Optimization of Area and Delay at Gate-Level in Multiple Constant Multiplications

Levent Aksoy
INESC-ID
Lisboa, PORTUGAL
Email: levent@algos.inesc-id.pt

Eduardo Costa
Universidade Católica de Pelotas
Pelotas, BRASIL
Email: ecosta@ucpel.tche.br

Paulo Flores and José Monteiro
INESC-ID/IST TU Lisbon
Lisboa, PORTUGAL
Email: {pff, jcm}@inesc-id.pt

***Abstract*—Although many efficient high-level algorithms have been proposed for the realization of Multiple Constant Multiplications (MCM) using the fewest number of addition and subtraction operations, they do not consider the low-level implementation issues that directly affect the area, delay, and power dissipation of the MCM design. In this paper, we initially present area efficient addition and subtraction architectures used in the design of the MCM operation. Then, we propose an algorithm that searches an MCM design with the smallest area taking into account the cost of each operation at gate-level. To address the area and delay tradeoff in MCM design, the proposed algorithm is improved to find the smallest area solution under a delay constraint. The experimental results show that the proposed algorithms yield low-complexity and high-speed MCM designs with respect to those obtained by the prominent algorithms designed for the optimization of the number of operations and the optimization of area at gate-level.**

***Keywords*-Multiple constant multiplications; addition and subtraction architectures; gate-level area optimization; delay aware area optimization; graph-based algorithms.**

I. Introduction

The multiplication of a set of constants by a variable, *i.e.*, the Multiple Constant Multiplications (MCM) operation, is a ubiquitous and crucial operation that has significant impact on the design of many Digital Signal Processing (DSP) systems including Finite Impulse Response (FIR) filters, Fast Fourier Transforms (FFT), and Discrete Cosine Transforms (DCT). In hardware, the MCM operation is generally realized in a shift-adds architecture [1] where each constant multiplication is implemented using addition/subtraction and shift operations rather than using a general multiplier due to the following two reasons. First, since the constants to be multiplied by a variable are determined beforehand by the DSP algorithms, the full-flexibility of a multiplier is not required. Second, the multiplication operation in hardware is expensive in terms of area, delay, and power dissipation[1].

The realization of the MCM operation in a shift-adds architecture also allows for possible reductions in the number of addition/subtraction operations, consequently in area and power dissipation of the design, when the common partial products are shared among the constant multiplications. Since shifts can be implemented using only wires in hardware without representing any area cost, the MCM problem is defined as finding the minimum number of addition/subtraction operations that implement the constant multiplications. The MCM problem has been proven to be an NP-complete problem in [2].

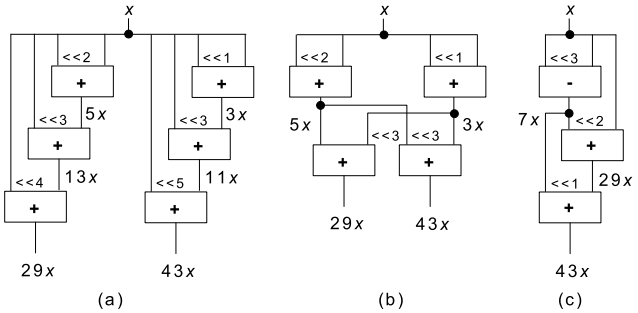

Figure 1. Solutions of the algorithms for the shift-adds implementation of constant multiplications $29x$ and $43x$: (a) the digit-based recoding technique [3]; (b) the CSE method [4]; (c) the graph-based algorithm [5].

A straightforward way for the multiplierless realization of constant multiplications, generally known as the digit-based recoding method [3], is to define the constants in multiplications in binary representation and for each 1 in the binary representation of the constant, is to shift the variable and add up the shifted variables. As a simple example, consider the constant multiplications $29x = (11101)_{bin}x$ and $43x = (101011)_{bin}x$. As shown in Figure 1(a), the constant multiplications can be realized using 6 operations. However, the algorithms that maximize the partial product sharing find the most promising solutions to the MCM problem. They are generally categorized in two classes as the Common Subexpression Elimination (CSE) algorithms [4], [6], [7] and the graph-based (GB) methods [5], [8], [9]. The CSE algorithms, that are also referred to as the pattern search methods, initially define the constants under a particular number representation, *e.g.*, binary, Canonical Signed Digit (CSD), or Minimal Signed Digit (MSD), and then recursively find the "best" subexpression, generally the most common. For our example, suppose that the constants in multiplications are defined in binary. The exact CSE algorithm [4] identifies the most common partial products $3x = (11)_{bin}x$ and $5x = (101)_{bin}x$ in both multiplications and obtains a solution with 4 operations as illustrated in Figure 1(b). On the other hand, the GB algorithms are not limited to any particular number representation and consider a larger number of alternative

[1]Although the relative cost of an adder and a multiplier depends on the adder and multiplier architectures, a $k \times k$ array multiplier has approximately k times the area and twice the latency of the slowest ripple carry adder.

978-0-7695-4171-6/10 $26.00 © 2010 IEEE
DOI 10.1109/DSD.2010.32

implementations of a constant multiplication, yielding better solutions than the CSE algorithms, as shown in [5], [9]. Returning to our example, the exact GB algorithm [5] finds a solution with 3 operations, $7x = x \ll 3 - x$, $29x = 7x \ll 2 + x$, and $43x = 7x \ll 1 + 29x$, as given in Figure 1(c).

Although the minimum number of operations solution in an MCM instance leads to a low-complexity MCM design, it may not yield an MCM design with the minimum area as shown in [10]. The reason is that the algorithms designed for the MCM problem do not take into account the actual area cost of each addition/subtraction operation at gate-level while finding the fewest number of operations solution. Although there exist a large number of algorithms designed for the MCM problem, there are only a few algorithms [10], [11] that target directly on the reduction of area in the MCM design at gate-level. However, the exact CSE algorithm of [10] considers a restricted number of possible implementations of the constant multiplications with respect to a GB algorithm and the GB algorithm of [11] is not equipped with the recently proposed efficient heuristics such as [5], [9].

In this paper, we start by introducing area efficient addition and subtraction architectures used in the MCM design, since the architectures proposed in [10] and [11] do not cover all possible addition and subtraction operations in MCM and do not consider additional simplifications respectively. Then, we propose a GB algorithm, called MINAS (MINimum Area Search algorithm), that considers more possible implementations of the constant multiplications than the exact CSE algorithm [10] and uses a better heuristic than the GB algorithm [11]. Also, based on the MINAS algorithm, we introduce the MINAS-DC algorithm that searches for a solution with the smallest area of the MCM design under a delay constraint that is defined as the maximal number of operations in series, generally known as the number of adder-steps. The experimental results show that MINAS finds MCM designs with significantly smaller area with respect to those obtained by the prominent GB algorithms [5], [8], [9] designed for the MCM problem and better solutions than those of the exact CSE algorithm [10] designed for the gate-level area optimization problem. Also, MINAS-DC obtains low-complexity and high-speed MCM designs with respect to solutions found by an efficient GB algorithm [12] designed for the MCM problem under a delay constraint.

The rest of the paper is organized as follows. Section II gives the background concepts. The addition and subtraction architectures are presented in Section III and the minimum area search algorithms are introduced in Section IV. Section V presents the experimental results and finally, the conclusions are given in Section VI.

II. Background

In this section, first, basic concepts on MCM and problem definitions are introduced. Then, an overview on the previously proposed algorithms is presented.

Since the common input is multiplied by multiple constants in MCM, the implementation of constant multiplications is equal to the implementation of constants. For the sake of clarity, this notation will be used in description of the main concepts and algorithms given in this section.

A. Definitions

In MCM, the main operation, called *A-operation* in [9], 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 follows:

$$w = A(u,v) = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r} \quad (1)$$

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. Thus, in the MCM problem, only positive and odd constants are considered. Observe from Eqn. (1) 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 the *A-operation* performs a subtraction. Note that while finding an *A-operation* for the implementation of a constant, it is necessary to constrain the left shifts, l_1 and l_2, otherwise there exist infinite ways of implementing a constant. In the GB algorithms of [5], [9], 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 under binary representation. Thus, the MCM problem [9] can be defined as follows:

Definition 1: THE MCM PROBLEM. Given the target set, $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$, composed of the positive and odd unrepeated target constants to be implemented, find the smallest ready set $R = \{r_0, r_1, \ldots, r_m\}$, with $T \subset R$, such that $r_0 = 1$ 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 [9].

In the MCM operation, the delay is generally defined as the maximum number of operations in series that generate the constant multiplications [13], although it depends on several implementation issues, such as circuit technology, placement, and routing. For a single constant t, its minimum

number of adder-steps implementation has $log_2 S(t)$ adder-steps, where $S(t)$ denotes the number of nonzero digits of the constant t under the CSD representation[2]. Hence, in the implementation of the target set $T = \{t_1, \ldots, t_n\}$, the minimum number of adder-steps [13] is computed as:

$$min\_delay = \max_{t_i}\{\lceil log_2 S(t_i)\rceil\}, \quad 1 \leq i \leq n \tag{2}$$

Thus, the MCM problem under a delay constraint can be defined as follows:

Definition 2: THE MCM PROBLEM UNDER A DELAY CONSTRAINT. Given the target set, $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$, composed of the positive and odd unrepeated target constants to be implemented and a delay constraint, dc with $dc \geq min\_delay$, find the smallest ready set $R = \{r_0, r_1, \ldots, r_m\}$ such that under the satisfied constraints on the ready set given in the MCM problem definition, the set of *A-operations* yields an MCM design without exceeding dc.

Although the cost of each addition and subtraction operation in hardware is assumed to be equal in the MCM problem, a constant can be implemented with a number of different operations each having a different cost at gate-level. The area of an operation [10], [11] depends on:

- the type of the operation, addition or subtraction,
- the bit-widths of the operation inputs,
- the number of shifts at the input or at the output of an operation, l_1, l_2, or r,
- the shifted input in a subtraction.

The gate-level area optimization problem is defined as:

Definition 3: THE GATE-LEVEL AREA OPTIMIZATION PROBLEM. Given the target set, $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$, composed of the positive and odd unrepeated target constants to be implemented, find the ready set $R = \{r_0, r_1, \ldots, r_m\}$ such that under the satisfied constraints on the ready set given in the MCM problem definition, the set of *A-operations* yields an MCM design with the smallest area at gate-level.

B. Related Work

For the MCM problem, the GB algorithm called RAG-n that includes two parts, optimal and heuristic, was introduced in [8]. In the optimal part, each target constant that can be implemented with available constants using a single operation are synthesized. If there exist unimplemented elements left in the target set, in its iterative heuristic part, RAG-n chooses a single unimplemented target constant with the smallest single coefficient cost and synthesizes it with a single operation including one(two) intermediate constant(s) that has(have) the smallest value among the possible constants. However, the intermediate constants chosen for the implementation of a single target constant in a previous iteration may not be shared for the implementation of not-yet synthesized target constants in later iterations, thus yielding a local minimum solution. Hence, the GB algorithm called Hcub [9], that has the same optimal part as RAG-n, in each iteration of its heuristic part, chooses a target constant to be implemented and synthesizes it by selecting an intermediate constant that has the best cumulative benefit over the not-yet synthesized target constants. On the other hand, the exact GB algorithms that search the minimum solution in breadth-first and depth-first manners were introduced in [5].

For the MCM problem under a delay constraint, the GB heuristic [13] realizes one constant at a time controlling the delay of design. The GB heuristic [15] initially finds a solution including more number of operations but, with a small number of adder-steps, and then reduces the number of operations without increasing the delay in an iterative loop.

For the gate-level area optimization problem, the exact CSE algorithm [10] formulates the problem as a 0-1 Integer Linear Programming (ILP) problem and finds the minimum area solution of the MCM operation when the possible implementations of constants are extracted from their particular number representations. The GB algorithm [11], that is based on RAG-n [8], initially takes an unimplemented constant that requires the smallest number of full adders (FAs), and then synthesizes it using a single operation including one or two intermediate constants that lead to the smallest number of FAs overhead.

III. ADDITION AND SUBTRACTION ARCHITECTURES

This section presents architectures for all possible addition and subtraction operations encountered in the MCM design and gives the cost of each operation in terms of the number of FAs, half adders (HAs), and additional logic gates. The ripple carry adder architecture is assumed for the realization of operations due to its area efficiency.

Note that the number of bits at the output of an operation implementing the constant multiplication tx is $\lceil log_2 t\rceil + N$, where N is the bit-width of the variable x. Hence, the area cost of an operation also depends on the bit-width of the input that the constants are multiplied with and the type of numbers considered, *i.e.*, unsigned or signed, since these lead to different implementations due to the sign extension. The parameters that are used to compute the gate-level area cost of an addition/subtraction operation which realizes a constant multiplication by a variable are given as follows:

l_1, l_2, or r : the number of shifts,
n_u : the bit-width of input u,
n_v : the bit-width of input v,
n_m : $min(n_u + l_1, n_v + l_2)$,
n_M : $max(n_u + l_1, n_v + l_2)$.

The costs of addition and subtraction operations are given in Table I and are explained in the following two sections. In this table, HA$'$ denotes a different type of HA block. It is the special implementation of an FA block when one of the inputs is 1, as opposed to an HA block, that is another

[2]The CSD representation of a constant includes the minimum number of nonzero digits [14].

Table I
IMPLEMENTATION COST OF ADDITION AND SUBTRACTION OPERATIONS.

Operation	$u+2^{l_2}v$		$(u+v)2^{-r}$		$2^{l_1}u-v$		$u-2^{l_2}v$		$(u-v)2^{-r}$	
Number	Unsigned	Signed	Unsigned	Signed	Unsigned	Signed	Unsigned	Signed	Unsigned	Signed
#FA	n_m-l_2-1	n_M-l_2-1	n_m-r	n_M-r	$max(l_1,n_v)-l_1$	n_u	n_v-1	n_u-l_2-1	n_v-r-1	n_u-r-1
#HA	n_M-n_m+1	1	n_M-n_m	0	$min(l_1,n_v)-1$	l_1-1	0	0	0	0
#HA′	0	0	0	0	$n_u+min(l_1-n_v,0)$	0	$n_u-n_v-l_2+1$	1	n_u-n_v+1	1
#inv	0	0	0	0	$max(l_1,n_v)$	n_v	n_v	n_v	n_v-r	n_v-r

special implementation of FA when one of the inputs is 0. In an FA block, if the input v_i is 1, the addition (*sum*) and carry output (*cout*) are the functions of the input u_i and the carry input (*cin*) given as $sum = \overline{cin \oplus u_i}$ and $cout = cin + u_i$.

A. Addition Operations

Addition operation $u+2^{l_2}v$: Observe from examples on the unsigned input model given in Figure 2(a)-(b) that larger number of shifts at the input achieves smaller area, since shifts are implemented with only wires. Note that the cost values given in Table I for the unsigned input model are valid, if the number of shifts of the operand v is less than the number of bits of the operand u, *i.e.*, $l_2 < n_u$. Otherwise, no hardware is needed for this operation as illustrated in Figure 2(b). In the signed input case, this situation never occurs, due to the sign extension of the operand u.

Addition operation $(u+v)2^{-r}$: The result of a constant multiplication to be computed by this operation is obtained after the output is shifted right by r times. Hence, there is no need to compute the first r digits of the output. However, observe from the example on the unsigned input model presented in Figure 2(c) that to determine the carry bit for the first FA, an OR gate whose inputs are the r^{th} digits of operands u and v is required, although it is not listed in Table I.

B. Subtraction Operations

The subtraction operation is implemented using 2's complement, *i.e.*, $u+\overline{v}+1$. So, the costs of inverter (inv) gates are included into the costs of the subtraction operations.

Subtraction operation $2^{l_1}u-v$: Observe from the example on unsigned input model given in Figure 2(d) that while the first bit of the result is simply the first bit of the operand v, the inputs of the first HA block are the inverted first and second bits of the operand v. Note that the values given in Table I also consider the case where the digits of operand u and v do not overlap, $l_1 \geq n_v$, that is not considered in [10].

Subtraction operation $u-2^{l_2}v$: Observe from the example on unsigned input model presented in Figure 2(e) that the shifts can be fully utilized by starting the addition with the first digit of the inverted operand v resulting in a smaller area. Also, note that its cost is computed without HA blocks as opposed to the subtraction operation $2^{l_1}u-v$.

Subtraction operation $(u-v)2^{-r}$: Observe from the example on unsigned input model given in Figure 2(f) that the operation output can be obtained by starting the addition from the $(r+1)^{\text{th}}$ digit of the operands u and v, since the operation output is shifted right by r times.

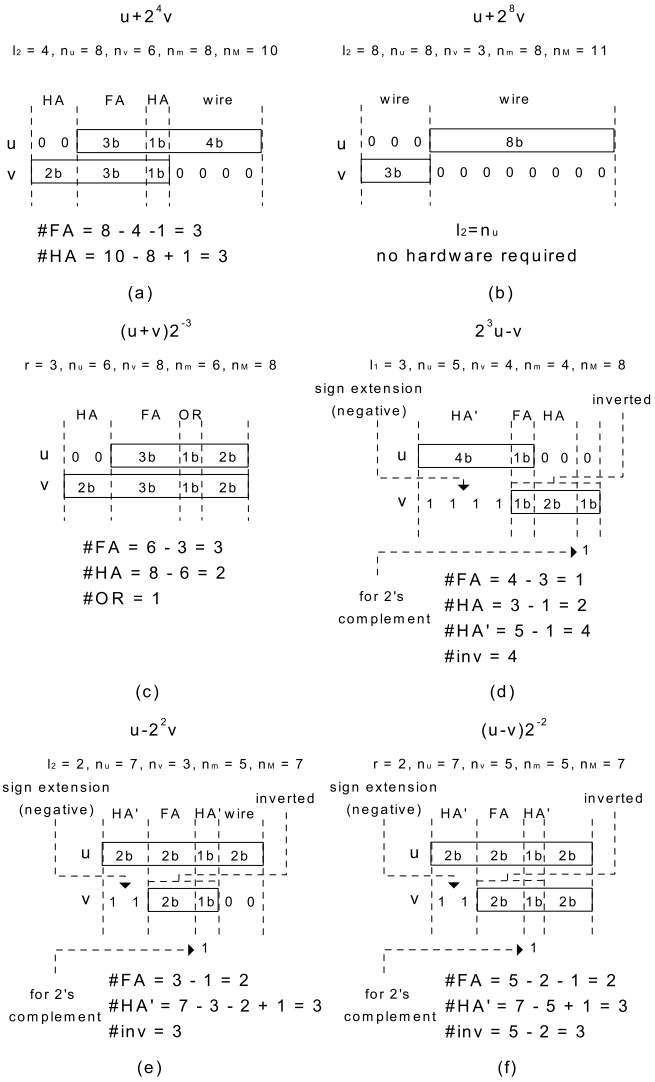

Figure 2. Examples on addition and subtraction operations under unsigned input: a-b) $u+2^{l_2}v$; c) $(u+v)2^{-r}$; d) $2^{l_1}u-v$; e) $u-2^{l_2}v$; f) $(u-v)2^{-r}$.

IV. THE GRAPH-BASED ALGORITHMS

In this section, initially we present the MINAS algorithm designed for the gate-level area optimization problem, and then show how it can be extended to find a solution with the smallest area under a delay constraint.

A. Optimization for Area

As done in algorithms designed for the MCM problem, in MINAS, we find the fewest number of intermediate constants such that all the target and intermediate constants

are synthesized using a single operation. However, while selecting an intermediate constant for the implementation of the not-yet synthesized target constants in each iteration, we favor the one among the possible intermediate constants that can be synthesized using the least hardware and enables to implement the not-yet synthesized target constants in a smaller area with the available constants. After the set of target and intermediate constants that realizes the MCM operation is found, each constant is synthesized using an *A-operation* that yields the minimum area in the MCM design.

In the preprocessing phase of the algorithm, the target constants to be implemented are made positive and odd, are added to the target set, T, without repetition, and the maximum bit-width of the target constants, bw, is determined. The main part of the algorithm is given in Algorithm 1.

In MINAS, the ready set, $R = \{1\}$, is formed initially and then the target constants that can be implemented with the elements of the ready set using a single operation are found and moved to the ready set iteratively using the *Synthesize* function. If there exist unimplemented constant(s) in the target set, then in each iteration of its heuristic part (line 3), an intermediate constant is added to the ready set until there is no element left in the target set. The MINAS algorithm considers the positive and odd constants that are not included in the current ready and target sets (lines 4-5) and that can be implemented with the elements of the current ready set using a single operation (lines 6-7) as possible intermediate constants. On line 6, the *ComputeCost* function searches all *A-operations* that compute the constant with the elements of the current ready set. If the implementations of the constant are found, then it determines the cost of each operation as described in Section III and returns its minimum implementation cost among possible operations. Otherwise, it returns 0 value indicating that the constant cannot be synthesized using an operation with the elements of the current ready set. After the possible intermediate constant is found, it is included into the working ready set, A, and its implications on the current target set are found by the *ComputeTCost* function. In this function, similar to the *ComputeCost*, the minimum implementation costs of the target constants that can be synthesized with the elements of the working ready set A are determined. For each target constant, t_k, that cannot be implemented with the elements of A, its cost value is determined as its maximum implementation cost, $maxcost(t_k)$, computed as if all its digits are implemented using FAs. Then, the cost of the intermediate constant is determined as its minimum implementation cost plus the implementation costs of the not-yet synthesized target constants. After the cost value of each possible intermediate constant is found, the intermediate constant with the minimum cost is added to the current ready set and its implications on the current target set are found using the *Synthesize* function. Also, we note that while searching for the "best" intermediate constant in each iteration, MINAS favors the one that has

Algorithm 1 The MINAS algorithm.

MINAS(T)
1: $R \leftarrow \{1\}$
2: (R, T) = Synthesize(R, T)
3: **while** $T \neq \emptyset$ **do**
4: **for** $j = 1$ to $2^{bw+1} - 1$ step 2 **do**
5: **if** $j \notin R$ and $j \notin T$ **then**
6: $impcost_j$ = ComputeCost($\{j\}$, R)
7: **if** $impcost_j \neq 0$ **then**
8: $A \leftarrow R \cup \{j\}$
9: $impcost_T$ = ComputeTCost(T, A)
10: $iccost_j = impcost_j + impcost_T$
11: Find the intermediate constant, ic, with the minimum $iccost_j$ cost among all possible constants, j
12: $R \leftarrow R \cup \{ic\}$
13: (R, T) = Synthesize(R, T)
14: D = SynthesizeMinArea(R)
15: **return** D

Synthesize(R, T)
1: **repeat**
2: $isadded = 0$
3: **for** each $t_k \in T$ **do**
4: **if** t_k can be implemented using a single *A-operation* whose inputs are the elements of R **then**
5: $isadded = 1$, $R \leftarrow R \cup \{t_k\}$, $T \leftarrow T \setminus \{t_k\}$
6: **until** $isadded = 0$
7: **return** (R, T)

ComputeCost({c}, C)
1: $cost_c = 0$
2: **for** all operations $c = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r}$, where $u, v \in C$ **do**
3: Determine the cost of each operation, compute the minimum implementation cost of constant c, assign it to $cost_c$
4: **return** $cost_c$

ComputeTCost(B, C)
1: $cost_B = 0$
2: **repeat**
3: $isadded = 0$
4: **for** each $b_k \in B$ **do**
5: $cost_{b_k}$ = ComputeCost($\{b_k\}$, C)
6: **if** $cost_{b_k} \neq 0$ **then**
7: $isadded = 1$, $C \leftarrow C \cup \{b_k\}$, $B \leftarrow B \setminus \{b_k\}$
8: $cost_B = cost_B + cost_{b_k}$
9: **until** $isadded = 0$
10: **for** each $b_k \in B$ **do**
11: $cost_B = cost_B + maxcost(b_k)$
12: **return** $cost_B$

SynthesizeMinArea(R)
1: Find all possible implementations of target and intermediate constants using the *GenerateImp(R)* function
2: Formalize the problem as a 0-1 ILP problem
3: Find D as a set of *A-operations* that yields minimum area
4: **return** D

GenerateImp(R)
1: $A \leftarrow \{1\}$, $R \leftarrow R \setminus \{1\}$
2: **repeat**
3: **for** each $r_k \in R$ **do**
4: (B, C) = Synthesize(A, $\{r_k\}$)
5: **if** $C = \emptyset$ **then**
6: Find all operations, $r_k = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r}$, where $u, v \in A$ and determine their implementation costs
7: $A \leftarrow A \cup \{r_k\}$, $R \leftarrow R \setminus \{r_k\}$
8: **until** $R = \emptyset$

the smallest cost and synthesizes all the not-yet synthesized target constants with the available constants among the ones that may have the minimum cost but cannot synthesize all the not-yet synthesized target constants.

When there are no elements left in the target set, the *SynthesizeMinArea* function is applied to find the set of *A-operations* that yields a solution with the smallest area on the final ready set. In this function, we formalize this problem as a 0-1 ILP problem, similar to the 0-1 ILP formalization described in [10]. In this case, the possible implementations of the constants are found by the *GenerateImp* function. Note that when no intermediate constant is required to implement the target constants, the minimum area solution can be simply obtained by choosing the operation with the minimum area cost among possible operations to implement each target constant. However, for the final ready set including intermediate constants, there can be more than one possible implementation of a constant that are not considered entirely while adding an intermediate constant to the ready set in each iteration. Also, the final ready set may include redundant intermediate constants, since the recently added intermediate constant may make the previously added intermediate constants redundant.

B. Optimization for Area under a Delay Constraint

The MINAS algorithm can be easily improved to deal with the area and delay tradeoff so that an increment in area can be compensated with a decrement in delay and vice versa. In this modified algorithm, called MINAS-DC, the preprocessing phase is the same as that of MINAS. Additionally, MINAS-DC takes as an input the user-specified delay constraint dc, given in terms of the number of adder-steps. It follows the similar procedure described in Algorithm 1. However, in its *Synthesize* function, similar to the one given in Algorithm 1, while finding a possible implementation of a constant using an *A-operation*, it considers the operation whose implementation does not exceed dc. Also, in finding the implementation cost of a constant, it considers the possible implementations that do not violate dc. After the set of target and intermediate constants that can generate MCM using a single operation is obtained, finding a solution with the smallest area is again formalized as a 0-1 ILP problem, but in this case, the possible implementations of a constant are determined as operations that also respect dc.

V. Experimental Results

In this section, we compare MINAS and MINAS-DC on FIR filter and randomly generated instances with the previously proposed algorithms designed for the MCM problem, the MCM problem under a delay constraint, and the gate-level area optimization problem. The low-level results of an MCM operation are obtained in two phases. First, the solution of an algorithm on an MCM instance (a set of *A-operations* that generates MCM) is found. Second, the MCM operation is designed at gate-level using the Sequential Interactive System (SIS) tool. In this tool, the addition and subtraction operations that implement MCM are initially described in a synthesizable format, and then the MCM design is mapped with the logic gates given in a library. The UMC Logic $0.18\mu m$ Generic II library was used as the design library.

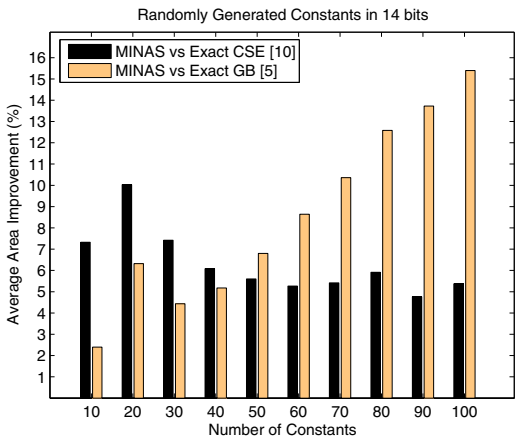

Figure 3. Comparison of MINAS with the exact algorithms designed for the MCM and gate-level area optimization problems.

As the first experiment set, we used uniformly distributed randomly generated instances where constants were defined under 14 bit-width. The number of constants ranges between 10 and 100, and we generated 30 instances for each of them. Figure 3 presents the average area improvements obtained by the MINAS algorithm over the exact GB algorithm [5] designed for the MCM problem and the exact CSE algorithm [10] designed for the gate-level area optimization problem. In the exact CSE algorithm [10], the constants were defined under MSD representation. In this experiment, the unsigned input model was used and the bit-width of the input that the constants are multiplied with was taken as 16.

Observe from Figure 3 that the solutions of MINAS lead to low-complexity MCM designs with respect to solutions obtained by the algorithms of [5] and [10]. We note that MINAS obtains the maximum average area improvement, 15.4%, over the exact GB algorithm [5] on instances including 100 constants. However, observe that the average area improvement values of MINAS over the exact GB algorithm decrease as the number of constants decreases, since the number of possible implementations of a constant considered in MINAS also decreases. On the other hand, MINAS obtains the maximum average area improvement, 10%, over the exact CSE algorithm [10] on instances including 20 constants. However, observe that the average area improvement values of MINAS over the exact CSE algorithm decrease as the number of constants increases. This is because the sharing of intermediate constants increases as the number of constants increases in the exact CSE algorithm.

As the second experiment set, we used the FIR filters given in [5]. Table II presents the high-level results of algorithms designed for the MCM problem and the gate-

level area optimization problem. Again, the results of the exact CSE algorithm [10] were obtained when filter coefficients were defined under MSD. In this table, *adder* denotes the number of operations and *step* indicates the number of adder-steps. Also, *CPU* stands for the required CPU time in seconds for an algorithm to find its solution on a PC with Intel Xeon at 2.33GHz and 4GB of memory under Linux. For the exact CSE algorithm, *CPU* represents the CPU time of the ILP-based pseudo-Boolean solver called glpPB.

Observe from Table II that while Hcub [9] obtains solutions close to the minimum as given by the exact GB algorithm of [5], RAG-n [8], whose heuristic is also used in the GB algorithm of [11] designed for the optimization of area at gate-level, finds results that are far away from the minimum. On the other hand, MINAS finds competitive results with Hcub and better results than RAG-n in terms of the number of operations. The solutions of the exact CSE algorithm [10] include larger number of operations than those of the GB algorithms designed for the MCM problem, since it considers less alternative implementations of the constant multiplications and it is primarily designed for the optimization of area at gate-level. However, its solutions have smaller number of adder-steps compared with those of the GB algorithms. Also, the solutions of the exact CSE and MINAS algorithms are obtained using higher computational effort than the GB heuristics designed for the MCM problem. In the exact CSE algorithm, to obtain the smallest area solution, a 0-1 ILP problem has to be solved, that heavily depends on the MCM instance and the performance of the 0-1 ILP solver. In MINAS, all implementations of each target and possible intermediate constant have to be considered to determine the smallest area cost of the constant in each iteration, as opposed to the algorithms designed for the MCM problem, where finding a single operation for the synthesis of a constant is sufficient.

Table III presents the low-level results of the MCM designs that are obtained using the solutions of algorithms given in Table II. In this table, *area* (μm^2) and *delay* (*ns*) denote the area and maximum delay of the MCM design respectively. Again, the unsigned input model was used and the bit-width of the filter input was taken as 16.

Observe from Tables II and III that in the algorithms designed for the MCM problem, the reduction of the number of operations directly takes effect on the area of the MCM design. This situation can be clearly observed when the results of RAG-n are compared with those of Hcub and the exact GB algorithm in Tables II and III on overall instances. On the other hand, although the exact CSE algorithm [10] obtains solutions including greater number of operations than those of the algorithms designed for the MCM problem, it finds better solutions in terms of area on Filters 3, 5, 7, and 8 than all these algorithms. Also, observe that MINAS obtains better area solutions than all algorithms given in Table III on each instance. This experiment indicates that a solution with the minimum number of operations may not lead to an MCM design with the minimum area and in order to reduce the area of design, the low-level implementation issues should be taken into account.

However, observe from Table III that the solutions of MINAS lead to slower MCM designs with respect to those of the exact CSE algorithm [10], since it increases the depth of the MCM design while finding a solution with the smallest area as can be observed in Table II. Table IV presents the high and low level results of the Hcub-DC [12] and MINAS-DC algorithms that can find a solution under a delay constraint. Note that Hcub-DC is the modified version of Hcub that finds the fewest number of operations solution under a delay constraint. In this experiment, the delay constraint *dc* was set to 4 and 3 in order to compare the solutions of these algorithms with those of the exact CSE algorithm [10]. Note that the *min_delay* value of each instance is 3, as computed in Eqn. (2).

Observe from Tables III and IV that the delay of MCM designs obtained by MINAS-DC is generally reduced as the delay constraint is decreased with respect to the MCM designs found by MINAS. It is also interesting to note that MINAS-DC may obtain better solutions in terms of area with respect to the solutions obtained by MINAS, *e.g.*, Filter 5. Note that MINAS-DC obtains the best area and delay values on Filters 2, 4, 6, and 8 than all algorithms given in Table III, except MINAS. Also, under its solutions with the minimum delay, it finds better area designs than the exact CSE algorithm on all instances, except Filters 3 and 7. Observe that MINAS-DC obtains competitive delay results with those of the exact CSE algorithm given in Table III and yields low-complexity and high-speed MCM designs with respect to designs obtained by Hcub-DC.

VI. Conclusions

In this paper, we introduced a new GB algorithm (MINAS) that searches the smallest area solution in the MCM design with the guide of implementation costs of the addition and subtraction operations formulated in terms of gate-level metrics. Also, to deal with the area and delay tradeoff, we proposed an algorithm (MINAS-DC) that can handle the user-specified delay constraint. The experimental results clearly indicate that MINAS achieves significant area improvements over the efficient GB algorithms designed for the MCM problem and the exact CSE algorithm designed for the gate-level area optimization problem. Also, MINAS-DC finds low-complexity and high-speed MCM designs with respect to those obtained by the prominent GB algorithm proposed for the MCM problem under a delay constraint.

VII. Acknowledgment

This work was partially supported by the *Portuguese Foundation for Science and for Technology* (FCT) under the research project *Architectural Optimization of DSP*

Table II
SUMMARY OF HIGH-LEVEL RESULTS OF ALGORITHMS ON FIR FILTER INSTANCES.

Objective	Optimization of the Number of Operations									Optimization of Area at Gate-Level					
Filter	RAG-n [8]			Hcub [9]			Exact GB [5]			Exact CSE [10]			MINAS		
	adder	step	CPU	adder	step	CPU	adder	step	CPU	adder	step	CPU	adder	step	CPU
1	24	10	5.2	23	7	0.1	22	10	21.1	28	4	314.5	23	8	57.7
2	27	6	5.7	24	6	0.1	23	7	1630.7	31	3	277.4	23	5	125.8
3	24	9	5.6	19	7	0.1	17	11	1761.0	23	3	50.0	18	8	99.4
4	23	5	5.4	18	7	0.1	17	8	67.4	22	4	38.5	18	7	92.3
5	44	9	5.5	42	8	0.1	41	10	36.7	56	4	1686.0	41	11	396.8
6	36	10	5.5	32	10	0.1	31	11	2169.7	42	4	427.6	33	8	491.3
7	28	7	5.9	24	7	0.1	23	10	499.4	30	4	40.0	25	6	367.5
8	40	5	4.8	38	5	0.1	37	6	2.2	45	4	156.3	37	9	18.3
Total	246	61	43.5	220	57	0.8	211	73	6188.1	277	30	2990.2	218	62	1649.1

Table III
SUMMARY OF LOW-LEVEL RESULTS OF ALGORITHMS ON FIR FILTER INSTANCES.

Objective	Optimization of the Number of Operations						Optimization of Area at Gate-Level			
Filter	RAG-n [8]		Hcub [9]		Exact GB [5]		Exact CSE [10]		MINAS	
	area	delay	area	delay	area	delay	area	delay	area	delay
1	29644	63.6	29774	59.0	29392	60.2	30420	44.8	28142	57.4
2	34734	57.6	31500	58.6	31246	63.0	32520	44.3	29538	57.8
3	26574	60.3	23252	59.9	23316	61.2	23224	43.0	22110	60.7
4	26870	39.8	23058	54.9	22542	60.8	22626	43.8	21626	52.8
5	62464	65.9	59426	68.3	59028	74.8	57172	51.7	55360	73.8
6	46416	64.4	43350	63.5	43214	66.5	43822	54.9	40186	63.0
7	33390	53.9	31908	61.5	32096	65.9	30668	46.3	29898	50.7
8	45668	60.2	45386	56.7	45796	58.3	45104	49.6	42900	56.7
Total	305760	465.6	287654	482.4	286630	510.7	285556	378.4	269760	472.7

Table IV
SUMMARY OF HIGH AND LOW LEVEL RESULTS OF HCUB-DC AND MINAS-DC ALGORITHMS WHEN *dc* WAS SET TO 4 AND 3.

Filter	step	Hcub-DC [12]			MINAS-DC		
		adder	area	delay	adder	area	delay
1	4	25	32058	58.0	24	29770	51.4
	3	25	30256	49.5	26	28294	45.6
2	4	26	33780	54.1	25	29226	50.9
	3	27	33668	54.8	28	30696	43.6
3	4	19	23932	57.2	19	21654	56.3
	3	23	27706	53.2	22	23704	48.6
4	4	21	27286	54.4	20	21642	35.8
	3	22	27828	51.8	22	22884	45.0
5	4	49	64720	59.9	48	54140	54.1
	3	55	70682	56.4	51	53270	54.5
6	4	34	45994	57.6	35	39452	50.4
	3	40	54032	55.8	38	41472	46.6
7	4	27	35728	52.3	26	30628	48.5
	3	30	39148	54.9	30	31548	46.1
8	4	39	47064	57.3	39	43188	53.9
	3	44	52564	57.4	43	43236	49.4

Systems with Multiple Constants Multiplications (Multicon) PTDC/EIA-EIA/103532/2008 and by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds.

REFERENCES

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

[2] 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.

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

[4] 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.

[5] 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.

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

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

[8] 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.

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

[10] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Optimization of Area in Digital FIR Filters using Gate-Level Metrics," in *Proc. DAC*, 2007, pp. 420–423.

[11] K. Johansson, O. Gustafsson, and L. Wanhammar, "A Detailed Complexity Model for Multiple Constant Multiplication and an Algorithm to Minimize the Complexity," in *Proc. ECCTD*, 2005, pp. 465–468.

[12] Spiral website, http://www.spiral.net.

[13] 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.

[14] A. Avizienis, "Signed-digit Number Representation for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, pp. 389–400, 1961.

[15] A. Dempster, S. Demirsoy, and I. Kale, "Designing Multiplier Blocks with Low Logic Depth," in *Proc. ISCAS*, 2002, pp. 773–776.