Exact and Approximate Algorithms for the Filter Design Optimization Problem

Levent Aksoy, *Member, IEEE*, Paulo Flores, *Senior Member, IEEE*, and José Monteiro, *Senior Member, IEEE*

***Abstract*—The filter design optimization (FDO) problem is defined as finding a set of filter coefficients that yields a filter design with minimum complexity, satisfying the filter constraints. It has received a tremendous interest due to the widespread application of filters. Assuming that the coefficient multiplications in the filter design are realized under a shift-adds architecture, the complexity is generally defined in terms of the total number of adders and subtractors. In this paper, we present an exact FDO algorithm that can guarantee the minimum design complexity under the minimum quantization value, but can only be applied to filters with a small number of coefficients. We also introduce an approximate algorithm that can handle filters with a large number of coefficients using less computational resources than the exact FDO algorithm and find better solutions than existing FDO heuristics. We describe how these algorithms can be modified to handle a delay constraint in the shift-adds designs of the multiplier blocks and to target different filter constraints and filter forms. Experimental results show the effectiveness of the proposed algorithms with respect to prominent FDO algorithms and explore the impact of design parameters, such as the filter length, quantization value, and filter form, on the complexity and performance of filter designs.**

***Index Terms*—Delay reduction, depth-first and local search methods, direct and transposed forms, filter design optimization problem, finite impulse response filters, multiplierless design.**

I. Introduction

DIGITAL filtering is a ubiquitous operation in digital signal processing (DSP) applications and is realized using infinite impulse response (IIR) or finite impulse response (FIR) filters. Although an FIR filter requires a larger number of coefficients than an equivalent IIR filter, it is preferred to the IIR filter due to its stability and phase linearity properties [1]. The computation of the output of an N-tap FIR filter is given by

$$y(n) = \sum_{i=0}^{N-1} h_i \cdot x(n-i) \qquad (1)$$

where N is the filter length, h_i is the ith filter coefficient, and $x(n-i)$ is the ith previous filter input. The straightforward realization of (1) is depicted in Fig. 1(a) which is known as the direct form. Alternatively, the realization of (1) in the transposed form is shown in Fig. 1(b).

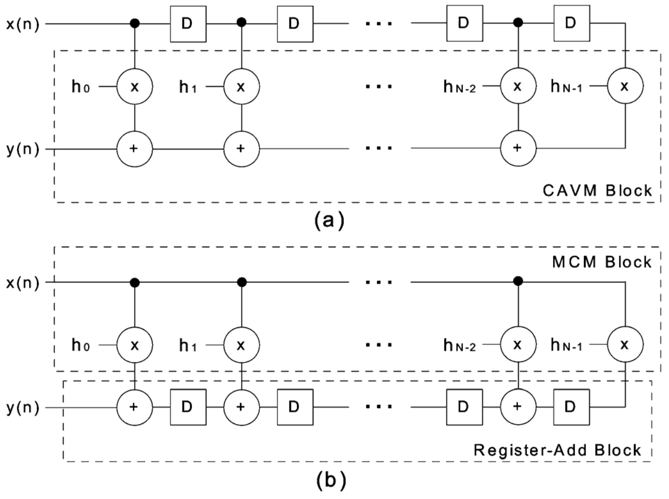

Fig. 1. Different forms of an N-tap FIR filter: (a) direct; (b) transposed.

The complexity of the FIR filter design is dominated by the multiplication of filter coefficients by the time-shifted versions of the filter input, i.e., the constant array-vector multiplication (CAVM) block in the direct form of Fig. 1(a) or by the multiplication of filter coefficients by the filter input, i.e., the multiple constant multiplications (MCM) block in the transposed form of Fig. 1(b). Since filter coefficients are fixed and determined beforehand and the realization of a multiplier in hardware is expensive in terms of area, delay, and power dissipation, these CAVM and MCM operations are generally implemented under a shift-adds architecture using only shifts, adders, and subtractors [2]. Note that shifts by a constant value can be implemented using only wires which represent no hardware cost. Thus, a well-known optimization problem [3] is defined as: given a set of constants, find the minimum number of adders/subtractors that realize the constant multiplications. Note that this is an NP-complete problem even in the case of a single constant multiplication [4]. In the last two decades, many efficient algorithms were proposed for the multiplierless design of the MCM block, targeting not only the minimization of the number of operations, but also the optimization of gate-level area, delay, throughput, and power dissipation of the MCM design [3], [5]–[14]. The algorithm of [15] guarantees the least number of operations in the CAVM design and incorporates efficient techniques to reduce the gate-level area and delay of the CAVM design.

On the other hand, the FDO problem [16] is defined as: given the filter specifications *fspec*, defined as a five-tuple (filter length N, passband w_p and stopband w_s frequencies,

Manuscript received March 21, 2014; revised August 01, 2014; accepted October 09, 2014. Date of publication October 31, 2014; date of current version December 04, 2014. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Joseph Cavallaro. This work was partially supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project PEst-OE/EEI/LA0021/2013.

L. Aksoy is with INESC-ID, 1000-029, Lisbon, Portugal (e-mail: levent@algos.inesc-id.pt).

P. Flores and J. Monteiro are with INESC-ID/Instituto Superior Técnico, University of Lisbon, 1000-029, Lisbon, Portugal (e-mail: pff@inesc-id.pt; jcm@inesc-id.pt).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2014.2366713

and passband δ_p and stopband δ_s ripples), find a set of filter coefficients that yields a filter design with the minimum number of adders/subtractors and satisfies the filter constraints. Many efficient FDO algorithms were proposed, considering different filter constraints, targeting different filter forms, using different search methods during the exploration of possible filter coefficients, and applying different techniques to reduce the filter design complexity [17]–[29]. However, none of these algorithms can guarantee that their solutions (a set of filter coefficients) lead to a filter design with the minimum number of adders/subtractors. This is due to two main facts: i) they do not explore the whole search space; and/or ii) they are not equipped with the exact techniques that can find the minimum number of operations for the constant multiplications.

In this article, we present the exact FDO algorithm [30], called SIREN, that can find a set of fixed-point filter coefficients, satisfying the filter constraints and leading to a filter design with the minimum number of adders/subtractors under the minimum quantization value. SIREN is equipped with a depth-first search (DFS) method to explore the search space exhaustively, the exact algorithm of [9] to find the minimum number of operations in the MCM block of the transposed form, and efficient search pruning and branching techniques to speed up the search process. Since the size of the search space of the FDO problem grows exponentially with the filter length N [29], SIREN can only handle filters with a small number of coefficients. It was observed that it can find solutions to the symmetric filters including less than 40 coefficients in a reasonable time.

Hence, in this article, we propose an approximate algorithm, called NAIAD, that can cope with the FDO problems which SIREN cannot handle and obtain solutions close to the minimum. NAIAD initially finds possible sets of filter coefficients satisfying the filter constraints. Then, a local search method is applied to each set of filter coefficients to explore the feasible solutions around its neighborhood, aiming to reduce the total number of adders/subtractors in the filter design. It was observed that NAIAD can handle symmetric filters including more than 100 coefficients up to 325.

Note that the direct form filters occupy less area and consume less power, but have higher delay than the transposed form filters [31]. Hence, we present the modifications made to SIREN and NAIAD to target both the transposed and direct filter forms. Note also that the number of adder-steps in the multiplier block of a filter, i.e., the number of adders/subtractors in series, has a significant impact on the delay of the filter design [31]. Hence, we describe how these algorithms can be modified to handle a delay constraint in the multiplierless design of the CAVM and MCM blocks of the direct and transposed filter forms.

The rest of the article is organized as follows. Section II presents the background concepts and related work. The exact and approximate FDO algorithms are introduced in Section III and experimental results are given in Section IV. Finally, Section V concludes the article.

II. Background

This section gives the background concepts and presents an overview on the methods proposed for the shift-adds design of the MCM and CAVM blocks and the FDO problem.

A. Linear Programming

Linear programming (LP) is a technique to minimize or maximize a linear cost function subject to a set of linear constraints. An LP problem is given as follows:[1]

$$\begin{aligned} minimize: &\quad f = \mathbf{c}^T \cdot \mathbf{x} \\ subject\ to: &\quad \mathbf{A}\cdot\mathbf{x} \geq \mathbf{b} \\ &\quad \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \end{aligned} \tag{2}$$

In (2), c_j in $\mathbf{c}$ is a cost value associated with each variable x_j, $1 \leq j \leq n$, in the cost function f, and $\mathbf{A}\cdot\mathbf{x} \geq \mathbf{b}$ denotes a set of m linear constraints. Also, $\mathbf{lb}$ and $\mathbf{ub}$ respectively consist of the lower and upper bounds of variables.

The variables are assumed to be real numbers in an LP problem, for which there exist polynomial-time algorithms [32], [33]. However, if all or some variables are restricted to integers, as in pure integer LP (ILP) or mixed ILP (MILP) problems, respectively, these LP problems become NP-complete, for which there is no polynomial-time algorithm [34].

B. Multiplierless Design of the CAVM and MCM Blocks

The CAVM block of the direct form filter realizes a linear transform in the form of $y = h_0x_0 + h_1x_1 + \ldots + h_{N-1}x_{N-1}$, where x_i stands for the time-shifted version of the filter input with $0 \leq i \leq N-1$ (Fig. 1(a)). Also, the MCM block of the transposed form filter implements the constant multiplications in the form of $y_0{=}h_0x, y_1{=}h_1x, \ldots, y_{N-1}{=}h_{N-1}x$, where x denotes the filter input (Fig. 1(b)).

For their shift-adds designs, the digit-based recoding (DBR) technique [35] first defines the constants under a number representation, e.g., binary or canonical signed digit (CSD)[2] [5]. Second, for the nonzero digits in the representations of constants, it shifts the variables according to the digit positions and adds/subtracts the shifted variables with respect to the digit values. As an example, consider $h_0 = 21$ and $h_1 = 53$ and suppose that the CSD representation is used. The decomposition of the linear transform $y = 21x_0 + 53x_1$ is as follows:

$$\begin{aligned} y = 21x_0 + 53x_1 &= (10101)_{CSD}x_0 + (10\bar{1}0101)_{CSD}x_1 \\ &= x_0 \ll 4 + x_0 \ll 2 + x_0 + x_1 \ll 6 - x_1 \ll 4 + x_1 \ll 2 + x_1 \end{aligned}$$

where 6 operations are required for this CAVM block as shown in Fig. 2(a). Also, the decompositions of constant multiplications $y_0 = 21x$ and $y_1 = 53x$ in an MCM block are as follows:

$$\begin{aligned} y_0 &= 21x = (10101)_{CSD}x = x \ll 4 + x \ll 2 + x \\ y_1 &= 53x = (10\bar{1}0101)_{CSD}x = x \ll 6 - x \ll 4 + x \ll 2 + x \end{aligned}$$

which lead to a design with 5 operations as shown in Fig. 2(b).

In the following two subsections, prominent algorithms designed for the optimization of the number of adders/subtractors in the MCM and CAVM blocks are described in detail. Their common aim is to maximize the sharing of partial products.

1) Multiplierless Design of the MCM Operation: The methods proposed for the shift-adds design of an MCM block are

[1]The minimization objective can be easily converted to a maximization objective by negating the cost function. Less-than-or-equal and equality constraints are accommodated by the equivalences, $\mathbf{A}\cdot\mathbf{x} \leq \mathbf{b} \Leftrightarrow -\mathbf{A}\cdot\mathbf{x} \geq -\mathbf{b}$ and $\mathbf{A}\cdot\mathbf{x} = \mathbf{b} \Leftrightarrow (\mathbf{A}\cdot\mathbf{x} \geq \mathbf{b}) \wedge (\mathbf{A}\cdot\mathbf{x} \leq \mathbf{b})$, respectively.

[2]An integer can be written in CSD using n digits as $\sum_{i=0}^{n-1} d_i 2^i$, where $d_i \in \{1, 0, \bar{1}\}$ and $\bar{1}$ denotes -1 with $0 \leq i \leq n-1$. Under CSD, nonzero digits are not adjacent and the minimum number of nonzero digits is used.

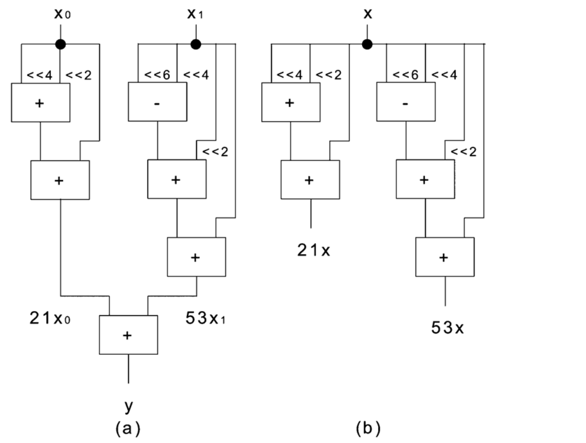

Fig. 2. Multiplierless realization of constant multiplications using the DBR technique [35]: (a) $21x_0{+}53x_1$; (b) $21x$ and $53x$.

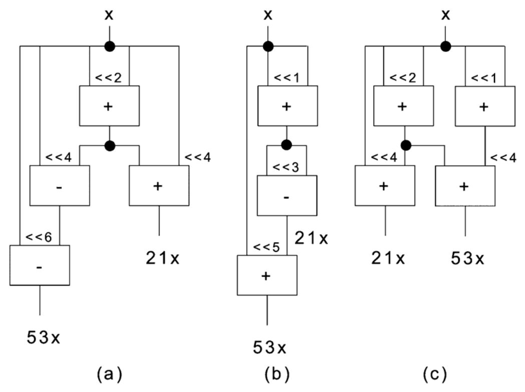

Fig. 3. Multiplierless realization of $21x$ and $53x$: (a) exact CSE algorithm [7]; (b) exact GB algorithm [9]; (c) approximate GB algorithm [9] modified to handle a delay constraint.

generally grouped in two categories as the *common subexpression elimination* (CSE) algorithms [5]–[7] and the *graph-based* (GB) techniques [3], [8], [9]. The CSE methods initially define the constants under a particular number representation. Then, they consider possible subexpressions, that can be extracted from the nonzero digits in the constant representations, and choose the "best" subexpression, generally the most common, to be shared among the constant multiplications. Their main drawback is their dependency on a number representation. The GB methods are not restricted to any particular number representation and aim to find intermediate subexpressions that enable the realization of constant multiplications with the minimum number of operations. They consider a larger number of realizations of a constant and obtain better solutions than the CSE methods, but require more computational resources due to a larger search space.

For our MCM example in Fig. 2(b), the exact CSE algorithm [7] obtains a minimum solution with 4 operations by finding the most common subexpression $5x = (101)_{CSD}x$ (Fig. 3(a)) when constants are defined under CSD. The exact GB algorithm of [9] obtains a minimum solution with 3 operations by finding the intermediate subexpression $3x$ (Fig. 3(b)).

The minimum adder-steps of a shift-adds design of a single constant multiplication, hx, is computed as $\lceil log_2 S(h)\rceil$, where $S(h)$ is the number of nonzero digits in the CSD representation of h. Given an MCM instance with N constants, its minimum adder-steps is determined as $\mathrm{MAS}_{MCM}{=}\max_i\{\lceil log_2 S(h_i)\rceil\}$ with $0{\le}i{\le}N{-}1$ [11]. Given a delay constraint dc with $dc{\ge}\mathrm{MAS}_{MCM}$, the algorithms of [7], [11], [12] can find the smallest number of operations that realize the constant multiplications without violating dc. For our example, the minimum adder-steps of both $21x$ and $53x$ is 2. The approximate algorithm [9] modified to handle a delay constraint finds a solution with 4 operations when dc is 2 (Fig. 3(c)). With respect to the solution of the exact GB algorithm [9] in Fig. 3(b), its solution has one more operation, but one less adder-step.

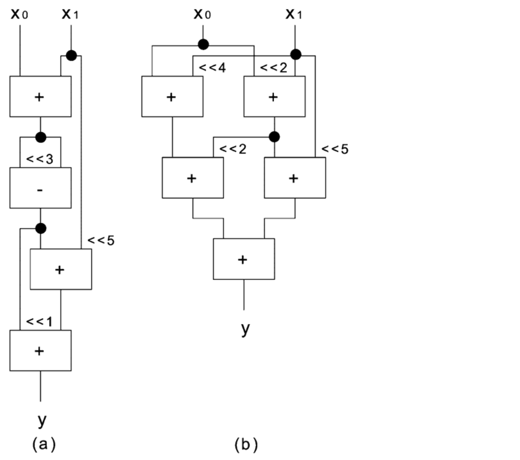

Fig. 4. Shift-adds design of $21x_0{+}53x_1$: (a) ECHO-A [15]; (b) ECHO-D [15].

2) Multiplierless Design of the CAVM Operation: The algorithm of [15], called ECHO, consists of two main parts. In its first part, the shift-adds realizations of constants in the CAVM operation are found using an MCM algorithm. In its second part, the constants in the linear transform are replaced with their realizations in the MCM solution and the common subexpressions are extracted iteratively using a set of transformations. ECHO has two variations, ECHO-A and ECHO-D, that target the optimization of area and delay of the CAVM operation, respectively. ECHO-A uses the exact MCM algorithm [9] and considers some area optimizations. ECHO-D is equipped with the approximate MCM algorithm [9] modified to handle a delay constraint and considers some delay optimizations. Both algorithms ensure to obtain a solution with $m{+}nzc{-}1$ operations, where m is the number of operations found by the MCM algorithm in the first part and nzc is the number of nonzero constants of the CAVM block.

For our CAVM example in Fig. 2(a), ECHO-A finds a solution with 4 operations and 4 adder-steps (Fig. 4(a)) that was obtained based on the MCM solution in Fig. 3(b). Also, the solution of *echo-d* includes 5 operations and 3 adder-steps (Fig. 4(b)) that was obtained based on the MCM solution in Fig. 3(c). This example shows the direct impact of the MCM solution on the number of operations and adder-steps of the CAVM design.

C. Filter Design Optimization

The zero-phase frequency response of a symmetric FIR filter is given as[3]:

$$G(w) = \sum_{i=0}^{\lfloor M\rfloor} d_i h_i cos\left(w(M-i)\right)$$

where $M = (N-1)/2$ and $d_i = 2 - K_{i,M}$ with $K_{i,M}$ is the Kronecker delta,[4] $h_i \in \mathbb{R}$ with $-1 \le h_i \le 1$, and $w \in \mathbb{R}$

[3]The frequency response of an asymmetric filter can be found in [36].

[4]The $K_{a,b}$ function is 1 when a is equal to b. Otherwise, it is 0.

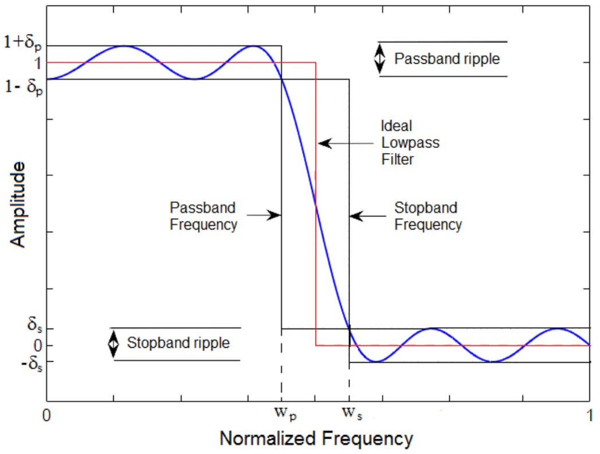

Fig. 5. Zero-phase frequency response of a low-pass FIR filter.

is the angular frequency. Considering a low-pass FIR filter as illustrated in Fig. 5 and assuming that the desired pass-band and stop-band gains are equal to 1 and 0, respectively, the filter must satisfy the following constraints [22]:

$$\begin{aligned} 1-\delta_p \le G(w) \le 1+\delta_p, &\quad w\in[0,w_p] \\ -\delta_s \le G(w) \le \delta_s, &\quad w\in[w_s,\pi] \end{aligned} \tag{3}$$

The pass-band gain is not relevant for many DSP applications and can be compensated in the filter design. Thus, a scaling factor (s) can be added into the filter constraints as a continuous variable as follows [25], [26]:

$$\begin{aligned} s(1-\delta_p) \le G(w) \le s(1+\delta_p), &\quad w\in[0,w_p] \\ s(-\delta_s) \le G(w) \le s(\delta_s), &\quad w\in[w_s,\pi] \\ s^l \le s \le s^u & \end{aligned} \tag{4}$$

where s^l and s^u are respectively the lower and upper bounds of s. Furthermore, in some DSP applications, it is desirable to minimize the peak weighted ripple [18], the normalized peak ripple (NPR) [28], [29], or the NPR magnitude [27].

A straightforward filter design technique (SFDT) follows two steps: i) given *fspec*, the filter coefficients, that respect the filter constraints, are found using a filter design method, such as windowing [37], McClellan-Parks-Rabiner algorithm [38], or linear programming [39]; ii) the multiplier block of the FIR filter is realized using the minimum number of adders/subtractors as described in Section II-B.

Since there exist many possible sets of coefficients satisfying the filter constraints, FDO algorithms incorporate sophisticated techniques such as local search [18], [24], [27] and exhaustive search methods, including branch-and-bound [22], [23], [25], DFS [28], [29], and MILP [17], [19], [21], [24], [26] techniques. The local search methods can be applied to filters with a large number of coefficients, but the optimal solution cannot be ensured, since the entire search space is not explored. The exhaustive search methods can only be run on filters with a small number of coefficients due to the exponential growth of the search space. Their runtime complexity can be reduced when the number of possible values of coefficients is limited [22], [28].

To reduce the complexity of the filter design, the algorithms of [17]–[24] search for coefficients with the fewest nonzero digits, since a coefficient represented with a few nonzero digits requires a small number of operations. The algorithms of [22], [24] find the common partial products using one of algorithms described in Section II-B after a solution is obtained. However, since the sharing of partial products is not considered during the search of coefficients, these methods may yield filters with a large number of operations as shown in Section IV-A.

The methods of [25]–[29] search for coefficients that exploit the partial product sharing. In [25], all possible values of coefficients are explored using a branch-and-bound algorithm and a CSE heuristic is used to share the common subexpressions among the coefficient multiplications. The algorithm of [26] uses an MILP method to consider all possible coefficients satisfying the filter constraints and finds coefficients that include the most common nonzero digits. The method of [27] finds the coefficients that include the most 101 and $10\overline{1}$ digit patterns (subexpressions) and satisfy the filter constraints. The method of [28] uses a subexpression basis set that is dynamically expanded as coefficients are synthesized at each depth of its search tree. The technique of [29] is based on the method of [28], but explores the entire search space under a given quantization value. It is able to be aware of whether an optimum solution is obtained.

III. Exact and Approximate FDO Algorithms

The following two subsections present SIREN and NAIAD, targeting the filter constraints given for a symmetric filter in (4), the transposed form of the FIR filter, and the optimization of the number of operations without a delay constraint. The last subsection describes the modifications required to target different filter constraints, the direct form of the FIR filter, and the optimization of the number of operations under a delay constraint in the multiplier blocks of filter forms.

A. SIREN: An Exact FDO Algorithm

SIREN was developed to find a set of filter coefficients yielding a minimum number of adders/subtractors in the filter design and satisfying the filter constraints. Its pseudo-code is given in Algorithm 1. It takes the five-tuple *fspec* denoting the filter specifications as input and returns a set of fixed-point coefficients *sol*. In Algorithm 1, Q stands for the quantization value used to convert floating-point numbers to integers. SIREN will be described in detail using a symmetric FIR filter with *fspec* (8, 0.2π, 0.7π, 0.01, 0.01) as an example.

Algorithm 1: The SIREN algorithm

```
SIREN(fspec)
1: $Q = 0$, $sol = \{\ \}$
2: $(h^l, h^u, s^l, s^u) = \mathrm{ComputeBounds}\ (fspec)$
3: $O = \mathrm{OrderCoefs}(h^l, h^u)$
4: repeat
5:    $Q = Q + 1$, $H^l = \lceil (h^l \cdot 2^Q) \rceil$, $H^u = \lfloor (h^u \cdot 2^Q) \rfloor$
6:    if CheckValidity($H^l$, $H^u$) then
7:       $sol = \mathrm{DFS}(fspec, O, Q, H^l, H^u, s^l, s^u)$
8: until $sol \neq \emptyset$
9: return $sol$
```

First, to restrict the search space, we find the lower and upper bounds of coefficients and scale factor s using the *ComputeBounds* function. To find the lower bounds of coefficients, is solved for each coefficient:

$$\begin{aligned} minimize: &\quad f = h_i \\ subject\ to: &\ s(1-\delta_p) \le G(w) \le s(1+\delta_p), \quad w \in [0, w_p] \\ &-s(\delta_s) \le G(w) \le s(\delta_s), \qquad w \in [w_s, \pi] \\ &\quad h^l \le h \le h^u \\ &\quad s^l \le s \le s^u \end{aligned}$$

where h^l and h^u denote the lower and upper bounds of all filter coefficients which were initially assigned to -1 and 1, respectively and the lower and upper bounds of s, s^l and s^u, were initially set to 0.01 and 100, respectively. The value of h_i in the LP solution corresponds to its lower bound h_i^l and is stored in h^l. In a similar way, the upper bound of each coefficient h_i^u is found when the cost function is $f{=}{-}h_i$ and is stored in h^u. Thus, the sets h^l and h^u consist of the floating-point lower and upper bounds of coefficients, respectively. The values of s^l and s^u are found similarly. For symmetric filters, the number of LP problems to be solved is $2\lfloor M \rfloor + 4$. Recall that an LP problem can be solved in polynomial time [33].

For our example, the floating-point lower and upper bounds of filter coefficients are computed as $h^l = \{h_0^l, h_1^l, h_2^l, h_3^l\} = \{-0.0966, -0.0915, 0.0015, 0.0039\}$ and $h^u = \{h_0^u, h_1^u, h_2^u, h_3^u\} = \{-0.0003, -0.0002, 0.4144, 1\}$, respectively. Also, s^l and s^u are 0.01 and 2.53, respectively.

Second, the *OrderCoefs* function finds an ordering of coefficients to be used in its DFS method while constructing the search tree (described ahead). We sort the coefficients in ascending order according to their $h_i^u - h_i^l$ values and store their indices i in this order in O. The reason behind finding such an ordering is that if coefficients with narrower upper and lower bound intervals are placed in lower depths of the search tree, fewer decisions are made and conflicts occur earlier. Thus, the runtime of SIREN can be reduced significantly, still exploring all possible values of coefficients. For our example, the ordering of coefficients is $O = \{1, 0, 2, 3\}$.

Third, starting with the quantization value Q equal to 1, the floating-point lower (upper) bound of each coefficient is multiplied by 2^Q, rounded to the smallest following (the largest previous) integer, and is stored in H^l (H^u). The validity of these sets H^l and H^u is tested by the *CheckValidity* function by simply checking each coefficient if H_i^l is less than or equal to H_i^u. If they are not valid, this function returns zero. In this case, Q is increased by one, H^l and H^u are updated, and the *CheckValidity* function is applied again. Otherwise, the DFS method, that explores all possible values of each coefficient in between H_i^l and H_i^u, is applied to find a set of filter coefficients which respects the filter constraints and yields the minimum design complexity, or to prove that there exists no such a set of filter coefficients. If the former condition occurs, sol is returned. If the latter condition occurs, Q is increased by one, H^l and H^u are updated, and the DFS method is applied again. Hence, SIREN ensures that its solution is a set of fixed-point filter coefficients obtained using the smallest Q value.

Note that Q is an important parameter in the filter design. When Q increases, the bitwidths (sizes) of coefficients increase. Thus, such coefficients lead to larger sizes of registers and structural adders in the register-add block of the transposed form (Fig. 1(b)). Also, most probably, they lead to a large number of operations in the multiplier blocks of both forms (Fig. 1). Similar to Q, the solution quality of an FDO algorithm is evaluated by the effective wordlength (EWL) of a set of coefficients [22], [28], [29], computed as $max\{\lceil log_2|h_i|\rceil\}$ with $0{\le}i{\le}N{-}1$ when fixed-point coefficients are considered.

In the DFS method of SIREN, the search tree is constructed based on the ordering of coefficients O, where a vertex at depth d, V_d, denotes the filter coefficient whose index is the dth element of O, i.e., $h_{O(d)}$. An edge at depth d of the search tree, i.e., a fanout of V_d, stands for an assignment to the vertex V_d from $[V_d^l, V_d^u]$ where V_d^l (V_d^u) denotes the lower (upper) bound of V_d. Note that the values of the vertex at depth d are assigned incrementally starting from V_d^l to V_d^u.

When d is 1, the DFS method assigns $H_{O(1)}^l$ and $H_{O(1)}^u$ to V_1^l and V_1^u, respectively and sets the value of the vertex V_1 to V_1^l. At any depth greater than 1, $d > 1$, although the lower and upper bounds of a vertex can be taken from H^l and H^u, respectively, tighter lower and upper bounds can be computed, since the values of $d-1$ coefficients are determined and fixed. The lower bound of the vertex V_d is computed by solving the following LP problem, where the non-determined coefficients and s are the continuous variables of the LP problem.

$$\begin{aligned} minimize: &\ f = h_{O(d)} \\ sbj.to: &\ s(1-\delta_p) \le G(w)/2^Q \le s(1+\delta_p), \quad w \in [0, w_p] \\ &-s(\delta_s) \le G(w)/2^Q \le s(\delta_s), \qquad w \in [w_s, \pi] \\ &\quad H_i^l \le h_i \le H_i^u, \quad i \in [O(d), O(\lfloor M \rfloor + 1)] \\ &\quad s^l \le s \le s^u \\ h_{O(1)} \ldots h_{O(d-1)} &: determined \end{aligned}$$

In this LP problem, the lower and upper bounds of all non-determined coefficients are taken from H^l and H^u, respectively. The upper bound of V_d is computed when the cost function is changed to $f{=}{-}h_{O(d)}$. If there exist feasible solutions for both LP problems, this lower (upper) bound is rounded to the smallest following (the largest previous) integer and assigned to V_d^l (V_d^u). If $V_d^u \ge V_d^l$, they are determined to be the lower and upper bounds of V_d. Whenever there is no feasible lower or upper bound for V_d or $V_d^u < V_d^l$, the search is backtracked chronologically to the previous vertex until there is a value to be assigned among its lower and upper bounds.

When the values of all coefficients are determined, i.e., the leaf at the final depth of the search tree is reached (when d is $\lfloor M \rfloor{+}1$ for symmetric filters), the implementation cost of the transposed form filter is computed as $TA{=}MA{+}SA$, where TA is the total number of operations in the filter, and MA and SA are the number of operations in the MCM block and the number of structural adders in the register-add block, respectively (Fig. 1(b)). While MA is found using the exact MCM method [9], SA is computed based on the nonzero coefficients. No adder is needed for a coefficient equal to 0 in the register-add

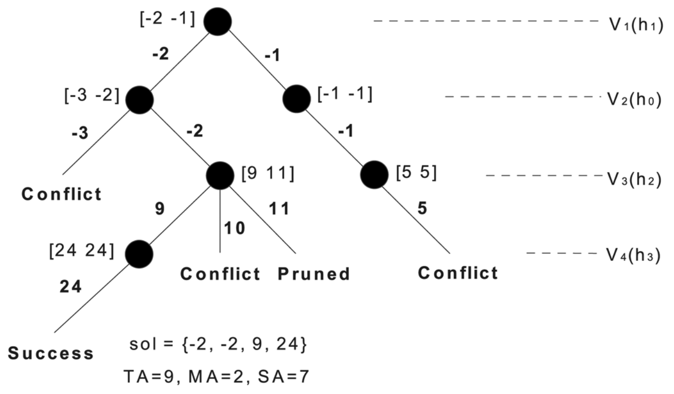

Fig. 6. Search tree formed by the DFS method.

block. This coefficient set is stored in *sol* if its TA value is smaller than that of the best one found so far which was set to infinity in the beginning of the DFS method.

To prune the search tree, the TA value is estimated when depth d is greater than $2M/3$ for symmetric filters. This value was chosen to be close to the bottom of the search tree not to waste an effort for computing an estimate that usually does not yield a backtrack. To compute this estimation, the lower bound on MA is found using the determined coefficients [40]. The lower bound on SA is found after all non-determined coefficients are set to a value. To do so, the upper and lower bound interval of each non-determined coefficient is checked if 0 is included. If so, this non-determined coefficient is set to 0. Otherwise, it is assumed to be a constant different from 0.

The DFS method terminates when all possible values of coefficients have been explored. If *sol* is empty, it guarantees that there is no set of filter coefficients which can be selected from their quantized lower and upper bounds respecting the filter constraints. Otherwise, *sol* consists of fixed-point coefficients that lead to a filter with minimum design complexity, satisfying the filter constraints.

For our example, when Q is 5, the quantized lower and upper bounds of filter coefficients are $H^l = \{-3,-2,1,1\}$ and $H^u = \{-1,-1,13,32\}$, respectively. Note that no solution was found with $Q < 5$. The search tree constructed by the DFS method when Q is 5 is shown in Fig. 6, where a and b in $[a\ b]$ given next to each vertex stand respectively for its lower and upper bounds which are dynamically computed as coefficients are fixed. In this figure, the actual traverse of the DFS method on filter coefficients can be followed from top to bottom and from left to right. Also, *Conflict* denotes that given determined coefficients, there exists no feasible lower/upper bound for the current depth vertex. *Pruned* indicates that the set of determined coefficients cannot lead to a better solution than the best one found so far. *Success* presents that the set of coefficients leads to a solution satisfying the filter constraints.

Observe that as the values of coefficients are determined, the intervals between the lower and upper bounds of coefficients are reduced when compared to those in the original H^l and H^u. If the DFS method was not equipped with techniques, that order the filter coefficients, determine the lower and upper bounds of coefficients dynamically, and prune the search space, in the worst case, it would consider $\prod_{i=0}^{\lfloor M \rfloor}(H_i^u - H_i^l + 1)$ possible sets of coefficients for a symmetric filter, i.e., the number of leafs at the final depth of the search tree, in each iteration of SIREN. For our example, this value is 2496 when Q is 5. However, the DFS method ensures the minimum solution with only 1 leaf at the final depth and 10 branches.

The performance of SIREN depends heavily on the minimum quantization Q value, the filter length N, and the exact MCM algorithm [9]. The Q value has an impact on the number of runs of the DFS method, the lower and upper bounds of coefficients, the number of branches in the search tree, and the sizes of coefficients which affect the performance of the exact MCM algorithm [9]. The N value has an effect on the performance of the exact MCM algorithm and on the depth of the search tree. The performance of the exact MCM algorithm is related to the number and size of coefficients [9]. To increase the performance of SIREN, a parallel version of the DFS method can be developed. Thus, the whole search space can be divided into many small parts and they can be explored in a reasonable time simultaneously. However, the FDO problem is NP-complete [29], and hence, heuristics are indispensable for filters with a large number of coefficients.

B. NAIAD: An Approximate FDO Algorithm

NAIAD was developed based on two observations: i) given filter specifications, finding a set of floating-point coefficients, that satisfies the filter constraints, takes a polynomial time; ii) given a set of coefficients, finding a multiplierless design of coefficient multiplications including a number of adders/subtractors very close to the minimum can be done in a reasonable time [3], [9]. Hence, NAIAD consists of two main parts: i) exploring sets of coefficients that satisfy the filter constraints and finding the ones with the smallest EWL value; ii) exploring the search area in the neighborhood of each solution obtained in the first part and finding the one that leads to the minimum design complexity. In following, these two parts are described in detail using a symmetric FIR filter with *fspec* $(11, 0.2\pi, 0.5\pi, 0.05, 0.05)$ as an example.

1) Exploring Coefficients Satisfying Filter Constraints: To explore possible sets of coefficients, which satisfy the filter constraints, in a systematic way, the variable ϵ is included into the left and right sides of filter constraints of (4). To find its lower bound, ϵ^l, the following LP problem is solved:

$$\begin{aligned}
&minimize: f = \epsilon \\
&subject\ to: s(1-\delta_p)-\epsilon \leq G(w) \leq s(1+\delta_p)+\epsilon, \quad w \in [0, w_p] \\
&\qquad\qquad s(-\delta_s)-\epsilon \leq G(w) \leq s(\delta_s)+\epsilon, \quad w \in [w_s, \pi] \\
&\qquad\qquad h^l \leq h \leq h^u \\
&\qquad\qquad s^l \leq s \leq s^u
\end{aligned}$$

where the filter coefficients h, the scale factor s, and ϵ are the continuous variables. The initial values of h^l, h^u, s^l, and s^u are -1, 1, 0.01, and 100, respectively. The upper bound of ϵ, ϵ^u, respecting the filter constraints of (4), is naturally equal to 0. We find the lower and upper bounds of each coefficient, h_i^l and h_i^u, when the cost function is $f{=}h_1$ and $f{=}-h_i$, respectively, and similarly, the lower and upper bounds of s.

For our example, ϵ^l, s^l and s^u are -0.086, 0.01, and 3.36, respectively. The lower and upper bounds of coefficients are $h^l=\{h_0^l,h_1^l,h_2^l,h_3^l,h_4^l,h_5^l\}=\{-0.1694, -0.1945, -0.1174, 0.0009, 0.0024, 0.0029\}$ and $h^u=\{h_0^u,h_1^u,h_2^u,h_3^u,h_4^u,h_5^u\}=\{0.0215, -0.0001, 0.1177, 0.4842, 0.9149, 1\}$, respectively.

Then, in an iterative loop, ϵ^l is increased in steps of $|\epsilon^l/r|$ to 0, where r is a predetermined integer search parameter that denotes the number of samples to be taken from the search space. A set of floating-point filter coefficients is obtained by solving the following LP problem.

$$\begin{aligned} s(1-\delta_p)-\epsilon \le G(w) &\le s(1+\delta_p)+\epsilon, \quad w\in[0,w_p]\\ s(-\delta_s)-\epsilon \le G(w) &\le s(\delta_s)+\epsilon, \quad w\in[w_s,\pi]\\ h^l \le h &\le h^u\\ s^l \le s &\le s^u\\ \epsilon^l \le \epsilon &\le 0 \end{aligned} \tag{5}$$

Starting with a quantization value Q equal to 1, the found coefficients are converted to integers as $round(h_i2^Q)$ and are checked if these fixed-point coefficients still satisfy the original filter constraints of (4) when they are converted to floating-point numbers as $round(h_i2^Q)/2^Q$. If not, Q is increased by 1. Thus, the coefficients are converted to integers using the minimum Q considering the quantization error. The EWL values of these sets of fixed-point coefficients are computed and the ones with the minimum EWL value are stored in a set, called $ISPset$, that will include initial search points (ISPs).

Note that having a coefficient equal to 0 can significantly reduce the filter design complexity (Fig. 1). Hence, for each ϵ^l value in the iterative loop, the LP problem of (5) is also solved when each coefficient h_i with $h_i^l\le 0$ and $h_i^u\ge 0$ is set to 0 by simply assigning 0 to its lower and upper bounds in the LP problem. For our example, they are h_0 and h_2. Similarly, the sets of coefficients are quantized to integers using the minimum Q value, their EWL values are computed, and the ones with the minimum EWL value are stored in $ISPset$.

Then, the cost of each element of $ISPset$ is computed in terms of TA as described in the previous subsection. To find the MA value, rather than the exact MCM algorithm [9], an efficient MCM heuristic [3] is used. Among all these ISPs, the minimum value of TA is found and is denoted as the best cost value found so far (BC). An ISP with the BC value is determined as the best solution (BS).

Note that r has a significant impact on the solution quality. While a small r value leads to a few possible solutions with a high EWL value, a large r value yields a large number of possible solutions with a small EWL value. In NAIAD, it was set to 5000 which was found empirically. This means that the number of LP problems need to be solved in this first part is $5001(b+1)$, where b denotes the number of filter coefficients of which 0 is in between their lower and upper bounds.

For our example, $ISPset$ has 3 elements, $\{-1,-2,0,5,13,16\}$, $\{-1,-1,0,4,9,11\}$, and $\{-1,-2,-1,5,12,15\}$, with TA values equal to 10, 10, and 13, respectively and an EWL value equal to 4. Thus, BS is the first element of $ISPset$ and BC is 10. Note that the first and second elements of $ISPset$, whose h_2 values are 0, have the smallest TA value.

Algorithm 2: The local search method of NAIAD

```
LSM($fspec$, $ISP$, $cost$, $Q$)
1: $bc = cost, bs = ISP$
2: loop
3:   repeat
4:     $w2repeat = 0$
5:     $O = \mathrm{GenerateAnOrdering}(\lfloor M \rfloor)$
6:     for $i = 1 to \lfloor M \rfloor + 1$ do
7:       $(h^l_{O(i)}, h^u_{O(i)}) = \mathrm{FindLUB}(O(i), fspec, ISP, Q)$
8:       for $c = \lceil h^l_{O(i)} \rceil to \lfloor h^u_{O(i)} \rfloor$ do
9:         if $c \neq ISP_{O(i)}$ then
10:          $NSP = ISP, NSP_{O(i)} = c$
11:          $impcost = \mathrm{ComputeImpCost}(NISP)$
12:          if $impcost < bc$ then
13:            $w2repeat = 1$
14:            $ISP_{O(i)} = c, bs = ISP, bc = impcost$
15:  until $w2repeat = 0$
16:  if Terminating conditions are not met then
17:    $NSP = \mathrm{ChangeCoefs}(fspec, ISP, Q)$
18:    if $NSP \neq ISP$
19:      $ISP = NSP$
20:      $impcost = \mathrm{ComputeImpCost}(ISP)$
21:      if $impcost \le bc$ then
22:        $bs = ISP, bc = impcost$
23:    else
24:      return $bs, bc$
25:  else
26:    return $bs, bc$
```

2) Exploring Search Area Around Coefficients: A local search method (LSM) is applied to each element of $ISPset$ (a set of coefficients denoted as ISP with its implementation cost value $cost$ and quantization value Q). Its pseudo-code is given in Algorithm 2, where bc and bs are respectively the best cost value in terms of TA and the best solution including the set of coefficients with bc. The bc and bs are initially set to $cost$ and ISP, respectively. The LSM function aims to explore the search area around ISP and to reduce the implementation cost of the filter design. To do that this function iteratively takes a coefficient, changes its value between its lower and upper bounds, finds the implementation cost of the new filter design every time, and keeps the one with the minimum cost.

Since the traversing order of filter coefficients affects the solution of the LSM function, in its iterative loop (lines 3–15), we determine an ordering of coefficients randomly using the *GenerateAnOrdering* function. Then, for a coefficient in the given order, $h_{O(i)}$ with $1\le i\le\lfloor M\rfloor+1$, using the *FindLUB* function, we compute its lower and upper bounds when the values of all coefficients except the $O(i)$th coefficient are determined as given in ISP. The following LP problem is generated to find

$\{h_0, h_1, h_2, h_3, h_4, h_5\}$ = {-1, -2, -1, 5, 12, 15} TA = 13
Considered coefficient: h_5, $\lceil h_5^l \rceil = 14$, $\lfloor h_5^u \rfloor = 16$
$\{h_0, h_1, h_2, h_3, h_4, h_5\}$ = {-1, -2, -1, 5, 12, $\boxed{16}$} TA = 12
Considered coefficient: h_2, $\lceil h_2^l \rceil = -1$, $\lfloor h_2^u \rfloor = 0$
$\{h_0, h_1, h_2, h_3, h_4, h_5\}$ = {-1, -2, $\boxed{0}$, 5, 12, 16} TA = 10
Considered coefficient: h_3, $\lceil h_3^l \rceil = 5$, $\lfloor h_3^u \rfloor = 6$
$\{h_0, h_1, h_2, h_3, h_4, h_5\}$ = {-1, -2, 0, $\boxed{6}$, 12, 16} TA = 9

Fig. 7. Changing the values of filter coefficients in the LSM function. The modified coefficients are shown in a box.

the lower bound of $h_{O(i)}$, $h_{O(i)}^l$, where only the $O(i)$th coefficient and s are continuous variables.

$$\begin{aligned} &minimize: f = h_i \\ &subject\ to: s(1-\delta_p) \leq G(w)/2^Q \leq s(1+\delta_p), \quad w \in [0, w_p] \\ &\qquad s(-\delta_s) \leq G(w)/2^Q \leq s(\delta_s), \quad w \in [w_s, \pi] \\ &\qquad s^l \leq s \leq s^u \\ &h_{O(1)}, \ldots, h_{O(i-1)}, h_{O(i+1)}, \ldots, h_{O(\lfloor M \rfloor+1)} : determined \end{aligned}$$

The upper bound of $h_{O(i)}$, $h_{O(i)}^u$, is found by changing the cost function to $f = -h_{O(i)}$. Then, for each possible fixed-point value of the $O(i)$th coefficient in between its lower and upper bounds, i.e., c, other than its value in ISP, we assign it to the $O(i)$th coefficient of a new search point NSP which was initially assigned to ISP. We find the implementation cost of NSP in terms of TA, $impcost$, using the *ComputeImpCost* function as described in the previous subsection. If $impcost$ is less than the best one found so far bc, then the $w2repeat$ variable is set to 1 and ISP, bs, and bc are updated. After all coefficients of ISP are traversed, if the $w2repeat$ variable is 1, we iterate this procedure once more, but with a different ordering of coefficients. Otherwise, it is decided that a local minima is reached. To escape from this local point, the *ChangeCoefs* function is applied, where -2, -1, 0, 1, or 2 (determined randomly) is added to the value of each coefficient in ISP. Note that the *ChangeCoefs* function can change the values of more than one coefficient simultaneously. If the new search point NSP satisfies the filter constraints, its implementation cost is computed and it is entered into the iterative loop again. Otherwise, the search is terminated. The local search algorithm has also two terminating conditions: i) the number of iterations of the iterative loop in the infinite loop (lines 2–26) is 30; and ii) the total number of runs of the MCM algorithm [3] is $40N$.

If the LSM function returns a solution with an implementation cost value bc better than BC, then BS and BC are replaced with its outputs bs and bc, respectively.

For our example, suppose that the third element of $ISPset$, i.e., {-1, -2, -1, 5, 12, 15} with a TA value 13, is given to the local search method. Fig. 7 shows how a solution with a better TA value is obtained by changing the values of coefficients. At the end of the assignments given in Fig. 7, a solution with a TA value equal to 9, i.e., {-1, -2, 0, 5, 12, 16}, is found. We note that this is the minimum solution that NAIAD could find. On this example, SIREN also finds a solution with TA and EWL values equal to 9 and 4, respectively.

C. Further Modifications in SIREN and NAIAD

To realize the MCM block of the transposed form with the minimum number of adder-steps, in SIREN and NAIAD, we respectively used the modified versions of the approximate algorithms of [9] and [3] that can handle the delay constraint. Whenever a set of fixed-point filter coefficients is determined in SIREN and NAIAD, the minimum adder-steps of coefficients is computed as given in Section II-B-1 and it is given to the algorithms of [9] and [3] as a delay constraint.

In order to target the direct form of the FIR filter, in SIREN and NAIAD, ECHO-A [15] is used to compute the smallest number of operations in the CAVM block and ECHO-D [15] is used for the design of the CAVM block with a small number of adder-steps. Note that in direct form filters, the total number of operations in the filter, i.e., TA, is determined by the solution of ECHO-A or ECHO-D on the set of filter coefficients.

The proposed methods can target different filter constraints. For example, when the lower and upper bounds of s, s^l and s^u, in (4) are set to 1, the filter constraints of (3) are aimed. Setting s^l and s^u respectively to 0.7 and 1.4 corresponds to the ±3 dB gain tolerance in the filter design [30]. The proposed algorithms can also target asymmetric filters taking into account the related filter constraints [36].

The proposed algorithms can target the optimization of the gate-level area of the filter design. In this case, whenever a set of coefficients is found, an algorithm [10], [15], that can find the shift-adds design of the multiplier block of the filter occupying minimum area, should be used. In the transposed form filter, the size of registers and adders in the register-add block should also be considered.

IV. Experimental Results

This section is divided in two subsections. In the first subsection, we explore the effectiveness of SIREN and NAIAD, comparing their results with those of prominent FDO algorithms and a straightforward filter design technique (SFDT). In the second subsection, we explore the impact of filter design parameters, such as filter length, quantization value, and filter form, on the filter design complexity, presenting the gate-level results of filter designs obtained based on the solutions of SIREN, NAIAD, and the algorithm of [29]. Note that SIREN and NAIAD were written in MATLAB, used lp_solve 5.5.2.0 as an LP solver, and were run on a PC with Intel Xeon at 2.33 GHz under Linux. The filter designs were described in VHDL and synthesized using the Synopsys Design Compiler with the UMCLogic 180 nm Generic II library when the bitwidth of the filter input bwi was 16. In the synthesis script, relaxed timing constraints were used in order to provide more freedom to the synthesis tool to optimize area. The functionality of filters was verified on 10,000 randomly generated input signals in simulation, from which the switching activity information, that was used by the synthesis tool to compute the power dissipation, was obtained. Note also that we did not utilize any truncation method [41], which is generally used to reduce the complexity of the filter design sacrificing the accuracy of the result, neither on the filter output nor on any register/adder. Thus, the sizes of filter input and coefficients have a direct impact on the complexity of the filter design.

Unless stated otherwise, it should be accepted that the results of SIREN and NAIAD were found when they targeted the constraints in (4), the transposed form, and the minimization of the number of operations without a delay constraint.

A. Comparisons on FDO Algorithms

Table I shows the specifications of 10 symmetric FIR filters which are commonly used in evaluation of FDO algorithms.

TABLE I
SPECIFICATIONS OF SYMMETRIC FIR FILTERS

Filter	Type	N	w_p	w_s	δ_p	δ_s
X1	Low-pass	15	0.2π	0.8π	0.0001	0.0001
G1	Low-pass	16	0.2π	0.5π	0.01	0.01
Y1	Low-pass	30	0.3π	0.5π	0.00316	0.00316
Y2	Low-pass	38	0.3π	0.5π	0.001	0.001
A	Low-pass	59	0.125π	0.225π	0.01	0.001
S2	Low-pass	60	0.042π	0.14π	0.012	0.001
L2	Low-pass	63	0.2π	0.28π	0.028	0.001
B	Low-pass	105	0.2π	0.24π	0.01	0.01
L1	High-pass	121	0.8π	0.74π	0.0057	0.0001
C	Low-pass	325	0.125π	0.14π	0.005	0.005

TABLE II
SUMMARY OF FDO ALGORITHMS ON FIR FILTERS OF TABLE I

Filter	Method	EWL	MA	SA	TA	BST	TT
X1	[27]	13	7	8	15	–	–
	NAIAD	10	5	8	13	1m3s	1m3s
	SIREN	10	5	8	13	<1s	2s
	[29]	10	5	8	13	–	<1s
G1	[26]	7	2	13	15	–	–
	NAIAD	6	3	15	18	50s	50s
	[29]	6	2	15	17	–	<1s
	SIREN	6	2	15	17	<1s	<1s
Y1	[28]	10	6	23	29	–	21m30s
	NAIAD	9	7	23	30	5m55s	6m3s
	[29]	9	7	23	30	–	6s
	SIREN	9	6	23	29	2m17s	7m56s
Y2	[25]	12	–	–	39	–	–
	NAIAD	11	9	29	38	15m21s	19m18s
	[29]	10	10	37	47	–	11s
	SIREN	10	9	29	38	3m52s	4m29s
A	[22]	10	18	58	76	3h2m	4h14m
	NAIAD	10	16	56	72	43m34s	44m13s
	[29]	10	14	54	68	–	2d2h
	SIREN	10	16	52	68	14h57m	2d
S2	[22]	11	27	59	86	23m	27m
	[29]	10	17	59	76	–	16h42m
	NAIAD	10	15	57	72	49m19s	1h10s
	SIREN	9	14	57	71	16h4m	2d
L2	NAIAD	11	17	60	77	55m1s	1h4m
	[22]	10	18	62	80	26m	54m
	SIREN	10	16	60	76	1d23h	2d
	[29]	10	17	56	73	–	16h28m
B	[24]	9	10	99	109	<1s	1m7s
	[22]	8	11	100	111	9m	1d
	NAIAD	8	13	96	109	2h2m	2h2m
L1	[19]	15	59	120	179	–	–
	NAIAD	15	40	116	156	2h18m	3h48m
	[22]	14	47	120	167	32m	56m
C	[20]	11	43	322	365	20h46m	1d
	[22]	10	22	306	328	13h47m	1d
	NAIAD	10	25	286	311	2h51m	4h

Table II presents the results of SIREN, NAIAD, and other algorithms whose results were taken from [22], [24], [29] as reported. In this table, BST and TT denote respectively the CPU time required to find the best solution and the total CPU time. For each filter, the FDO methods were sorted according to their results on i) EWL, ii) TA, iii) TT, and iv) BST in descending order. Note that the CPU time limit for SIREN and NAIAD was 2 days and 4 hours, respectively.

On filters including less than 40 coefficients, X1, G1, Y1, and Y2, SIREN finds a solution with the minimum number of operations and minimum EWL value using little computational resources. NAIAD obtains solutions very close to the minimum in terms of EWL and TA. However, it requires more CPU time than SIREN, especially on filter Y2, for which, many ISPs were considered. This experiment indicates that the previously proposed FDO algorithms obtain solutions with the same values of EWL and TA as those of SIREN or very close to them. On filters including around 60 coefficients, A, S2, and L2, although SIREN cannot ensure the minimum TA value under the minimum quantization value due to its CPU time limit, it can find a better solution than other FDO algorithms on filter S2. Also, NAIAD can complete the search in a reasonable time and obtain better solutions than the algorithm of [22], which was also applied to filters including more than 100 coefficients, except on filter L2 where NAIAD finds a solution with a higher EWL value. SIREN was not applied to filters including more than 100 coefficients, B, L1, and C. Instead, NAIAD finds solutions with equal or less number of operations than other FDO methods and obtains a solution with a higher EWL value only on filter L1.

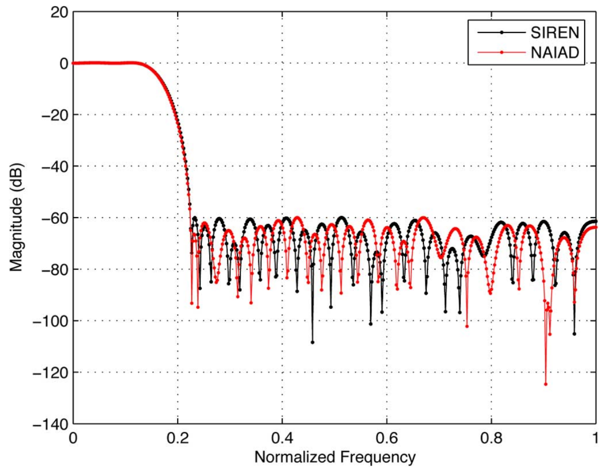

Fig. 8. Zero-phase frequency responses of the filter A of Table I.

Fig. 8 presents the zero-phase frequency responses of filter A of Table I based on the coefficients determined by SIREN and NAIAD. Observe from Fig. 8 and Table II that while the solutions of both algorithms satisfy the filter constraints, SIREN needs fewer operations than NAIAD.

To further analyze the proposed algorithms, we used randomly generated 21 symmetric low-pass FIR filters whose N values range between 20 and 40. Fig. 9 presents the results of SIREN, NAIAD, and SFDT in terms of EWL, TA, and CPU time in seconds. In SFDT, given *fspec*, the filter coefficients were computed using the *firgr* function of MATLAB and were quantized to integers with the minimum Q value determined incrementally. Then, the minimum number of operations in the MCM block of the filter was found by the exact algorithm [12].

Observe from Fig. 9(a) that SIREN finds a set of coefficients with the smallest EWL value and the solutions of NAIAD have EWL values equal or very close to those of SIREN. On these instances, SFDT obtains solutions with the largest EWL values. Observe from Fig. 9(b) that SIREN finds a solution with the smallest number of operations, except on filters with 25, 28, 31, and 34 coefficients. Recall that SIREN guarantees a solution with the minimum number of operations under the minimum Q value. Hence, on these instances using a higher Q value than the minimum, NAIAD can find a solution with fewer operations than SIREN, but with a higher EWL value. Note that NAIAD obtains solutions in terms of TA very close to SIREN, i.e., 0.95

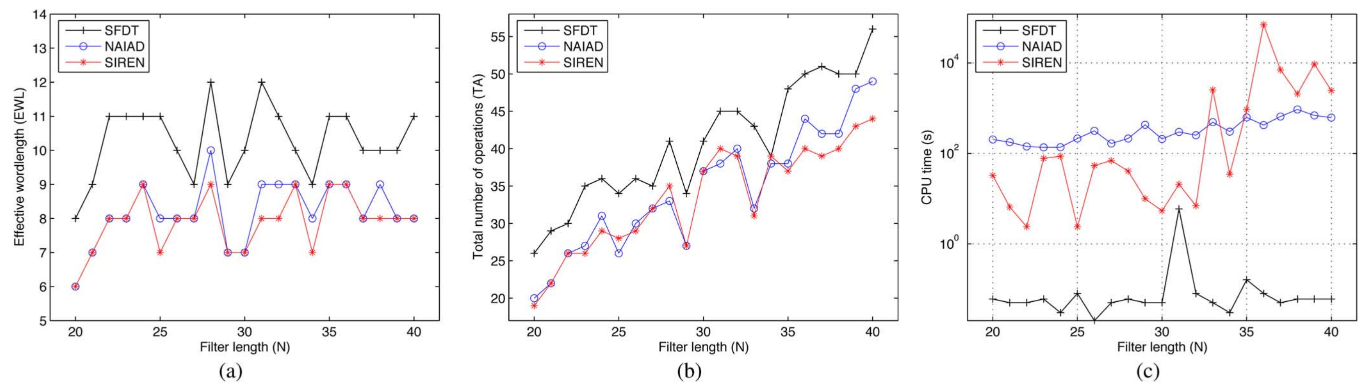

Fig. 9. Results of SIREN, NAIAD, and SFDT on randomly generated filters: (a) EWL; (b) TA; (c) CPU time in log scale.

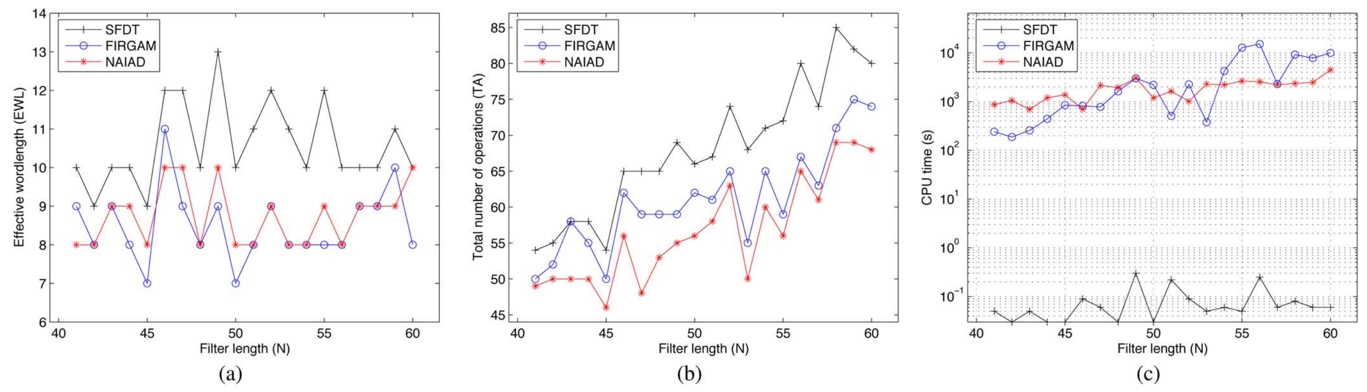

Fig. 10. Results of NAIAD, FIRGAM [22], and SFDT on randomly generated filters: (a) EWL; (b) TA; (c) CPU time in log scale.

more adders/subtractors on average. On these instances, SFDT yields sets of coefficients requiring the largest number of operations. Observe from Fig. 9(c) that as N increases, the run-time of SIREN increases dramatically and the run-time of NAIAD increases slightly. On these instances, SFDT uses a little computational effort to find a solution.

Moreover, Fig. 10 presents the results of NAIAD, FIRGAM [22], and SFDT on randomly generated 20 low-pass symmetric FIR filters, where the N value ranges between 41 and 60. In FIRGAM, the objective was to minimize the number of nonzero digits in the coefficients. FIRGAM was run on a PC with Intel Core i5-2410M at 2.3 GHz under Windows 7.

Observe from Fig. 10 that while the solutions of SFDT have higher EWL values than NAIAD and FIRGAM, the solutions of NAIAD and FIRGAM have 8.75 and 8.5 EWL values on average, respectively. NAIAD finds a solution requiring less number of total operations than both FIRGAM and SFDT. We note that while the difference of average TA values between the solutions of FIRGAM and NAIAD is 4.45, this value between the solutions of SFDT and NAIAD is 11.5. However, SFDT obtains a solution using the least CPU time and NAIAD requires less CPU time than FIRGAM on average.

To explore the effectiveness of the local search method of NAIAD, we developed the local search method of the FDO algorithm POTx [24]. Given a set of fixed-point filter coefficients, it aims to reduce the total number of power-of-two (POT) terms in coefficients, still satisfying the filter constraints. Thus, in our version of POTx, we first find the possible sets of fixed-point coefficients with the minimum EWL value as done in the first part of NAIAD. Then, we apply the local search method of POTx to

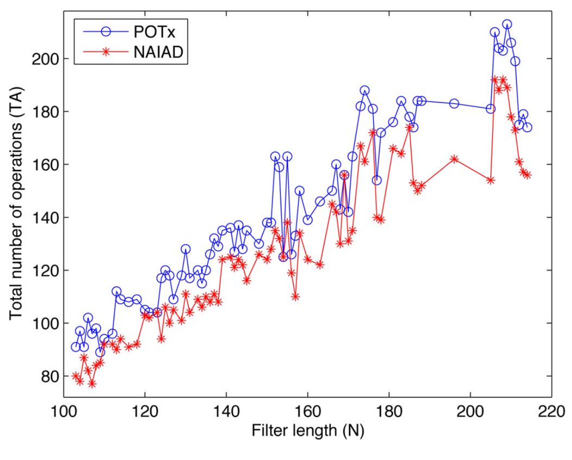

Fig. 11. TA values found by our version of POTx and NAIAD.

only one set of coefficients. Finally, we apply the MCM method [3] to the final set of coefficients to maximize the sharing of partial products in the MCM block of the filter. In this experiment, we used randomly generated 75 low-pass symmetric FIR filters whose filter lengths range between 103 and 214. Fig. 11 and 12 present the total number of operations in the filter designs obtained by our version of POTx and NAIAD and the CPU time of the local search methods in POTx and NAIAD, respectively.

Observe from Fig. 11 that NAIAD always yields FIR filter designs with the same or less number of operations than our version of POTx. On average, it leads to filters including less than 15.3 operations than our version of POTx. Observe from Fig. 12 that the local search method of POTx generally requires

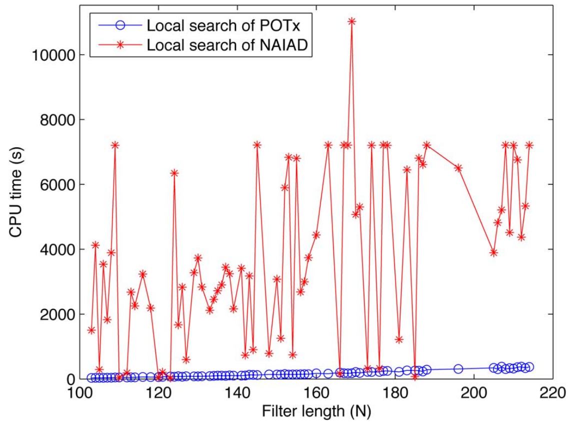

Fig. 12. CPU time of the local search methods in POTx and NAIAD.

TABLE III
SUMMARY OF RESULTS OF SIREN ON FILTER Y1 OF TABLE I

N	Q	MA	SA	TA	CA	NCA	A	D	P
28	9	9	21	30	18.2	11.4	29.6	6.4	3.2
29	9	7	24	31	18.2	11.6	29.7	6.1	3.1
30	9	6	23	29	17.5	12.1	29.6	5.7	3.2
31	9	7	24	31	18.2	12.0	30.2	6.0	3.1
32	8	7	29	36	21.1	12.5	33.5	6.3	3.5

TABLE IV
SUMMARY OF RESULTS OF SIREN ON FILTER Y2 OF TABLE I

N	Q	MA	SA	TA	CA	NCA	A	D	P
34	11	10	27	37	23.2	15.1	38.3	6.8	4.2
35	11	12	28	40	25.0	15.0	40.0	7.4	4.4
36	11	10	27	37	23.3	15.3	38.6	7.1	3.9
37	10	12	36	48	30.0	15.6	45.6	7.4	4.6
38	10	9	29	38	23.5	16.1	39.6	6.6	4.4
39	10	11	32	43	26.4	16.3	42.7	7.1	4.4

less CPU time than the local search method of NAIAD. These two observations are due to the fact that the local search method of NAIAD may be applied to more than one possible set of fixed-point filter coefficients and targets the optimization of the number of operations.

B. Explorations on Filter Design Parameters

To explore the impact of the filter length N on the filter complexity, we consider the filters Y1 and Y2 of Table I. Table III presents the results of SIREN on filter Y1 when N is in between 28 and 32. Table IV shows the results of SIREN on filter Y2 when N is in between 34 and 39. Note that the minimum value of N, i.e., 28 for filter Y1 and 34 for filter Y2, was found by decrementing N by 1 at each time. These tables also present the gate-level results of filter designs, where CA, NCA, and A denote the combinational area, non-combinational area, and total area, all in mm^2, respectively, D stands for the critical path delay in ns, and P is the total dynamic power dissipation in mW. Note that the EWL value of each set of coefficients is the same as the Q value and the number of coefficients equal to 0 can be computed as $N-SA-1$.

Observe from Tables III and IV that as N increases, the Q value decreases, reducing the bitwidth of filter coefficients, which can also yield reductions on the size of structural adders and on the number of operations in the MCM block (although that also depends on the filter coefficients). However, the number of registers and structural adders is increased in this case. Hence, due to this tradeoff and because the filter design complexity in terms of TA is also related to the explored search space (the search space of the FDO problem differs under different N values), there is no clear evidence that a smaller N value always leads to a filter design requiring the smallest number of operations or occupying the smallest area. This observation is also true for delay and power dissipation of the filter design. Thus, it is useful to design a filter with different N values and choose the one that fits best in an application.

TABLE V
SUMMARY OF RESULTS OF SIREN ON FILTERS G1 AND Y1 OF TABLE I

Filter	Q	MA	SA	TA	CA	NCA	A	D	P
G1	6	2	15	17	9.1	5.6	14.7	4.9	1.5
	7	2	13	15	8.1	5.9	14.0	5.4	1.5
Y1	9	6	23	29	17.5	12.1	29.6	5.7	3.2
	10	6	23	29	18.1	12.8	30.9	6.9	3.3

To explore the impact of the quantization value Q on the filter complexity, Table V presents the results of SIREN on the filter G1 of Table I when Q is 6 and 7 and on the filter Y1 of Table I when Q is 9 and 10. Note that the EWL value of each set of coefficients is the same as the Q value and the number of coefficients equal to 0 can be computed as $N-SA-1$.

The results on filter G1 clearly indicate that a filter with coefficients having a smaller EWL value does not always yield a filter design occupying smaller area. In this case, it is because of fewer structural adders in the register-add block, that reduces the *CA* value, even though as EWL increases, the size of registers increases, increasing the *NCA* values. However, the results on filter Y1, where the total number of operations is the same in filters with Q values 9 and 10, show that an increase in Q increases the bitwidths of coefficients, increasing the size of adders/subtractors and registers. This can be observed from the CA and NCA values. In this case, the delay and power dissipation of the filter design also increase.

To explore the impact of the filter form, the number of adder-steps in the multiplier block of the filter, and the solution quality of an FDO algorithm on the filter design complexity, Table VI presents the solutions of the algorithm of [29], NAIAD, and SIREN on the filters A and S2 of Table I. Note that the solutions of the algorithm [29] (sets of fixed-point filter coefficients) were taken from [29] and the FIR filters were designed after the shift-adds realization of the multiplier block is found using the algorithms which were also used in SIREN. In this table, AS denotes the number of adder-steps in the CAVM and MCM block of the direct and transposed form FIR filter, respectively. For each FDO algorithm, the CAVM and MCM blocks were designed under two objectives. Under the *oper* objective, the FDO algorithms target the optimization of the number of operations without a restriction on the number of adder-steps. Under the *step* objective, they target the optimization of the number of operations considering a delay constraint as described in Sections II-B-1 and II-B-2.

First, consider the results of FDO algorithms on the direct and transposed forms. Observe that while the total number of operations in both forms is equal or very close to each other, the number of adder-steps in the CAVM block of the direct form is

TABLE VI
SUMMARY OF RESULTS OF FDO ALGORITHMS ON FILTERS A AND S2 OF TABLE I

Filter	Algorithm	Objective	EWL	Direct Form							Transposed Form						
				TA	AS	CA	NCA	A	D	P	TA	AS	CA	NCA	A	D	P
A	NAIAD	oper	10	72	12	36.1	15.8	51.9	8.7	2.3	72	3	45.4	26.7	72.1	7.2	7.3
		step		72	8	36.8	15.8	52.6	8.3	2.4	72	3	45.4	26.7	72.1	7.2	7.3
	[29]	oper	10	68	11	34.5	15.8	50.3	9.2	2.3	68	4	43.1	26.6	69.7	8.0	7.1
		step		69	8	34.8	15.8	50.6	8.4	2.3	69	3	43.4	26.6	70.0	7.5	7.5
	SIREN	oper	10	68	12	35.5	15.8	51.3	9.0	2.2	68	4	43.3	26.1	69.4	7.8	6.9
		step		69	8	35.8	15.8	51.6	8.8	2.2	69	3	45.0	26.2	71.2	7.3	7.8
S2	[29]	oper	10	76	12	38.7	16.0	54.7	9.2	2.4	76	3	48.8	27.5	76.3	8.1	7.4
		step		76	9	38.7	16.0	54.7	8.7	2.3	76	3	48.8	27.5	76.3	8.1	7.4
	NAIAD	oper	10	72	12	36.0	16.0	52.0	8.7	2.4	72	4	46.5	27.5	74.0	7.8	7.6
		step		73	9	37.0	16.0	53.0	8.6	2.4	73	3	46.7	27.5	74.2	7.8	7.3
	SIREN	oper	9	71	12	36.0	16.0	52.0	9.2	2.3	71	4	44.5	26.4	70.9	7.3	7.9
		step		72	8	36.2	16.0	52.2	8.3	2.3	73	2	45.9	26.4	72.3	6.7	7.7

larger than that of the MCM block of the transposed form. The direct form filters occupy significantly less area than the transposed form, which is primarily because of two facts: i) while the size of registers is equal to the bitwidth of the filter input bwi in the direct form, it is increasing up to the bitwidth of the filter output in the transposed form. This fact can be observed from the *NCA* results; ii) the CAVM algorithm [15] can exploit the common subexpressions in (1) due to the symmetric coefficients using $\lfloor N/2 \rfloor$ adders with a size of $bwi + 1$, which is 17 in our experiment. This fact can be observed from the *CA* results. The transposed form filters have less delay compared to the direct form filters because of fewer adder-steps in the multiplier block. But, they consume more power which is primarily related to the area of the design.

Second, consider the results of filters obtained with and without a restriction on the number of adder-steps in the multiplier blocks. Observe that reducing the number of adder-steps generally decreases the delay of the design, where the maximum gain is obtained as 9.7% on the direct form filter S2 obtained by SIREN. However, a restriction on the number of adder-steps may increase the total number of operations, and consequently, the area of the design. Note that the reduction of the number of adder-steps increases the clock frequency of the filter design and can also reduce the complexity of the pipelined realization of the filter as shown in [13].

Third, consider the results of FDO algorithms. Observe that although both the method of [29] and SIREN obtain a solution with the same TA and EWL values on filter A, the filter design obtained by the solution of SIREN occupies larger area than that realized based on the solution of the method of [29], expect on the transposed form under the *oper* objective. This example indicates that there may exist many solutions to the FDO problem with the same TA and EWL values, but yielding filter designs with different gate-level area. Also, the solution of NAIAD on filter A leads to a filter design occupying the largest area, since its solution includes the largest number of operations. In turn, the solutions of SIREN on filter S2 lead to the least complex filter designs, since its solutions have less TA and EWL values than those of the method of [29] and have less TA values than those of NAIAD. Also, since the solutions of NAIAD include fewer operations than those of the method of [29] on filter S2, filters designed based on the solutions of NAIAD have less complexity than those obtained by the solutions of the method of [29].

V. CONCLUSIONS

This article addressed the problem of optimizing the number of operations in the FIR filter design while satisfying the filter constraints, generally known as the FDO problem. It presented exact and approximate FDO algorithms, all of which are equipped with efficient methods to find the fewest operations in the shift-adds design of the coefficient multiplications. Moreover, it showed how these algorithms can be modified to target different filter constraints and filter forms and to handle a delay constraint in the multiplier blocks of filters. It was observed that the exact FDO method can handle filters with a small number of coefficients, on which approximate FDO methods can find solutions very close to the minimum. It was also shown that heuristic methods are indispensable for filters with a large number of coefficients, on which the proposed approximate method can find better solutions in terms of the number of operations than prominent FDO algorithms. It was indicated that the total number of operations, EWL value, filter length, quantization value, and filter form have a significant impact on the gate-level area, delay, and power dissipation results of filter designs.

ACKNOWLEDGMENT

The authors would like to thank Prof. Arda Yurdakul for providing the source code of their algorithm FIRGAM [22].

REFERENCES

[1] L. Wanhammar, *DSP Integrated Circuits*. New York, NY, USA: Academic, 1999.

[2] H. Nguyen and A. Chatterjee, "Number-splitting with shift-and-add decomposition for power and hardware optimization in linear DSP synthesis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 4, pp. 419–424, 2000.

[3] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, 2007, doi: 10.1145/1240233.1240234.

[4] P. Cappello and K. Steiglitz, "Some complexity issues in digital signal Processing," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 5, pp. 1037–1041, 1984.

[5] R. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, no. 10, pp. 677–688, 1996.

[6] I.-C. Park and H.-J. Kang, "Digital filter synthesis based on minimal signed digit representation," in *Proc. Design Autom. Conf.*, 2001, pp. 468–473.

[7] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Exact and approximate algorithms for the optimization of area and delay in multiple constant Multiplications," *IEEE Trans. Comput.-Aided Design Intrgr. Circuits Syst.*, vol. 27, no. 6, pp. 1013–1026, 2008.

[8] A. Dempster and M. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 9, pp. 569–577, 1995.
[9] L. Aksoy, E. Gunes, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *Elsevier J. Microprocessors Microsyst.*, vol. 34, no. 5, pp. 151–162, 2010.
[10] K. Johansson, O. Gustafsson, and L. Wanhammar, "A detailed complexity model for multiple constant multiplication and an algorithm to minimize the complexity," in *Proc. IEEE Eur. Conf. Circuit Theory Design*, 2005, pp. 465–468.
[11] H.-J. Kang and I.-C. Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders," *IEEE Trans. Circuits Syst. II*, vol. 48, no. 8, pp. 770–777, 2001.
[12] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Finding the optimal tradeoff between area and delay in multiple constant multiplications," *Elsevier J. Microprocessors Microsyst.*, vol. 35, no. 8, pp. 729–741, 2011.
[13] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in *Proc. IEEE Int. Symp. Circuit Syst.*, 2012, pp. 49–52.
[14] K. Johansson, O. Gustafsson, L. DeBrunner, and L. Wanhammar, "Minimum adder depth multiple constant multiplication algorithm for low power FIR filters," in *Proc. IEEE Int. Symp. Circuit Syst.*, 2011, pp. 1439–1442.
[15] L. Aksoy, P. Flores, and J. Monteiro, "ECHO: A novel method for the multiplierless design of constant array vector multiplication," in *Proc. IEEE Int. Symp. Circuit Syst.*, 2014, pp. 1456–1459.
[16] B. Y. Kong and I.-C. Park, "FIR filter synthesis based on interleaved processing of coefficient generation and multiplier-block synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 8, pp. 1169–1179, 2012.
[17] Y. C. Lim and S. R. Parker, "FIR filter design over a discrete powersof-two coefficient space," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 31, no. 3, pp. 583–591, 1983.
[18] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with power-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. 36, no. 7, pp. 1044–1047, 1989.
[19] C.-Y. Yao and C.-J. Chien, "A Partial MILP algorithm for the design of linear phase FIR filters with SPT coefficients," *IEICE Trans. Fundam. Electron., Commun., Comput. Sci.*, vol. E85-A, no. 10, pp. 2302–2310, 2002.
[20] C.-L. Chen and A. N. Willson, Jr., "A trellis search algorithm for the design of FIR filters with signed-powers-of-two coefficients," *IEEE Trans. Circuits Syst. II*, vol. 46, no. 1, pp. 29–39, 1999.
[21] O. Gustafsson, H. Johansson, and L. Wanhammar, "An MILP approach for the design of linear-phase FIR filters with minimum number of signed-power-of-two terms," in *Proc. Eur. Conf. Circuit Theory Design*, 2001, pp. 217–220.
[22] M. Aktan, A. Yurdakul, and G. Dündar, "An algorithm for the design of low-power hardware-efficient FIR filters," *IEEE Trans. Circuits Syst.*, vol. 55, no. 6, pp. 1536–1545, 2008.
[23] N. Takahashi and K. Suyama, "Design of CSD coefficient FIR filters based on branch and bound method," in *Proc. Int. Symp. Commun. Inf. Technol.*, 2010, pp. 575–578.
[24] A. Shahein, Q. Zhang, N. Lotze, and Y. Manoli, "A novel hybrid monotonic local search algorithm for FIR filter coefficients optimization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 3, pp. 616–627, 2012.
[25] J. Yli-Kaakinen and T. Saramaki, "A systematic algorithm for the design of multiplierless FIR filters," in *Proc. IEEE Int. Symp. Circuit Syst.*, 2001, pp. 185–188.
[26] O. Gustafsson and L. Wanhammar, "Design of linear-phase FIR filters combining subexpression sharing with MILP," in *Proc. Midwest Symp. Circuits Syst.*, 2002, pp. 9–12.
[27] F. Xu, C. H. Chang, and C. C. Jong, "Design of low-complexity FIR filters based on signed-powers-of-two coefficients with reusable common subexpressions," *IEEE Trans. Comput.-Aided Design Intrgr. Circuits Syst.*, vol. 26, no. 10, pp. 1898–1907, 2007.
[28] Y. J. Yu and Y. C. Lim, "Optimization of linear phase FIR filters in dynamically expanding subexpression space," *Circuits, Syst., Signal Process.*, vol. 29, no. 1, pp. 65–80, 2010.
[29] D. Shi and Y. J. Yu, "Design of linear phase FIR filters with high probability of achieving minimum number of adders," *IEEE Trans. Circuits Syst.*, vol. 58, no. 1, pp. 126–136, 2011.
[30] L. Aksoy, P. Flores, and J. Monteiro, "SIREN: A depth-first search method for the filter design optimization problem," in *Proc. Great Lakes Symp. VLSI*, 2013, pp. 179–184.
[31] L. Aksoy, P. Flores, and J. Monteiro, "A tutorial on multiplierless design of FIR filters: Algorithms and architectures," *Circuits, Syst., Signal Process.*, vol. 33, no. 6, pp. 1689–1719, 2014.
[32] G. Dantzig, *Linear Programming and Extensions*. Princeton, NJ, USA: Princeton Univ. Press, 1963.
[33] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.
[34] R. Vanderbei, *Linear Programming: Foundations and Extensions*. New York, NY, USA: Springer, 2001.
[35] M. Ercegovac and T. Lang, *Digital Arithmetic*. San Mateo, CA, USA: Morgan Kaufmann, 2003.
[36] Y. C. Lim, "Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude," *IEEE Trans. Circuits Syst.*, vol. 37, no. 12, pp. 1480–1486, 1990.
[37] T. Parks and C. Burrus, *Digital Filter Design*. New York, NY, USA: Wiley, 1987.
[38] J. McClellan, T. Parks, and L. Rabiner, "A computer program for designing optimum FIR linear phase digital filters," *IIEEE Trans. Audio Electroacoust.*, vol. 21, no. 6, pp. 506–526, 1973.
[39] L. Rabiner, "Linear program design of FIR digital filters," *IIEEE Trans. Audio Electroacoust.*, vol. 20, no. 4, pp. 280–288, 1972.
[40] O. Gustafsson, "Lower bounds for constant multiplication problems," *IEEE Trans. Circuits Syst. II*, vol. 54, no. 11, pp. 974–978, 2007.
[41] M. Faust and C.-H. Chang, "Low error bit width reduction for structural adders of FIR filters," in *Proc. IEEE Eur. Conf. Circuit Theory Design*, 2011, pp. 713–716.

Levent Aksoy (S'06–M'09) received the M.S. and Ph.D. degrees in electronics and communication engineering and electronics engineering from Istanbul Technical University (ITU), Istanbul, Turkey, in 2003 and 2009, respectively. He was a Research Assistant with the Faculty of Electrical and Electronics Engineering, ITU from 2001 to 2009. Since November 2009, he has been with the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, where he is currently a post-doctoral researcher in Algorithms for Optimization and Simulation (ALGOS) group. His research interests include optimization algorithms, high-level synthesis, and DSP system design.

Paulo Flores (S'92–M'02–SM'13) received the five-year engineering degree, M.Sc., and Ph.D. degrees in electrical and computer engineering from Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal, in 1989, 1993, and 2001, respectively. Since 1990, he has been teaching at Instituto Superior Técnico, University of Lisbon, where he is currently an Assistant Professor in the Department of Electrical and Computer Engineering. He has also been with the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, since 1988, where he is currently a Senior Researcher in Algorithms for Optimization and Simulation (ALGOS) group. His research interests are computer architecture and CAD for VLSI circuits in the area of embedded systems, test and verification of digital systems, and computer algorithms, with particular emphasis on optimization of hardware/software problems using satisfiability (SAT) models. He has contributed to more than 50 papers to journals and international conferences. Dr. Flores is a senior member of the IEEE Circuit and Systems Society.

José Monteiro (S'93–M'96–SM'10) received a five-year engineering degree and the M.Sc. degree in electrical and computer engineering from Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal, in 1989 and 1992, respectively, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, in 1996. Since 1996, he has been with Instituto Superior Técnico, University of Lisbon, where he is currently an Associate Professor in the Department of Computer Science and Engineering. He is currently Director of the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon. His main interests are computer architecture and CAD for VLSI circuits, with emphasis on synthesis, power analysis, low-power, and design validation. Dr. Monteiro received the Best Paper Award from the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS in 1995.