Circuits Syst Signal Process (2014) 33:1689–1719
DOI 10.1007/s00034-013-9727-8

A Tutorial on Multiplierless Design of FIR Filters: Algorithms and Architectures

Levent Aksoy · Paulo Flores · José Monteiro

Received: 10 April 2013 / Revised: 12 December 2013 / Published online: 23 January 2014

Abstract Finite impulse response (FIR) filtering is a ubiquitous operation in digital signal processing systems and is generally implemented in full custom circuits due to high-speed and low-power design requirements. The complexity of an FIR filter is dominated by the multiplication of a large number of filter coefficients by the filter input or its time-shifted versions. Over the years, many high-level synthesis algorithms and filter architectures have been introduced in order to design FIR filters efficiently. This article reviews how constant multiplications can be designed using shifts and adders/subtractors that are maximally shared through a high-level synthesis algorithm based on some optimization criteria. It also presents different forms of FIR filters, namely, direct, transposed, and hybrid and shows how constant multiplications in each filter form can be realized under a shift-adds architecture. More importantly, it explores the impact of the multiplierless realization of each filter form on area, delay, and power dissipation of both custom (ASIC) and reconfigurable (FPGA) circuits by carrying out experiments with different bitwidths of filter input, design libraries, reconfigurable target devices, and optimization criteria in high-level synthesis algorithms.

Keywords FIR filter · Direct, transposed, and hybrid forms · Multiplierless design · High-level synthesis · Area and delay optimization · Custom and reconfigurable circuits

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

P. Flores · J. Monteiro
INESC-ID/Tecnico ULisboa, Lisbon, Portugal
e-mail: pff@inesc-id.pt

J. Monteiro
e-mail: jcm@inesc-id.pt

1 Introduction

FIR filters are widely used in digital signal processing (DSP) applications due to their stability and linear-phase properties. The output of an N-tap FIR filter is computed as

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

where N is the filter length, h_i is the ith filter coefficient, and $x(n-i)$ is the ith previous filter input with $0 \leq i \leq N-1$. The straightforward implementation of Eq. 1 when N is equal to 6 is illustrated in Fig. 1a, which is known as the direct form. Alternatively, it can be realized in the transposed form, as shown in Fig. 1b. Observe from Fig. 1a–b that the direct and transposed forms require exactly N multipliers and $N-1$ adders and registers. However, a careful observation reveals three main differences: (i) while the size of registers in the direct form is fixed and equal to the bitwidth of the filter

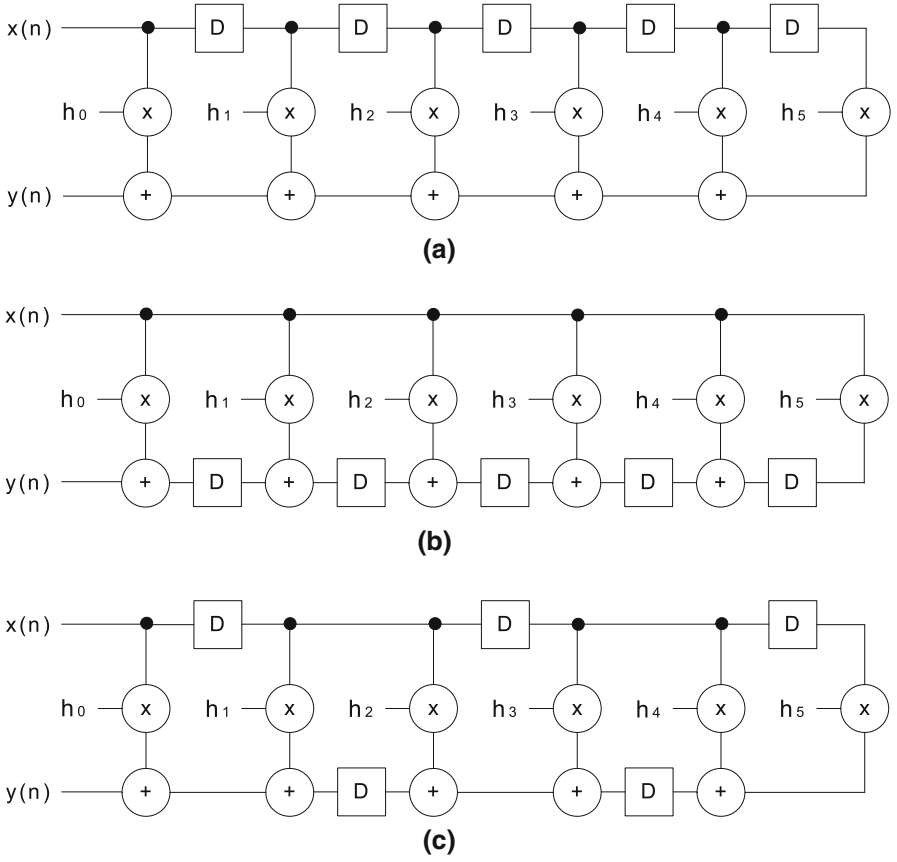

Fig. 1 Different realizations of a 6-tap FIR filter: **a** direct form; **b** transposed form; **c** one type of the hybrid form among many others

input bwi, in the transposed form, their size depends on N, bwi, and the bitwidth of coefficients, increasing up to $bwi + \lceil log_2 \sum_{i=1}^{N-1} |h_i| \rceil$; (ii) the critical path of the transposed form (a multiplier and an adder) is shorter than that of the direct form (a multiplier and $\lceil log_2 N \rceil$ adders, assuming that the adders are organized in a binary tree); (iii) the transposed form has a larger input capacitance than the direct form due to the higher fan-out of the filter input. The expression of Eq. 1 can also be realized in a hybrid form as shown in Fig. 1c, enabling the exploration of these tradeoffs between the direct and transposed forms [28,40].

Although area, delay, and power efficient multiplier architectures, such as Wallace [55] and modified Booth [24] multipliers, have been proposed, the full-flexibility of a multiplier is not necessary in the filter design, since filter coefficients are fixed and determined beforehand [45]. Hence, the multiplication of filter coefficients by the filter input or its time-shifted versions is generally implemented under the shift-adds architecture, where constant multiplications are realized using shifts and addition/subtraction operations [48].

In the last two decades, many efficient high-level synthesis algorithms have been introduced for the multiplierless design of constant multiplications using the fewest number of addition/subtraction operations [1,6,7,9,10,15,17,23,26,29–32,34,35,42,43,47,49,50,53,54,57,59]. These algorithms target different types of constant multiplications and aim to optimize the number of operations by maximizing the partial product sharing among the constant multiplications. Also, the algorithms of [3–6,10,11,38] take into account the implementation cost of each adder/subtractor realizing a constant multiplication in a custom or a reconfigurable circuit and can optimize the area of a shift-adds design. Moreover, the algorithms of [1,3,5,34,36,39] can find a shift-adds design of constant multiplications under a delay constraint, that is defined in terms of the maximum number of operations in series, and can optimize the number of operations or gate-level area. Thus, they can explore the tradeoff between area and delay in shift-adds designs.

In this article, we review prominent algorithms designed for the multiplierless realization of different types of constant multiplications, targeting the optimization of the number of operations, and the area and delay of the design. We also present different FIR filter forms, namely, direct, transposed, and hybrid, and describe how constant multiplications in these filter forms can be efficiently realized under a shift-adds architecture so that the sharing of partial products is maximized using a high-level synthesis algorithm. The most important contribution of this article is the introduction of low-level results of direct, transposed, and hybrid filter forms. In this work, symmetric and asymmetric FIR filters were synthesized for both application-specific integrated circuits (ASIC) and field programmable gate arrays (FPGA), and experiments were carried out to explore the effect of key parameters in the algorithms and filter architectures on the complexity of FIR filters.

Among many others, three important results drawn from experiments can be stated as follows: (i) it is well known that the shift-adds designs of filters with the use of high-level synthesis algorithms yield significant savings in area with respect to filters using generic multipliers in custom circuits. Our experimental results showed that this is also true for FPGAs when an FIR filter has a number of coefficients larger than the number of DSP48 slices in the target device, which support many independent

functions including a multiplier; (ii) for the multiplierless design of filters in custom circuits, the transposed form filters occupy more area and consume more power than the direct form filters, but still having less delay. Also, the hybrid form, which is best suitable for asymmetric filters, offers intermediate designs occupying less area than the transposed form and having less delay than the direct form; (iii) for the multiplierless design of filters on FPGAs, the transposed form is the best architecture in terms of delay and is generally better than the direct and hybrid forms in terms of power dissipation. However, the target FPGA device with its available resources and the bitwidth of the filter input play key roles on the area of the filter design.

The rest of this article proceeds as follows. Section 2 gives the main concepts related to the multiplierless constant multiplications and presents prominent high-level synthesis algorithms. Section 3 introduces different forms of FIR filters and describes how constant multiplications in each form can be implemented under a shift-adds architecture. Experimental results on different forms of FIR filters synthesized in custom and reconfigurable circuits are presented in Sect. 4, and finally, this article is concluded in Sect. 5.

2 Multiplierless Constant Multiplications

This section introduces different types of constant multiplications that appear in direct, transposed, and hybrid form FIR filters as well as in many DSP systems and presents algorithms designed for their shift-adds realizations.

2.1 Types of Constant Multiplications

The multiplication of data samples by constant coefficients in DSP systems can be categorized in four main classes as illustrated in Fig. 2:

1. The *single constant multiplication* (SCM) operation realizes the multiplication of a single coefficient c by a single variable x, i.e., $y = cx$. It is frequently used in the

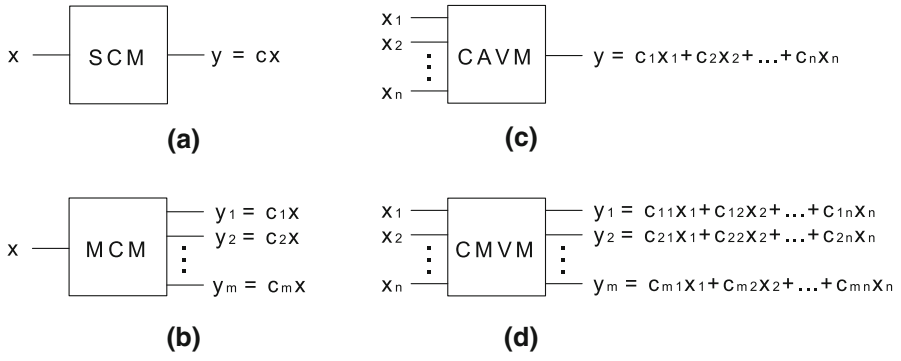

Fig. 2 Four types of constant multiplications: **a** SCM; **b** MCM; **c** CAVM; **d** CMVM

design of fast Fourier transforms (FFTs) [51] and fast discrete cosine transforms (DCTs) [53].

2. The *multiple constant multiplications* (MCM) operation computes the multiplication of a set of m constants C by a single variable x, i.e., $y_j = c_j x$ with $1 \leq j \leq m$. It occurs in the transposed form FIR filters [39].
3. The *constant array-vector multiplication* (CAVM) operation implements the multiplication of a $1 \times n$ constant array C by an $n \times 1$ input vector X, i.e., $y = \sum_k c_k x_k$ with $1 \leq k \leq n$. It appears in infinite impulse response (IIR) and direct form FIR filters [32].
4. The *constant matrix-vector multiplication* (CMVM) operation realizes the multiplication of an $m \times n$ constant matrix C by an $n \times 1$ input vector X, i.e., $y_j = \sum_k c_{jk} x_k$ with $1 \leq j \leq m$ and $1 \leq k \leq n$. It is used in the design of linear DSP transforms [10] and hybrid form FIR filters [28].

Observe that the CMVM operation is the most general case of constant multiplications and corresponds to an SCM operation when both m and n are 1, to an MCM operation when $m > 1$ and n is 1, and to a CAVM operation when m is 1 and $n > 1$.

2.2 High-Level Synthesis Algorithms

A straightforward way of realizing constant multiplications under a shift-adds architecture, called the digit-based recoding (DBR) technique [19], is first to define the constants under a particular number representation, namely, binary or canonical signed digit (CSD)[1] [32], and second, for the nonzero digits in the representation of constants, is to shift the input variables according to digit positions and add/subtract the shifted variables with respect to digit values.

As a simple MCM example, consider the constant multiplications $51x$ and $77x$. The decompositions of constants under CSD are given as follows:

$$\begin{aligned} 51x &= (10\bar{1}010\bar{1})_{CSD}x = x \ll 6 - x \ll 4 + x \ll 2 - x \\ 77x &= (1010\bar{1}01)_{CSD}x = x \ll 6 + x \ll 4 - x \ll 2 + x \end{aligned}$$

which lead to a design with 6 operations, as shown in Fig. 3a.

As a CMVM example, consider the linear transforms, $y_1 = 11x_1 + 17x_2$ and $y_2 = 19x_1 + 33x_2$. The constants in the linear transforms are defined under CSD as follows:

$$\begin{aligned} y_1 &= 11x_1 + 17x_2 = (10\bar{1}0\bar{1})_{CSD}x_1 + (10001)_{CSD}x_2 \\ &= x_1 \ll 4 - x_1 \ll 2 - x_1 + x_2 \ll 4 + x_2 \\ y_2 &= 19x_1 + 33x_2 = (1010\bar{1})_{CSD}x_1 + (100001)_{CSD}x_2 \\ &= x_1 \ll 4 + x_1 \ll 2 - x_1 + x_2 \ll 5 + x_2 \end{aligned}$$

[1] 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. The nonzero digits are not adjacent, and a constant is represented with minimum number of nonzero digits under CSD.

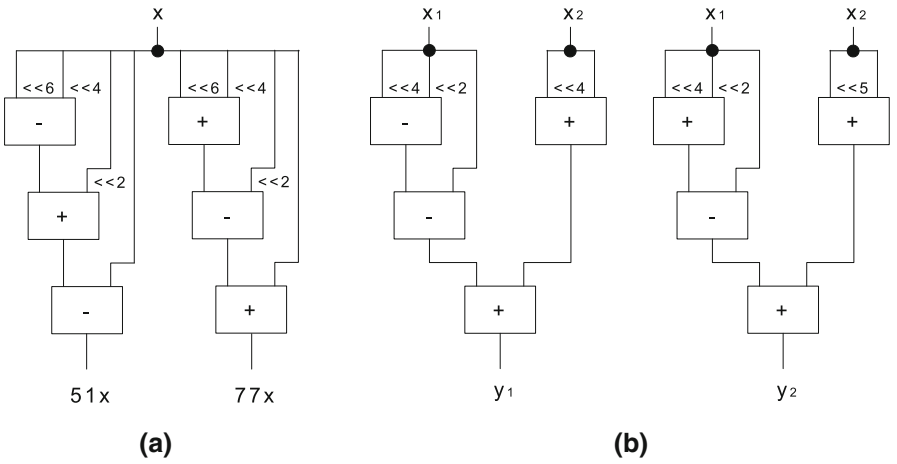

Fig. 3 Shift-adds implementations of constant multiplications using the DBR technique [19]: **a** an MCM operation realizing $51x$ and $77x$; **b** a CMVM operation realizing $y_1 = 11x_1 + 17x_2$ and $y_2 = 19x_1 + 33x_2$

which require 8 operations, as depicted in Fig. 3b.

In the shift-adds design of constant multiplications, the fundamental optimization problem is to find the minimum number of adders/subtractors that realize the constant multiplications, since shifts can be realized using only wires which represent no hardware cost. Note that this is an NP-complete problem even in the case of SCM [12]. The proposed algorithms aim to maximize the sharing of partial products and are generally grouped in two categories based on the search space where they look for a solution.

1. The *common subexpression elimination* (CSE) methods [1,6,10,32,34,35,42,43,49,50,57,59] define the constants under a number representation, such as, binary, CSD, or minimal signed digit (MSD)[2] [49]. 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. Their main drawback is their dependency on a number representation.
2. The *graph-based* (GB) techniques [7,15,17,26,29,30,47,53,54] are not restricted to any particular number representation and aim to find intermediate subexpressions that enable to realize the constant multiplications with minimum number of operations. They consider a larger number of realizations of a constant and obtain better solutions than the CSE methods, but requiring more computational resources, due to the larger search space.

Additionally, hybrid techniques combine methods from both CSE and GB algorithms [5] and increase the number of possible implementations of a constant obtained by a CSE method considering alternative realizations [18,34].

[2] MSD differs from CSD in one property which allows the nonzero digits to be adjacent. Thus, a constant may have alternative representations in MSD, all including minimum number of nonzero digits.

Observe from Fig. 2 that although any algorithm designed for a CMVM instance can be applied to an SCM, MCM, or CAVM instance, there is no exact algorithm that finds the global minimum solution in terms of the number of operations for a CMVM instance, while the exact algorithms of [53] and [7] can be applied to real size SCM and MCM instances, respectively. This is because the complexity of the problem increases from SCM to CMVM as the number of input and output variables increases. Hence, in the next two subsections, we first describe prominent algorithms designed for the SCM and MCM operations and then, those designed for the CAVM and CMVM instances.

2.2.1 Methods for the Multiplierless Design of SCM and MCM Operations

For the shift-adds design of SCM operations, the CSE heuristic of [42] initially defines the constant under CSD and then, in each iteration, it finds the "best" subpattern with the maximal number of terms having at least 2 occurrences and replaces its occurrences in the representation of the constant. Due to its polynomial time complexity, it can be applied to constants with thousands of bits. The minimum number of operations required for all constants up to 12 bits was found by the exhaustive search algorithm of [17], and this approach was extended to constants up to 19 bits in [29]. The exhaustive search technique of [53] equipped with efficient search pruning techniques can easily find the minimum solution of 32-bit constants.

For our example in Fig. 3a, $51x$ requires minimum 2 operations, such as, $3x = x \ll 1 + x$ and $51x = 3x \ll 3 + 3x$, and $77x$ needs minimum 3 operations, such as, $31x = x \ll 5 - x$, $93x = 31x \ll 2 - 31x$, and $77x = 93x - x \ll 4$, which lead to a total of 5 operations for both constant multiplications.

For the shift-adds design of MCM operations, the exact CSE algorithms of [23,31] define the constants under a number representation, obtain all possible implementations of constant multiplications that are extracted from the representations of constants, and formalize the problem of finding the minimum number of operations as a 0-1 integer linear programming (ILP) problem. The problem reduction and model simplification techniques for the exact CSE algorithms were presented in [1,34]. On the other hand, the prominent CSE heuristics of [32,49,50,57] iteratively find the most common subexpressions and replace them in constant multiplications.

Returning to our example in Fig. 3a, the exact CSE algorithm [1] obtains a solution with 4 operations when constants are defined under CSD by finding the most common subexpression $17x = (10001)_{CSD}x$, as shown in Fig. 4a.

The exact GB algorithms of [7] search a minimum solution in breadth-first and depth-first manners. The prominent approximate GB algorithms of [15,54] include two parts, optimal and heuristic. In their optimal parts, each constant, which can be implemented with a single operation, is synthesized. If there exist unimplemented constants left, then they switch to their heuristic parts where the required intermediate constants are found iteratively. The algorithm of [15] initially chooses a single unimplemented constant with the smallest single coefficient cost [17] and then synthesizes it with a single operation including one(two) intermediate constant(s) that has(have) the smallest value. The algorithm of [54] selects a single intermediate con-

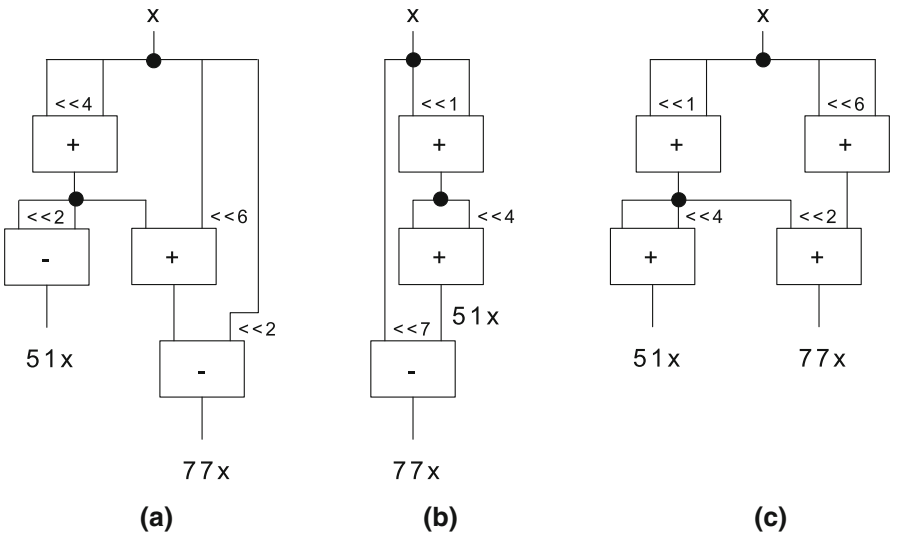

Fig. 4 Shift-adds design of $51x$ and $77x$: **a** the exact CSE method [1]; **b** the exact GB method [7]; **c** the algorithm [3] under a delay constraint

stant that yields the best cumulative benefit over all unimplemented constants for their implementations.

For our example in Fig. 3a, the exact GB algorithm [7] finds a solution with 3 operations, exploiting the intermediate constant 3 (Fig. 4b).

However, the algorithms designed for the minimization of the number of operations cannot guarantee that their solutions always lead to a design with optimal area at gate-level, since they do not take into account the implementation cost of each adder/subtractor. Hence, in [3,38], the complexity of each adder and subtractor in MCM was formulated in terms of basic logic blocks and the algorithms, which find a set of operations realizing the constant multiplications and leading to an MCM design with the smallest gate-level area, were introduced. A similar approach [4,11] was applied to the multiplierless design of constant multiplications on FPGAs, where the proposed methods find a set of operations that require minimum number of 4-input look-up tables (LUTs).

Although the delay of a shift-adds design is dependent on several issues, such as circuit technology, placement, and routing, at this stage, it is considered as the number of adder-steps, i.e., the maximal number of adders/subtractors in series that produce any constant multiplication. The minimum adder-steps of an SCM, cx, is computed as $MAS_{SCM} = \lceil log_2 S(c) \rceil$, where $S(c)$ denotes the number of nonzero digits in the CSD representation of c. Thus, given an MCM operation consisting of a set of m constants, its minimum adder-steps is determined as $MAS_{MCM} = \max_j \{\lceil log_2 S(c_j) \rceil\}$ with $1 \leq j \leq m$ [27,39]. Hence, given a delay constraint dc, greater than or equal to MAS_{MCM}, the algorithms of [1,3,34,39] can minimize the number of operations or gate-level area realizing the constant multiplications without violating dc.

For our example in Fig. 3a, the minimum adder-steps of both $51x$ and $77x$ are 2. When dc is set to 2, the GB algorithm [3] finds a solution with 4 operations, as shown

in Fig. 4c. Thus, a solution with one more operation than the minimum (Fig. 4b) is obtained, but with one less adder-step.

Moreover, as shown in power dissipation estimation models [13,14] and in algorithms [2,16], the area and depth of each operation in an MCM design have a significant impact on its power consumption. This is because a larger number of logic gates produce more transitions, and the transitions generated at the output of an operation produce more transitions on the next level operations, generating and propagating more glitching along the reconvergent paths. Thus, the algorithms of [2,21,37] aim to find a realization of an MCM operation, where each constant is implemented at its minimum depth, minimizing the number of operations or gate-level area.

2.2.2 Methods for the Multiplierless Design of CAVM and CMVM Operations

To the best of our knowledge, there is no algorithm that directly targets the optimization of the multiplierless design of a CAVM operation. In practice, a CMVM algorithm described in this section is used for its shift-adds design.

For the shift-adds design of CMVM operations, given a set of linear transforms, the CSE algorithm of [59] finds all possible implementations of linear transforms by extracting only the 2-term subexpressions and formalizes the problem of maximizing the sharing of subexpressions as a 0-1 ILP problem. The exact CSE algorithm of [6] follows a similar approach but considers all possible realizations of linear transforms. However, these CSE algorithms can only be applied to CMVM instances with small size of constant matrices due to the exponential growth in the size of 0-1 ILP problems. The CSE heuristics of [35,43] iteratively find the most common 2-term subexpression and replace it within the linear transforms. They differ in the selection of subexpressions that have the same number of occurrences. The CSE algorithm [10] relies on an efficient CSE algorithm [42] that iteratively searches a subexpression with the maximal number of terms and with at least 2 occurrences. The CSE heuristic of [9] chooses its subexpressions based on a cost value which is computed as the product of the number of terms in the subexpression and the number of its occurrences in the linear transforms. In turn, the GB algorithm of [30] initially computes the differences between each two linear transforms and determines their implementation cost values. Then, it uses a minimum spanning tree algorithm to find the realizations of linear transforms with differences, that have the minimum cost, and replaces the linear transforms with the required differences. Furthermore, the hybrid algorithm of [5] iteratively finds the most promising differences of linear transforms and applies an improved CSE heuristic to further reduce the complexity of the CMVM operation.

For our example in Fig. 3b, the algorithm of [5] obtains a solution with 5 operations by exploiting $x_1 + 2x_2$, as shown in Fig. 5a.

Although there exists no algorithm that can find a set of adders/subtractors realizing a CMVM operation using optimal gate-level area, the algorithms of [5,10] include hardware optimization techniques which take into account the gate-level implementation costs of adders/subtractors.

The minimum adder-steps of a CAVM operation, i.e., a linear transform $y = c_1x_1 + c_2x_2 + \ldots + c_nx_n$, is computed as $MAS_{CAVM} = \lceil log_2(\sum_k S(c_k)) \rceil$ with

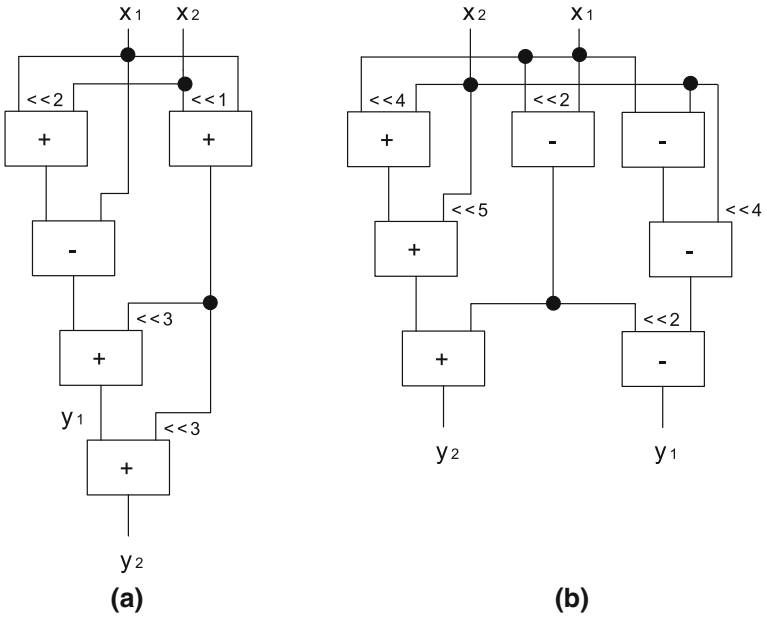

Fig. 5 Shift-adds design of $y_1 = 11x_1 + 17x_2$ and $y_2 = 19x_1 + 33x_2$ using the algorithm [5]: **a** without a delay constraint; **b** with a delay constraint

$1 \leq k \leq n$. The minimum adder-steps of a CMVM operation, i.e., a set of linear transforms, is computed as $MAS_{CMVM} = \max_j\{\lceil log_2(\sum_k S(c_{jk}))\rceil\}$ with $1 \leq j \leq m$ and $1 \leq k \leq n$ [27]. Thus, given a delay constraint dc, greater than or equal to MAS_{CMVM}, the algorithms [5,36] can find the fewest number of operations that realize the CMVM operation without exceeding dc.

For our example in Fig. 3b, the minimum adder-steps of y_1 and y_2 is computed as 3. Given the delay constraint equal to 3, the algorithm of [5] obtains a solution with 7 operations, as depicted in Fig. 5b. Thus, a solution with 2 more operations, but one less adder-step than that of Fig. 5a is obtained.

Observe from Figs. 4 and 5 that since the reduction of the number of adder-steps generally increases the number of operations [39], the algorithms, which can handle a delay constraint, can explore the tradeoff between area and delay of a shift-adds design by changing the delay constraint.

3 FIR Filter Architectures

3.1 Multiplierless Design of Direct Form FIR Filters

The shift-adds design of a direct form FIR filter can be realized not only by sharing the common subexpressions among the constant multiplications but also by sharing of previous values of the filter input [10,32,58]. However, although the number of operations can be reduced slightly, the number of registers may be increased significantly

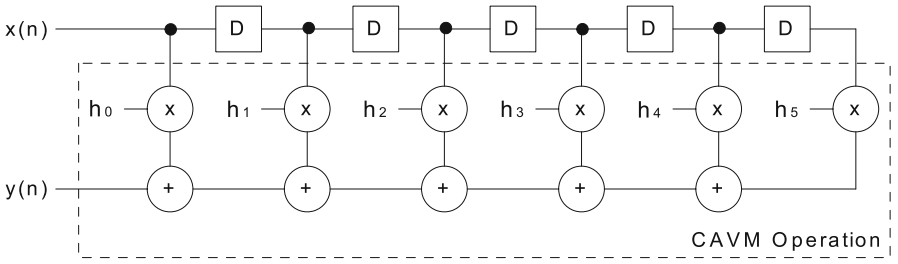

Fig. 6 Design of a 6-tap direct form FIR filter

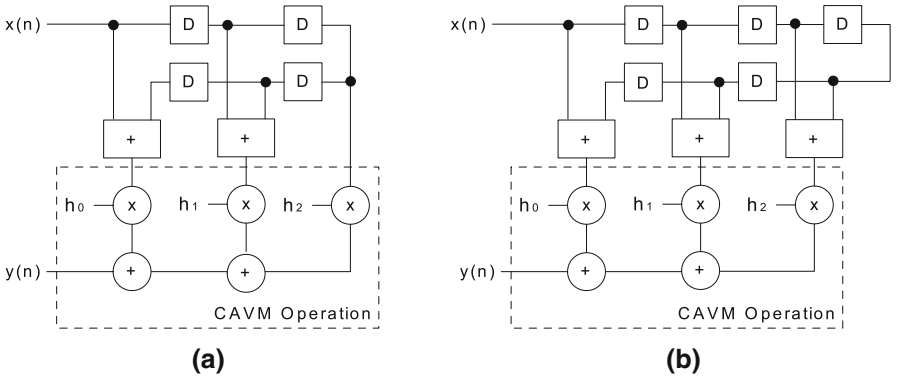

Fig. 7 Design of direct form FIR filters with symmetric coefficients: **a** with 5 coefficients; **b** with 6 coefficients

in this case [10]. Hence, in this work, the multiplications of filter coefficients by the latest N filter input values and their summations are regarded as a CAVM operation, where these latest N filter input values are considered as input variables of the CAVM operation, and the output of the CAVM operation is the filter output, as shown in Fig. 6

Some a-prior hardware modifications can be considered for symmetric filters, although the high-level synthesis algorithms can bring out the same realizations. Since the same constants are multiplied by different input variables, they can be added first, and then, the outputs of these adders can be assigned to the inputs of the CAVM operation, as shown in Fig. 7. In this case, the number of inputs of the CAVM operation is equal to $\lceil N/2 \rceil$, and the number of adders outside the CAVM operation is equal to $\lfloor N/2 \rfloor$, which is due to the even or odd number of filter coefficients. These adders always generate an output with a bitwidth equal to the bitwidth of the filter input plus 1.

In its hardware description under a shift-adds architecture, first, one of CMVM methods given in Sect. 2.2.2 is applied to the CAVM operation including the filter coefficients, and the CAVM operation is described based on the found adders/subtractors. Then, the filter is described including the CAVM operation as well as the necessary circuit consisting of the registers, and also, the adders outside the CAVM operation, if the filter is symmetric.

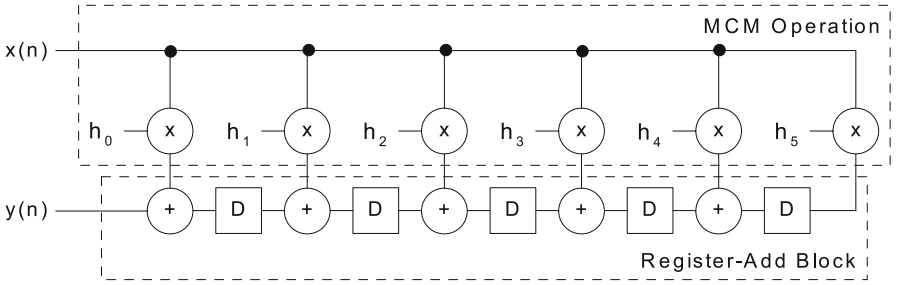

Fig. 8 Design of a 6-tap transposed form FIR filter

3.2 Multiplierless Design of Transposed Form FIR Filters

In the transposed form FIR filter, the multiplication of coefficients by the filter input is regarded as an MCM operation, as shown in Fig. 8. In its hardware description under a shift-adds architecture, initially, one of MCM algorithms introduced in Sect. 2.2.1 is applied to the filter coefficients, and a solution is found. Then, the MCM operation is described based on the found adders/subtractors. Finally, the filter is described including the MCM operation, and the registers and adders in the register-add block of the filter. Note that the MCM methods convert the target constants to positive and odd numbers, eliminate the repeated target constants in their preprocessing phase, and generate a solution based on this reduced set of constants. Thus, whenever an even or a negative version of an output of the MCM operation is required at the input of an adder in the register-add block, the associated output is simply shifted or negated, respectively. Observe that the symmetric coefficients are handled naturally by the MCM algorithm, requiring no dedicated hardware.

Further reduction of hardware in the register-add block can be obtained using the technique of [20]. In this approach, whenever a large partial number at the output of an register is added to a set of small outputs of the MCM operation, the large partial number is divided into two parts. Its most significant part is passed through registers, requiring additional registers, and its least significant part is added to these small outputs of the MCM operation, requiring smaller size adders. Finally, its most significant part is added to this sum. In this case, the hardware for addition operations can be reduced so significantly that the increase in the area of registers can be compensated.

3.3 Multiplierless Design of Hybrid Form FIR Filters

A straightforward way of generating a hybrid form FIR filter is to keep/put some registers in the direct form on the input-line[3]/output-line[4] [40]. Thus, there exist $2^{N-1}-2$ different hybrid forms of an FIR filter. In this work, we present three types of the hybrid form in order to explore the tradeoffs between the direct and transposed forms.

[3] The top signal path carrying the delayed filter input in Fig. 1a.

[4] The bottom signal path including the adders that compute the filter output in Fig. 1a.

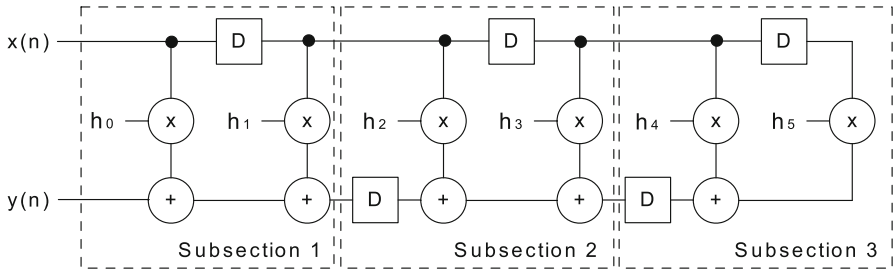

Fig. 9 Design of a 6-tap *Hybrid-I-3* FIR filter

Note that by changing the value of their parameters, many alternative hybrid form FIR filters can be generated.

3.3.1 Hybrid Form I

The hybrid form FIR filter can be seen as a series of cascading *subsections*, where each subsection has one and only one register on the output-line[5], as shown in Fig. 9. The algorithm of [40] determines the subsections of the hybrid form such that the given cycle-time constraint is not violated. However, the multiplierless realization of constant multiplications is not considered in [40].

The first type of the hybrid form filter, called *Hybrid-I-s*, is obtained by dividing the filter into s subsections, where each subsection includes at least 2 filter coefficients, which is to increase the sharing of partial products among the constant multiplications. In general, s is a user-defined parameter to be in between 2 and $\lfloor N/2 \rfloor$. If N is a multiple of s, each subsection includes the same number of coefficients, i.e., N/s. Otherwise, the number of filter coefficients in the subsections, except the one computing the filter output, is determined as $\lfloor N/s \rfloor$. Thus, the one that computes the filter output has $N - \lfloor N/s \rfloor (s-1)$ filter coefficients. Hence, when s is increased, the number of filter coefficients in each subsection is decreased, reducing the partial product sharing among the constant multiplications, thus yielding a filter design that includes a larger number of operations and a greater number of registers on the output-line. However, the critical path is decreased in this case.

In the hardware description of this hybrid form FIR filter, given the number of subsections s, initially the filter coefficients in each subsection are determined, and the summations of coefficient multiplications are regarded as CAVM operations. Then, one of CMVM algorithms given in Sect. 2.2.2 is applied to each CAVM operation, a solution is found, and each subsection is described including the adders/subtractors found for the CAVM operation, and the registers on the input-line and output-line as well. Finally, the filter is described by connecting these subsections.

[5] The adder that computes the filter output is always assumed to be connected to a register even this register is not synthesized in hardware.

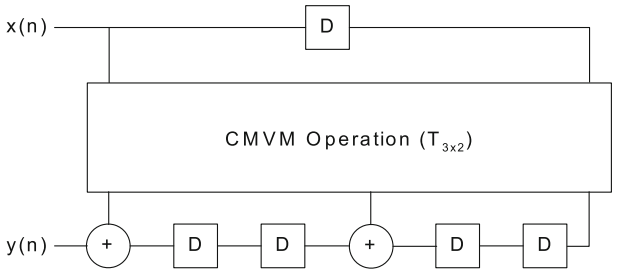

Fig. 10 Design of a 6-tap *Hybrid-II-2* FIR filter

3.3.2 Hybrid Form II

The second type of the hybrid form FIR filter, called *Hybrid-II-k*, is obtained by dividing its transfer function in the *z-domain* into subsections of k consecutive taps [28]. As a simple example, consider a 6-tap FIR filter and suppose that k is 2. The transfer function can be arranged as

$$\begin{aligned} H(z) &= h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + h_5 z^{-5} \\ &= h_0 + h_1 z^{-1} + (h_2 + h_3 z^{-1}) z^{-2} + (h_4 + h_5 z^{-1}) z^{-4} \end{aligned}$$

so that the filter output can be written as

$$Y(z) = [1 \quad z^{-2} \quad z^{-4}] \cdot T \cdot \begin{bmatrix} 1 \\ z^{-1} \end{bmatrix} \cdot X(z), \quad T = \begin{bmatrix} h_0 & h_1 \\ h_2 & h_3 \\ h_4 & h_5 \end{bmatrix}$$

This implementation is illustrated in Fig. 10, where the CMVM operation realizes the multiplication of the constant matrix T by the latest 2 filter inputs. If N is not a multiple of k, additional $mk - N$ coefficients, whose values are equal to 0, must be generated, where m is $\lceil N/k \rceil$. Thus, the constant matrix T is formed as follows:

$$T_{m \times k} = \begin{bmatrix} h_0 & h_1 & \dots & h_{k-1} \\ h_k & h_{k+1} & \dots & h_{2k-1} \\ \vdots & \vdots & & \vdots \\ h_{(m-1)k} & h_{(m-1)k+1} & \dots & h_{mk-1} \end{bmatrix}$$

Hence, the number of adders on the output-line is $m - 1$, the number of registers on the input-line and output-line is $k - 1$ and $(m - 1)k$, respectively, a total of $mk - 1$ registers.

Since the complexity of a CMVM operation depends on its constants and the high-level synthesis algorithm, it is hard to predict its complexity from the size of the constant matrix. However, we observed that the constant matrices with a large number of rows and a small number of columns lead to less number of operations than those which have a small number of rows and a large number of columns. This is simply

because the sharing of partial products increases as the number of input variables (corresponds to the number of columns) decreases and the number of linear transforms (corresponds to the number of rows) increases. Hence, as k is increased (decreasing the value of m), the complexity of the CMVM operation increases, although the number of adders on the output-line is decreased. In this case, the number of registers on the input-line is also increased (decreasing the number of registers on the output-line), which decreases the complexity of the whole filter, since the size of input-line registers is generally less than those on the output-line.

In the hardware description of this hybrid form filter, initially, the constant matrix T is generated, the multiplierless design of the CMVM operation is found using one of CMVM algorithms introduced in Sect. 2.2.2, and the CMVM operation is described based on the found adders/subtractors. Then, the filter is described including the CMVM operation and the necessary registers and adders.

3.3.3 Hybrid Form III

The third type of the hybrid form FIR filter, called *Hybrid-III-r*, can be obtained by dividing its transfer function in the *z-domain* into subsections of r successive taps [28]. As a simple example, again consider a 6-tap FIR filter and suppose that r is 2. The transfer function can be arranged as

$$\begin{aligned} H(z) &= h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + h_5 z^{-5} \\ &= h_0 + h_2 z^{-2} + h_4 z^{-4} + (h_1 + h_3 z^{-2} + h_5 z^{-4}) z^{-1} \end{aligned}$$

so that the filter output can be written as

$$Y(z) = [1 \quad z^{-1}] \cdot S \cdot \begin{bmatrix} 1 \\ z^{-2} \\ z^{-4} \end{bmatrix} \cdot X(z), \quad S = \begin{bmatrix} h_0 & h_2 & h_4 \\ h_1 & h_3 & h_5 \end{bmatrix}$$

whose implementation is illustrated in Fig. 11. If N is not a multiple of r, additional $rp - N$ filter coefficients, whose values are equal to 0, must be generated, where p is $\lceil N/r \rceil$. Thus, the constant matrix S is formed as follows:

$$S_{r \times p} = \begin{bmatrix} h_0 & h_r & \dots & h_{r(p-1)} \\ h_1 & h_{r+1} & \dots & h_{r(p-1)+1} \\ \vdots & \vdots & & \vdots \\ h_{r-1} & h_{2r-1} & \dots & h_{rp-1} \end{bmatrix}$$

In this hybrid form, the number of adders on the output-line is $r-1$, and the number of registers on the input-line and output-line is $(p-1)r$ and $r-1$, respectively, a total of $pr-1$ registers. Note that when r is increased (decreasing p), the number of adders on the output-line increases, but the complexity of the CMVM operation decreases. In this case, the number of registers on the output-line increases, and the number of registers on the input-line decreases.

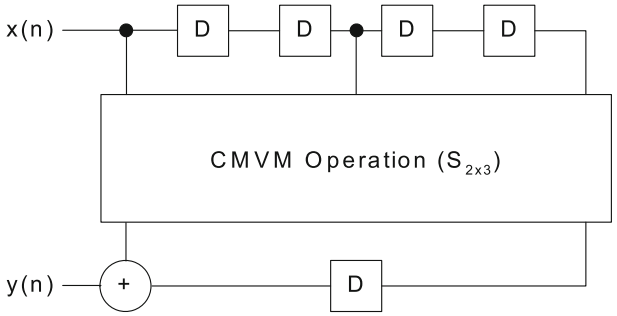

Fig. 11 Design of a 6-tap *Hybrid-III-2* FIR filter

In the hardware description of this hybrid form filter, initially, one of CMVM algorithms introduced in Sect. 2.2.2 is applied to the constant matrix S, and a solution is obtained. Then, the CMVM operation is described based on the found adders/subtractors. Finally, the filter is described including the CMVM operation and the necessary registers and adders.

Note that the number of registers in *Hybrid-III-r* (*Hybrid-II-k*) is larger than those of direct, transposed, and *Hybrid-I-s* filters when N is not a multiple of r (k).

4 Experimental Results

This section is divided into two subsections. In the first, symmetric FIR filters are used, and the results of direct and transposed forms are introduced. The results of hybrid form filters are not presented in this subsection, since the symmetric coefficients are not utilized appropriately in the given types of the hybrid form in the same natural way as in the direct and transposed forms. In the second, asymmetric filters are used, and the results of direct, transposed, and hybrid form filters are given. The results of different filter forms were obtained using different bitwidths of the filter input, design libraries, target FPGA devices, and optimization criteria in a high-level synthesis algorithm.

For the multiplierless design of MCM, CAVM, and CMVM operations in FIR filters, the state-of-art algorithms of [3,5] were used. Since these algorithms are already compared with other prominent algorithms in [3,5], and in order not to divert the reader from the main topic of this article (to investigate the impact of key parameters of filter architectures on the complexity of FIR filters), no comparison between different algorithms is presented here. The reader is referred to the papers [1–3,5–7,10,34,39,54] for the comparisons of state-of-art MCM and CMVM algorithms in terms of the number of operations and adder-steps, and gate-level area, delay, and power dissipation.

Also, in order to make a fair comparison between different FIR filter forms, the truncation method [22,25,56], which is generally used to reduce the complexity of the filter design sacrificing the accuracy of the result, was not allowed, neither on the filter output nor on a register/adder in the filter design.

Table 1 Specifications of filters with symmetric coefficients

Filter	N	Pass	Stop	Q
1	40	0.15	0.25	16
2	60	0.15	0.20	16
3	80	0.12	0.15	16
4	100	0.18	0.20	16
5	120	0.20	0.22	16

FIR filters were described in VHDL, and the Synopsys Design Compiler and the Xilinx ISE Design Suite 13.1 were used as synthesis tools for the custom and reconfigurable designs, respectively. In the synthesis script, relaxed timing constraints were used in order to provide more freedom to the tools to optimize area. The functionality of filters was verified on 10,000 randomly generated input signals in simulation, from which the switching activity information, which was used by the tools to compute the power dissipation, was also obtained.

4.1 Results of Symmetric Filter Designs

Table 1 presents five low-pass filters with symmetric coefficients computed by the *firgr* function of MATLAB, where N is the filter length, $pass$ and $stop$ are the normalized passband and stopband frequencies, respectively, and Q is the quantization value used to convert the floating-point coefficients to integers.

4.1.1 Experiments with Different Optimization Criteria and Filter Architectures

The experimental settings were defined as

1. The bitwidth of the filter input (*bwi*) is 16. Thus, the bitwidth of the filter output[6] (*bwo*) of Filter 1 is 34, and the others have an output with 35 bits.
2. For custom designs, the UMCLogic 180 nm Generic II library is used.
3. For the reconfigurable designs, the Virtex 4 FPGA xc4vfx12-12sf363 is used. A Virtex 4 slice consists of two LUTs and two flip-flops, and this target device includes 32 DSP48 slices, each having an 18-bit by 18-bit two's-complement multiplier.

In this experiment, the FIR filters given in Table 1 were designed using generic multipliers (*GM*), where the multiplication of coefficients by the filter input or its time-shifted versions was described as constant multiplications in VHDL, which was implemented by the synthesis tools. In their direct forms, the additions of filter coefficient multiplications (Eq. 1) were organized in a binary tree to reduce the delay of the filter design. These filters were also designed under a shift-adds architecture, where the solutions of algorithms [5] and [3] were respectively used to realize the CAVM and MCM operations in the direct and transposed forms. These algorithms were run

[6] The bitwidth of the filter output is computed as $bwi + \lceil log_2 \sum_{i=0}^{N-1} |h_i| \rceil$, where h_i is the ith fixed-point filter coefficient with $0 \leq i \leq N-1$.

without a delay constraint (*SA-A*) and with a delay constraint (*SA-D*), which was set to the minimum adder-steps of the operation as described in Sect. 2.2.

Table 2 presents the high-level and low-level results of the direct and transposed forms of FIR filters. For the high-level results, we present the number of operations in the multiplier block (*MBO*) and the number of adder-steps (*AS*) in the multiplier block. In the direct (transposed) form, they denote the number of adders/subtractors in the CAVM (MCM) block and its number of adder-steps, respectively. For all filter designs, *TO* denotes the total number of adders/subtractors. This value includes the $N-1$ adders in both filter forms using generic multipliers, the $\lfloor N/2 \rfloor$ adders used for the symmetric coefficients in the multiplierless design of the direct form and the $N-1$ adders in the register-add block of the multiplierless design of the transposed form. For the custom designs, we present the combinational area (*CA*), non-combinational area (*NCA*), and the total area (*TA*), all in mm^2. The delay (*D*) in the critical path in *ns* and the total dynamic power dissipation (*P*) in *mW* are also given. For the reconfigurable designs, the number of LUTs (*LUTs*), the number of flip-flops (*FFs*), the number of slices (*Sls*), and the number of DSP48 slices (*DSP48s*) are shown. Again, *D* denotes the delay in *ns* and *P* stands for the total power dissipation in *mW*. The best values of *TA*, *Sls*, *D*, and *P* in each filter form are given in bold.

For the **custom designs**, the multiplierless design of both direct and transposed form FIR filters leads to significant savings in area with respect to filters using generic multipliers, where the maximum gain is 41 %, achieved on the direct form of Filter 4. However, there exist filter designs using generic multipliers that have the smallest delay. This is due to a large number of operations in series in the shift-adds designs. Also, the direct form leads to less complex FIR filters when compared to the transposed form, which is due to two main reasons: (i) although the number of registers ($N-1$) is the same in both forms, in the direct form, the size of all registers is equal to *bwi*, i.e., 16 in this experiment. In the transposed form, it is increasing and reaching to *bwo*. This fact can be observed on the *NCA* values of the filter designs, where the *NCA* value of the transposed form is always larger than two times of the *NCA* value of the direct form; (ii) although the *TO* value in the direct form is larger than that of the transposed form, the outputs of $\lfloor N/2 \rfloor$ adders of the direct form, which is at most 30 % of its *TO* value for the shift-adds designs, are $bwi+1$ bits wide, i.e., 17 in this experiment. In turn, the $N-1$ adders of the transposed form, which is at most 67 % of its *TO* value for the shift-adds designs, are in the register-add block, and their sizes are increasing up to *bwo*. This fact can be observed from the *CA* results, where the direct form has always less value than the transposed form. Although the delay of a multiplierless FIR filter also depends on the adder architecture used by the synthesis tool, the reduction of the number of adder-steps in the MCM operation of the transposed form decreases its delay significantly, where the maximum gain is 35 %, achieved on Filter 3, taking into account a slight increase in gate-level area. This is also valid for the direct form filters, except Filters 3 and 4. Note that there is always a transposed form filter that has less delay than its relevant direct form. Furthermore, the direct form FIR filters consume less power than the transposed form, which is mainly due to less area.

For the **reconfigurable designs**, both filter forms using generic multipliers require less number of slices when the number of coefficients is less than 80 with respect to the multiplierless filters. Recall that these are symmetric filters, and the total number

Table 2 Summary of results of FIR filters with symmetric coefficients using a 16-bit filter input

Filter	Form	Arch/Algo	High-level results			Low-level results on custom designs					Low-level results on reconfigurable designs				
			MBO	AS	TO	CA	NCA	TA	D	P	LUTs	FFs	Sls (DSP48s)	D	P
1	Direct	GM			39	65.2	10.6	75.8	**9.6**	**1.7**	810	624	**788 (19)**	28.1	**377**
		SA-A [5]	54	18	74	37.7	10.6	**48.3**	10.1	1.8	1476	624	1096	35.1	449
		SA-D [5]	54	7	74	38.9	10.6	49.5	9.8	**1.7**	1517	624	1104	**23.8**	392
	Trans.	GM			39	71.5	21.1	92.6	**8.1**	9.5	1185	1240	**655 (19)**	**10.9**	**349**
		SA-A [3]	26	6	65	45.4	21.1	**66.5**	10.2	**6.6**	1818	1240	973	12.9	384
		SA-D [3]	31	3	70	46.5	21.1	67.6	8.5	7.2	1857	1240	997	11.0	356
2	Direct	GM			59	103.5	16.0	119.5	10.4	**2.4**	1456	944	**1284 (26)**	28.2	403
		SA-A [5]	79	20	109	55.7	16.0	**71.7**	11.5	**2.4**	2194	944	1629	36.8	447
		SA-D [5]	79	7	109	58.3	16.0	74.3	**10.1**	2.5	2275	944	1643	**25.2**	**402**
	Trans.	GM			59	113.4	32.9	146.3	**8.5**	15.0	1888	1920	**1012 (29)**	12.0	379
		SA-A [3]	33	8	92	67.2	32.9	**100.1**	12.4	10.3	2666	1920	1413	14.9	388
		SA-D [3]	38	3	97	68.2	32.9	101.1	9.1	**10.2**	2699	1920	1431	**11.0**	**369**
3	Direct	GM			79	143.2	21.5	164.7	**10.4**	**3.3**	2809	1264	2199 (25)	31.4	**431**
		SA-A [5]	105	22	145	76.2	21.5	**97.7**	10.6	**3.3**	2917	1264	**2179**	38.5	519
		SA-D [5]	110	8	150	78.6	21.5	100.1	10.6	**3.3**	3059	1264	2205	**27.2**	459
	Trans.	GM			79	156.4	44.6	201.0	**8.6**	20.7	3130	2625	**1702 (32)**	11.3	**421**
		SA-A [3]	42	10	121	90.9	44.6	**135.5**	13.3	**13.9**	3693	2625	1952	16.3	482
		SA-D [3]	51	3	130	92.5	44.6	137.1	**8.6**	14.7	3659	2625	1941	**10.9**	434

Table 2 continued

Filter	Form	Arch/Algo	High-level results			Low-level results on custom designs					Low-level results on reconfigurable designs				
			MBO	AS	TO	CA	NCA	TA	D	P	LUTs	FFs	Sls (DSP48s)	D	P
4	Direct	GM			99	171.9	26.9	198.8	11.0	**4.0**	3868	1584	2945 (29)	30.4	**471**
		SA-A [5]	123	20	173	89.6	26.9	**116.5**	**10.5**	4.1	3475	1584	**2611**	37.9	548
		SA-D [5]	127	8	177	91.8	26.9	118.7	10.8	**4.0**	3594	1584	2691	**29.4**	485
	Trans.	GM			99	187.8	55.6	243.4	**8.6**	24.9	4425	3270	2422 (32)	11.4	**441**
		SA-A [3]	48	9	147	111.7	55.6	**167.3**	13.0	**18.2**	4399	3270	**2316**	16.8	509
		SA-D [3]	59	3	158	114.5	55.6	170.1	10.0	18.6	4481	3270	2367	**11.1**	448
5	Direct	GM			119	185.9	32.4	218.3	**10.4**	**4.8**	5121	1904	3827 (28)	31.2	**478**
		SA-A [5]	135	22	195	100.6	32.4	**133.0**	11.0	4.9	3861	1904	**2964**	37.7	565
		SA-D [5]	142	8	202	103.7	32.4	136.1	10.5	**4.8**	4023	1904	3061	**26.3**	487
	Trans.	GM			119	205.9	66.4	272.3	**8.7**	27.8	5500	3900	3063 (32)	**11.0**	**468**
		SA-A [3]	57	7	176	130.3	66.4	**196.7**	11.4	21.6	5081	3900	2693	15.6	532
		SA-D [3]	61	3	180	130.7	66.4	197.1	9.6	**21.5**	5060	3900	**2691**	11.1	479

of DSP48 slices in the target device is 32. Thus, in Filters 4 and 5, all the constant multiplications cannot be realized in 32 DSP48 slices, requiring additional LUTs which increase the number of slices. Also, in general, the filters using generic multipliers consume less power with respect to the multiplierless filters, which is due to the use of a DSP48 slice as a generic multiplier and a large number of adder-steps in the shift-adds architectures. Note that no DSP48 slices were used in the multiplierless filter designs, leaving these resources available for any other multiplication operations required in the application. For the filters designed under the shift-adds architecture, although the number of flip-flops and the number of LUTs in the transposed form are greater than those in the direct form, the transposed form filters require significantly fewer slices than the direct form. This is because the configurable logic blocks (CLBs) in the slices of the Virtex 4 FPGA include LUTs and flip-flops. Hence, the registers in the register-add block of the transposed form share the same CLBs of the adders, decreasing the total number of slices significantly. The transposed form also leads to FIR filters with less delay and power dissipation than the direct form. Furthermore, the use of CAVM and MCM operations with the minimum number of adder-steps decreases the delay significantly, which directly reduces the power dissipation in both filter forms. While the maximum gain on delay is obtained as 33 % in the transposed form of Filter 3, the maximum gain on power dissipation is computed as 13 % in the direct form of Filter 5, when the results of *SA-A* and *SA-D* are compared. However, the reduction of the number of adder-steps increases the number of slices, since the number of operations is increased, except those of the transposed forms of Filters 3 and 5. This may occur since the algorithms of [3] do not target the FPGA design platform.

4.1.2 An Experiment with Different Bitwidths of Filter Input

Tables 3 and 4 present the results of the direct and transposed forms of Filter 5 when *bwi* is 8 and 24 on custom and reconfigurable designs, respectively. In this experiment, the other settings given in Sect. 4.1.1 were kept the same.

With respect to the results given in Table 2, as *bwi* is decreased (increased) from 16 to 8 (24), the *bwo* value of the filter is also decreased (increased) from 35 to 27 (43), respectively. Consequently, the complexity of the filter design in terms of area, delay, and power dissipation is decreased (increased) on both custom and reconfigurable circuits. This is due to the fact that *bwi* directly affects the sizes of registers, adders/subtractors, and multipliers in the design.

For the custom designs of Filter 5 under a shift-adds architecture given in Table 3, it is obvious that all the observations made in Sect. 4.1.1 are still valid when *bwi* is decreased or increased. However, for the reconfigurable designs when *bwi* is 8, contrary to the results obtained when *bwi* is 16 and 24, the multiplierless design of the direct form filter requires less number of slices than the multiplierless design of the transposed form, as shown in Table 4. This is because the direct form requires significantly less number of flip-flops than the transposed form. Note that the flip-flops of the direct form need slices to be realized as opposed to those in the transposed form, which are implemented in the same CLBs with adders. When *bwi* is 8, the direct form also consumes less power than the transposed form if the constant multiplications are realized using minimum number of adder-steps. In turn, when *bwi* is 24, all the

Table 3 Summary of results of custom designs with different bitwidths of the filter input

Filter	Form	Arch/Algo	8-bit filter input					24-bit filter input				
			CA	NCA	TA	D	P	CA	NCA	TA	D	P
5	Direct	GM	108.8	16.2	125.0	**8.6**	**2.5**	257.7	48.6	306.3	12.5	**7.1**
		SA-A [5]	60.1	16.2	**76.3**	9.4	2.6	141.0	48.6	**189.6**	12.6	7.2
		SA-D [5]	61.4	16.2	77.6	9.0	**2.5**	146.1	48.6	194.7	**12.1**	**7.1**
	Transposed	GM	128.7	50.2	178.9	**6.9**	18.2	280.1	82.6	362.7	**10.3**	39.8
		SA-A [3]	92.6	50.2	142.8	9.1	**15.4**	167.3	82.6	**249.9**	13.4	**27.6**
		SA-D [3]	92.2	50.2	**142.4**	7.6	**15.4**	168.7	82.6	251.3	11.0	27.9

Table 4 Summary of results of reconfigurable designs with different bitwidths of the filter input

Filter	Form	Arch/Algo	8-bit filter input					24-bit filter input				
			LUTs	FFs	Sls (DSP48s)	D	P	LUTs	FFs	Sls (DSP48s)	D	P
5	Direct	GM	3522	952	2383 (20)	30.3	391	could not be implemented				
		SA-A [5]	2299	952	**1693**	36.4	446	5423	2856	**4237**	39.5	651
		SA-D [5]	2405	952	1758	**24.6**	**371**	5641	2856	4364	**27.9**	**577**
	Transposed	GM	4662	2948	2661 (9)	11.4	**373**	could not be implemented				
		SA-A [3]	3627	2948	**1959**	**11.3**	416	6535	4838	**3417**	16.0	635
		SA-D [3]	3617	2948	1969	12.3	383	6522	4838	3419	**11.0**	**569**

observations made in Sect. 4.1.1 for the multiplierless designs are still valid. We note that the implementation of both filter forms using generic multipliers was not possible due to the limited resources of the target FPGA device.

4.1.3 An Experiment with Different FPGA Devices

Table 5 presents the results of the direct and transposed forms of Filter 5, when the Virtex 5 FPGA xc5vlx50-2ff324 and the Virtex 6 FPGA xc6vlx75T-2ff484 are used as target devices, and *bwi* is set to 16 as done in Sect. 4.1.1. Note that a Virtex 5 slice consists of four LUTs and four flip-flops, which are two times of those in a Virtex 4 slice, and this Virtex 5 device has 48 DSP48E slices. A Virtex 6 slice contains four LUTs and eight flip-flops, 4 more than that of a Virtex 5 slice, and this Virtex 6 device has 288 DSP48E1 slices.

For the filter designs using generic multipliers, considering also the results given in Table 2, which were obtained using a Virtex 4 FPGA, the complexity of both filter forms depends on the number of DSP48 slices used to realize the constant multiplications. For example, while the transposed form leads to a design using less number of slices in Virtex 4 and 5 FPGAs than the direct form, the direct form filter occupies the least

Table 5 Summary of results of reconfigurable designs with different FPGA devices

Filter	Form	Arch/Algo	Virtex 5 FPGA					Virtex 6 FPGA				
			LUTs	FFs	Sls (DSP48Es)	D	P	LUTs	FFs	Sls (DSP48E1s)	D	P
5	Direct	GM	4357	1904	1542 (35)	29.7	**797**	2826	1904	**938 (59)**	29.9	**1477**
		SA-A [5]	3838	1904	**1274**	37.6	870	3785	1904	1085	31.2	1559
		SA-D [5]	4013	1904	1377	**27.7**	812	4021	1904	1146	**23.7**	1498
	Transposed	GM	4236	3886	**1169 (48)**	11.5	**785**	3777	3886	**1018 (57)**	8.8	1516
		SA-A [3]	5094	3886	1402	15.8	848	5020	3886	1332	11.7	1537
		SA-D [3]	5070	3886	1390	**10.9**	804	5054	3886	1354	**8.4**	**1503**

number of slices in a Virtex 6 FPGA, where all the constant multiplications are realized in DSP48E1 slices.

For the multiplierless filter designs, although the number of LUTs and FFs in Virtex 4, 5, and 6 FPGAs are very close to each other in both direct and transposed forms, the difference in the number of slices used in these FPGAs is related to how many LUTs and FFs a slice of a Virtex FPGA includes. Thus, an interesting result observed in this experiment is that the direct form of Filter 5 requires fewer slices than its transposed form in Virtex 5 and 6 FPGAs, as opposed to the results obtained in a Virtex 4 FPGA. However, the transposed form has always the best delay when compared to those of the direct form in these different target FPGA devices. Furthermore, the reduction of the number of adder-steps in the shift-adds design of constant multiplications leads to filters that have less delay and consume less power even different target FPGA devices are used. Finally, the filters synthesized in the Virtex 6 FPGA generally have less delay and consume more power than those implemented in the Virtex 4 and 5 FPGAs.

4.1.4 An Experiment with a Different Design Library

Table 6 presents the low-level results of the custom designs of direct and transposed forms of Filter 5 when the NanGate 45 nm Open Cell library[7] is used. In this experiment, *bwi* is set to 16 as done in Sect. 4.1.1.

As the design library is changed from 180 nm (Table 2) to 45 nm technology, the area, delay, and power dissipation values of the filter design are decreased, but all the observations made in Sect. 4.1.1 are still valid.

4.2 Results of Asymmetric Filter Designs

Table 7 presents two low-pass FIR filters with asymmetric coefficients computed by the *firgr* function of MATLAB with the *minphase* option. Table 8 presents the high-level and low-level results of FIR filters, where the algorithms of [3,5] were applied

[7] The design library is available at www.nangate.com.

Table 6 Summary of results of custom designs with a different design library

Filter	Form	Arch/Algo	CA	NCA	TA	D	P
5	Direct	GM	33.2	10.1	43.4	**5.0**	**0.7**
		SA-A [5]	17.2	10.1	**27.3**	5.2	**0.7**
		SA-D [5]	17.7	10.1	27.9	**5.0**	**0.7**
	Transposed	GM	36.1	20.8	56.9	**3.9**	3.1
		SA-A [3]	22.3	20.8	**43.1**	4.6	2.4
		SA-D [3]	22.4	20.8	43.2	4.2	**2.3**

Table 7 Specifications of filters with asymmetric coefficients

Filter	N	Pass	Stop	Q
6	45	0.15	0.20	16
7	75	0.20	0.25	16

to obtain the shift-adds design of constant multiplications when the delay constraint was set to the minimum number of adder-steps in CAVM/MCM/CMVM operations as described in Sect. 2.2. The experimental settings given in Sect. 4.1.1 were used, and the *bwo* values of Filter 6 and 7 were computed as 33 and 34, respectively, when *bwi* is 16. For each type of a hybrid form filter, *Hybrid-I-s*, *Hybrid-II-k*, and *Hybrid-III-r*, different s, k, and r values, which are divisors of N, were used to explore their impact on the filter complexity. In Table 8, *OLO* is the number of adders on the output-line, and *MBO* is the number of operations in the CAVM, MCM, and CMVM blocks in the direct, transposed, and hybrid forms, respectively. Also, *ILR* and *OLR* are the number of registers on the input-line and output-line, respectively. All the forms of Filters 6 and 7 include the same total number of registers, i.e., 44 and 74, respectively. Other parameters have the same meaning as explained for Table 2 in the previous subsection. In this table, the best and worst values of *TA*, *Sls*, *D*, and *P* are given in bold and bold–italic, respectively.

Observe that the transposed form includes the minimum total number of operations (*TO*). This value increases for the *Hybrid-I-s* form, as the number of subsections (s) increases. This is simply because, as s is increased, the number of filter coefficients in each subsection is decreased, which consequently decreases the partial product sharing. For the second and third types of the hybrid form, as the number of rows (columns) of the constant matrix is increased (decreased), the number of operations in the CMVM operation is decreased. For Filter 6, the *TO* values of these hybrid forms are in between those of the transposed and direct forms, close to the *TO* value of the direct form. For Filter 7, it is generally larger than the *TO* value of the direct form.

For the **custom designs**, the third type of the hybrid form leads to filter designs with the least complexity among the hybrid forms, and for Filter 6 with the r value 3, it yields a design with minimum total area among all the forms. This is due to the smaller combinational area, which is the consequence of having a *TO* value less than that of the direct form. Also, the combinational area of the transposed form Filter 6 is less than that of the direct form, which was never observed on symmetric filters given in Table 2. Although this fact heavily depends on the filter coefficients, it has

Table 8 Summary of results of FIR filters with asymmetric coefficients using a 16-bit filter input

Filter	Form	High-level results					Low-level custom design results					Low-level reconfigurable design results				
		OLO	MBO	TO	ILR	OLR	CA	NCA	TA	D	P	LUTs	FFs	Sls	D	P
6	Direct	0	120	120	44	0	63.8	12.0	75.8	9.7	2.3	2451	704	1659	23.8	441
	Hybrid-I-3			130	42	2	68.1	12.5	80.6	9.4	**2.2**	2654	733	1724	22.6	450
	Hybrid-I-5			136	40	4	74.3	13.0	87.3	8.9	2.3	2906	762	1836	22.4	453
	Hybrid-I-9			144	36	8	79.6	14.0	93.5	8.9	2.4	3228	823	1966	18.7	436
	Hybrid-I-15			152	30	14	84.3	15.5	***99.8***	8.8	2.6	3535	911	***2077***	17.0	406
	Hybrid-II-3	14	92	106	2	42	64.9	22.5	87.4	9.3	6.3	2701	1334	1846	18.9	441
	Hybrid-II-5	8	108	116	4	40	67.1	21.9	89.0	9.4	5.3	3173	722	1706	20.2	447
	Hybrid-II-9	4	108	112	8	36	60.1	20.8	80.9	***9.8***	4.3	2665	474	1469	22.0	467
	Hybrid-II-15	2	114	116	14	30	64.1	19.4	83.5	9.5	3.4	2600	412	1483	***25.1***	***480***
	Hybrid-III-3	2	108	110	42	2	62.0	12.5	**74.5**	9.4	2.9	2404	737	1580	23.2	444
	Hybrid-III-5	4	114	118	40	4	67.5	13.1	80.6	9.4	3.6	3010	401	1597	21.8	425
	Hybrid-III-9	8	106	114	36	8	66.6	14.2	80.8	9.7	4.4	2825	403	1479	19.4	415
	Hybrid-III-15	14	95	109	30	14	68.0	15.7	83.7	9.4	5.8	2927	546	1535	20.8	440
	Transposed	44	51	95	0	44	62.6	23.0	85.6	**8.1**	***9.3***	2557	1351	**1332**	**10.5**	**371**

Table 8 continued

Filter	Form	High-level results					Low-level custom design results					Low-level reconfigurable design results				
		OLO	MBO	TO	ILR	OLR	CA	NCA	TA	D	P	LUTs	FFs	Sls	D	P
7	Direct	0	164	164	74	0	89.3	20.1	**109.4**	***10.3***	**3.1**	3417	1184	2381	25.8	488
	Hybrid-I-3			185	72	2	98.3	20.6	118.9	9.9	3.3	3900	1212	2614	23.7	440
	Hybrid-I-5			197	70	4	106.1	21.1	127.2	9.3	3.2	4148	1240	2751	24.1	474
	Hybrid-I-15			220	60	14	122.4	23.4	145.8	8.4	3.6	4942	1382	3055	20.0	462
	Hybrid-I-25			229	50	24	126.8	25.8	***152.6***	**8.2**	3.8	5187	1522	***3082***	17.3	437
	Hybrid-II-3	24	132	156	2	72	94.8	37.3	132.1	9.8	12.6	3895	2205	2779	19.2	508
	Hybrid-II-5	14	151	165	4	70	94.5	36.7	131.2	10.2	8.2	4736	1217	2514	20.7	483
	Hybrid-II-15	4	167	171	14	60	94.3	34.4	128.7	9.9	6.5	3984	558	2193	23.9	521
	Hybrid-II-25	2	173	175	24	50	94.5	32.0	126.5	9.9	5.1	4029	578	2328	24.0	496
	Hybrid-III-3	2	168	170	72	2	94.8	20.7	115.5	9.7	3.9	3831	1221	2607	***26.0***	***526***
	Hybrid-III-5	4	164	168	70	4	96.0	21.2	117.2	9.9	5.0	4284	589	2243	24.1	480
	Hybrid-III-15	14	155	169	60	14	98.5	23.8	122.3	9.6	7.5	4197	588	2197	23.0	467
	Hybrid-III-25	24	151	175	50	24	105.4	26.4	131.8	9.8	8.6	4406	916	2362	20.2	455
	Transposed	74	69	143	0	74	91.4	37.6	129.0	9.2	***13.7***	3607	2214	**1901**	**10.5**	**435**

Table 9 Summary of results of reconfigurable designs with different bitwidths of the filter input

Filter	Form	8-bit filter input					24-bit filter input				
		LUTs	FFs	Sls	D	P	LUTs	FFs	Sls	D	P
7	Direct	2103	592	1413	25.5	382	4732	1776	3354	27.3	569
	Hybrid-I-3	2395	620	1564	22.8	355	5423	1804	3670	26.4	**472**
	Hybrid-II-15	2525	360	1398	22.8	404	5452	755	2998	25.2	590
	Hybrid-III-15	2707	443	1449	22.2	378	5703	764	2974	23.3	488
	Transposed	2468	1622	**1334**	**11.4**	**342**	4778	2796	**2483**	**12.1**	517

two main reasons: (i) the total number of operations in the transposed form is less than that of the direct form; (ii) $\lfloor N/2 \rfloor$ adders with a size of $bwi + 1$ used for the symmetric coefficients outside the CAVM operation (Fig. 7) are not used in asymmetric filters. However, the direct form filters occupy less total area than the transposed form filters, since they require less number of flip-flops. Moreover, the *Hybrid-I-s* form leads to a design of Filter 7 with the smallest delay, where the maximum gain is 20.3 % with respect to the direct form, but incurring in a significant increase in area. The hybrid form filters consume power in between those of the direct and transposed forms.

For the **reconfigurable designs**, the transposed form is the best way of synthesizing filters on the Virtex 4 FPGA target device, and the hybrid forms lead to designs having the values of occupied slices, delay, and power dissipation in between those of the transposed and direct form filters. In the second and third types of the hybrid form (Figs. 10-11), some cascaded registers were designed using shift-registers by the synthesis tool, whose cost values were counted under the number of LUTs.

Since experiments on different bitwidths of the filter input and design libraries on the custom designs revealed similar conclusions to those obtained for symmetric filters in previous subsection, the impact of the bitwidth of the filter input and target FPGA device on the reconfigurable designs is investigated. To do so, five implementations of Filter 7 were chosen from Table 8, i.e., direct, *Hybrid-I-3*, *Hybrid-II-15*, *Hybrid-III-15*, and transposed forms. Among the hybrid forms, the ones, which yield the least number of slices in the Virtex 4 FPGA, were selected.

Table 9 presents the low-level results of Filter 7 when *bwi* is 8 and 24. In this experiment, the Virtex 4 FPGA target device is used as done in obtaining the results of Table 8. Considering the results in Table 8, as *bwi* is decreased (increased), the complexity of the filter designs is decreased (increased) directly. Among these different forms of Filter 7, its transposed form occupies the least number of slices and has the smallest delay when *bwi* is 8 and 24, as also observed when *bwi* is 16.

Table 10 presents the low-level results of Filter 7 when the previously mentioned Virtex 5 and 6 FPGAs are used as a target device. In this experiment, *bwi* is set to 16 as done in obtaining the results of Table 8. Observe from Table 10 that while Filter 7 can be realized most efficiently in its transposed form on Virtex 5 FPGA, its direct form requires less number of slices than its transposed form on Virtex 6 FPGA. However, the transposed form filter design has the least delay and power dissipation on both FPGAs.

Table 10 Summary of results of reconfigurable designs with different FPGA devices

Filter	Form	Virtex 5 FPGA					Virtex 6 FPGA				
		LUTs	FFs	Sls	D	P	LUTs	FFs	Sls	D	P
7	Direct	3398	1184	1148	25.7	820	3166	1185	**891**	21.8	1523
	Hybrid-I-3	3821	1212	1179	21.7	762	3542	1212	985	19.8	1493
	Hybrid-II-15	3941	546	1105	24.8	834	3751	538	1020	18.9	1551
	Hybrid-III-15	4121	598	1120	23.1	787	3672	606	988	16.4	1507
	Transposed	3617	2204	**980**	**10.3**	**760**	3585	2204	955	**8.8**	**1487**

5 Concluding Remarks

This article reviewed prominent algorithms designed for the multiplierless realization of constant multiplications and described how the direct, transposed, and hybrid forms of an FIR filter can be synthesized under a shift-adds architecture efficiently. It introduced the results of a set of experiments that focused on the exploration of key factors in the filter design which have a direct impact on area, delay, and power dissipation of the design. It provided insights into the multiplierless design of filters and discussed the advantages and disadvantages of this technique with respect to designs using generic multipliers.

It is shown that the complexity of a multiplierless FIR filter design depends on the filter form (direct, transposed, and hybrid), high-level synthesis algorithms (targeting the optimization of area and delay), design platform (ASIC and FPGA), and design parameters (bitwidth of the filter input, design library, and target FPGA device). For applications demanding less area, the direct form is the best architecture to design symmetric FIR filters in custom circuits. In case of asymmetric filters, the hybrid form is an another option to be considered, paying attention to its different types and parameters. For reconfigurable designs, the choice of the filter form depends on the target FPGA device and bitwidth of the filter input. For high-speed applications, the transposed form is the best architecture for both design platforms. It is also important to use algorithms that target the reduction of the number of adder-steps. For low-power applications, it is better to implement the filter in the direct form if the design platform is ASIC, which is due to less combinational area and less number of registers. If the design platform is FPGAs, the transposed form should be preferred, considering the target device.

Although the choice of the filter form depends on so many parameters for a designer, this article showed that there also exist the direct and hybrid forms that may introduce more promising filter designs than the commonly used transposed form. Hence, the optimization algorithms, which only focus on the transposed form in certain problems, should consider the realization of direct and hybrid forms. For example, (i) the problem of realizing constant multiplications with minimum number of carry-save adders [33], which are preferred to ripple carry adders in high-speed applications taking into account the increase in area; (ii) the problem of maximizing the throughput of the filter design [41,44,46], which is an important design parameter in many DSP

applications; (iii) the filter design optimization problem [8,52], which is to find a set of filter coefficients which yields a filter design using minimum number of operations, satisfying the filter constraints.

The experimental results also indicated that the high-level synthesis algorithms have a significant impact on the multiplierless design of FIR filters in terms of complexity, performance, and power dissipation. Thus, developing more efficient algorithms and improving the solution quality of algorithms, especially those targeting the CAVM and CMVM operations, will naturally yield FIR filter designs with less complexity and higher performance.

Acknowledgments This work was supported by the national funds through FCT, Fundação para a Ciência e a Tecnologia, under Project PEst-OE/EEI/LA0021/2013.

References

1. L. Aksoy, E. Costa, P. Flores, J. Monteiro, Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications. IEEE Trans. Comput.-Aided Des. Integr. Circuits **27**(6), 1013–1026 (2008)
2. L. Aksoy, E. Costa, P. Flores, J. Monteiro, Design of low-power multiple constant multiplications using low-complexity minimum depth operations. in *Proceedings of Great Lakes Symposium on VLSI* (2011), pp. 79–84
3. L. Aksoy, E. Costa, P. Flores, J. Monteiro, Finding the optimal tradeoff between area and delay in multiple constant multiplications. Elsevier J. Microprocess. Microsyst. **35**(8), 729–741 (2011)
4. L. Aksoy, E. Costa, P. Flores, J. Monteiro, Design of low-complexity digital finite impulse response filters on FPGAs. in *Proceedings of Design, Automation and Test in Europe Conference* (2012), pp. 1197–1202
5. L. Aksoy, E. Costa, P. Flores, J. Monteiro, Multiplierless design of linear DSP transforms. in *VLSI-SoC: Advanced Research for Systems on Chip*, Chap. 5 (Springer, 2012), pp. 73–93
6. L. Aksoy, E. Costa, P. Flores, J. Monteiro, Optimization algorithms for the multiplierless realization of linear transforms. ACM Trans. Des. Autom. Electron. Syst. **17**(1), Art. No. 3 (2012). doi:10.1145/2071356.2071359
7. L. Aksoy, E. Gunes, P. Flores, Search algorithms for the multiple constant multiplications problem: exact and approximate. Elsevier J. Microproces. Microsyst. **34**(5), 151–162 (2010)
8. M. Aktan, A. Yurdakul, G. Dnndar, An algorithm for the design of low-power hardware-efficient FIR filters. IEEE Trans. Circuits Syst. **55**(6), 1536–1545 (2008)
9. A. Arfaee, A. Irturk, N. Laptev, F. Fallah, R. Kastner, Xquasher: a tool for efficient computation of multiple linear expressions. in *Proceedings of Design Automation Conference* (2009), pp. 254–257
10. N. Boullis, A. Tisserand, Some optimizations of hardware multiplication by constant matrices. IEEE Trans. Comput. **54**(10), 1271–1282 (2005)
11. N. Brisebarre, F. de Dinechin, J.M. Muller, Integer and floating-point constant multipliers for FPGAs. in *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors* (2008), pp. 239–244
12. P. Cappello, K. Steiglitz, Some complexity Issues in digital signal processing. IEEE Trans. Acoust. Speech Signal Process. **32**(5), 1037–1041 (1984)
13. J. Chen, C.H. Chang, H. Qian, New power index model for switching power analysis from adder graph of FIR filter. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2009), pp. 2197–2200
14. S. Demirsoy, A. Dempster, I. Kale, Power analysis of multiplier blocks. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2002), pp. 297–300
15. A. Dempster, Use of minimum-adder multiplier blocks in FIR digital filters. IEEE Trans. Circuits Syst. II **42**(9), 569–577 (1995)
16. A. Dempster, S. Demirsoy, I. Kale, Designing multiplier blocks with low logic depth. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2002), pp. 773–776

17. A. Dempster, M. Macleod, Constant integer multiplication using minimum adders. IEE Proc. Circuits Devices Syst. **141**(5), 407–413 (1994)
18. A. Dempster, M. Macleod, Digital filter design using subexpression elimination and all signed-digit representations. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2004), pp. 169–172
19. M. Ercegovac, T. Lang, *Digital Arithmetic*. (Morgan Kaufmann, 2003)
20. M. Faust, C.H. Chang, Optimization of structural adders in fixed coefficient transposed direct form FIR filters. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2009), pp. 2185–2188
21. M. Faust, C.H. Chang, Minimal logic depth adder tree optimization for multiple constant multiplication. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2010), pp. 457–460
22. M. Faust, C.H. Chang, Low error bit width reduction for structural adders of FIR filters. in *Proceedings of IEEE European Conference on Circuit Theory and Design* (2011), pp. 713–716
23. P. Flores, J. Monteiro, E. Costa, An exact algorithm for the maximal sharing of partial terms in multiple constant multiplications. in *Proceedings of International Conference on Computer-Aided Design* (2005), pp. 13–16
24. W. Gallagher, E. Swartzlander, High radix booth multipliers using reduced area adder trees. in *Proceedings of Asilomar Conference on Signals, Systems and Computers* (1994), pp. 545–549
25. R. Guo, L. Wang, L. DeBrunner, A novel FIR filter implementation using truncated MCM technique. in *Proceedings of Asilomar Conference on Signals, Systems and Computers* (2009), pp. 718–722
26. O. Gustafsson, A difference based adder graph heuristic for multiple constant multiplication problems. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2007) pp. 1097–1100
27. O. Gustafsson, Lower bounds for constant multiplication problems. IEEE Trans. Circuits Syst. II **54**(11), 974–978 (2007)
28. O. Gustafsson, J. Coleman, A. Dempster, M. Macleod, Low-complexity hybrid form FIR filters using matrix multiple constant multiplication. in *Proceedings of Asilomar Conference on Signals, Systems and Computers* (2004), pp. 77–80
29. O. Gustafsson, A. Dempster, L. Wanhammar, Extended results for minimum-adder constant integer multipliers. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2002), pp. 73–76
30. O. Gustafsson, H. Ohlsson, L. Wanhammar, Low-complexity constant coefficient matrix multiplication using a minimum spanning tree. in *Proceedings of Nordic Signal Processing Symposium* (2004), pp. 141–144
31. O. Gustafsson, L. Wanhammar, ILP modelling of the common subexpression sharing problem. in *Proceedings of International Conference on Electronics, Circuits and Systems* (2002), pp. 1171–1174
32. R. Hartley, Subexpression sharing in filters using canonic signed digit multipliers. IEEE Trans. Circuits Syst. II **43**(10), 677–688 (1996)
33. R. Hawley, B. Wong, T.J. Lin, J. Laskowski, H. Samueli, Design techniques for silicon compiler implementations of high-speed FIR digital filters. IEEE J. Solid-State Circuits **31**(5), 656–667 (1996)
34. Y.H. Ho, C.U. Lei, H.K. Kwan, N. Wong, Global optimization of common subexpressions for multiplierless synthesis of multiple constant multiplications. in *Proceedings of Asia and South Pacific Design Automation Conference* (2008), pp. 119–124
35. A. Hosangadi, F. Fallah, R. Kastner, Reducing hardware complexity of linear DSP systems by iteratively eliminating two-term common subexpressions. in *Proceedings of Asia and South Pacific Design Automation Conference* (2005), pp. 523–528
36. A. Hosangadi, F. Fallah, R. Kastner, Simultaneous optimization of delay and number of operations in multiplierless implementation of linear systems. in *Proceedings of International Workshop on Logic Synthesis* (2005)
37. K. Johansson, O. Gustafsson, L. DeBrunner, L. Wanhammar, Minimum adder depth multiple constant multiplication algorithm for low power FIR filters. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2011), pp. 1439–1442.
38. K. Johansson, O. Gustafsson, L. Wanhammar, A detailed complexity model for multiple constant multiplication and an algorithm to minimize the complexity. in *Proceedings of IEEE European Conference on Circuit Theory and Design* (2005), pp. 465–468
39. H.J. Kang, I.C. Park, FIR filter synthesis algorithms for minimizing the delay and the number of adders. IEEE Trans. Circuits Syst. II **48**(8), 770–777 (2001)

40. K.Y. Khoo, Z. Yu, A. Willson, Design of optimal hybrid form FIR filter. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2001), pp. 621–624
41. M. Kumm, P. Zipf, M. Faust, C.H. Chang, Pipelined adder graph optimization for high speed multiple constant multiplication. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2012), pp. 49–52
42. V. Lefevre, Multiplication by an Integer Constant. Tech. rep., Institut National de Recherche en Informatique et en Automatique (2001)
43. M. Macleod, A. Dempster, A common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers. Electron. Lett. **40**(11), 651–652 (2004)
44. K. Macpherson, R. Stewart, Rapid prototyping—area efficient FIR filters for high speed FPGA implementation. IEE Proc. Vision Image Signal Process. **153**(6), 711–720 (2006)
45. J. McClellan, T. Parks, L. Rabiner, A computer program for designing optimal FIR linear phase digital filters. IEEE Trans. Audio Electroacoustics **21**(6), 506–526 (1973)
46. U. Meyer-Baese, J. Chen, C.H. Chang, A. Dempster, A comparison of pipelined RAG-n and DA FPGA-based multiplierless filters. in *Proceedings of IEEE Asian-Pacific Conference on Circuits and Systems* (2006), pp. 1555–1558
47. K. Muhammad, K. Roy, A graph theoretic approach for synthesizing very low-complexity high-speed digital filters. IEEE Trans. Comput.-Aided Des. Integr. Circuits **21**(2), 204–216 (2002)
48. H. Nguyen, A. Chatterjee, Number-splitting with shift-and-add decomposition for power and hardware optimization in linear DSP synthesis. IEEE Trans. VLSI **8**(4), 419–424 (2000)
49. I.C. Park, H.J. Kang, Digital filter synthesis based on minimal signed digit representation. in *Proceedings of Design Automation Conference* (2001), pp. 468–473
50. M. Potkonjak, M. Srivastava, A. Chandrakasan, Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination. IEEE Trans. Comput.-Aided Des. Integr. Circuits **15**(2), 151–165 (1996)
51. F. Qureshi, O. Gustafsson, Low-complexity reconfigurable complex constant multiplication for FFTs. in *Proceedings of IEEE International Symposium on Circuits and Systems* (2009), pp. 24–27
52. D. Shi, Y.J. Yu, Design of linear phase FIR filters with high probability of achieving minimum number of adders. IEEE Trans. Circuits Syst. **58**(1), 126–136 (2011)
53. J. Thong, N. Nicolici, A novel optimal single constant multiplication algorithm. in *Proceedings of Design Automation Conference* (2010), pp. 613–616
54. Y. Voronenko, M. Püschel, Multiplierless multiple constant multiplication. ACM Trans. Algorithms **3**(2), Art. No. 11 (2007). doi:10.1145/1240233.1240234
55. C. Wallace, A suggestion for a fast multiplier. IEEE Trans. Electron. Comput. **13**(1), 14–17 (1964)
56. E. Walters III, Design Tradeoffs Using Truncated Multipliers in FIR Filter Implementations. Master's thesis, Lehigh University (2002)
57. F. Xu, C.H. Chang, C.C. Jong, Contention resolution algorithm for common subexpression elimination in digital filter design. IEEE Trans. Circuits Syst. II: Express Briefs **52**(10), 695–700 (2005)
58. S.H. Yoon, J.W. Chong, C.H. Lin, An area optimization method for digital filter design. ETRI J. **26**(6), 545–554 (2004)
59. A. Yurdakul, G. Dündar, Multiplierless realization of linear DSP transforms by using common two-term expressions. J. VLSI Signal Process. **22**(3), 163–172 (1999)