Efficient Design of FIR Filters Using Hybrid Multiple Constant Multiplications on FPGA

Levent Aksoy[†], Paulo Flores[†‡] and José Monteiro[†‡]
[†] INESC-ID, Lisbon, Portugal
[‡] Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
levent@algos.inesc-id.pt, pff@inesc-id.pt, jcm@inesc-id.pt

***Abstract*—The multiple constant multiplication (MCM) block, which realizes the multiplication of constants by a variable, is a ubiquitous operation in digital signal processing (DSP) systems. It can be implemented using generic multipliers or shifts and adders/subtractors. This paper addresses the problem of finding the minimum number of adders/subtractors to realize the MCM block while a number of multipliers are available to realize some constant multiplications. Such a situation appears in the design of DSP systems on field programmable gate arrays (FPGAs) which also include generic multipliers. We present a 0-1 integer linear programming (ILP) formulation of this problem, yielding an exact common subexpression elimination (CSE) method. Due to the NP-completeness of this problem, we also introduce an approximate graph-based (GB) algorithm. Experimental results show that the proposed methods can find better solutions than a state-of-art algorithm and the use of different number of multipliers in the MCM block leads to filter designs with different number of slices, delay, and power dissipation which enable a designer to choose the one that fits best in an application.**

I. Introduction

Finite impulse response (FIR) filters are used in many DSP applications such as audio, image, and video processing. The transposed form FIR filter is depicted in Fig. 1. Its design complexity is dominated by the MCM block, because the implementation of a multiplier in hardware is expensive in terms of area. Since the filter coefficients are determined beforehand, the MCM block of the FIR filter is generally realized under the shift-adds architecture using only shifts and adders/subtractors [1]. Note that shifts by a constant value require only wires which represent no hardware cost. The *MCM problem* is defined as finding the minimum number of operations which realize the MCM block and was proven to be NP-complete in [2]. Over the years, many efficient algorithms have been introduced for the MCM problem [3]–[7].

FPGAs have been successfully used in many DSP applications due to their programmable nature and simple design cycle. They consist of configurable logic blocks (CLBs) connected via programmable interconnects. Today's FPGAs have configurable embedded SRAM, high-speed transceivers, high-speed I/Os, and DSP blocks which support many independent functions including multipliers with two non-constant inputs.

This paper presents the realization of the MCM operation on an FPGA using multipliers in its DSP blocks together with adders/subtractors realized in its CLBs, as previously presented in [8]. In this so-called *hybrid MCM problem*, given the constants to be multiplied by an input variable and the number of multipliers available for constant multiplications, *i.e.*, m, the aim is to implement the MCM block using a minimum number of adders/subtractors and at most m multipliers. Note that if m is 0, the hybrid MCM problem turns to an MCM problem.

To this end, we present a 0-1 ILP formulation of the hybrid MCM problem, where possible realizations of constant multiplications are extracted from the number representations of constants, yielding an exact CSE algorithm called FREYJA. We also introduce an approximate algorithm called FREYR. In an iterative loop, FREYR first finds a different solution to the MCM problem using an efficient GB algorithm [6] at each time. Second, based on this solution, it decides which constant multiplications should be realized using at most m multipliers so that a minimum number of adders/subtractors is required for the MCM design. This decision problem is also formulated as a 0-1 ILP problem similar to the one described in FREYJA, but generating a 0-1 ILP problem whose size is much smaller than that obtained by FREYJA. Experimental results show that FREYJA can be applied to real-size instances. However, FREYR can find better solutions than FREYJA, since it considers more possible realizations of constants than FREYJA. Also, both algorithms can obtain better solutions than the state-of-art algorithm of [8]. The results of filter designs on FPGA indicate that the use of generic multipliers in the MCM block, which are realized in the DSP blocks of FPGA, leads to filter designs with less delay and power consumption as well as less number of slices. It is shown that by changing the m value, the tradeoffs between the number of slices and delay and between the number of slices and power dissipation can be explored and an FIR filter that fits best in an application can be found.

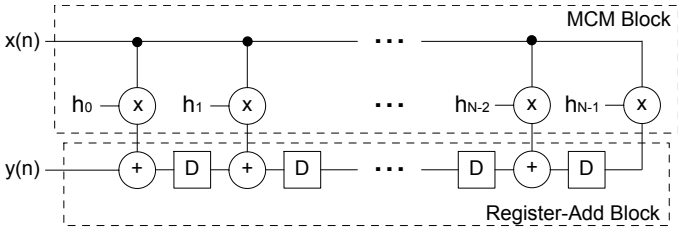

Fig. 1. Transposed form FIR filter.

II. Background

This section presents the background concepts on the number representation, 0-1 ILP problem, shift-adds design of the MCM operation, and hybrid MCM operation.

A. Number Representation

The *binary* representation decomposes a number in a set of additions of powers of two. The signed digit system makes the use of positive and negative digits. The *canonical signed digit* (CSD) representation [3] has two main properties: i) two nonzero digits are not adjacent; ii) the number of nonzero digits is minimum. The *minimal signed digit* (MSD) representation [4] is obtained by dropping the first property of CSD. Thus, a constant may have several representations under MSD, but all with a minimum number of nonzero digits.

Consider 23 defined in six bits. Its binary representation, 010111, includes 4 nonzero digits. It is represented as $10\bar{1}00\bar{1}$

978-1-4799-6492-5/14/$31.00 ©2014 IEEE

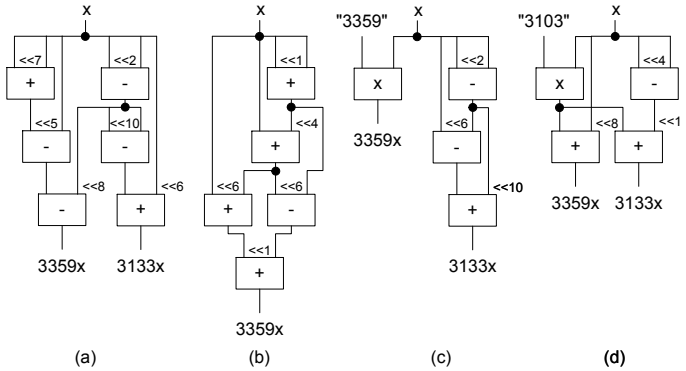

Fig. 2. Multiplierless designs of $3133x$ and $3359x$: (a) exact CSE algorithm [5]; (b) exact GB algorithm [7]; designs of $3133x$ and $3359x$ when one generic multiplier is available: (c) FREYJA; (d) FREYR.

in CSD and both $10\bar{1}00\bar{1}$ and $01100\bar{1}$ denote 23 in MSD using 3 nonzero digits, where $\bar{1}$ stands for -1.

B. 0-1 ILP Problem

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

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

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

In the objective function of the 0-1 ILP problem given in Eq. 1, w_i in $\mathbf{w}$ is a weight value associated with each variable x_i, where $1 \leq i \leq k$. In Eq. 2, $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ denotes the set of j linear constraints, where $\mathbf{b} \in \mathbb{Z}^j$ and $\mathbf{A} \in \mathbb{Z}^j \times \mathbb{Z}^k$.

C. Multiplierless Design of the MCM Operation

The algorithms proposed for the MCM problem can be categorized in two classes as CSE and GB techniques. Their aim is to maximize the sharing of common partial products. The CSE methods [3]–[5] first define the constants under a number representation such as, binary, CSD, or MSD. Then, considering possible subexpressions which can be extracted from the nonzero digits in representations of constants, the "best" subexpression, generally, the most common, is chosen to be shared among the constant multiplications. The drawback of CSE methods is their dependency on a number representation. The GB algorithms [6], [7] are not restricted to any particular number representation and aim to find intermediate partial products which enable to realize the constant multiplications with a minimum number of operations. They consider a larger number of realizations of a constant and obtain better solutions than the CSE methods, but require more computational resources due to a larger search space. As an example, consider the constant multiplications $3133x$ and $3359x$. The exact CSE method [5] finds a solution with 6 operations when constants are defined under CSD, sharing the common subexpressions $3x$ among the constant multiplications (Fig. 2a). The exact GB method [7] obtains a solution with 5 operations, finding the common intermediate partial products $3x$ and $49x$ (Fig. 2b).

The delay of a multiplierless MCM design is generally defined as the number of adder-steps that denotes the maximum number of operations in series [9]. For our example, the MCM designs in Figs. 2a-b have 3 and 4 adder-steps, respectively.

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

D. Hybrid MCM Operation

To the best of our knowledge, the problem of finding a hybrid MCM realization consisting of at most m multipliers and the smallest number of adders/subtractors was only addressed in [8]. The algorithm of [8] is based on the method of [10], targets the pipelined design of the MCM block, and aims to find a design, including adders/subtractors and pipeline registers, with the least complexity. Note that a CLB block also includes flip-flops and thus, the registers for an adder/subtractor come for free. This is also true for the generic multipliers in the DSP blocks of FPGA. However, in order to carry the output of an adder/subtractor or a multiplier located at depth d to the input of another adder/subtractor at depth $d+2$ or more, non-free pipeline registers are required. Hence, the method of [8] targets the design of an MCM block with the minimum number of adder-steps as the method of [10].

This paper considers the most fundamental form of the problem presented in [8], *i.e.*, the *hybrid MCM problem*. For our MCM example, suppose that one generic multiplier is made available to be used in the MCM block. The solutions of FREYJA and FREYR include 3 adders/subtractors (Fig. 2c-d), leading to a 50% and 40% gain in terms of the number of adders/subtractors with respect to the solutions of the exact CSE algorithm [5] (Fig. 2a) and the exact GB algorithm [7] (Fig. 2b), respectively. Observe that the constant multiplication chosen to be realized using a multiplier may be a required constant multiplication as in the solution of FREYJA or an intermediate partial product as in the solution of FREYR.

III. Proposed Algorithms

This section introduces an exact CSE algorithm and an approximate GB method proposed for the *hybrid MCM problem*. In their preprocessing phase, each constant to be multiplied by the variable x is multiplied by -1 if it is negative and divided by 2 until it is odd. Then, this positive and odd constant, except 1 that denotes the variable x, is stored in a set called target set T without repetition. Note also that the shift-adds realization of constant multiplications is in fact equal to the realization of constants. For example, $3x$ realized as $3x{=}x{\ll}1{+}x$ can be rewritten as $3{=}1{\ll}1{+}1$ by removing the variable x from both sides. This terminology is used interchangeably in this section.

A. FREYJA: An Exact CSE Algorithm

To increase the performance of FREYJA, we previously computed all possible realizations of odd constants between 3 and $2^{16}-1$ under binary, CSD, and MSD representations and made them available to FREYJA by storing them in a file. To do so, a constant is represented under a number representation and the nonzero digits in its representation are decomposed into two parts, considering all possible combinations. The same realizations, that can be obtained due to the commutative law of addition, are not considered. As an example, consider 45 under CSD, $10\bar{1}0\bar{1}01$. Its realizations are given in Fig. 3a. Note that a constant with a representation having n nonzero digits has $2^{n-1}-1$ possible realizations. Under MSD, each of multiple representations of a constant is considered similarly.

FREYJA has four parts which are described in detail next.

1) Determining Partial Terms: The steps of the first part are given as follows, where P is a set of sets which will include all partial terms required to realize each constant of T.

I) Take an unimplemented element from T, t_i, and extract its realizations from the previously generated file.

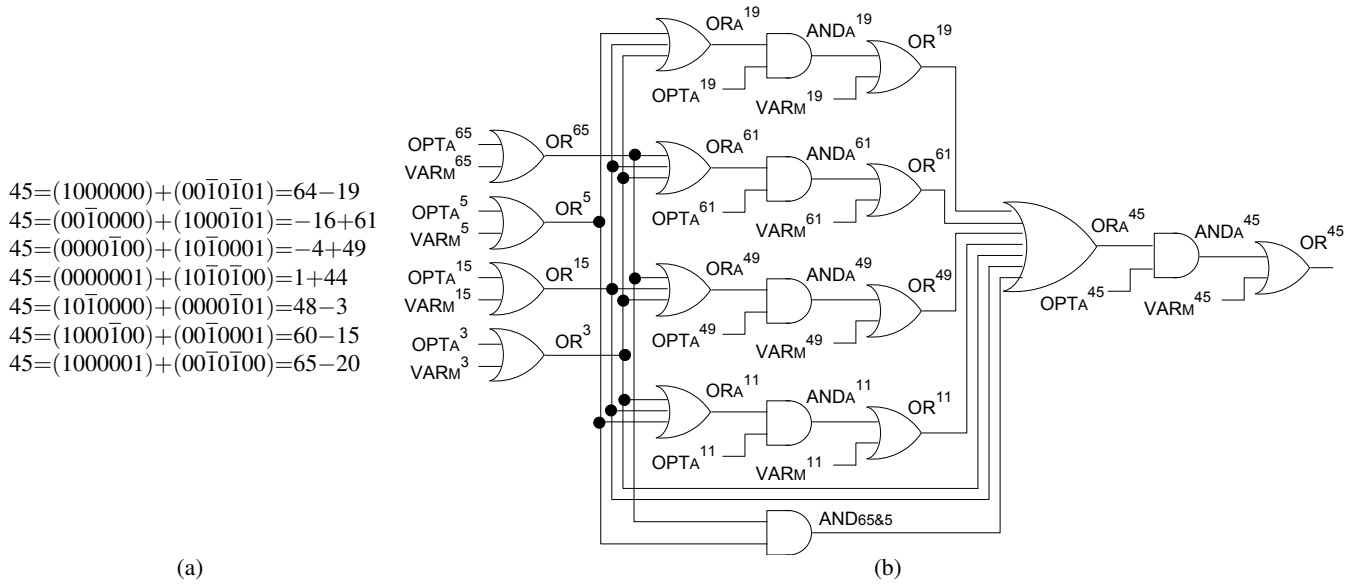

Fig. 3. (a) Possible implementations of 45 under CSD; (b) Boolean network generated for the realizations of 45 under CSD.

II) For each operation that realizes t_i;
   a) Convert the inputs of an adder/subtractor to positive and odd integers, determine the non-repeated partial terms that are not equal to 1, and store them in a set called *Iset*. Thus, *Iset* may be empty or may contain a single partial term or a pair of partial terms.
   b) If *Iset* is empty, make P_i empty and go to Step IV.
   c) Otherwise, check each element of P_i;
      i) If $P_i(k)$ dominates[2] *Iset*, go to Step II.
      ii) If *Iset* dominates $P_i(k)$, delete $P_i(k)$.
   d) Add *Iset* to P_i.

III) Add all partial terms in P_i to T without repetition and label them as unimplemented.

IV) Label t_i as implemented and repeat Step I until all elements in T are labeled as implemented.

In Step IIb, we determine that an element of T can be realized using a single operation whose inputs are 1 or the shifted version of 1. In Step IIc, we avoid the repetition of a single partial term or a pair of partial terms and remove the redundant partial terms determined by the dominance rule [11].

For our example in Fig. 3a, P_1, which includes the partial terms required for the implementation of 45, is found as $\{19, 61, 49, 11, 3, 15, (65,5)\}$. Then, the partial terms in P_1 are added to T and the partial terms required for their implementations are found in a similar way. Note that while the target set T only consists of target constants to be realized in the MCM block in the beginning of this part, it is augmented with partial terms that are required for the realization of these constants.

2) Constructing a Boolean Network: The Boolean network represents the implementations of constants both using adders/subtractors and multipliers[3]. It includes only AND and OR gates. The representation of realizations of constants under the shift-adds architecture is done as follows:

I) For each constant in T, t_i, if P_i is empty, assign t_i as a primary input (PI) of the network.

II) Otherwise, generate an OR gate corresponding to the constant t_i, $\text{OR}_A^{t_i}$. For each single partial term in P_i, a, assign the output of an OR gate related to this constant, OR^a (to be described below), to the input of $\text{OR}_A^{t_i}$. For each pair of partial terms in P_i, b and c, generate an AND gate corresponding to this pair, $\text{AND}_{b\&c}$, whose inputs are respectively the outputs of OR gates related to these constants, OR^b and OR^c (to be described below). Assign the outputs of these AND gates to the input of $\text{OR}_A^{t_i}$.

To associate the optimization variables of the 0-1 ILP problem with the constants of T for their realizations under the shift-adds architecture, for each OR gate $\text{OR}_A^{t_i}$, we generate a 2-input AND gate, $\text{AND}_A^{t_i}$, where one of its inputs is the output of $\text{OR}_A^{t_i}$ and the other is the optimization variable associated with t_i, $\text{OPT}_A^{t_i}$. Each PI of the network denoting a constant of T, t_i, is also represented with an optimization variable $\text{OPT}_A^{t_i}$.

To represent the realizations of constants using multipliers, for each constant of T, t_i, we generate a variable $\text{VAR}_M^{t_i}$.

To indicate that a constant multiplication can be realized using a multiplier or an adder/subtractor, for each constant of T, t_i, we generate a 2-input OR gate, OR^{t_i}, where one of its inputs is $\text{VAR}_M^{t_i}$ and the other is $\text{OPT}_A^{t_i}$ if t_i is a PI of the network, otherwise it is $\text{AND}_A^{t_i}$.

Fig. 3b shows the network generated for the constant 45.

3) Generating the 0-1 ILP Problem: The objective function of the 0-1 ILP problem is a linear function of optimization variables with weight values set to 1. To obtain the constraints of the 0-1 ILP problem, first, we find the conjunctive normal form (CNF) formulas of each gate in the network and express each clause of the CNF formulas as a linear inequality [12]. For example, a 2-input AND gate, $c = a \wedge b$, is translated to CNF as $(a+\bar{c})(b+\bar{c})(\bar{a}+\bar{b}+c)$ and converted to linear constraints as $a-c \geq 0$, $b-c \geq 0$, $-a-b+c \geq -1$. Second, to avoid the violation of the number of available multipliers, a single constraint $\sum_{i=1}^{|T|} \text{VAR}_M^{t_i} \leq m$ is added to the 0-1 ILP problem. Third, for each element of the target set T obtained in the preprocessing phase, t_i, its variable OR^{t_i} is set to 1, since these are the constants to be realized in the MCM block.

4) Finding a Minimum Solution: A 0-1 ILP solver is used to find a solution of the 0-1 ILP problem. In its solution, the variables $\text{VAR}_M^{t_i}$ set to 1 are the constants to be realized using multipliers. The variables $\text{OPT}_A^{t_i}$ set to 1 are the constants to be

[2] A dominates B, if $A \bigcap B = A$.

[3] The variables denoted with the subscript A and M are related to the adders/subtractors and multipliers, respectively.

Algorithm 1 The FREYR algorithm.

FREYR(**T, m**)

1: $BC = \infty, BS \leftarrow \{\}, seed = 0, S_{set} \leftarrow \{\}$
2: **repeat**
3: $seed = seed + 1$
4: O = MCMwHcub(T, $seed$)
5: S = GenerateSynthSet(O)
6: S = AddDepth1Constants(S)
7: **if** $S \notin S_{set}$ **then**
8: $S_{set} \leftarrow S_{set} \cup S$
9: $[HS, cost]$ = FindHybridMCM(T, m, S)
10: **if** $cost < BC$ **then**
11: $BC = cost$, $BS \leftarrow HS$
12: **until** Termination conditions meet
13: **return** BS

implemented under the shift-adds architecture. The realizations of these constants are found from the previously generated file as adders/subtractors whose inputs are 1, the constants selected by the 0-1 ILP solver, or their shifted versions.

B. FREYR: An Approximate GB Algorithm

The size of a 0-1 ILP problem generated by FREYJA grows exponentially as the number of constants and the number of nonzero digits in the representations of constants increase, limiting its application to certain MCM instances. Hence, we introduce an approximate algorithm FREYR which can be applied to instances that the exact CSE algorithm cannot handle and find a solution in a reasonable time. FREYR is an iterative procedure and consists of two parts. First, a different MCM solution to the MCM problem is found by the MCM algorithm Hcub [6] at each time. Second, based on this MCM solution, the problem of determining which constant multiplications to be realized using the fewest adders/subtractors and at most m multipliers is formulated as a 0-1 ILP problem and a hybrid MCM solution is found. Its pseudo-code is given in Algorithm 1, where *BC* is the best cost value in terms of the number of adders/subtractors, *BS* is the best solution consisting of at most m multipliers and *BC* adders/subtractors realizing the constant multiplications, *seed* is an integer parameter used in Hcub to find a different MCM solution, and S_{set} includes all the sets of constants considered in FREYR. It is described using an example with T=$\{171, 211\}$ when m is 1.

In the iterative loop of FREYR, the *MCMwHcub* function finds a solution to the MCM operation including the constants of T with a different *seed* value at each time. For our example, the solution of Hcub to T when *seed* is 1 includes 5 operations (Fig. 4a). In the *GenerateSynthSet* function, from the solution of Hcub, *i.e.*, a set of adders/subtractors, we extract all the outputs of operations realizing constants, compute their depths in the MCM design, and store these constants (including the constants of T) in a set called S in an ascending order based on their depth values. For our example, S is formed as $\{65, 69, 211, 341, 171\}$. Then, the *AddDepth1Constants* function augments S with the constants $2^i \pm 1$ without repetition, where i ranges between 2 and $mbw+1$, paying attention to the order of constants based on their depth values. Note that *mbw* is the maximum bitwidth of constants of T. The reason behind adding depth 1 constants is to increase the number of possible realizations of a constant to be found in the *FindHybridMCM* function (described next). For our example, S becomes {3, 5, 7, 9, 15, 17, 31, 33, 63, 65, 127, 129, 255, 257, 511, 513, 69, 211, 341, 171} after the depth 1 constants are added. If S is

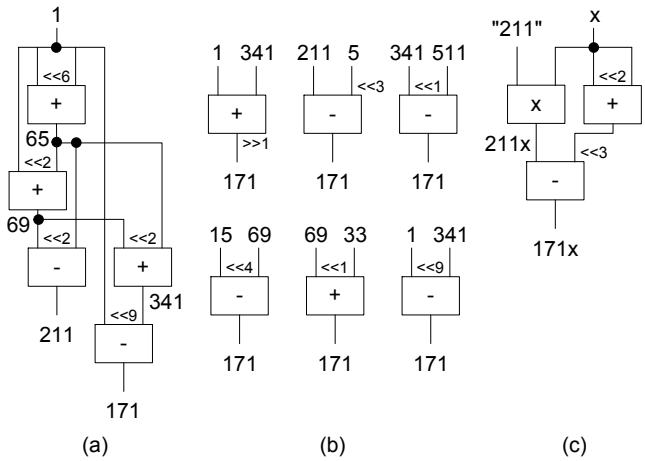

Fig. 4. (a) A solution of Hcub on $T = \{171, 211\}$; (b) 6 possible realizations of 171 found based on the solution of Hcub in the *FindHybridMCM* function; (c) a solution of the *FindHybridMCM* function on T when m is 1.

not considered before, the *FindHybridMCM* function decides which constants of S to be realized using a minimum number of adders/subtractors and at most m multipliers, implementing all the constant multiplications of T. This problem is formulated as a 0-1 ILP problem similar to the one described in the exact CSE algorithm FREYJA. The only difference is finding the realizations of constants. In this function, the realization of a constant is in the form of *A-operation* [6] given as follows:

$$A(x,y) = w = |x \ll l_x + (-1)^{s_y} y \ll l_y| \gg r \tag{3}$$

where l_x and l_y, which are greater than or equal to 0, stand for the left shifts of the operands x and y respectively, s_y, which is equal to 0 or 1, determine the sign of y, and r, which is greater than or equal to 0, is the right shift of the output. Note that finding a realization of a constant means finding the values of these parameters of the *A-operation* in Eq. 3. Thus, for a constant of S, s_i, we find all possible *A-operations* realizing s_i when x and y are 1 or a constant of S, s_j, where $j<i$, l_x and l_y are less than or equal to $mbw+1$, and r is less than or equal to $mbw-\lceil log_2 s_i \rceil+1$. Note that all these parameters are searched exhaustively. In our example, other than the depth 1 constants, which have a single realization, 6, 1, 4, and 6 possible realizations are found for 69, 211, 341, and 171, respectively, all of which include the ones found by Hcub. For example, 6 possible realizations found for 171 are shown in Fig. 4b. Thus, the size of the 0-1 ILP problem depends on the constants of S and is much smaller than that of the one generated by FREYJA. After the 0-1 ILP problem is generated and solved, the *FindHybridMCM* function returns a hybrid solution *HS*, *i.e.*, a set of multipliers and adders/subtractors realizing the MCM operation, together with the number of required adders/subtractors, *i.e.*, *cost*. For our example, a solution with 2 adders/subtractors is found (Fig. 4c). If *cost* is better than the best cost value found so far, *BC*, the best solution *BS* and *BC* are updated with *HS* and *cost*, respectively. FREYR terminates if one of the following three conditions is satisfied: i) if *seed* reaches 100; ii) if the last 20 *HS* values found in sequence have cost values greater than or equal to *BC*; iii) if the last 10 S sets obtained in sequence have been considered before. For our example, FREYR tries all other solutions of Hcub until its termination, but cannot find a better solution than the one given in Fig. 4c.

TABLE I. SUMMARY OF AVERAGE RESULTS OF ALGORITHMS ON RANDOMLY GENERATED INSTANCES.

N	MCM									Hybrid MCM									
	[5]-CSD			[5]-MSD			[6]			m	FREYJA-CSD			FREYJA-MSD			FREYR		
	op	as	cpu	op	as	cpu	op	as	cpu		op	as	cpu	op	as	cpu	op	as	cpu
2	6.0	3.7	5.4	5.5	3.7	29.2	5.4	3.9	6.2	1	3.2	3.1	6.4	3.0	2.9	38.3	2.9	2.6	6.8
3	8.1	3.8	9.1	7.6	3.8	42.5	7.3	4.6	8.1	1	5.4	3.5	13.1	5.2	3.6	52.0	5.0	3.5	7.9
5	11.7	3.5	10.4	11.1	3.6	36.7	10.0	5.4	8.1	2	6.9	3.5	13.9	6.6	3.3	38.6	5.9	3.5	8.3
7	15.6	3.9	23.8	14.4	3.6	129.7	12.8	5.8	9.3	3	8.6	3.7	24.7	8.0	3.5	119.2	7.0	3.8	9.4
10	20.5	3.8	32.1	19.2	3.8	274.6	16.3	7.5	9.4	5	9.9	3.6	37.8	9.3	3.7	145.8	7.7	3.7	10.2
15	27.2	3.8	41.6	25.8	4.0	480.6	21.1	8.0	10.0	7	13.8	3.6	44.2	12.9	3.7	350.8	10.2	3.7	10.3
20	34.9	3.9	60.9	32.6	4.1	666.3	26.5	9.4	12.1	10	16.6	3.6	63.3	15.7	3.8	320.2	12.1	4.3	12.5
30	46.6	4.0	104.2	44.0	4.1	1400.3	35.1	11.1	10.9	15	22.2	4.0	92.3	20.5	3.7	414.0	16.0	4.2	10.0
50	69.7	4.1	267.0	66.2	4.2	3143.2	53.1	12.0	6.8	25	32.5	4.0	118.8	30.3	3.9	783.4	25.0	4.3	0.6
75	96.7	4.3	504.4	92.4	4.3	3705.8	76.7	12.1	3.6	37	45.6	4.3	143.1	42.3	4.2	937.4	38.0	4.4	0.8
100	122.3	4.5	684.8	117.6	4.5	2539.3	101.8	12.1	3.0	50	56.4	4.5	143.2	52.9	4.2	280.1	50.0	4.5	1.1

TABLE II. SUMMARY OF RESULTS OF ALGORITHMS ON FIR FILTERS.

Fil.	N_u	$m = \lfloor N_u/4 \rfloor$						$m = \lfloor N_u/2 \rfloor$							$m = \lfloor 3N_u/4 \rfloor$						
		m	[8]		FREYJA		FREYR		m	[8]		FREYJA		FREYR		m	[8]		FREYJA		FREYR	
			op	as	op	as	op	as		op	as	op	as	op	as		op	as	op	as	op	as
A	2	0	7	3	6	3	6	3	1	4	3	3	3	3	3	1	4	3	3	3	3	3
B	4	1	7	3	6	4	7	4	2	4	2	4	3	4	4	3	2	2	2	2	2	2
C	6	1	10	3	9	3	9	6	3	5	3	5	3	5	3	4	3	2	3	3	3	3
D	8	2	10	3	9	4	8	3	4	7	3	6	3	4	2	6	2	1	2	1	2	1
E	13	3	17	3	15	3	13	4	6	13	2	11	3	8	4	9	6	2	6	3	4	2
F	20	5	22	3	19	3	15	6	10	13	2	11	2	10	4	15	6	2	5	2	5	2
G	30	7	33	2	28	3	23	4	15	22	2	18	3	15	3	22	12	2	9	3	8	2
H	53	13	44	3	40	4	40	6	26	30	2	27	4	27	5	39	13	2	14	2	14	2
I	70	17	63	3	54	5	53	10	35	39	2	35	4	35	5	52	21	2	18	3	18	2
Avg.			23.7	2.9	20.7	3.6	19.3	5.1		15.2	2.3	13.3	3.1	12.3	3.7		7.7	2.0	6.9	2.4	6.6	2.1

IV. EXPERIMENTAL RESULTS

This section presents the results of FREYJA and FREYR on randomly generated instances and FIR filters and compares them with those of the method of [8]. FREYJA and FREYR were written in MATLAB and run on a PC with Intel Xeon at 2.4GHz. They use SCIP2.0 [13] as the 0-1 ILP solver. This section also introduces the results of FIR filters designed based on the solutions of FREYR. In this experiment, the Xilinx Virtex 6 xc6vlx75T-2ff484 FPGA device was used. Note that a Virtex 6 slice contains four LUTs and eight flip-flops and this device has 288 DSP48E1 blocks. The FIR filters were described in VHDL when the bitwidth of the filter input was 16 and the Xilinx ISE Design Suite 13.1 was used as the synthesis tool. The functionality of filters was verified on 10,000 randomly generated input signals in simulation, from which we obtained the switching activity information that was used by the synthesis tool to compute the power dissipation.

As the first experiment set, we used randomly generated MCM instances with the number of constants (N) 2, 3, 5, 7, 10, 15, 20, 30, 50, 75, and 100. For each group, there were 30 instances. The positive and odd constants were generated in between $2^{12}+1$ and $2^{13}-1$. Table I presents the results of MCM algorithms, *i.e.*, the exact CSE method [5] and Hcub [6]. The exact CSE method [5] was run when constants are defined under CSD and MSD and Hcub [6] was run with different seeds as done in FREYR and its best solution with the fewest operations was found. This table also shows the results of FREYJA and FREYR when the number of multipliers, m, is $\lfloor N/2 \rfloor$. In this table, *op*, *as*, and *cpu* stand respectively for the number of adders/subtractors, the number of adder-steps, and CPU time in seconds, all in average.

For the MCM instances, Hcub [6] finds significantly better solutions than the exact CSE algorithm [5] in terms of the number of adders/subtractors and CPU time. However, its solutions include a large number of adder-steps. Also, the use of MSD representation leads to MCM designs including less number of operations with respect to the CSD representation, since a constant may have multiple representations under MSD, including the one under CSD. However, due to this fact, the exact CSE algorithm [5] under MSD generates a larger 0-1 ILP problem and requires more CPU time than the exact CSE algorithm [5] under CSD. On the other hand, for the hybrid MCM instances, the observations stated for the CSE and GB MCM algorithms are also valid for FREYJA and FREYR. However, the solutions of FREYJA and FREYR are closer to each other than those of the MCM algorithms. This is because the use of multipliers reduces the number of constants to be realized using adders/subtractors. For example, the use of 1 multiplier eliminates 2.5 adders/subtractors on average which can be observed on the solutions of Hcub and FREYR on instances with 2 constants. Also, the hybrid MCM designs have less number of adder-steps than the MCM designs. As N increases, the CPU time required for FREYJA decreases with respect to those of the exact CSE algorithm [5]. Although the sizes of 0-1 ILP problems generated by FREYJA are larger than those obtained by the exact CSE algorithm [5], this is because the 0-1 ILP problem becomes simpler to the 0-1 ILP solver when $m>0$. This fact is also valid between Hcub and FREYR.

As the second experiment set, we used the filters given in [8]. Table II presents the results of the hybrid MCM algorithms when m was $\lfloor N_u/4 \rfloor$, $\lfloor N_u/2 \rfloor$, and $\lfloor 3N_u/4 \rfloor$, where N_u denotes the number of unique positive and odd filter coefficients. Note that the results of FREYJA were obtained under MSD. Observe that the proposed methods always find solutions including the same or less number of adders/subtractors than the algorithm of [8]. This is simply because the algorithm of [8] does not primarily target the optimization of the number of operations. However, its hybrid MCM designs have the same or less number of adder-steps than the proposed algorithms, since it aims to find a hybrid MCM design with the minimum

TABLE III. SPECIFICATIONS OF FIR FILTERS.

Filter	filter length	N_u	normalized passband	normalized stopband	quantization value
1	100	46	0.11	0.13	16
2	180	83	0.20	0.21	16

TABLE IV. SUMMARY OF RESULTS OF FIR FILTER DESIGNS.

Filter	m	op	as	Sl	DSP48E1	D	P
1	0	50	11	1160	0	15.5	1531
	$\lfloor N_u/4 \rfloor$	36	8	1078	12	12.1	1492
	$\lfloor N_u/2 \rfloor$	24	4	1008	24	8.8	1494
	$\lfloor 3N_u/4 \rfloor$	12	4	934	36	8.7	1498
	N_u	0	0	875	46	8.9	1505
2	0	84	9	2065	0	17.2	1574
	$\lfloor N_u/4 \rfloor$	63	8	1918	21	12.6	1536
	$\lfloor N_u/2 \rfloor$	42	7	1776	42	11.0	1544
	$\lfloor 3N_u/4 \rfloor$	21	3	1685	63	10.2	1556
	N_u	0	0	1559	83	10.8	1563

number of adder-steps. Also, the solutions of FREYJA and FREYR in terms of the number of adders/subtractors are very close to each other and they get closer as m increases.

As the third experiment set, we generated two low-pass FIR filters using the *firgr* function of MATLAB. Table III presents their specifications. The MCM blocks of these filters were realized based on the solutions of FREYR. Table IV presents the results of filter designs when m is 0, $\lfloor N_u/4 \rfloor$, $\lfloor N_u/2 \rfloor$, $\lfloor 3N_u/4 \rfloor$, and N_u, where *Sl*, *DSP48E1*, *D*, and *P* denote the number of slices, the number of DSP48E1 blocks, the critical path delay in *ns*, and power dissipation in *mW*, respectively.

Observe from Table IV that as the number of multipliers m is increased, the number of required adders/subtractors op, and consequently, the number of occupied slices, decrease. Also, the increase in m causes a reduction in the number of adder-steps of the MCM design as, which generally reduces the delay of filter designs. Since m, op, and as have an impact on the power dissipation, a change in these parameters leads to filter designs with different power consumption values.

To explore the tradeoff between the number of slices and delay and between the number of slices and power dissipation, we used the Filter 1 of Table III and obtained its designs when m ranges between 0 and 46 in steps of 2. Figs. 5 and 6 present these tradeoffs. Observe that FIR filters with different number of slices, delay, and power dissipation values are obtained when m is changed. We note that the maximum and minimum values of delay are obtained as 17.1*ns* and 8.6*ns* when m is 2 and 36, respectively and the maximum and minimum values of power dissipation are found as 1531*mW* and 1490*mW* when m is 0 and 14, respectively. This is a clear evidence that the minimum delay and power dissipation values may be found when m is neither 0 nor N_u, but in between these values. Note that as m increases, the generic multipliers become more dominant on the delay and power dissipation of the design.

V. CONCLUSIONS

This paper addressed the problem of finding the minimum number of adders/subtractors in a hybrid MCM design, where the constant multiplications can be realized using both multipliers and adders/subtractors on FPGA. To the best of our knowledge, FREYJA and FREYR are the first algorithms proposed for this problem. Experimental results showed that they can obtain better solutions than an existing algorithm targeting pipelined hybrid MCM designs. Also, it was shown that the use of hybrid MCM design leads to FIR filter designs requiring less number of slices, having less delay, and consuming less power with respect to those including only adders/subtractors.

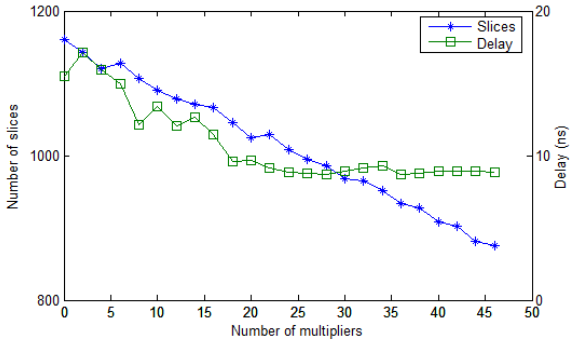

Fig. 5. Tradeoff between number of slices and delay of Filter 1.

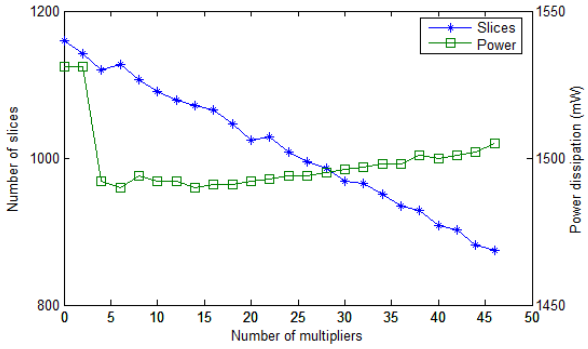

Fig. 6. Tradeoff between number of slices and power dissipation of Filter 1.

VI. ACKNOWLEDGMENT

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

REFERENCES

[1] H. Nguyen and A. Chatterjee, "Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis," *IEEE Tran. on VLSI*, vol. 8, no. 4, pp. 419–424, 2000.

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

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

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

[5] 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.

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

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

[8] M. Kumm and P. Zipf, "Hybrid Multiple Constant Multiplication for FPGAs," in *Proc. of ICECS*, 2012, pp. 556–559.

[9] H.-J. Kang and I.-C. Park, "FIR Filter Synthesis Algorithms for Minimizing the Delay and the Number of Adders," *IEEE TCAS-II*, vol. 48, no. 8, pp. 770–777, 2001.

[10] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined Adder Graph Optimization for High Speed Multiple Constant Multiplication," in *Proc. of ISCAS*, 2012, pp. 49–52.

[11] O. Coudert, "On Solving Covering Problems," in *Proc. of DAC*, 1996, pp. 197–202.

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

[13] SCIP website, http://scip.zib.de.