Design of Low-Complexity Digital Finite Impulse Response Filters on FPGAs

Levent Aksoy
INESC-ID
Lisboa, Portugal

Eduardo Costa
UCPEL
Pelotas, Brazil

Paulo Flores
INESC-ID/IST TU Lisbon
Lisboa, Portugal

José Monteiro
INESC-ID/IST TU Lisbon
Lisboa, Portugal

***Abstract*—The multiple constant multiplications (MCM) operation, which realizes the multiplication of a set of constants by a variable, has a significant impact on the complexity and performance of the digital finite impulse response (FIR) filters. Over the years, many high-level algorithms and design methods have been proposed for the efficient implementation of the MCM operation using only addition, subtraction, and shift operations. The main contribution of this paper is the introduction of a high-level synthesis algorithm that optimizes the area of the MCM operation and, consequently, of the FIR filter design, on field programmable gate arrays (FPGAs) by taking into account the implementation cost of each addition and subtraction operation in terms of the number of fundamental building blocks of FPGAs. It is observed from the experimental results that the solutions of the proposed algorithm yield less complex FIR filters on FPGAs with respect to those whose MCM part is implemented using prominent MCM algorithms and design methods.**

I. Introduction

FIR filters are widely used in Digital Signal Processing (DSP) applications due to their stability and linear-phase property. The multiplier block of the FIR filter (Figure 1), where the multiplications of filter coefficients by the filter input are realized, dominates the complexity of the design since a large number of constant multiplications are required. This block is generally known as the MCM operation and is a central operation and performance bottleneck in many DSP applications, such as video coding and image compression.

In DSP applications that demand high performance, since general purpose or DSP processors cannot handle the amount of data to be processed, Application Specific Integrated Circuits (ASIC) or FPGAs is the common design platform. With the advancement of technology for FPGAs, it is now possible to implement high-speed DSP algorithms using bit-parallel arithmetic on FPGAs. However, while constant multiplication plays a key role in DSP, the multiplication is a costly operation in FPGAs since it consumes significant FPGA resources and operates with relatively long latencies.

In the last two decades, prominent high-level synthesis algorithms [1]–[6] have been introduced for the efficient design of constant multiplications. Since the constant coefficients are determined beforehand by the DSP algorithms, the main idea is to replace the constant multiplications with addition/subtraction and shift operations without requiring the full-flexibility of a multiplier. Thus, their primary objective is to find the minimum number of addition and subtraction operations that realize the constant multiplications. Note that shifts can be realized using only wires in a bit-parallel design without representing any hardware cost.

This work was supported by the *Portuguese Foundation for Science and Technology* (FCT) under the research project *Multicon - Architectural Optimization of DSP Systems with Multiple Constants Multiplications* and INESC-ID multiannual funding through the PIDDAC Program funds.

978-3-9810801-8-6/DATE12/©2012 EDAA

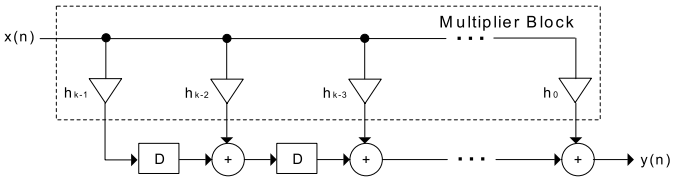

Fig. 1. Transposed form of a digital FIR filter implementation.

For the implementation of constant multiplications under the shift-adds architecture, a straightforward method, generally known as the digit-based recoding [7], initially defines the constants in multiplications under binary representation. Then, for each 1 in the representation of the constant, according to its bit position, it shifts the variable and adds up the shifted variables to obtain the result. As a simple example, consider the constant multiplications $29x$ and $43x$. Their decompositions in binary are listed as:

$$29x = (11101)_{bin}x = x \ll 4 + x \ll 3 + x \ll 2 + x$$
$$43x = (101011)_{bin}x = x \ll 5 + x \ll 3 + x \ll 1 + x$$

and require 6 addition operations as given in Figure 2(a).

The complexity of an MCM operation can be further reduced by sharing partial products among the constant multiplications. The prominent algorithms can be categorized in two classes: Common Subexpression Elimination (CSE) [1]–[3] techniques and graph-based (GB) [4]–[6] methods. The CSE algorithms first define the constants under a particular number representation, namely, binary, Canonical Signed Digit (CSD) [1], or Minimal Signed Digit (MSD) [2], and then, find the "best" subexpression, generally the most common, among the constant multiplications. The GB algorithms are not restricted to any particular number representation and consider a large number of alternative implementations of a constant multiplication, yielding better solutions than the CSE algorithms [5], [6]. Returning to our simple example, the exact CSE algorithm [3] obtains a solution with 4 operations by finding the most common partial products $3x = (11)_{bin}x$ and $5x = (101)_{bin}x$ when constants are defined under binary (Figure 2(b)). The exact GB algorithm [6] finds the minimum

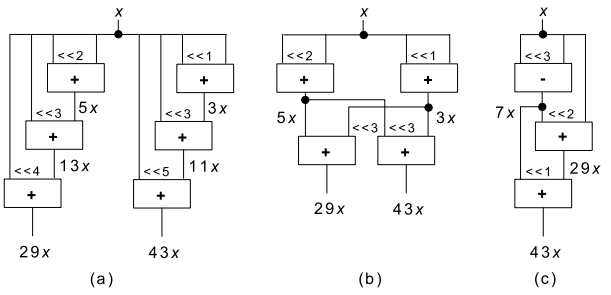

Fig. 2. Shift-adds implementations of $29x$ and $43x$: (a) without partial product sharing [7]; with partial product sharing: (b) the algorithm of [3]; (c) the algorithm of [6].

number of operations solution with 3 operations by sharing the common partial product $7x$ in both multiplications (Figure 2(c)). Observe that the partial product $7x = (111)_{bin}x$ cannot be extracted from the binary representations of both multiplications $29x$ and $43x$ in the exact CSE algorithm [3].

Although the solutions of these algorithms lead to less complex MCM designs on FPGAs due to fewer number of operations, it is obvious that their solutions may not yield a design using the minimum amount of resources in FPGAs. This is simply because these algorithms do not consider the implementation cost of each addition/subtraction operation in terms of the main building blocks of FPGAs, *e.g.*, look-up tables (LUTs) or configurable logic blocks (CLBs). Although there exist high-level techniques [8], [9] whose solutions are realized on FPGAs, to the best our knowledge, there exists no high-level MCM algorithm that targets the optimization of MCM area on FPGAs considering its specifications. Hence, this paper introduces an approximate algorithm, called LUTOR, that initially applies the Hcub algorithm [5] on an MCM instance to find a solution with the fewest number of operations. Then, a 0-1 Integer Linear Programming (ILP) technique is applied on the solution of Hcub to obtain a set of operations which leads to an MCM design using minimum number of 4-input LUTs on FPGAs. Although LUTOR focuses on the synthesis of the MCM operation on Xilinx Virtex 4 FPGAs, its cost function can be easily modified for other Xilinx families including 5 or 6-input LUTs or for the Altera FPGAs. It is observed that LUTOR leads to significant reductions in the number of 4-input LUTs and, consequently, in the area of the MCM design on FPGAs, with respect to efficient MCM algorithms. The experimental results show that the shift-adds design of MCM operations using high-level algorithms yield less complex multiplier blocks and, consequently, less complex FIR filters, when compared to those designed using LUT-based constant multipliers [10] and generic multipliers.

The rest of the paper proceeds as follows. Section II presents the background concepts and the LUTOR algorithm is introduced in Section III. The experimental results are given in Section IV and finally, Section V concludes the paper.

II. BACKGROUND

This section presents the background concepts, introduces the problem definitions, and gives an overview on previously proposed prominent MCM algorithms and design methods.

A. 0-1 Integer Linear Programming

The 0-1 ILP problem is the minimization or the maximization of a linear cost function subject to a set of linear constraints and is generally defined as follows[1]:

$$\text{Minimize} \quad \mathbf{w}^T \cdot \mathbf{x} \tag{1}$$

$$\text{Subject to} \quad \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}, \quad \mathbf{x} \in \{0,1\}^n \tag{2}$$

In (1), w_j in $\mathbf{w}$ is an integer value associated with each of n variables x_j, $1 \leq j \leq n$, in the cost function, and in (2), $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ denotes the set of m linear constraints, where $\mathbf{b} \in \mathbb{Z}^m$ and $\mathbf{A} \in \mathbb{Z}^m \times \mathbb{Z}^n$.

B. Multiplierless Constant Multiplication

The fundamental operation in multiplierless constant multiplications, called *A-operation* in [5], is an operation with two integer inputs and one integer output that performs a single addition or a subtraction, and an arbitrary number of shifts. It is defined as:

$$w = A(u,v) = |(u \ll l_1) + (-1)^s (v \ll l_2)| \gg r \tag{3}$$

where $l_1, l_2 \geq 0$ are integers denoting left shifts of the operands, $r \geq 0$ is an integer indicating a right shift of the result, and $s \in \{0,1\}$ is the sign, which determines if an addition or a subtraction operation is to be performed.

In the MCM problem, the complexity of an adder and a subtracter in hardware is assumed to be equal. It is also assumed that the sign of the constant can be adjusted at some part of the design and the shifting operation has no cost in hardware due to the bit-parallel processing. Thus, only positive and odd constants are considered in the MCM problem. Observe from Eqn. (3) that in the implementation of an odd constant with any two odd constants at the inputs, one of the left shifts, l_1 or l_2, is zero and r is zero, or both l_1 and l_2 are zero and r is greater than zero. Hence, only one of the shifts, l_1, l_2, or r, is greater than zero. Thus, any *A-operation* that realizes an addition can be in the form of $u + 2^{l_2}v$ or $(u+v)2^{-r}$, where in the former, only one of the left shifts and the right shift are zero and in the latter, both of the left shifts are zero. Also, the subtraction operations in the form of $2^{l_1}u - v$, $u - 2^{l_2}v$, and $(u-v)2^{-r}$ cover all the cases where *A-operation* performs a subtraction. It is also necessary to constrain the left shifts, l_1 and l_2, otherwise there exist infinite ways of implementing a constant. In the exact algorithm of [6], the number of shifts is allowed to be at most $bw+1$, where bw is the maximum bit-width of the constants to be implemented. Thus, the MCM problem [5] is defined as:

Definition 1. THE MCM PROBLEM. *Given the target set composed of positive and odd unrepeated target constants to be implemented, $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$, find the smallest ready set, $R = \{r_0, r_1, \ldots, r_m\}$, with $T \subset R$, under the*

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

TABLE I
IMPLEMENTATION COSTS OF ALL POSSIBLE *A-operations*.

A-operation	# 4-input LUTs
$w = u + 2^{l_2}v$	$^*b_{wx} - l_2$
$w = (u + v)2^{-r}$	$b_{wx} + r$
$w = 2^{l_1}u - v$	b_{wx}
$w = u - 2^{l_2}v$	$b_{wx} - l_2$
$w = (u - v)2^{-r}$	$b_{wx} + r$

* This applies when there is a overlap in between input operands. Otherwise, *i.e.*, if $l_2 \geq \lceil log_2 u \rceil$, no LUT is required.

conditions of $r_0 = 1$ *and for all* r_k *with* $1 \leq k \leq m$*, there exist* r_i, r_j *with* $0 \leq i, j < k$ *and an A-operation* $r_k = A(r_i, r_j)$.

Hence, the number of operations required to be implemented for the MCM problem is $|R| - 1$, as given in [5]. Note that the MCM problem is an NP-complete problem [11].

C. Addition and Subtraction Operations on FPGAs

We now consider the design of addition and subtraction operations on Xilinx Virtex FPGAs [12]. The building block of the Virtex CLB contains four logic cells (LCs) organized as two slices. An LC includes a 4-input LUT, a carry chain logic, and a storage element. While the 4-input LUT can implement any 4-input logic function, dedicated carry logic provides fast arithmetic carry capability. In Xilinx Virtex FPGAs, in general, an addition/subtraction performed on two n-bit operands requires n 4-input LUTs [12].

However, an *A-operation* has always a left shift or a right shift value greater than 0, which may decrease the general cost of an operation as indicated in [9]. Considering the advanced optimization techniques available in the Xilinx ISE design tool, that can exploit hardware simplifications when shifts exist, we determined the costs of all possible *A-operations* in terms of the number of 4-input LUTs as given in Table I. In this table, b_{wx} denotes the bit-width of the constant multiplication wx computed as $\lceil log_2 w \rceil + N$, where N is the bit-width of the input x. Note that these values were confirmed by experiments carried on all types of *A-operation* with different size of input operands and different shift values under signed input on a Virtex 4 FPGA using the Xilinx ISE 13.1 design tool. In the case of MCM, despite the optimization techniques used in the design tool, our estimates are close to the synthesis values as shown in Section IV. Hence, the optimization of area problem in MCM design on FPGAs can be defined as follows:

Definition 2. THE OPTIMIZATION OF AREA PROBLEM IN MCM DESIGN ON FPGA. *Given the target set* $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$*, find the ready set* $R = \{r_0, r_1, \ldots, r_m\}$ *such that under the same conditions on the ready set given in Definition 1, the set of A-operations yields an MCM design using minimum number of 4-bit LUTs in FPGAs.*

D. Related Work

The exact CSE algorithms designed for the MCM problem were introduced in [3], [13]. They initially define each target constant under a number representation and extract all its possible implementations from its representation. Then, the MCM problem is defined as a 0-1 ILP problem and a solution is obtained using a generic 0-1 ILP solver. Prominent CSE heuristics were also presented in [1]–[3].

The exact GB algorithms that search a solution with the minimum number of operations in breadth-first and depth-first manners were introduced in [6]. Efficient GB algorithms, that includes two parts, optimal and heuristic, were introduced in [4], [5]. In their optimal parts, each target constant, that can be implemented with a single operation, is synthesized. If there exist unimplemented elements left in the target set, then they switch to their heuristic parts where the required intermediate constants are found. The RAG-n algorithm [4] initially chooses a single unimplemented target constant with the smallest single coefficient cost and then, synthesizes it with a single operation including one(two) intermediate constant(s) that has(have) the smallest value in its heuristic part. The Hcub algorithm [5] selects a single intermediate constant that yields the best cumulative benefit over all unimplemented target constants for the implementation of each target constant.

There exists no high-level synthesis algorithm targeting the optimization of area in MCM design on FPGAs. However, a LUT-based constant multiplier designed especially for FPGAs was proposed in [14], where the input x is split into 4 bits and the constant multiplication kx is realized using additions and shifts. Assuming an 8-bit input variable x, kx is realized as $(k.x_{7-4}) \ll 4 + k.x_{3-0}$, where each partial product is implemented using 4-input LUTs. This technique was improved in [15] by sharing the common LUTs in a partial product, removing redundant LUTs, and merging LUTs and single bit adders. In [10], the design method of [15] was carried to MCM by sharing the common LUTs among constant multiplications.

III. THE APPROXIMATE ALGORITHM

The LUTOR algorithm consists of two main parts. In the first part, a solution with the fewest number of operations, that generates MCM, is found by Hcub [5]. In the second part, the optimization of area problem in MCM design on FPGAs is formalized as a 0-1 ILP problem based on the solution of Hcub and a set of operations, that yields an MCM design requiring the minimum number of 4-input LUTs, is obtained.

A. Implementation of the LUTOR Algorithm

In the preprocessing phase of LUTOR, the constants to be multiplied by a variable are converted to positive and then, made odd by successive divisions by 2. The resulting constants are stored without repetition in a set called target set T and the maximum bit-width of the target constants, bw, is determined. The LUTOR algorithm, whose pseudo-code is given in Figure 3, takes these parameters and also N, the bit-width of the input variable x, as an input.

In the iterative loop of LUTOR (lines 3-14), we initially find a set of operations, O, that implements the constant multiplications using Hcub [5] with a different seed at each time (line 5). Then, we determine the intermediate and target constants in O and store them in a set called ready set, R (line 6). In order to increase the number of possible implementations of a constant in the *ILP* function, described

LUTOR**(T, bw, N)**

1: $seed = 0, R_{set} \leftarrow \{\}$
2: $icost_b = \infty$, $O_b \leftarrow \{\}$
3: **repeat**
4: $seed = seed + 1$
5: O = Hcub(T, $seed$)
6: R = GenerateReadySet(O)
7: R = AddDepth1Constants(R, bw)
8: **if** $R \notin R_{set}$ **then**
9: $R_{set} \leftarrow R_{set} \cup R$
10: O = ILP(T, R, bw, N)
11: $icost$ = ComputeImplementationCost(O, N)
12: **if** $icost < icost_b$ **then**
13: $icost_b = icost$, $O_b \leftarrow O$
14: **until** Termination conditions meet
15: **return** O_b

Fig. 3. The LUTOR algorithm.

in Section III-B, we add the depth-1 constants to R if they do not exist (line 7). The depth-1 constants are in the form of $2^{i+1} - 1$ and $2^i + 1$, where i ranges in between 1 and bw, which can be realized using a single *A-operation* whose inputs are 1. To avoid unnecessary computations, we check if R is already included in R_{set} which is a set that stores all the ready sets given to the *ILP* function (line 8). Then, the *ILP* function is applied on this ready set R to find a set of *A-operations* that yields an MCM design requiring minimum number of 4-input LUTs (line 10). After a solution is obtained by the *ILP* function, the cost of the MCM design on FPGAs, $icost$, is computed based on the costs of *A-operations* given in Table I (line 11). If its cost value is smaller than the best one ($icost_b$) found so far, then O_b and $icost_b$ are updated (lines 12-13). The iterative loop terminates whenever the number of elements in R_{set} reaches to 100 or the last 20 ready sets obtained by Hcub are identical[2] (line 14).

B. Implementation of the ILP Technique

The *ILP* function consists of four main parts: 1) generation of constant implementations; 2) construction of a network that represents the implementations of constants; 3) formalization of the problem as a 0-1 ILP problem; 4) obtaining the minimum solution. These parts are described in detail next.

1) Generation of Constant Implementations: After the set of operations (*O*) realizing the MCM instance is found by Hcub, in the *GenerateReadySet* function, we also compute the depth (or adder-step [3]) of each intermediate and target constant of *R* in the network of addition and subtraction operations of *O*.[3] After the depth-1 constants are included into *R*, we sort the constants in *R* according to their depth values in ascending order. The part of the algorithm, where the implementations of constants are found, is given as follows:

i) Take an element from R, r_i, except '1' that denotes the variable which the constants are multiplied with. Form an empty set, I_i, associated with r_i that will include the inputs of each *A-operation* which computes r_i.
ii) For each *A-operation* that generates r_i,
   a) Make each of its inputs positive and odd.
   b) Add its positive and odd inputs to the set I_i.
iii) Repeat Step (i) until all elements of R are considered.

To find all possible *A-operations* that implement an element of *R*, r_i, we assign r_i to the output of an *A-operation* given in Eqn. (3). Then, for each constant r_j in R, where $0 \leq j \leq i-1$, we assign r_j to the input u of the *A-operation* and by changing the shift (l_1, l_2, and r) and sign (s) values, we compute the input v of the *A-operation*. Note that left shifts are restricted to $bw+1$ and only one of l_1, l_2, and r is set to a value greater than 0 and the others are set to 0, since only positive and odd constants are considered. If v is an element of *R*, r_k, where $0 \leq k \leq i-1$, this operation is determined as a possible implementation of r_i. These restrictions come from the fact that the MCM operation forms a directed acyclic graph and does not include feedback loops [16]. By doing so, we ensure that the implementation of each constant determined by Hcub is considered in the 0-1 ILP formalization. Thus, we guarantee that the optimized MCM design will always have less or equal number of 4-input LUTs as the one obtained by Hcub.

As a simple example, consider one of the solutions of Hcub on a single target constant 21 which requires two operations, *i.e*, $17 = 1 \ll 4 + 1$ and $21 = 17 + 1 \ll 2$. After the depth-1 constants are included into R, we have the ready set $\{1, 3, 5, 7, 9, 15, 17, 31, 33, 63, 21\}$. Among many others, five implementations of the constant 21 including the solution of Hcub are $21 = 17+1 \ll 2$, $21 = 7 \ll 1+7$, $21 = 1 \ll 4+5$, $21 = 31 - 5 \ll 1$, and $21 = 3 \ll 3 - 3$. Note that each of these operations leads to a different implementation cost when they are realized on FPGAs.

2) Construction of the Boolean Network: After all possible implementations of constants in *R* are found, these implementations are represented in a Boolean network that includes only AND and OR gates. Its properties are as follows:

i) The primary input of the network is the input to be multiplied with the constants denoted by '1'.
ii) An AND gate in the network represents an addition or a subtraction operation and has two inputs.
iii) An OR gate in the network represents a target or an intermediate constant and combines all possible implementations of the constant.
iv) The outputs of the network are the OR gate outputs associated with the target constants.

The Boolean network is constructed as follows:

i) Take an element from R, r_i, except '1'.
ii) For each input pair of an *A-operation* in I_i, generate a two-input AND gate. The inputs of the AND gate are the elements of the input pair, *i.e.*, '1' or the outputs of the OR gates representing the target and intermediate constants.
iii) Generate an OR gate associated with r_i where its inputs are the outputs of AND gates determined in Step (ii).
iv) If r_i is a target constant, assign the output of the corresponding OR gate as the output of the network.
v) Repeat Step (i) until all elements in R are considered.

[2]These values are determined empirically based on experiments.

[3]Considering the shift-adds network of Figure 2(c) as an example, the depth of 7, 29, and 43 is computed as 1, 2, and 3, respectively.

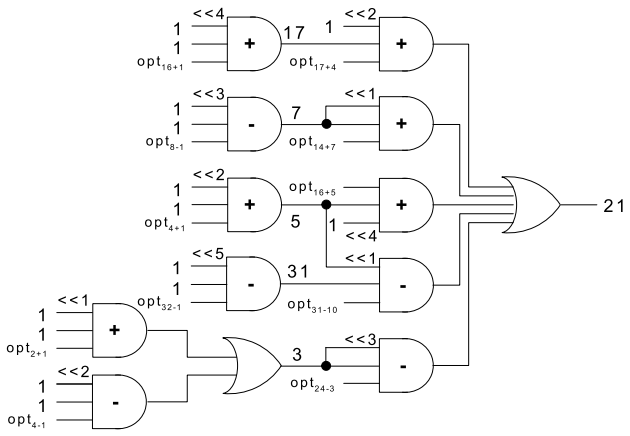

Fig. 4. Inclusion of optimization variables into the network constructed for the target constant 21 with limited implementations.

3) The 0-1 ILP Formalization: We need to include optimization variables into the network, so that the area optimization problem in MCM design on FPGAs can be easily formed as a 0-1 ILP problem. To do this, we associate the optimization variables with addition/subtraction operations that have different implementation costs on FPGAs. Thus, for each AND gate that represents an addition/subtraction operation in the network, we introduce an optimization variable, *i.e.*, $opt_{a\pm b}$, where a and b denote the inputs of an operation, and we add this variable to the input of the AND gate. In the cost function to be minimized, the cost value of the optimization variable is determined as the cost of the *A-operation* given in Table I.

Figure 4 presents the Boolean network generated for the target constant 21 after the optimization variables are added. For simplicity, only five realizations of 21 listed previously are illustrated in this figure. Also, 1-input OR gates for the intermediate constants 5, 7, 17, and 31 are omitted and the type of each operation is shown inside of each AND gate.

After the optimization variables are added into the network, the generation of the 0-1 ILP problem is straightforward. The cost function of the 0-1 ILP problem is constructed as a linear function of optimization variables, where the cost value of each optimization variable is determined as described before. Also, the constraints of the 0-1 ILP problem are obtained by finding the Conjunctive Normal Form (CNF) formulas of each gate in the network and expressing each clause of the CNF formulas as a linear inequality, as described in [17]. For example, a 3-input AND gate, $d = a \wedge b \wedge c$, is translated to CNF as $(a + \overline{d})(b + \overline{d})(c + \overline{d})(\overline{a} + \overline{b} + \overline{c} + d)$ and converted to linear constraints as $a - d \geq 0$, $b - d \geq 0$, $c - d \geq 0$, $-a - b - c + d \geq -2$. The outputs of the network, *i.e.*, the outputs of OR gates associated with the target constants, are set to 1, since the implementation of target constants is aimed.

4) Finding the Minimum Solution: A generic 0-1 ILP solver will search for the minimum value of the cost function on the generated 0-1 ILP problem by satisfying the constraints that represent how the target and intermediate constants are implemented. The addition/subtraction operations, which yield the minimum area solution, are those whose optimization variables are set to 1 by the 0-1 ILP solver.

TABLE II
FIR FILTER SPECIFICATIONS.

Filter	pass	stop	#tap	width
1	0.10	0.15	200	16
2	0.10	0.25	180	16
3	0.10	0.20	240	16
4	0.10	0.20	300	16
5	0.15	0.25	200	16
6	0.15	0.25	240	16
7	0.10	0.15	240	16

IV. EXPERIMENTAL RESULTS

As an experiment set, we used the FIR filters given in Table II, where *pass* and *stop* are the normalized passband and stopband frequencies, respectively, *#tap* is the number of coefficients, and *width* is the bit-width of filter coefficients.

Table III presents the results of high-level algorithms (the exact CSE algorithm [3] when constants are defined under MSD and the Hcub algorithm [5], both designed for the MCM problem, and the proposed LUTOR algorithm) on the multiplier blocks of the FIR filters. In this table, *op* and *as* denote the number of operations and the number of adder-steps in MCM, *i.e.*, the maximum number of operations in series, respectively. Also, *cpu* is the required CPU time in seconds for the high-level algorithms to find a solution on a PC with Intel Xeon at 2.33GHz and 4GB of memory.

Observe that the exact CSE algorithm obtains worse solutions than Hcub and LUTOR in terms of the number of operations due to its limitation on the number representation. The solutions of LUTOR include the same number of operations as Hcub, except Filter 5, but it requires more CPU time to find a solution than Hcub. This is because it iterates on more than one solution of Hcub and the MCM design, that requires the minimum number of 4-input LUTs on FPGAs, is obtained by the 0-1 ILP solver SCIP 2.0 [18]. Note that the minimum and maximum CPU time of SCIP 2.0 to solve the generated 0-1 ILP problems was 0.17s and 0.28s, respectively, indicating that these 0-1 ILP problems are in fact easy to be solved since the implementations of a constant are obtained only from the constants in the solution of Hcub and depth-1 constants.

Table III also presents the synthesis results of the multiplier blocks on Xilinx Virtex 4 FPGA, xc4vfx12-12sf363, where *lut* denotes the number of 4-input LUTs and *slice* stands for the number of slices showing the DSP48 slices in parenthesis. The filter input was assumed to be a signed 8-bit number. On the solutions of LUTOR, the estimated 4-input LUTs (*elut*) values computed from the costs of *A-operations* as given in Table I are also presented. Moreover, the synthesis results of the multiplier blocks designed using LUT-based constant multipliers (*LBCM*) [10] and generic multipliers (*GM*) are given. In the latter, the multiplications of filter coefficients by the filter input are described in VHDL as constant multiplications and the Xilinx ISE 13.1 design tool synthesized the circuit. All the synthesis results given in Table III are obtained before mapping, because each multiplier block does not fit into the target FPGA due to its large number of outputs.

Observe from the results of exact CSE and Hcub algorithms that the reduction of the number of operations in an MCM design also reduces the number of 4-input LUTs when it

TABLE III
SUMMARY OF RESULTS OF DESIGN METHODS AND ALGORITHMS ON MULTIPLIER BLOCKS OF FIR FILTERS.

Fil.	GM		LBCM [10]		Exact CSE - MSD [3]					Hcub [5]					LUTOR					
	lut	slice	lut	slice	op	as	cpu	lut	slice	op	as	cpu	lut	slice	op	as	cpu	elut	lut	slice
1	2155	1156 (17)	1199	635	82	5	1.9	1169	615	79	5	0.2	1114	596	79	7	59.1	869	854	469
2	1187	638 (11)	763	408	53	5	12.6	767	407	47	5	0.1	673	358	47	8	33.5	563	549	301
3	1935	1046 (11)	977	518	66	5	2.4	984	519	63	4	0.1	946	500	63	6	48.7	719	719	393
4	2095	1129 (8)	1043	554	72	4	2.4	1028	544	68	5	0.2	931	499	68	9	51.0	765	754	414
5	1662	896 (10)	905	478	64	4	0.5	910	483	59	4	0.1	811	434	60	9	6.5	663	658	368
6	2092	1122 (7)	1002	527	72	4	0.8	1013	537	69	5	0.2	941	504	69	7	63.8	727	720	397
7	2508	1347 (12)	1240	654	87	5	1.6	1225	645	83	5	0.2	1170	620	83	7	99.2	914	887	488
Avg.	1948	1048 (11)	1018	539	71	5	3.2	1014	536	67	5	0.1	941	502	67	8	51.7	746	734	404

TABLE IV
SUMMARY OF RESULTS OF DESIGN METHODS AND ALGORITHMS ON FIR FILTERS.

Fil.	GM			LBCM [10]			Exact CSE - MSD [3]			Hcub [5]			LUTOR		
	slice	delay	power	slice	delay	power	slice	delay	power	slice	delay	power	slice	delay	power
1	3740 (17)	3.422	377	3161	3.019	362	3101	3.108	372	3093	3.278	384	2948	3.014	381
2	2372 (11)	3.363	335	2141	2.926	323	2084	2.854	327	2035	2.973	317	1970	2.942	326
3	3517 (11)	3.109	358	2963	3.525	362	2904	3.158	362	2898	3.619	363	2773	2.790	358
4	3927 (8)	3.180	367	3324	3.198	366	3258	3.821	367	3226	3.484	365	3119	3.448	369
5	2995 (10)	3.328	342	2562	3.450	347	2508	3.251	342	2463	3.382	340	2395	3.297	346
6	3483 (7)	3.350	357	2892	3.694	358	2830	3.323	360	2805	3.816	364	2687	3.353	360
7	4372 (12)	2.968	384	3634	3.247	380	3570	3.220	411	3552	3.195	389	3399	2.965	391
Avg.	3487 (11)	3.246	360	2954	3.294	357	2894	3.248	363	2867	3.392	360	2756	3.116	362

is synthesized on FPGAs. However, LUTOR obtains the best solutions in terms of the number of 4-bit LUTs, where the maximum gain over the exact CSE and Hcub algorithms is 29% and 24%, respectively. Also, the estimated number of 4-input LUT values are close to the synthesis values. Moreover, the shift-adds design of multiplier blocks with the use of high-level algorithms reduces the complexity of the MCM design significantly with respect to those designed using LUT-based constant multipliers [10] and generic multipliers.

Table IV presents the implementation results of complete FIR filters whose multiplier blocks are realized as given in Table III. These results are obtained after the mapping and placement and routing. In this table, *delay* denotes the delay in ns in the critical path and *power* stands for the power dissipation in mW obtained by the XPower tool based on simulations with 10,000 random inputs using the ISim simulator. The functionality of the FIR filter designs was also verified during these simulations.

Observe from Tables III and IV that the register-add circuit that computes the filter output as shown in Figure 1 dominates the complexity of the FIR filters due to the large number of filter coefficients. However, the solutions of LUTOR lead to the least complex FIR filters on all instances, where its maximum gain on the number of slices over the exact CSE and Hcub algorithms is 5.5% and 4.7%, respectively. The FIR filters designed based on the solutions of LUTOR have also similar delay and power dissipation results on average to those of filter designs obtained by the MCM algorithms [3], [5]. The shift-adds design of FIR filters incorporation with high-level algorithms yields the best designs in terms of area with respect to FIR filters whose MCM parts are realized using LUT-based constant multipliers [10] or generic multipliers. Note that some of generic multipliers are realized with DSP48 slices, which are not used in the LUT-based and shift-adds implementations.

V. CONCLUSIONS

We introduced a high-level synthesis algorithm that optimizes the number of required 4-input LUTs in an MCM design and its efficiency was shown on design of multiplier blocks and FIR filters on FPGAs. The proposed approach can incorporate any MCM heuristic and can be applied to various optimization problems by only changing the 0-1 ILP formalization.

REFERENCES

[1] R. Hartley, "Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers," *IEEE TCAS-II*, vol. 43, no. 10, pp. 677–688, 1996.

[2] I.-C. Park and H.-J. Kang, "Digital Filter Synthesis Based on Minimal Signed Digit Representation," in *DAC*, 2001, pp. 468–473.

[3] 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*, vol. 27, no. 6, pp. 1013–1026, 2008.

[4] A. Dempster and M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *IEEE TCAS-II*, vol. 42, no. 9, pp. 569–577, 1995.

[5] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, p. ., 2007.

[6] L. Aksoy, E. Gunes, and P. Flores, "Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate," *Elsevier Journal on Microprocessors and Microsystems*, vol. 34, no. 5, pp. 151–162, 2010.

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

[8] F. de Dinechin and V. Lefevre, "Constant Multipliers for FPGAs," in *PDPTA*, 2001, pp. 167–173.

[9] N. Brisebarre, F. de Dinechin, and J.-M. Muller, "Integer and Floating-Point Constant Multipliers for FPGAs," in *ASAP*, 2008, pp. 239–244.

[10] M. Faust and C. H. Chang, "Bit-Parallel Multiple Constant Multiplication Using Look-Up Tables on FPGA," in *ISCAS*, 2011, pp. 657–660.

[11] P. Cappello and K. Steiglitz, "Some Complexity Issues in Digital Signal Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 5, pp. 1037–1041, 1984.

[12] Xilinx Corporation, Design Tips for HDL Implementation of Arithmetic Functions, Application Note 215, 2000.

[13] O. Gustafsson and L. Wanhammar, "ILP Modelling of the Common Subexpression Sharing Problem," in *ICECS*, 2002, pp. 1171–1174.

[14] Xilinx Corporation, Constant Coefficient Multipliers for the XC4000E, Application Note 054, 1996.

[15] M. Wirthlin, "Constant Coefficient Multiplication Using Look-Up Tables," *Journal of VLSI Signal Processing*, vol. 36, no. 1, pp. 7–15, 2004.

[16] O. Gustafsson, "Towards Optimal Constant Multiplication: A Hypergraph Approach," in *Proceedings of Asilomar Conference on Signals, Systems and Computers*, 2008, pp. 1805–1809.

[17] P. Barth, "A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization," Max-Planck-Institut Fur Informatik, Tech. Rep., 1995.

[18] Solving Constraint Integer Programs website, http://scip.zib.de/.