{-4,-3,-2,-1,0,1,2,3,4\}$. According to (4), $w_i(n+1)$ is given by

$$w_i(n+1)=v(n)\cdot x(n-i)+w_i(n)=[\delta_v,1]\cdot\begin{bmatrix}\overline{\mathbf{v}}(n)\cdot x(n-i)\\ w_i(n)\end{bmatrix}, \tag{12}$$

where $\delta_v=[2^0,2^3,\cdots,2^{3\lceil (m-q)/3\rceil -3}]$, and $\overline{\mathbf{v}}(n)=[\overline{v}_0(n),\overline{v}_1(n),\cdots,\overline{v}_{\lceil (m-q)/3\rceil -1}(n)]^T$. Therefore, a $(\lceil (m-q)/3\rceil+1)$-input Wallace tree and a final m-bit adder are sufficient for implementing the accumulation in (12).

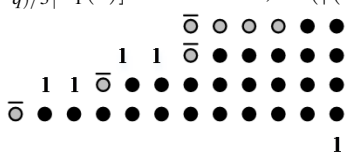


Fig. 7. Partial product tree of an approximate 12×20 radix-8 Booth multiplier with truncation. ●: a partial product; ◍: the sign bit; $\overline{◍}$: the inverted sign bit.

Fig. 8 shows the proposed weight update circuit, only one radix-8 Booth encoder is required for the M multiplications because $\mu e(n)$ is the same for the M weights. Also, the recoding adders for calculating $3x(n-i)$ are shared with the ones in the error computation module as they share the same input multiplicands ($[x(n),x(n-1),\cdots,x(n-M+1)]$). Similarly, a PPG is used to compute the partial product vectors $\overline{\mathbf{v}}(n)\cdot x(n-i)$. Then, the partial product vectors and the weight at the former iteration $w_i(n)$ are accumulated by a $(\lceil (m-q)/3\rceil+1)$-input Wallace tree. The new weight $w_i(n+1)$ is obtained by adding the two output vectors of the Wallace tree using an m-bit CLA. As the weight update module is more sensitive to errors, a smaller number of LSBs is approximated in the Wallace tree.

Consequently, the proposed weight update design saves $(M-1)$ radix-8 Booth encoders and M recoding adders compared with a conventional multiplier based design. It significantly reduces the area and power dissipation when M is large. Moreover, the critical path delay of the proposed design is reduced by $2\times$ of the delay of an adder (i.e., by the delays of the recoding adder and the final adder in the multiplication) compared with the design in Fig. 4.

IV. Truncated Partial Product Generation and Approximate Accumulation

To reduce area, power dissipation and critical path delay of the proposed deign, the partial products in DA are generated by truncating some LSBs of the inputs.

In a parallel DA architecture, accumulation is usually implemented by an adder tree. As the carry-propagating adders in an adder tree are very slow, a Wallace tree is used in this design to speed up the accumulation stage. Moreover, the Wallace tree is approximated to lower the hardware cost.

A. Truncated Partial Product Generation

Due to the partial product accumulation, the final result of an inner product will not be significantly affected if the average error of the approximate partial products is small.

An m-bit number A in 2's complement can be represented as $A=-a_{m-1}2^{m-1}+\sum_{i=0}^{m-2}a_i2^i$, where a_i is the i^{th} LSB of A, and the most significant bit a_{m-1} is the sign bit. Let A_H be the remaining value of A with k $(1\le k\le m/2)$ LSBs truncated. Then, $A_H=-a_{m-1}2^{m-1}+\sum_{i=k}^{m-2}a_i2^i$. Let A_L be $\sum_{i=0}^{k-1}a_i2^i$, the truncation error is then $A_H-A=-A_L$. Let the probability of $a_i=1$ be p, where $0\le p\le 1$. The average error due to truncation is given by

$$E[-A_L]=-p\sum_{i=0}^{k-1}2^i=p(1-2^k), \tag{13}$$

where $E[\cdot]$ denotes an expected value. The maximum error distance (in the absolute value of the error) occurs when the k LSBs of A are all ones. So, the maximum error distance

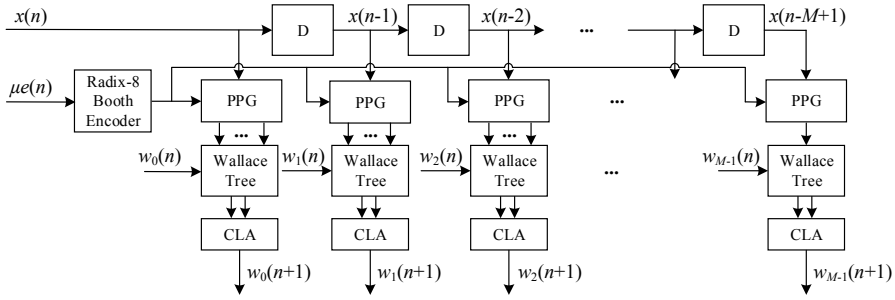


Fig. 8. Proposed weight update scheme. PPG: the partial product generator; CLA: the m-bit carry lookahead adder.

(D_{max}) of A_H is

$$D_{max} = \sum_{i=0}^{k-1} 2^i = 2^k - 1. \tag{14}$$

As per (13), the average error of a truncated number is approximately $-2^k p$. To compensate this error, $2^k p$ is added to A_H. Assume 0 and 1 are equally likely to occur, i.e., the probability of $a_i = 1$ or $a_i = 0$ is 2^{-1}. In this case, the compensation error is 2^{k-1}. The compensated number A' is given by

$$A' = A_H + 2^{k-1} = -a_{m-1}2^{m-1} + \sum_{i=k-1}^{m-2} a_i 2^i, \tag{15}$$

where a_{k-1} is '1', and m is the bit width of A. In this case, truncation error becomes $A' - A = -A_L + 2^{k-1}$; the average error of the truncated number in (13) is reduced to $E[-A_L] + 2^{k-1} = 2^{-1}$. The D_{max} occurs when k LSBs of A are zeros; it is reduced to 2^{k-1}. Using this error compensation scheme for the truncated input operands, the average error of the partial products can be computed in a signed multiplication. Assume that $X = X_H + X_L$ and $Y = Y_H + Y_L$ are the multiplicand and multiplier, respectively, the average error of the partial products is then given by

$$\begin{aligned} E[E_{PP}] = E[&(X_H + 2^{k-1})(Y_H + 2^{k-1}) \\ &- (X_H + X_L)(Y_H + Y_L)] \end{aligned}, \tag{16}$$

where $X_H = -x_{m-1}2^{m-1} + \sum_{i=k}^{m-2} x_i 2^i$, $X_L = \sum_{i=0}^{k-1} x_i 2^i$, $Y_H = -y_{m-1}2^{m-1} + \sum_{i=k}^{m-2} y_i 2^i$ and $Y_L = \sum_{i=0}^{k-1} y_i 2^i$. When the probability of $x_i = 1$ and $y_i = 1$ is 0.5, $E[X_H] = E[Y_H]$ is $2^{-1}(-2^{m-1} + \sum_{i=k}^{m-2} 2^i) = -2^{k-1}$, and $E[X_L] = E[Y_L]$ is $2^{k-1} - 2^{-1}$ as per (13). As X and Y are independent, $E[Y_H X_L] = E[Y_H]E[X_L]$, $E[X_H Y_L] = E[X_H]E[Y_L]$ and $E[X_L Y_L] = E[X_L]E[Y_L]$. The average error of the partial products in (16) becomes

$$\begin{aligned} E[E_{PP}] &= (2^{k-1}(E[X_H] + E[Y_H]) + 2^{2k-2}) - \\ &\quad (E[X_H]E[Y_L] + E[Y_H]E[X_L] + E[X_L]E[Y_L]). \\ &= -2^{-2} \end{aligned} \tag{17}$$

This result indicates that the number of partial products in a DA architecture can be reduced by truncating some LSBs of the input data, and the accumulated sum can be rather accurate by using the proposed error compensation.

For a fixed-width implementation of DA, the partial products at the less significant bit positions can be truncated as in the fixed-width multiplication. Thus, the partial product generation and error compensation schemes for a fixed-width multiplier are further applied to the proposed DA partial product generation. In the fixed-width multiplier design, the partial products at the lower half bit positions are truncated, and the error is compensated by an error compensation strategy. Several error compensation strategies have been proposed for fixed-width Booth multipliers [29]–[32]. Among them, the probabilistic [32] and approximate recoding adder based approaches are very efficient and applicable to the radix-8 Booth algorithm. The comparison in [29] shows that the approximate recoding adder based scheme is significantly more accurate and hardware-efficient than the probabilistic approach for a radix-8 Booth fixed-width multiplier.

In the proposed FIR adaptive filter, therefore, the m-bit input data are truncated by k LSBs and compensated first. The partial products are then approximately generated using the radix-8 Booth encoder and the PPG in the $(m-k+1) \times (m-k+1)$ ABM2-R15. To assess the accuracy of the approximate partial product generation scheme for DA, the inner product with a length of 64 is simulated. In this simulation, 5 LSBs of the inputs are truncated and compensated. The inputs are five million combinations; each combination consists of 64 16-bit random integers generated from the normal distribution. The inputs are divided by 2^{15} to ensure that the inputs are in the range of $[-1, 1)$ and in the fixed-point representation with 1 sign bit and 15 fractional bits. The input combinations for the simulation are selected to make sure their inner products are in the range of $[-1, 1)$. Thus, the inner products are also represented by 16-bit fixed-point numbers with 1 sign bit and 15 fractional bits. Errors are then computed as the difference between the approximate results and the accurate results. To show the errors in integers, both the accurate and approximate inner products are multiplied by 2^{15}. The simulation results show that about 99.79% of the errors are within $(-400, 400)$. Fig. 9 shows the distribution of the errors, where the mean and standard deviation of the errors are around 4 and 122, respectively. Since the range for the accurate outputs is $[-32768, 32767)$, the simulation results indicate that most of the errors due to the approximate partial product generation are very small.

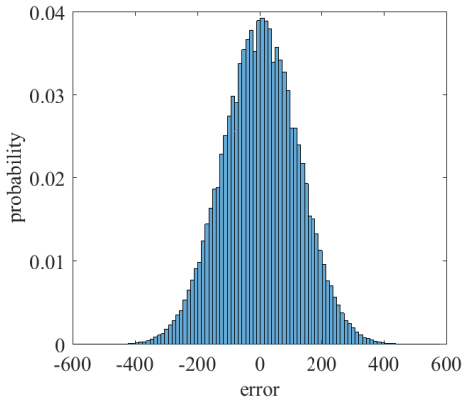


Fig. 9. The error distribution of the proposed approximate partial product generation for DA.

B. Approximate Accumulation

Fig. 10(a) and (b) show the structures of a traditional adder tree (AT) and a Wallace tree (WT) for six m-bit inputs, respectively. For an AT, there are $(M-1)$ m-bit adders in $\lceil log_2 M \rceil$ stages for M inputs ($M > 2$). Thus, the circuit area and the critical path delay are $C_{AT} = (M-1) \times C_{mA}$ and $t_{AT} = \lceil log_2 M \rceil \times t_{mA}$, where C_{mA} and t_{mA} are the circuit area and critical path delay of an m-bit adder. However, the WT requires $\lfloor log_{1.5} M \rfloor$ (for $M > 13$; there is not a general formula to represent the number of required stages in a WT for $M \leq 13$) carry-save stages and one final m-bit carry propagate adder for M inputs. Thus, the circuit area and the critical path delay of the WT are $C_{WT} = (M-2) \times m \times C_{FA} + C_{mA}$ and $t_{WT} = \lfloor log_{1.5} M \rfloor \times t_{FA} + t_{mA}$, where C_{FA} and t_{FA} are the circuit area and critical path delay of a full adder [33]. It is evident that $C_{AT} \geq C_{WT}$ when $C_{mA} \geq m \times C_{FA}$, and $t_{AT} > t_{WT}$ when $t_{mA} > \frac{log_{1.5}2}{1-1/log_2 M} \times t_{FA}$. As $\frac{log_{1.5}2}{1-1/log_2 M}$ decreases with the increase of M, a WT is more efficient in delay than an AT when M is large. In an extreme case where $M = 4$, $t_{AT} = 2 \times t_{mA}$ and $t_{WT} = 2 \times t_{AF} + t_{mA}$ (a 4-input WT requires 2 stages). Therefore, a WT is faster than an AT as long as $t_{mA} > 2 \times t_{FA}$. For the ripple carry adder (RCA), C_{mA} and t_{mA} are proportional to m, while they are proportional to $log_2 m$ and $m log_2 m$, respectively, for a fast carry lookahead adder (CLA). Obviously, a WT has a similar size of circuit with an AT when RCAs are used. On the other hand, a WT has a smaller circuit than an AT when CLAs are used. Additionally, the speed of a WT can be improved by up to 30% by optimizing the signal connections among full adders using the algorithm in [34]. Thus, a speed-optimized WT is implemented for the parallel mode DA in the proposed FIR adaptive filter design.

To further reduce circuit complexity, approximation is applied to the less significant part of a WT as in the lower-part-OR adder (LOA) [27]. In the LOA, the less significant bits are "added" by OR gates and an AND gate is used to generate a carry-in signal for the more significant bits that are summed by a precise adder. LOA is an efficient approximate adder for the accumulative operation due to its low average error [25]. Fig. 10(c) shows an approximate Wallace tree (AWT), in which the less significant bits are accumulated by 3-input OR gates instead of full adders, and 2-input AND gates are used to generate the carry bits for the more significant bits (that are accurately accumulated by full adders). The number of approximate LSBs determines the accuracy of an AWT. Thus, by changing the number of approximate LSBs, the AWT is configured into a circuit with variable accuracy. As the number of '1's in the intermediate results increases within a Wallace tree due to the OR operation, it is more likely to generate an error in a later stage. Therefore, the last few stages in a Wallace tree can be accurately accumulated by using full adders to ensure a high accuracy.

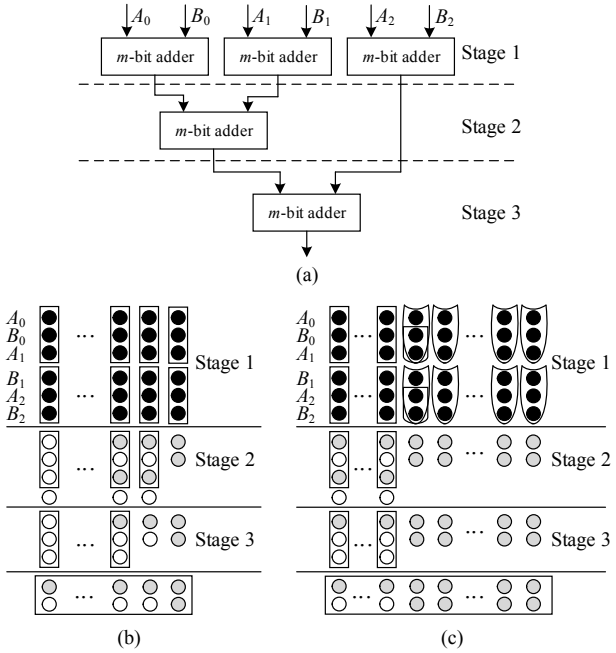


Fig. 10. Accumulation of partial products by (a) a traditional adder tree, (b) a Wallace tree and (c) an approximate Wallace tree. ●: an input bit; ◍: the sum bit from the previous layer; ○: the carry bit from the previous layer; □ : a full adder; : an OR gate; : an AND gate.

The accuracy and measurement of various accumulation circuits are shown in Table II. The accuracy and power dissipation are obtained using 10 million input combinations. Each input combination consists of 64 or 128 16-bit random integer numbers. Specifically, the critical path delay and area are reported by the Synopsys design compiler (DC) by synthesizing the designs in ST's 28 nm CMOS technology with a supply voltage of 1.0 V. The power dissipation is estimated by the PrimeTime-PX with a clock period of 1 ns. Table II shows that the accurate WT is slightly faster and consumes similar or slightly lower power than the AT using CLAs. The area of the WT is significantly smaller than that of its AT counterpart. More significant improvement in latency, area and power dissipation is obtained for a larger bit width.

For the AWTs, their average errors are very small when the number of approximate LSBs is smaller than 5. Also, the standard deviation increases rapidly when the number of approximate LSBs is larger than 4. For hardware, the AWTs

TABLE II. Error and circuit measurements of designs for partial product accumulation.

# inputs	Design	# approximated LSBs	Average error	Standard deviation (10^3)	Delay (ns)	Area (μm^2)	Power (mW)	ADP ($\mu m^2 \cdot ns$)	PDP (pJ)
64	AT	–	0	0	0.83	6,091	6.98	5,055	5.80
	WT	–	0	0	0.81	4,595	6.65	3,722	5.39
	AWT	2	0.82	2.17	0.79	4,521	6.49	3,572	5.13
	AWT	3	1.74	3.38	0.79	3,889	5.64	3,072	4.46
	AWT	4	6.76	4.99	0.79	3,630	4.87	2,868	3.84
	AWT	5	15.07	7.19	0.77	3,582	4.74	2,758	3.65
128	AT	–	0	0	0.96	10,984	12.40	10,544	11.90
	WT	–	0	0	0.94	9,206	12.40	8,654	11.66
	AWT	2	0.14	3.09	0.92	8,809	6.56	8,104	11.22
	AWT	3	4.52	4.84	0.93	7,743	10.50	7,201	9.77
	AWT	4	8.11	7.11	0.92	7,073	9.32	6,507	8.58
	AWT	5	10.62	10.17	0.92	6,341	8.24	5,833	7.58

with 4 approximate LSBs achieve more than 43% reduction in area-delay product (ADP) and about 30% reduction in power-delay product (PDP) compared with conventional ATs.

V. Simulation and Synthesis Results

The adaptive filter is employed to simulate an unknown system as an application of system identification. 64-tap and 128-tap FIR adaptive filters are considered to assess the proposed design as low and high order applications. The unknown systems under consideration are a 48-tap bandpass FIR filter and a 103-tap high-pass FIR filter, which are identified by a 64-tap FIR adaptive filter and a 128-tap FIR adaptive filter, respectively. The step size for the adaptive algorithm is 2^{-8}. The input signal is a random vector generated from the standard normal distribution in $[-1,1)$. White Gaussian noise with a signal-to-noise ratio of 40 dB is added to the output signals of the unknown systems as interference noise.

For an m-bit fixed-point implementation of the FIR adaptive filter, 1 bit is used for the sign bit and $m-1$ bits are used for the fractional part as the input is within the range $[-1,1)$.

A. Accuracy Evaluation

To evaluate the accuracy and convergence of the designs, the mean squared error (MSE) and the normalized misalignment are considered. The MSE measures the difference between the outputs of an unknown system and the adaptive filter. To show the performance in convergence, the MSE is computed at each iteration of the algorithm. Considering the variance in the MSE and computation time, the MSE is averaged over 20 independent trials smoothed by a 20-point moving-average filter. The normalized misalignment indicates the difference between an unknown system's weights and the weights estimated by the adaptive filter at each iteration. It is given by [35]

$$\eta(n) = 20log_{10}\frac{\|\mathbf{h}-\mathbf{w}(n)\|}{\|\mathbf{h}\|}, \tag{18}$$

where $\|\cdot\|$ is the Euclidean norm operation, $\mathbf{h}$ is the weight vector of the unknown system, and $\mathbf{w}(n)$ is the adaptive weight vector at the n^{th} iteration.

Initially, the accurate direct-form FIR adaptive filters in Figs. 3 and 4 at different resolutions (or bit widths) are simulated to investigate the effect of the resolution on accuracy.

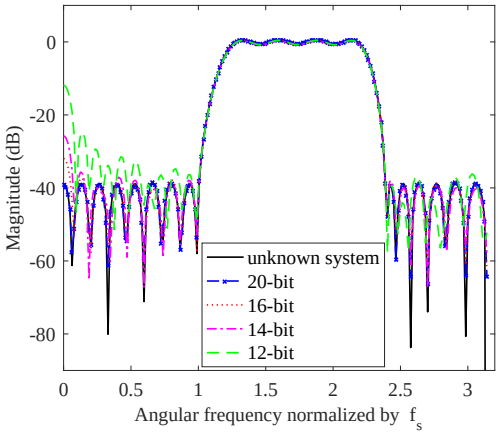
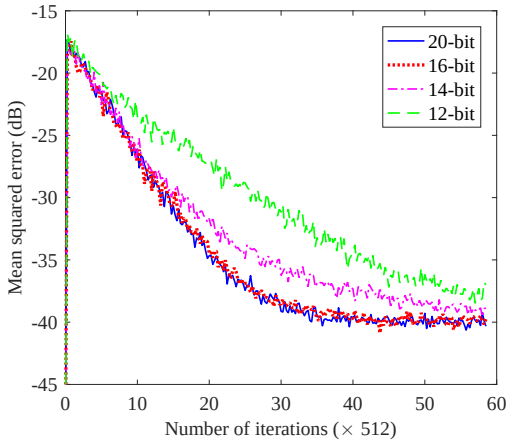


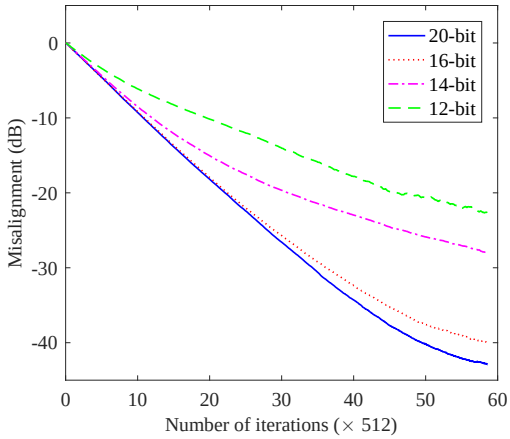
Fig. 11. The impulse responses of the identified systems by using accurate FIR adaptive filters at different resolutions.

For an m-bit implementation, the multiplication and addition are implemented by an accurate $m \times m$ radix-8 Booth multiplier and an accurate m-bit CLA, respectively. The $2m$-bit product by an $m \times m$ multiplier is truncated and rounded to m-bit. For the "unknown system" of a 48-tap FIR bandpass filter, Fig. 11 shows the impulse responses of the identified systems using 20-bit, 16-bit, 14-bit and 12-bit fixed-point FIR adaptive filters after 30,000 iterations. It can be seen that the results by the 12-bit and 14-bit implementations are far off from the "unknown system", while the results by the 16-bit and 20-bit implementations are more accurate due to the higher resolutions.

Based on the comparison results of the accurate FIR adaptive filters, the 20-bit implementation for the proposed FIR adaptive filter is selected to compare with the most efficient DLMS-based designs in [7] at the same resolution. Four configurations of the proposed design are considered for different numbers of truncated LSBs on the input data: T0 (with no truncated bit), T5 (with 5 truncated LSBs), T7 (with 7 truncated LSBs) and T9 (with 9 truncated LSBs). The simulation results in Table II show the tradeoff between accuracy and hardware usage of the AWT. It shows that the AWT with 4 approximate LSBs achieves the best tradeoff with a high accuracy and low power dissipation. Thus, in the error computation module, 4 LSBs are approximated in the four least significant WTs, and

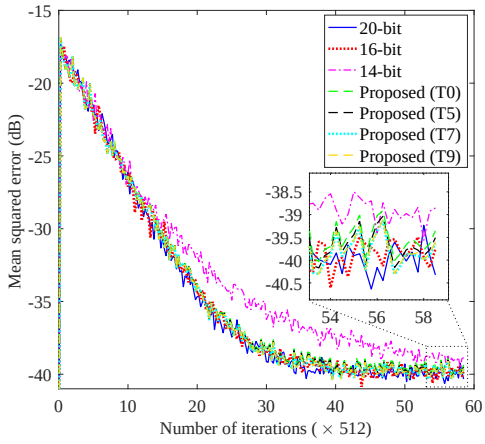


(a)

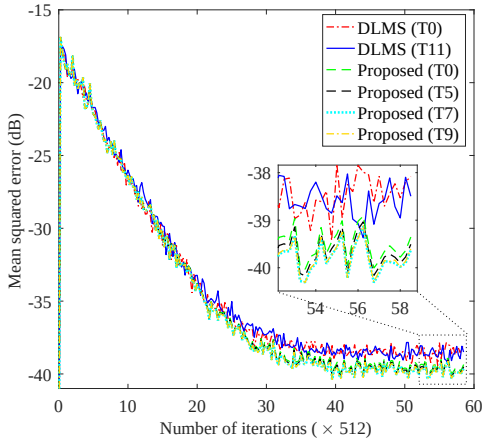


(b)

Fig. 12. Convergence of accurate FIR adaptive filters at different resolutions (a) learning curves in the mean squared error and (b) learning curves in the normalized misalignment.



(a)



(b)

Fig. 13. Comparison of learning curves in the mean squared error between the proposed 64-tap adaptive filters and (a) accurate implementations and (b) DLMS-based designs.

2 LSBs are approximated in the two more significant WTs. The other Wallace trees used in the proposed design remain accurate. For the DLMS design, the schemes without pruning and with a pruning parameter of 11, referred to as DLMS (T0) and DLMS (T11), are considered as well.

As shown in the learning curves for the 64-tap filters in Fig. 13, the proposed designs have a similar convergence speed and steady-state MSE as the 20-bit and 16-bit accurate designs. Compared with the DLMS design, the proposed designs converge slightly faster to a lower MSE, as shown in Fig. 13(b). The normalized misalignment shown in Fig. 14 indicates that the proposed designs result in similar learning processes as the 20-bit accurate design; these designs outperform the other considered designs. The DLMS design causes a high misalignment, which indicates that the system weights identified by the DLMS design are far from those of the actual system.

For the 128-tap FIR adaptive filter designs, the learning results are shown in Fig. 15. As can be seen, the convergence

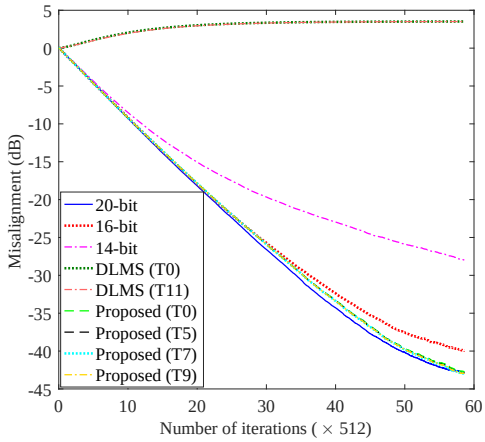
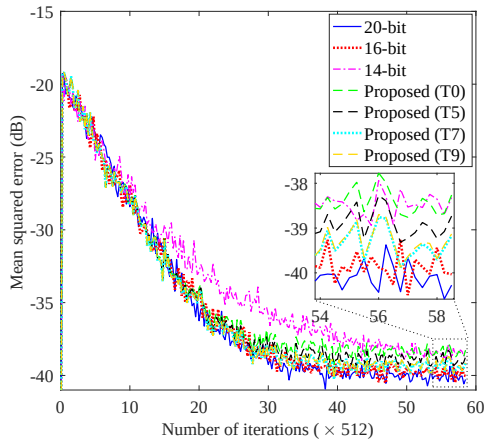
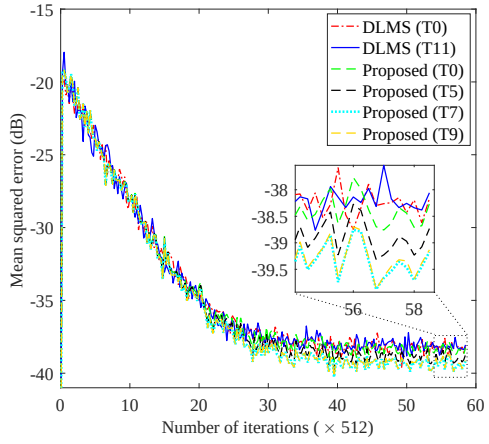


Fig. 14. Learning curves in the normalized misalignment of 64-tap FIR adaptive filter designs.



(a)



(b)

Fig. 15. Comparison of learning curves in the mean squared error between the proposed 128-tap adaptive filters and (a) accurate implementations and (b) DLMS-based designs.

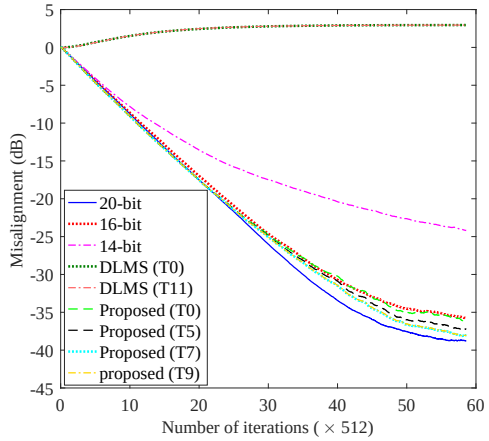


Fig. 16. Learning curves in the normalized misalignment of 128-tap FIR adaptive filter designs.

speeds of the proposed T0 and T5 are slightly slower, whereas the learning curves for the MSE of the T7 and T9 are similar to the accurate 20-bit and 16-bit designs. Fig. 15(b) shows that the proposed designs (except for the T0) perform better than the DLMS designs with lower steady-state MSEs. Similar learning curves in the normalized misalignment are obtained for the 128-tap designs and shown in Fig. 16. However, the differences between the proposed designs are rather noticeable. In this case, the learning curves in the misalignment of T0 and T5 are closer to the accurate 16-bit design, and the curves for T7 and T9 are closer to the accurate 20-bit design. Moreover, the steady-state MSEs of the considered designs (reported in Table III) show a similar trend.

B. Hardware Efficiency

To evaluate the hardware efficiency, the filter designs are implemented in VHDL and synthesized by the Synopsys DC in ST's 28 nm CMOS technology. For ease of comparison, all designs are synthesized in the same process with the same supply voltage, temperature, optimization option and clock period. The supply voltage and temperature are 1.0 V and 25 °C, respectively. The critical path delay, area and power dissipation of the designs are reported by the Synopsys DC. The average power dissipation is estimated by using the PrimeTime-PX with the same inputs as in the accuracy evaluation. The clock period for the power estimation is 4 *ns*.

For the performance evaluation, the values of the energy per operation (EPO) and throughput per area (TPA) [36] are computed for the considered designs. The EPO is defined as the energy consumed per operation during one clock period, and the TPA is defined as the number of operations per unit time and per unit area. They are respectively given by

$$EPO = t_{op} \times Power, \tag{19}$$

and

$$TPA = 1/(t_{min} \times Area), \tag{20}$$

where t_{op} and t_{min} are respectively the time required per operation, i.e., the operating clock period of a circuit, and the shortest time required per operation (or the critical path delay of a combinational circuit). *Power* is the total power consumption including the dynamic and leakage powers. *Area* is the circuit area.

Table IV shows the hardware measurements of the FIR adaptive filter circuits. The "shared-LUT" denotes an accurate 20-bit fixed-point implementation of the FIR adaptive filter using shared LUTs (16-word) [20]; CLAs are used to implement the additions in this design. For a fair comparison, in the other accurate implementations without using DA (20-bit, 16-bit, 14-bit and 12-bit), the multiplications and additions are implemented by radix-8 Booth multipliers and CLAs, respectively. The additions in the DLMS-based design are implemented by CLAs too. During the synthesis, the shortest critical path delay is found such that the tightest timing constraint is applied to each design with no timing violation. Table IV shows that the shared-LUT design is the slowest and that its improvements in area and power are very small

TABLE III. Steady-state MSEs of considered FIR adaptive filter designs in an increasing order (dB).

Filter length	20-bit	16-bit	proposed (T7)	proposed (T9)	proposed (T5)	proposed (T0)	14-bit	DLMS (T0)	DLMS (T11)
64	-39.98	-39.86	-39.80	-39.78	-39.67	-38.59	-38.94	-39.53	-38.56
128	-39.97	-39.85	-39.37	-39.32	-38.90	-38.40	-38.59	-38.21	-38.20

TABLE IV. Hardware characteristics of FIR adaptive filter designs.

Filter length	Design	Delay (ns)	Area (μm^2)	Total power (mW)	Leakage power (μW)	EPO (pJ /operation)	TPA (operation $/(s \cdot \mu m^2)$)	EPO reduction (%)	TPA increase (%)
64	20-bit	2.78	246,472	85.3	191.6	341.2	1,459	0.00	0.00
	16-bit	2.73	222,021	78.4	171.8	313.6	1,650	8.09	13.05
	14-bit	2.67	214,142	75.4	172.9	301.6	1,749	11.61	19.84
	12-bit	2.65	193,616	67.6	154.7	270.4	1,949	20.75	33.54
	DLMS (T0)	0.73	208,178	84.2	119.0	336.8	6,580	1.29	350.88
	DLMS (T11)	0.74	188,785	76.8	103.9	306.8	7,158	10.08	390.47
	shared-LUT	2.92	217,296	67.9	253.7	271.6	1,576	20.40	7.99
	proposed (T0)	2.13	98,644	37.2	76.43	148.8	4,759	56.39	226.11
	proposed (T5)	2.13	93,897	35.5	74.27	142.0	5,000	58.39	242.60
	proposed (T7)	2.11	94,593	36.0	77.29	144.0	5,010	57.80	243.30
	proposed (T9)	2.11	84,577	32.8	67.61	131.2	5,604	61.55	283.95
128	20-bit	2.90	448,025	150.4	327.3	601.6	770	0.00	0.00
	16-bit	2.86	433,610	146.8	331.4	587.2	806	2.39	4.77
	14-bit	2.85	368,019	125.4	271.5	501.6	953	16.62	23.88
	12-bit	2.81	362,233	123.5	279.0	494.0	982	17.88	27.65
	DLMS (T0)	0.77	384,163	154.3	207.2	617.2	3,380	-2.59	339.23
	DLMS (T11)	0.76	359,219	145.1	193.5	508.4	3,663	3.52	375.91
	shared-LUT	3.10	405,515	123.3	271.0	493.2	795	18.02	3.36
	proposed (T0)	2.29	193,439	70.3	146.4	281.2	2,257	53.26	193.31
	proposed (T5)	2.27	184,281	67.4	141.1	269.6	2,391	55.19	210.59
	proposed (T7)	2.25	175,127	65.7	137.9	262.8	2,537	56.32	229.74
	proposed (T9)	2.24	170,672	63.7	134.8	254.8	2,615	57.65	239.85

Note: In the accurate implementations (20-bit, 16-bit, 14-bit and 12-bit), the multiplications and additions are implemented by radix-8 Booth multipliers and CLAs, respectively. The additions in the DLMS-based design and the shared-LUT design are implemented by CLAs. .

compared to the 20-bit implementation. The long delay is mainly due to the update and access of the LUTs. The DA architecture using LUTs is more efficient for an FIR filter with fixed coefficients, for which no update is required for the LUTs. The hardware efficiency of the shared-LUT design decreases with the increase of the filter length. The proposed designs require shorter critical path delays than the accurate designs; however, the DLMS designs use the shortest delays due to the pipelining implementation. Increasing the number of truncated LSBs on the inputs has a more significant effect on reductions in area and power consumption than on delay, because the critical path of the Wallace tree in the proposed design is very short and reducing the accumulated partial product bits does not change it much. Among the considered designs, the proposed designs require the lowest area and power dissipation. The accurate designs incur the longest critical path delay, and the DLMS designs require slightly smaller area than the accurate ones. Furthermore, the DLMS designs incur higher power dissipations than some accurate designs due to the large hardware overhead caused by the additional latches used for pipelining. The proposed designs show the lowest EPO, whereas the DLMS designs require the highest EPO.

Finally, the EPO reduction and TPA increase of the filter designs are reported in the last two columns of Table IV. The proposed designs achieve nearly a 55% EPO reduction and a 3.2× TPA on average compared to the accurate 20-bit implementation. Additionally, they show a 45%-51% reduction in EPO and 2.3× to 2.9× TPA compared with an accurate 12-bit implementation. The EPO of the DLMS designs is larger by 2%-9% due to the high power dissipation. However, the TPAs are larger by 3.4× to 3.9× due to the shorter critical path delay. Compared with DLMS designs, the proposed ones show lower TPAs and smaller EPOs by 15%-40% and 45%-64%, respectively.

VI. Cerebellar Model Evaluation

The cerebellum plays a key role in the control of eye movement in the saccadic system; this involuntary eye movement is referred to as the vestibulo-ocular reflex (VOR). The VOR stabilizes a visual stimulus into the center of the retina (fovea) for a clear vision when the head moves [37]. Fig. 17 shows a simplified model of the VOR, where the cerebellum predicts the eye plant output and indirectly compensates the movement command. In the saccadic system, the head movements are sensed by the vestibular system consisting of semicircular canals and otolith organs [38]. For simplicity, only the horizontal head velocity sensed by the horizontal canal is considered as the input. The horizontal canal is modeled as a high-pass filter, $V(s) = \frac{s}{s+1/T_c}$, where $T_c = 6s$ [38]. The brainstem acts as a control center that receives the sensory information and compensation signals from the cerebellum. It then generates commands to drive the eye muscles for movement. The transfer

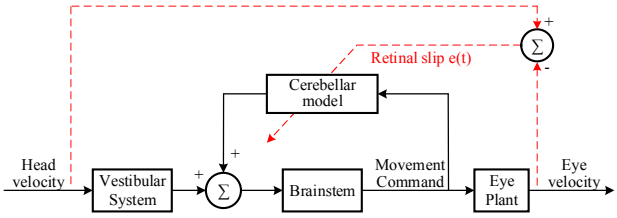


Fig. 17. A simplified model of the VOR.

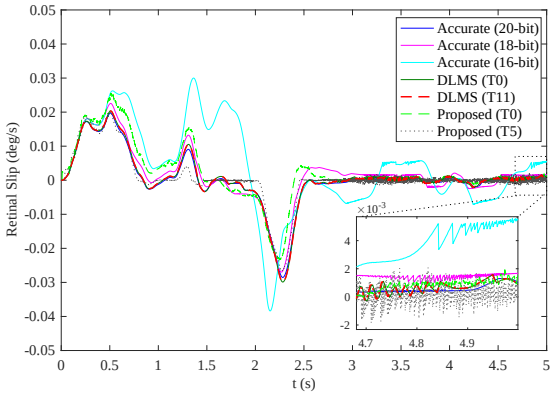


Fig. 18. The retinal slip during a 5s VOR training.

functions of the brainstem and the eye plant are given by $B(s) = G_d + \frac{G_i}{s+1/T_i}$ and $P(s) = \frac{s(s+1/T_z)}{(s+1/T_1)(s+1/T_2)}$, respectively, where $G_d = 1$, $G_i = 5.05$, $T_i = 500ms$, $T_1 = 370ms$, $T_2 = 57ms$ and $T_z = 200ms$ [39].

To evaluate the accuracy of the approximate cerebellar model, the saccadic system in Fig. 17 is implemented in MATLAB. The cerebellar model is implemented in an n-bit fixed-point format consisting of 1 sign bit and $(n-1)$ fractional bits. Fig. 18 shows the retinal slip (i.e., error signal) during a 5-s training, where the constant delay T is 1 ms, M is 128, and the step size μ is set to 2^{-8}. It can be seen that the accurate 20-bit fixed-point cerebellar model produces the lowest stable retinal slip, followed by the 18-bit implementation, whereas the retinal slip of the 16-bit implementation does not converge. The proposed T0 and DLMS designs achieve a similarly small retinal slip as the accurate 20-bit design. However, the DLMS designs show more fluctuations than the proposed T0 at the stable phase, as shown in the inset. The proposed T5 converges faster than the other designs, but it generates a similar retinal slip as the accurate 18-bit design that is slightly higher than the accurate 20-bit design. As the VOR system requires a higher accuracy than the system identification application, a converged retinal slip cannot be obtained by using the other configurations of the proposed design.

VII. Conclusion

This paper proposes a high-performance and energy-efficient fixed-point FIR adaptive filter design. It utilizes an integrated circuit of approximate distributed arithmetic (DA), so it achieves significant improvements in delay, area and power dissipation. Approximate partial product generation and accumulation schemes are proposed for the error computation and weight update modules in the adaptive filter. Moreover, an approximate radix-8 Booth algorithm is applied to the DA. The critical path and hardware complexity are significantly reduced due to the use of approximate and distributed arithmetic.

Two system identification applications using 64-tap and 128-tap FIR adaptive filters are considered to assess the quality of the proposed design. At a similar accuracy, the proposed design consumes more than 55% lower EPO and achieves a 3.2× TPA compared with the corresponding accurate design. Compared to a state-of-the-art design, the proposed design achieves a 45-64% reduction in EPO with a higher accuracy. A visual saccadic system using the proposed approximate adaptive filter in a cerebellar model achieves a similar retinal slip in vestibulo-ocular reflex as using an accurate filter. These results indicate that approximate arithmetic circuits are applicable to integrated circuit design for a better performance and energy efficiency.

References

[1] D. Marr and W. T. Thach, "A theory of cerebellar cortex," *The Journal of Physiology*, vol. 202, no. 3, pp. 437–470, 1969.

[2] J. S. Albus, "A theory of cerebellar function," *Mathematical Biosciences*, vol. 10, no. 1-2, pp. 25–61, 1971.

[3] T. W. Calvert and F. Meno, "Neural systems modeling applied to the cerebellum," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 363–374, 1972.

[4] M. Hassul and P. D. Daniels, "Cerebellar dynamics: the mossy fiber input," *IEEE Transactions on Biomedical Engineering*, no. 5, pp. 449–456, 1977.

[5] M. Fujita, "Adaptive filter model of the cerebellum," *Biological cybernetics*, vol. 45, no. 3, pp. 195–206, 1982.

[6] D. Comminiello, M. Scarpiniti, L. A. Azpicueta-Ruiz, J. Arenas-Garcia, and A. Uncini, "Functional link adaptive filters for nonlinear acoustic echo cancellation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 7, pp. 1502–1512, 2013.

[7] P. K. Meher and S. Y. Park, "Area-delay-power efficient fixed-point LMS adaptive filter with low adaptation-delay," *IEEE transactions on VLSI systems*, vol. 22, no. 2, pp. 362–371, 2014.

[8] M. Ito, "Cerebellar circuitry as a neuronal machine," *Progress in neurobiology*, vol. 78, no. 3, pp. 272–303, 2006.

[9] A. Lenz, S. R. Anderson, A. G. Pipe, C. Melhuish, P. Dean, and J. Porrill, "Cerebellar-inspired adaptive control of a robot eye actuated by pneumatic artificial muscles," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 6, pp. 1420–1433, 2009.

[10] E. H. Krishna, M. Raghuram, K. V. Madhav, and K. A. Reddy, "Acoustic echo cancellation using a computationally efficient transform domain LMS adaptive filter," in *International Conference on Information Sciences Signal Processing and their Applications*, 2010, pp. 409–412.

[11] T. K. Paul and T. Ogunfunmi, "On the convergence behavior of the affine projection algorithm for adaptive filters," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 8, pp. 1813–1826, 2011.

[12] P. K. Meher and S. Y. Park, "Critical-path analysis and low-complexity implementation of the LMS adaptive algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 3, pp. 778–788, 2014.

[13] N. V. Thakor and Y.-S. Zhu, "Applications of adaptive filtering to ECG analysis: noise cancellation and arrhythmia detection," *IEEE Transactions on Biomedical Engineering*, vol. 38, no. 8, pp. 785–794, 1991.

[14] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE Assp Magazine*, vol. 6, no. 3, pp. 4–19, 1989.

[15] P. K. Meher, S. Chandrasekaran, and A. Amira, "FPGA realization of FIR filters by efficient and flexible systolization using distributed arithmetic," *IEEE transactions on signal processing*, vol. 56, no. 7, pp. 3009–3017, 2008.

[16] H. Yoo and D. V. Anderson, "Hardware-efficient distributed arithmetic architecture for high-order digital filters," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, 2005, pp. 125–128.

[17] S. Mirzaei, A. Hosangadi, and R. Kastner, "FPGA implementation of high speed FIR filters using add and shift method," in *International Conference on Computer Design*, 2007, pp. 308–313.

[18] R. Guo and L. S. DeBrunner, "Two high-performance adaptive filter implementation schemes using distributed arithmetic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 9, pp. 600–604, 2011.

[19] B. K. Mohanty and P. K. Meher, "A high-performance energy-efficient architecture for FIR adaptive filter based on new distributed arithmetic formulation of block LMS algorithm," *IEEE transactions on signal processing*, vol. 61, no. 4, pp. 921–932, 2013.

[20] S. Y. Park and P. K. Meher, "Efficient FPGA and ASIC realizations of DA-based reconfigurable FIR digital filter," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 7, pp. 511–515, 2014.

[21] G. Long, F. Ling, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 9, pp. 1397–1405, 1989.

[22] P. Kabal, "The stability of adaptive minimum mean square error equalizers using delayed adjustment," *IEEE transactions on Communications*, vol. 31, no. 3, pp. 430–432, 1983.

[23] Y.-H. Chen, J.-N. Chen, T.-Y. Chang, and C.-W. Lu, "High-throughput multistandard transform core supporting MPEG/H.264/VC-1 using common sharing distributed arithmetic," *IEEE Transactions on VLSI Systems*, vol. 22, no. 3, pp. 463–474, 2014.

[24] M. Martina, G. Masera, M. R. Roch, and G. Piccinini, "Result-biased distributed-arithmetic-based filter architectures for approximately computing the DWT," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 8, pp. 2103–2113, 2015.

[25] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, no. 4, p. 60, 2017.

[26] E. J. King and E. E. Swartzlander, "Data-dependent truncation scheme for parallel multipliers," in *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems & Computers*, vol. 2, 1997, pp. 1178–1182.

[27] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.

[28] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM Sigplan Notices*, vol. 49, no. 4, pp. 269–284, 2014.

[29] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 Booth multipliers for low-power and high-performance operation," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2638–2644, 2016.

[30] K.-J. Cho, K.-C. Lee, J.-G. Chung, and K. K. Parhi, "Design of low-error fixed-width modified booth multiplier," *IEEE Transactions on VLSI Systems*, vol. 12, no. 5, pp. 522–531, 2004.

[31] J.-P. Wang, S.-R. Kuang, and S.-C. Liang, "High-accuracy fixed-width modified booth multipliers for lossy applications," *IEEE Transactions on VLSI Systems*, vol. 19, no. 1, pp. 52–60, 2011.

[32] C.-Y. Li, Y.-H. Chen, T.-Y. Chang, and J.-N. Chen, "A probabilistic estimation bias circuit for fixed-width booth multiplier and its DCT applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 4, pp. 215–219, 2011.

[33] K. C. Bickerstaff, E. E. Swartzlander, and M. J. Schulte, "Analysis of column compression multipliers," in *IEEE Symposium on Computer Arithmetic*, 2001, pp. 33–39.

[34] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 294–306, 1996.

[35] K.-A. Lee, W.-S. Gan, and S. M. Kuo, *Subband adaptive filtering: theory and implementation*. John Wiley & Sons, 2009.

[36] R. Wang, J. Han, B. Cockburn, and D. Elliott, "Stochastic circuit design and performance evaluation of vector quantization for different error measures," *IEEE Transactions on VLSI Systems*, vol. 24, no. 10, pp. 3169–3183, 2016.

[37] M. Antonelli, A. J. Duran, E. Chinellato, and A. P. Del Pobil, "Adaptive saccade controller inspired by the primates' cerebellum," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 5048–5053.

[38] M. Ranjbaran and H. L. Galiana, "Hybrid model of the context dependent vestibulo-ocular reflex: implications for vergence-version interactions," *Frontiers in computational neuroscience*, vol. 9, 2015.

[39] P. Dean, J. Porrill, and J. V. Stone, "Visual awareness and the cerebellum: possible role of decorrelation control," *Progress in brain research*, vol. 144, pp. 61–75, 2004.