SIREN: A Depth-First Search Algorithm for the Filter Design Optimization Problem

Levent Aksoy
INESC-ID
Lisboa, Portugal
levent@algos.inesc-id.pt

Paulo Flores
INESC-ID/IST TU Lisbon
Lisboa, Portugal
pff@inesc-id.pt

José Monteiro
INESC-ID/IST TU Lisbon
Lisboa, Portugal
jcm@inesc-id.pt

ABSTRACT

This paper addresses the filter design optimization (FDO) problem that is to find a set of filter coefficients which yields the least design complexity while meeting the required filter constraints. The design complexity of a filter is defined in terms of the total number of adders/subtracters, assuming that the multiplication of coefficients by the filter input is realized under a shift-adds architecture. Existing algorithms use efficient search methods, but none of them can guarantee the minimum design complexity. Hence, we propose an exact algorithm, called SIREN, that finds an optimum solution of the FDO problem under the minimum quantization value. It is based on a depth-first search method equipped with an exact technique, that finds the minimum number of adders/subtracters in the multiplier block of the filter, and search pruning techniques that enable it to be applicable to practical instances. Experimental results show that SIREN can still find better solutions than efficient FDO algorithms and its solutions lead to filters with significantly less area when compared to a straightforward filter design technique.

Categories and Subject Descriptors

G.1.6 [**Mathematics of Computing**]: Optimization

General Terms

Algorithms, Design

Keywords

Filter design optimization, depth-first search, linear programming, multiplierless filter design

1. INTRODUCTION

Finite impulse response (FIR) filters are widely used in digital signal processing (DSP) applications due to their output stability and phase linearity. The output of an N-tap FIR filter $y(n)$ is computed as $\sum_{i=0}^{N-1} h_i \cdot x(n-i)$, where h_i is the i^{th} filter coefficient and $x(n-i)$ is the i^{th} previous filter input. The realization of this expression in the transposed form is shown in Figure 1, where the multiplication of coefficients by the filter input dominates the design complexity.

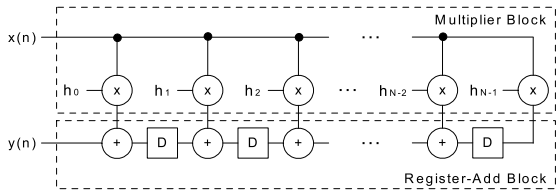


Figure 1: Transposed form FIR filter design.

However, the multiplier block of the FIR filter can be realized using only shifts and adders/subtracters, since the filter coefficients are determined to be constant [10]. Note that shifts can be implemented using only wires without representing any hardware cost. Thus, a fundamental optimization problem, called the multiple constant multiplications (MCM) [13], is defined as: given a set of constants to be multiplied by an input variable, find the minimum number of adders/subtracters that realize the constant multiplications. Existing MCM algorithms [1, 2, 8, 13] aim to maximize the sharing of partial products among the constant multiplications. In this manner, a straightforward way to design a multiplierless FIR filter requires two steps. First, a set of coefficients, that respects the filter constraints, is obtained, and second, the multiplier block of the filter is designed under a shift-adds architecture using an MCM algorithm.

In turn, the FDO problem [12] is defined as: given the filter specifications (*fspec*) denoted as a 5-tuple (filter length N, pass-band w_p and stop-band w_s frequencies, and maximum allowed pass-band δ_p and stop-band δ_s ripples), find a set of coefficients that leads to a filter design with minimum number of adders/subtracters and satisfies the filter constraints. The design complexity of the transposed form FIR filter is computed based on the number of adders/subtracters both in its multiplier and register-add blocks, assuming that the constant multiplications in its multiplier block are realized under a shift-adds architecture. Existing FDO algorithms include efficient techniques that minimize the number of nonzero digits in coefficients [3, 4, 11], maximize the common nonzero digits in coefficients [7, 14], and dynamically expand a set of partial products which are shared among the coefficient multiplications [12, 16]. However, to the best of our knowledge, there is no algorithm that guarantees a filter with the minimum design complexity.

In this paper, we first present the fundamental concepts on multiplierless design of constant multiplications and give an overview on prominent FDO algorithms. Then, we intro-

GLSVLSI'13, May 2–3, 2013, Paris, France.

duce the SIREN algorithm which can find a set of fixed-point filter coefficients that satisfies the filter constraints and leads to a filter design requiring minimum number of adders/subtraters under the smallest quantization value (Q). In SIREN, first, the floating-point lower and upper bounds of coefficients are computed. In an iterative loop, these lower and upper bounds of coefficients are quantized using Q, that is initially set to 0 and is increased by 1 in each iteration. Then, a depth-first search (DFS) method is applied to find a solution that yields a filter design with minimum complexity, satisfying the filter constraints or to prove that there exists no solution to be selected from the quantized lower and upper bounds of coefficients. The most important properties of the DFS method of SIREN are: i) it constructs a search tree based on a predetermined ordering of coefficients; ii) as the values of coefficients are determined, it updates the lower and upper bounds of coefficients in the search tree; iii) it computes the lower bound on the filter complexity and backtracks if this lower bound is greater than or equal to the best solution found so far; iv) it uses the exact MCM technique of [2] to find the minimum number of operations in the multiplier block of the filter. This paper also presents two variations of SIREN that are applied to slightly different FDO problems where search space is highly restricted. Finally, we introduce their time-complexity on randomly generated FIR filters, compare their solutions with prominent FDO algorithms, and give the gate-level results of FIR filters synthesized based on the solutions of SIREN. Experimental results indicate that they can be applied to real-size FIR filters and can find a minimum solution where prominent FDO algorithms cannot obtain a solution with this value or cannot ensure that their solutions are the minimum solutions.

2. BACKGROUND

2.1 Multiplierless Constant Multiplications

A straightforward method to realize constant multiplications under a shift-adds architecture, called the digit-based recoding (DBR) [5], initially defines the constants in multiplications under a number representation, *e.g.*, canonical signed digit (CSD)[1] [8] or binary. Then, for the nonzero digits in the representation of the constant, it shifts the input variable according to the digit positions and adds/subtracts the shifted variable with respect to the digit values. As a simple example, consider the constant multiplications $39x$ and $83x$. Their decompositions in CSD are listed as:

$$39x = (10100\overline{1})_{CSD}x = x \ll 5 + x \ll 3 - x$$
$$83x = (101010\overline{1})_{CSD}x = x \ll 6 + x \ll 4 + x \ll 2 - x$$

and require 5 operations as shown in Figure 2(a).

The complexity of an MCM design can be further reduced by sharing the partial products among the constant multiplications. Existing MCM methods can be grouped in two classes: common subexpression elimination (CSE) [1, 8] and graph-based (GB) [2, 13] techniques. The CSE algorithms initially define the constants under a particular number representation. Then, considering possible subexpressions that can be extracted from the nonzero digits in representations of constants, the "best" subexpression, generally, the most common, is chosen to be shared among the constant multiplications. The GB methods are not restricted to any particular number representation and find the intermediate partial products that enable to realize the constant multiplications with minimum number of operations. They consider a larger number of realizations of a constant and may obtain better solutions than CSE methods [2, 13].

[1]In CSD, a constant is written as $\sum_{i=0}^{n-1} b_i 2^i$ using n digits, where $b_i \in \{1, 0, \overline{1}\}$ and $\overline{1}$ denotes -1. In its CSD representation, nonzero digits are not adjacent and minimum number of nonzero digits is used.

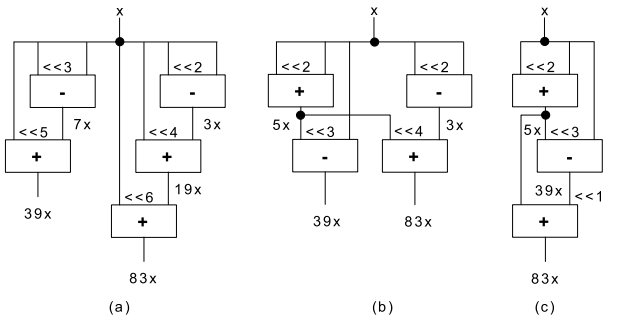


Figure 2: Shift-adds realizations of $39x$ and $83x$: (a) the DBR technique [5]; (b) the exact CSE algorithm of [1]; (c) the exact GB algorithm of [2].

Returning to our example, the exact CSE algorithm [1] obtains a solution with 4 operations when constants are defined under CSD (Figure 2b) and the exact GB method [2] finds a minimum solution with 3 operations (Figure 2c).

2.2 Filter Design Optimization Algorithms

The zero-phase frequency response of a symmetrical FIR filter is given as[2]:

$$G(w) = \sum_{j=0}^{\lfloor M \rfloor} d_j h_j cos(w(M-j))$$

where $M = (N-1)/2$ and $d_j = 2 - K_{j,M}$ with $K_{j,M}$ is the Kronecker delta[3], $h_j \in \mathbb{R}$ with $-1 \leq h_j \leq 1$, and $w \in \mathbb{R}$ is the frequency in radians. Assuming that the desired pass-band and stop-band gains are equal to 1 and 0, respectively, the filter must satisfy the following constraints [3]:

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

The pass-band gain is not relevant for many DSP applications. Thus, a scaling factor (s) can be added into the filter constraints as a continuous variable as follows, where s^l and s^u are respectively the lower and upper bounds of s [7, 15].

$$\begin{aligned} s(1 - \delta_p) \leq G(w) \leq s(1 + \delta_p), \quad & w \in [0, w_p] \\ s(-\delta_s) \leq G(w) \leq s(\delta_s), \quad & w \in [w_s, \pi] \\ s^l \leq s \leq s^u & \end{aligned} \tag{2}$$

In some DSP applications, it is also desirable to minimize the normalized peak ripple (NPR) [12, 16], which is defined as δ/b, where δ and b are the peak ripple and pass-band gain, respectively. Other considered parameters are the peak weighted ripple [11] and the NPR magnitude [14].

FDO algorithms generally reduce the design complexity in two different ways. The algorithms of [3, 4, 11] try to find a set of coefficients with a minimum number of nonzero digits, since a constant represented with fewer number of

[2]Frequency response of an asymmetrical filter can be found in [9].

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

SIREN($fspec$**)**
1: $Q = 0$, $sol\_found = 0$, $sol = [\,]$
2: $(h^l, h^u, s^l, s^u) = \text{CLUBFCSF}(fspec)$
3: $O = \text{OrderCoefs}(h^l, h^u)$
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** are_lubfc_valid(H^l, H^u) **then**
7: $(sol\_found, sol) = \text{DFS}(fspec, O, Q, H^l, H^u, s^l, s^u)$
8: **until** sol_found
9: **return** sol

Figure 3: The SIREN algorithm.

nonzero digits requires fewer number of operations in general. However, the reduction of the number of nonzero digits does not always lead to a design with minimum number of operations, since the sharing of partial products among the constant multiplications is not considered (Figure 2). In order to exploit the sharing of partial products, the algorithms of [7, 14] search for the coefficients that have the most common nonzero digits. Also, in the methods of [12, 16], the common partial products are expanded dynamically as the filter coefficients are determined. However, these techniques also cannot ensure the minimum number of operations in the filter, since they are not equipped with exact methods.

FDO algorithms explore the filter coefficients using exhaustive [3, 7, 12, 15, 16] and local search [4, 11, 14] methods. The exhaustive search techniques use mixed-integer linear programming [7], branch-and-bound [3, 15], and DFS [12, 16] methods. While they can only be applied to low-order filters due to an exponential increase in run-time [3, 12], the local search methods can be applied to high-order filters, but cannot ensure the minimum design complexity, since all possible values of coefficients are not considered.

Meanwhile, SIREN is designed to handle the filter constraints in Eqn. 2, where the search space is increased due to s, yielding better solutions [7]. Its variations, that can handle the same filter constraints with a restricted s and those in Eqn. 1, are also described. It is based on the exhaustive DFS method and uses the exact MCM algorithm [2].

3. THE SIREN ALGORITHM

As shown in its pseudo-code given in Figure 3, SIREN takes the 5-tuple *fspec* denoting the filter specifications as input and returns a set of fixed-point filter coefficients *sol*. First, it computes the lower and upper bounds of filter coefficients and s in its *CLUBFCSF* function, where the lower bounds of coefficients are found by solving the following linear programming (LP) problem for each coefficient.

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

The filter coefficients and s are *continuous* variables of this LP problem and the initial s^l and s^u were set to 0.01 and 100, respectively. Thus, 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 changed to $f = -h_i$ and is stored in h^u. The sets h^l and h^u consist of the floating-point lower and upper bound of each filter coefficient, respectively. The values of s^l and s^u are also updated similarly.

Then, its *OrderCoefs* function determines an ordering of filter coefficients to be used by the DFS method described in the following subsection while constructing the search tree. The main idea on finding an ordering of coefficients is based on a fact observed in experiments that if the coefficients with narrower upper and lower bound intervals are placed in lower depths of a search tree, less number of decisions are made and conflicts occur earlier. Thus, the runtime of SIREN can be reduced significantly, still exploring all possible values of coefficients. This function sorts the coefficients in ascending order according to their $h_i^u - h_i^l$ values and stores their indices i in this order in O.

In the iterative loop of SIREN, 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 is tested by the *are_lubfc_valid* function by simply checking each coefficient if H_i^l is less than or equal to H_i^u. If they are not valid, Q is increased by one, and H^l and H^u are updated. 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 latter 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. Note that Q is an important parameter in the filter design complexity, because as it increases, the bitwidths of coefficients are increased, requiring larger sizes of registers and structural adders in the register-add block (Figure 1), and most probably, requiring more number of operations in the multiplier block since larger integer coefficients are used. Similarly, in [3, 12, 16], the effective wordlength (EWL) of a set of coefficients, computed as $max\{\lceil log_2|h_i| \rceil\}$ when fixed-point coefficients are considered, was used to evaluate the quality of a solution in addition to the total number of adders/subtracters.

3.1 The Depth-First Search Method

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 d^{th} 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.

The DFS method first assigns $H_{O(1)}^l$ and $H_{O(1)}^u$ respectively to the lower and upper bounds of the vertex at depth 1, *i.e.*, the initial depth, and sets the value of the vertex V_1 to $H_{O(1)}^l$. However, 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, we can use tighter lower and upper bounds of a vertex, since the values of $d-1$ coefficients are determined and fixed at this stage. Thus, 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.

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

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

When the values of all coefficients are determined, the implementation cost of the filter based on this coefficient set is computed as $TA = MA+SA$, where TA is the total number of adders/subtracters in the filter, and MA and SA are the number of adders/subtracters in the multiplier block and the number of structural adders in the register-add block, respectively. While MA is found using the exact MCM algorithm [2], SA is computed as $N-ncz-1$, where ncz is the number of coefficients equal to zero. Naturally, no adder is required for a coefficient set to zero in the register-add block. This coefficient set is stored in *sol* and *sol_found* is set to 1, if its TA value is smaller than that of the best one found so far, which was set to infinity in the beginning of search.

In order to prune the search tree, the implementation cost of the filter (TA) is estimated at depth d greater than $2M/3$ ($2N/3$) for symmetric (asymmetric) filters. This value is found empirically not to waste an effort for computing an estimate which usually does not yield a backtrack. The lower bound on MA is found using the determined coefficients as described in [6]. The lower bound on SA is computed 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_found* is 0, it ensures that there exists no solution which respects the filter constraints under the given quantized lower and upper bounds of coefficients. Otherwise, its solution is a set of integer coefficients found using minimum Q that leads to a filter with minimum design complexity, satisfying the filter constraints.

As a simple example, consider a symmetric filter with *fspec* (8, 0.2π, 0.7π, 0.01, 0.01). The floating-point lower and upper bounds of coefficients are respectively 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 $h^u = \{h_0^u, h_1^u, h_2^u, h_3^u\} = \{-0.0003, -0.0002, 0.4144, 1\}$. The ordering of coefficients is found as $O = \{1, 0, 2, 3\}$. The quantized lower and upper bounds of coefficients are determined as $H^l = \{-3, -2, 1, 1\}$ and $H^u = \{-1, -1, 13, 32\}$, respectively when Q is 5 (no solution was found with $Q < 5$).

The search tree constructed by the DFS method is shown in Figure 4, 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 determined. 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. Here, **Conflict** denotes that given determined coefficients, there exists no feasible lower/upper bound for the current depth vertex. Also, **Success** presents that the set

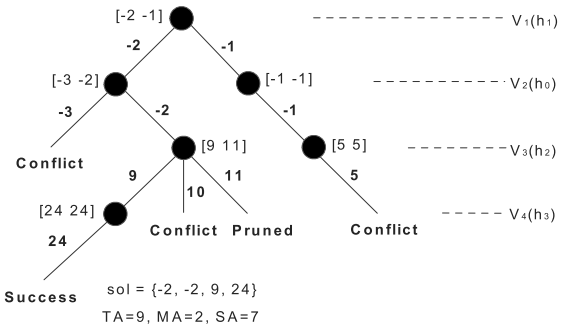


Figure 4: Search tree formed by the DFS method.

of determined coefficients leads to a solution satisfying the filter constraints and **Pruned** indicates that the set of determined coefficients cannot lead to a better solution than the best one found so far. Observe that as values are assigned to coefficients, the intervals between the lower and upper bounds of coefficients are significantly reduced when compared to those in the original H^l and H^u.

3.2 Modifications for Similar FDO Problems

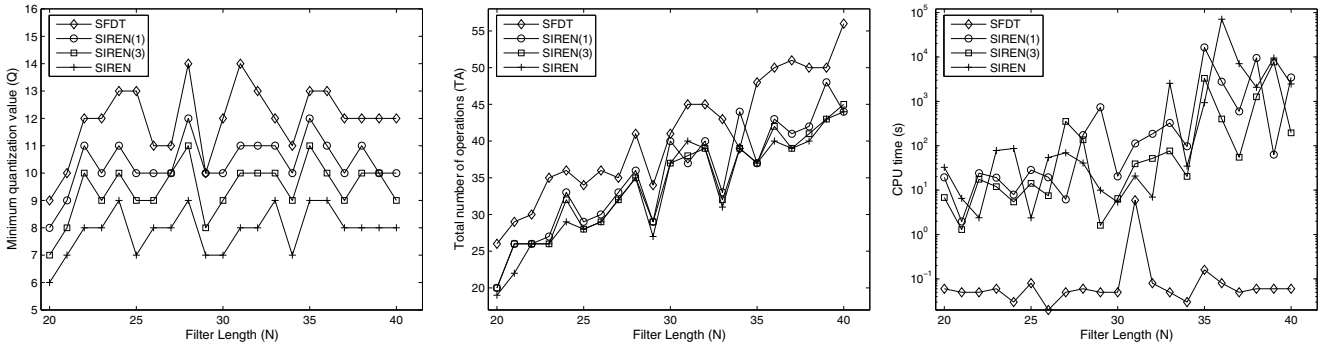
Since SIREN is a general algorithm, it can easily be modified to handle filter constraints given in Eqn. 1 or those in Eqn. 2 with a restricted s. For the former, the lower and upper bounds of s are simply set to 1 and this type of the SIREN algorithm is denoted as SIREN(1). For the latter, its lower and upper bounds are set to given values. In another type of the SIREN algorithm called SIREN(3), they were respectively set to 0.7 and 1.4, $\mp$3dB gain tolerance.

4. EXPERIMENTAL RESULTS

In this section, we present the results of SIREN and its variations, and compare them with those of prominent FDO algorithms. We also introduce the results of a straightforward filter design technique (SFDT), where given *fspec*, the *firgr* function of MATLAB was used to obtain the filter coefficients which were quantized to integers with minimum Q, avoiding the effect of quantization errors on the filter constraints. Then, the exact MCM algorithm [2] was applied to these filter coefficients to realize the multiplier block of the filter using minimum number of adders/subtracters. Note that SIREN was written in MATLAB and was run on a PC with Intel Xeon at 2.33GHz and 4GB memory. As an LP solver, it uses *lp_solve 5.5.2.0*.

As the first experiment set, we used 21 randomly generated symmetric low-pass FIR filters whose N values range between 20 and 40. Figure 5 presents the results of algorithms in terms of Q, TA, and CPU time in seconds.

Recall that SIREN and its variations iteratively increase Q until they find a solution that satisfies the filter constraints. Observe from Figure 5(a) that SIREN always finds a solution on these instances using a smaller Q than its variations. The difference between the average Q value in its solutions and those in the solutions of SIREN(3), SIREN(1), and SFDT is 1.5, 2.5, and 4, respectively. In this case, the EWL values of filter coefficients in its solutions are also smaller than those in the solutions of these algorithms. This comes from the fact that since there is no restriction on s in SIREN, the filter constraints of Eqn. 2, can be satisfied with smaller coefficients, but with a larger s. Simply because of this fact and because no restriction on s increases the number of pos-



(a) (b) (c)

Figure 5: Results of algorithms on randomly generated filters (a) Q; (b) TA; (c) CPU.

Table 1: Specifications of symmetric FIR filters.

Filter	N	w_p	w_s	δ_p	δ_s
X1	15	0.2π	0.8π	0.0001	0.0001
G1	16	0.2π	0.5π	0.01	0.01
Y1	30	0.3π	0.5π	0.00316	0.00316
Y2	38	0.3π	0.5π	0.001	0.001
A	59	0.125π	0.225π	0.01	0.001
S2	60	0.042π	0.14π	0.012	0.001
L2	63	0.2π	0.28π	0.028	0.001

sible constants considered for a coefficient, SIREN generally obtains better solutions in terms of TA than these algorithms, as can be observed from Figure 5(b). On average, it finds 0.6, 1.7, and 7.2 less number of adders/subtracters than SIREN(3), SIREN(1), and SFDT, respectively. However, there is only one exception, the filter with an N value 31, where it obtains a worse solution than its variations. This is because the use of a larger Q in its variations may enable them to consider such filter coefficients that require no adder/subtracter, like 0 or shifted versions of 1, or that maximize the sharing of partial products[4]. On the other hand, SIREN requires the most CPU time on average, simply because no restriction on s increases the upper and lower bound intervals of coefficients, leading to a large number of branches (assignments) in the DFS method. Additionally, N and Q are other parameters that affect the runtime of SIREN and its variations. While an increase in N increases the number of vertices in the DFS method, an increase in Q increases the number of runs of the DFS method, the number of branches in the DFS method that is due to a larger upper and lower bound interval of a vertex, and the runtime of the exact MCM algorithm [2] since larger coefficients are considered. Hence, SIREN may obtain a solution using less CPU time than its variations due to a smaller Q.

As the second experiment set, we used the symmetric FIR filters given in Table 1. Table 2 presents the results of SIREN, SIREN(3), and previously proposed algorithms whose results were taken from [3, 12] as reported. The CPU time limit was set to 2 days for SIREN and SIREN(3). In this table, BST denotes the CPU time required to find the best solution and TT stands for the total CPU time. The algorithms were sorted according to their results on i) EWL, ii) TA, iii) TT, and iv) BST. Note that the solutions of [12] had been determined to be the best ones on these filters.

Observe from Table 2 that both SIREN and SIREN(3) can find a solution with an EWL value less or the same

[4]As SIREN can be initiated with any Q (Figure 3), it was applied to this instance with the same Q that SIREN(3) and SIREN(1) used and obtained better results than its variations.

Table 2: Summary of algorithms on FIR filters.

Filter	Method	EWL	MA	SA	TA	BST	TT
X1	[14]	13	7	8	15	–	–
	SIREN(3)	10	5	8	13	4s	6s
	SIREN	10	5	8	13	<1s	2s
	[12]	10	5	8	13	–	<1s
G1	[7]	7	2	13	15	–	–
	[12]	6	2	15	17	–	<1s
	SIREN(3)	6	2	15	17	<1s	<1s
	SIREN	6	2	15	17	<1s	<1s
Y1	[16]	10	6	23	29	–	21m30s
	SIREN(3)	9	7	29	36	2s	4s
	[12]	9	7	23	30	–	6s
	SIREN	9	6	23	29	2m17s	7m56s
Y2	[15]	12	–	–	39	–	–
	[12]	10	10	37	47	–	11s
	SIREN(3)	10	9	29	38	6m36s	7m29s
	SIREN	10	9	29	38	3m52s	4m29s
A	[3]	10	18	58	76	3h2m	4h14m
	[12]	10	14	54	68	–	2d2h
	SIREN(3)	*10*	*16*	*52*	*68*	*1d6h*	*2d**
	SIREN	*10*	*16*	*52*	*68*	*14h57m*	*2d**
S2	[3]	11	27	59	86	23m	27m
	[12]	10	17	59	76	–	16h42m
	SIREN	*9*	*14*	*57*	*71*	*16h4m*	*2d**
	SIREN(3)	*9*	*14*	*57*	*71*	*7h27m*	*2d**
L2	[3]	10	18	62	80	26m	54m
	SIREN	*10*	*16*	*60*	*76*	*1d23h*	*2d**
	SIREN(3)	*10*	*15*	*58*	*73*	*1d22h*	*2d**
	[12]	10	17	56	73	–	16h28m

* Time-limit is exceeded.

as prominent algorithms. The solutions of SIREN on small size filters X1, G1, Y1, and Y2 include the least or the same number of operations as those of the FDO algorithm [12]. In turn, on medium size filters A, S2, and L2, both SIREN and SIREN(3) cannot complete the search due to the CPU time limit. However, while SIREN(3) yields a solution with less or the same number of operations as the FDO algorithm [12], SIREN obtains a worse solution than SIREN(3) only on Filter L2. This is because it generally requires more CPU time than SIREN(3) to explore the whole search space.

We also designed Filter Y2 based on the solutions of SFDT and SIREN when N was in between 34 (the minimum N value found using the *firgr* function of MATLAB) and 38, and Filter G1 based on the solutions of SIREN when Q was 6 and 7. FIR filters were described in VHDL when the bitwidth of the filter input was 16, and they were synthesized using the Synopsys Design Compiler and UMCLogic 180nm Generic II library. Tables 3 and 4 present the results of FIR filters, where CA, NCA, and A denote the combinational area, non-combinational area, and total area, all in mm^2, respectively.

Table 3: Summary of results of algorithms on Filter Y2.

N	SFDT									SIREN									
	Q	EWL	MA	SA	TA	CPU	CA	NCA	A	Q	EWL	MA	SA	TA	CPU	CA	NCA	A	gain
34	15	14	18	31	49	5m6s	33.2	16.5	49.7	11	11	10	27	37	19m48s	23.2	15.1	38.3	23.0
35	15	14	18	34	52	7s	34.6	16.8	51.4	11	11	12	28	40	34m19s	25.0	15.0	40.0	22.2
36	14	13	15	29	44	1s	28.8	16.7	45.5	11	11	10	27	37	18m38s	23.3	15.3	38.6	15.3
37	14	13	17	36	53	15s	35.9	17.2	53.1	10	10	12	36	48	23s	30.0	15.6	45.6	14.1
38	14	13	15	35	50	15s	33.1	17.6	50.7	10	10	9	29	38	4m29s	23.5	16.1	39.6	21.9

Table 4: Summary of results of SIREN on Filter G1.

$Q = 6$								$Q = 7$							
EWL	MA	SA	TA	CPU	CA	NCA	A	EWL	MA	SA	TA	CPU	CA	NCA	A
6	2	15	17	<1s	9.1	5.6	14.7	7	2	13	15	25s	8.1	5.9	14.0

Also, *gain* in Table 3 is the area gain in percentage between the designs obtained by the solutions of SIREN and SFDT.

Observe from Table 3 that the solutions of SIREN with less number of adders/subtracters and with filter coefficients having smaller EWL values lead to FIR filters occupying smaller area with respect to the solutions of SFDT. Note that as N is increased, the EWL values of coefficients is decreased. Thus, the area values of Filter Y2 designed using the solutions of SIREN when N is 34 and 38 are close to each other, indicating the tradeoff between N and the EWL value of coefficients on the filter complexity. Hence, it is useful to design a filter with different N values, since different sets of coefficients yield filter designs with different complexity.

Also, observe from Table 4 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 less number of structural adders in the register-add block. Thus, it is useful to run SIREN with also $Q+1$ to obtain such solutions. More importantly, these experiments suggest that to evaluate the quality of a solution of an FDO algorithm more accurately, the metrics, which take into account the gate-level implementation cost, should be derived.

5. CONCLUSIONS

This paper introduced the SIREN algorithm equipped with a DFS method, an exact MCM technique, and efficient search pruning techniques, and its variations applied to slightly different FDO problems. To our knowledge, they are the only algorithms that can guarantee the minimum filter design complexity defined as the total number of adders/subtracters using the minimum Q. Experimental results showed that they can be applied to medium size FIR filters and can still obtain better solutions than previously proposed prominent FDO algorithms. Also, the solutions of SIREN lead to less complex FIR filters with respect to the solutions of SFDT.

6. ACKNOWLEDGMENTS

This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia.

7. REFERENCES

[1] 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 TCAD*, 27(6):1013–1026, 2008.

[2] L. Aksoy, E. Gunes, and P. Flores. Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate. *Elsevier MICPRO*, 34(5):151–162, 2010.

[3] M. Aktan, A. Yurdakul, and G. Dündar. An Algorithm for the Design of Low-Power Hardware-Efficient FIR Filters. *IEEE TCAS-I*, 55(6):1536–1545, 2008.

[4] 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 TCAS-II*, 46(1):29–39, 1999.

[5] M. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann, 2003.

[6] O. Gustafsson. Lower Bounds for Constant Multiplication Problems. *IEEE TCAS-II*, 54(11):974–978, 2007.

[7] O. Gustafsson and L. Wanhammar. Design of Linear-Phase FIR Filters Combining Subexpression Sharing With MILP. In *MWSCAS*, pages 9–12, 2002.

[8] R. Hartley. Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers. *IEEE TCAS-II*, 43(10):677–688, 1996.

[9] Y. C. Lim. Design of Discrete-Coefficient-Value Linear Phase FIR Filters With Optimum Normalized Peak Ripple Magnitude. *IEEE TCAS*, 37(12):1480–1486, 1990.

[10] H. Nguyen and A. Chatterjee. Number-Splitting With Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis. *IEEE TVLSI*, 8(4):419–424, 2000.

[11] H. Samueli. An Improved Search Algorithm for the Design of Multiplierless FIR Filters with Power-of-Two Coefficients. *IEEE TCAS*, 36(7):1044–1047, 1989.

[12] D. Shi and Y. J. Yu. Design of Linear Phase FIR Filters With High Probability of Achieving Minimum Number of Adders. *IEEE TCAS-I*, 58(1):126–136, 2011.

[13] Y. Voronenko and M. Püschel. Multiplierless Multiple Constant Multiplication. *ACM Transactions on Algorithms*, 3(2), 2007.

[14] 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 TCAD*, 26(10):1898–1907, 2007.

[15] J. Yli-Kaakinen and T. Saramaki. A Systematic Algorithm for the Design of Multiplierless FIR Filters. In *ISCAS*, pages 185–188, 2001.

[16] Y. J. Yu and Y. C. Lim. Optimization of Linear Phase FIR Filters in Dynamically Expanding Subexpression Space. *Circuits, Systems, and Signal Processing*, 29:65–80, 2010.