Microprocessors and Microsystems 35 (2011) 729–741

Contents lists available at SciVerse ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Finding the optimal tradeoff between area and delay in multiple constant multiplications

Levent Aksoy [a,*], Eduardo Costa [b], Paulo Flores [c], José Monteiro [c]

[a] INESC-ID, Lisboa, Portugal
[b] Universidade Católica de Pelotas, Pelotas, Brazil
[c] INESC-ID/IST, TU Lisbon, Lisboa, Portugal

ARTICLE INFO

Article history:
Available online 22 August 2011

Keywords:
Multiple constant multiplications
Addition and subtraction architectures
Gate-level area optimization
Delay aware area optimization
0–1 Integer linear programming
Graph-based algorithms

ABSTRACT

Over the years many efficient algorithms for the multiplierless design of multiple constant multiplications (MCMs) have been introduced. These algorithms primarily focus on finding the fewest number of addition/subtraction operations that generate the MCM. Although the complexity of an MCM design is decreased by reducing the number of operations, their solutions may not lead to an MCM design with optimal area at gate-level since they do not consider the implementation costs of the operations in hardware. This article introduces two approximate algorithms that aim to optimize the area of the MCM operation by taking into account the gate-level implementation of each addition and subtraction operation which realizes a constant multiplication. To find the optimal tradeoff between area and delay, the proposed algorithms are further extended to find an MCM design with optimal area under a delay constraint. Experimental results clearly indicate that the solutions of the proposed algorithms lead to significantly better MCM designs at gate-level when compared to those obtained by the solutions of algorithms designed for the optimization of the number of operations.

1. Introduction

The multiple constant multiplications (MCMs) operation, that realizes the multiplication of a set of known constants by a variable, is a central operation and performance bottleneck in many Digital Signal Processing (DSP) applications such as, digital Finite Impulse Response (FIR) filters, linear DSP transforms, and error correcting codes. Since the multiplication operation is expensive in terms of area, delay, and power dissipation in hardware and the constants to be multiplied with the same variable are known beforehand, the full-flexibility of a multiplier is not necessary in the design of the MCM operation. Hence, constant multiplications are generally realized using addition, subtraction, and shift operations [1].

For the shift-adds implementation of constant multiplications, a straightforward method, generally known as the digit-based recoding [2], initially defines the constants in multiplications in binary. Then, for each 1 in the binary 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 follows:

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

and require six addition operations as illustrated in Fig. 1a.

However, the sharing of common partial products in the shift-adds architecture allows for great reductions in the number of operations and consequently, in area and power dissipation of the MCM design. Hence, the MCM problem is defined as finding the minimum number of addition/subtraction operations that implement the constant multiplications, since shifts can be realized using only wires in hardware without representing any area cost. Note that the MCM problem is an NP-complete problem [3].

The last two decades have seen tremendous progress in the design of efficient algorithms proposed for the MCM problem. These algorithms can be categorized in two classes: Common Subexpression Elimination (CSE) algorithms [4–6] and graph-based (GB) methods [7–9]. Although both CSE and GB algorithms aim to maximize the sharing of partial products, they differ in the search space that they explore. The CSE algorithms initially define the constants under a number representation, namely binary, Canonical Signed Digit (CSD), or Minimal Signed Digit (MSD). Then, all possible subexpressions are extracted from the representations of the constants and the "best" subexpression, generally the most common,

* Corresponding author. Tel.: +351 21 3100260; fax: +351 21 3145843.
E-mail addresses: levent@algos.inesc-id.pt (L. Aksoy), ecosta@ucpel.tche.br (E. Costa), pff@inesc-id.pt (P. Flores), jcm@inesc-id.pt (J. Monteiro).

0141-9331/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2011.08.009

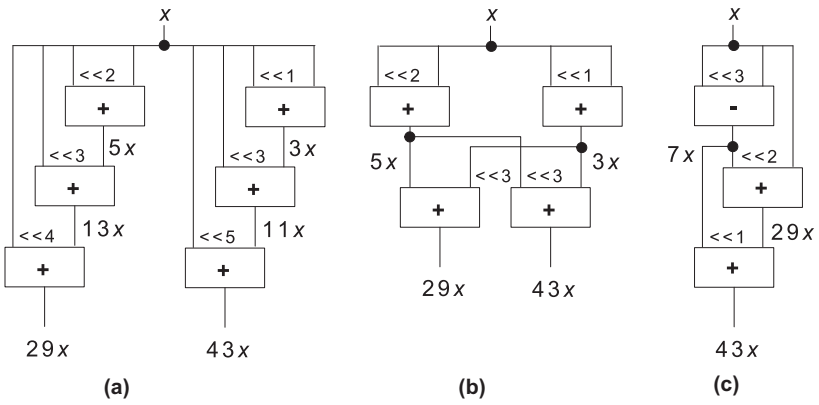

Fig. 1. Shift-adds implementations of $29x$ and $43x$: (a) without partial product sharing [2]; with partial product sharing: (b) the algorithm of [6]; (c) the algorithm of [9].

is chosen to be shared among the constant multiplications. On the other hand, the GB algorithms are not limited to any particular number representation and consider a larger number of alternative implementations of a constant multiplication, yielding better solutions than the CSE algorithms as shown in [8,9].

Returning to our example in Fig. 1, the exact CSE algorithm of [6] gives 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, as illustrated in Fig. 1b. On the other hand, the exact GB algorithm [9] finds the minimum number of operations solution with 3 operations by sharing the common partial product $7x$ in both multiplications, as shown in Fig. 1c. 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 [6].

However, the minimum number of operations solution in an MCM instance does not always yield an MCM design with optimal area as shown in [10,11]. This is simply because 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 searching for a solution with the fewest number of operations. Although there exist a large number of algorithms designed for the MCM problem, there are only a few algorithms [10–12] that target directly the reduction of area in the MCM design at gate-level.

Moreover, performance is also a crucial parameter, with circuit area is being in many cases expandable in order to achieve a given performance target. Although the delay parameter is dependent on several implementation issues, such as circuit technology, placement, and routing, the delay of the MCM operation is generally considered as the number of adder-steps, which denotes the maximal number of adders/subtracters in series to produce any constant multiplication [13]. The algorithms of [6,13,14] find the fewest number of operations that implement the MCM such that the delay constraint is never violated. However, the tradeoff between area and delay in the MCM design can be explored more extensively when high-level algorithms target the optimization of gate-level area under a delay constraint [11].

In this article, we start by presenting area efficient addition and subtraction architectures used in the MCM design and formalize the implementation cost of addition and subtraction operations at gate-level. It is followed by the introduction of the HCUB + ILP algorithm that combines two efficient techniques to optimize the area of the MCM design. In HCUB + ILP, we iteratively obtain an approximate solution with the fewest number of operations using the Hcub heuristic [8] designed for the MCM problem. Then, we find a set of operations that lead to an MCM design with optimal area by formalizing the gate-level area optimization problem as a 0–1 integer linear programming (ILP) problem based on the solution of Hcub. We also describe the HCUB-DC + ILP algorithm, that iteratively finds a solution with the fewest number of operations under a delay constraint using Hcub and optimizes the area of the MCM design on this solution without violating the delay constraint. Furthermore, we present the MINimum Area Search (MINAS) algorithm [11] that searches the "best" partial products, which lead to an MCM design with optimal area at gate-level, while synthesizing the constant multiplications. We also introduce its modified version, MINAS-DC, that searches for an MCM design with optimal area under a delay constraint.

The experimental results indicate that HCUB + ILP and MINAS find MCM designs with significantly smaller area when compared to those obtained by prominent algorithms designed for the MCM problem. Also, the solutions of HCUB-DC + ILP and MINAS-DC lead to high-speed and low-power MCM designs with respect to those implemented by the solutions of HCUB + ILP, MINAS, and algorithms proposed for the MCM problem under a delay constraint. Moreover, HCUB-DC + ILP and MINAS-DC can find the optimal tradeoff between area and delay by changing the delay constraint.

The rest of the article proceeds as follows. Section 2 presents the background concepts and gives the problem definitions. A detailed overview on previously proposed algorithms is given in Section 3 and the addition and subtraction architectures are described in Section 4. The HCUB + ILP and MINAS algorithms are introduced in Sections 5 and 6 respectively. Section 7 presents the experimental results and finally, Section 8 concludes the article.

2. Background

This section gives the basic concepts related with the proposed algorithms and introduces the problem definitions.

2.1. 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 defined as[1]:

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

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

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

[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, $\mathbf{A} \cdot \mathbf{x} \leqslant \mathbf{b} \Longleftrightarrow -\mathbf{A} \cdot \mathbf{x} \geqslant -\mathbf{b}$ and $\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \Longleftrightarrow (\mathbf{A} \cdot \mathbf{x} \geqslant \mathbf{b}) \wedge (\mathbf{A} \cdot \mathbf{x} \leqslant \mathbf{b})$, respectively.

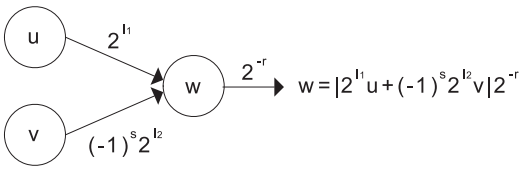

Fig. 2. The graph representation of an A-operation.

2.2. Multiplierless constant multiplications

Since the common input is multiplied by multiple constants in MCM, the implementation of constant multiplications is equal to the implementation of constants. For example, the implementation of $3x$ given as $3x = x \ll 1 + x = (1 \ll 1 + 1)x$ can be rewritten as $3 = 1 \ll 1 + 1$ by eliminating the variable x from both sides. These notations will be used interchangeably in this article.

In multiplierless constant multiplications, the fundamental operation, called *A-operation* in [8], 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:

$$w = A(u, v) = |2^{l_1}u + (-1)^s 2^{l_2} v| 2^{-r} = |(u \ll l_1) + (-1)^s (v \ll l_2)| \gg r \quad (3)$$

where $s \in \{0, 1\}$ is the sign, which determines if an addition or a subtraction operation is to be performed, $l_1, l_2 \geqslant 0$ are integers denoting left shifts of the operands, and $r \geqslant 0$ is an integer indicating a right shift of the result. An *A-operation* can be represented in a graph, where the vertices are labeled with the constants and the edges are labeled with the sign and shifts, as illustrated in Fig. 2.

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. Thus, only positive and odd constants are considered in the MCM problem. Observe from Eq. (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 [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. Thus, the MCM problem [8] 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 $T \subset R$, such that $r_0 = 1$ and for all r_k with $1 \leqslant k \leqslant m$, there exist r_i, r_j with $0 \leqslant i, j < k$ 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 [8].

The minimum adder-step realization of a single target constant t includes $\lceil \log_2 S(t) \rceil$ adder-steps, where $S(t)$ stands for the number of non-zero digits of t in its CSD representation. Note that the CSD representation of a constant includes the minimum number of non-zero digits [15]. Hence, for a target set, $T = \{t_1, \ldots, t_n\}$, the minimum adder-steps of the MCM operation [13] is computed as:

$$min\_delay_{MCM} = \max_{t_i} \{\lceil \log_2 S(t_i) \rceil\}, \quad 1 \leqslant i \leqslant n \quad (4)$$

The optimization of the number of operations problem under a delay constraint can be defined as follows:

Definition 2. THE MCM PROBLEM UNDER A DELAY CONSTRAINT.Given the target set $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$ and the delay constraint dc with $dc \geqslant min\_delay_{MCM}$, find the smallest 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 without exceeding dc.

To illustrate the implementations of an MCM operation with the minimum number of operations and with the minimum number of adder-steps, consider the target set $T = \{5, 11, 171, 215\}$ as an example. The minimum adder-steps of the target constants 5, 11, 171, and 215 are 1, 2, 3, and 2 respectively. Thus, the minimum adder-step realization of the MCM operation includes 3 adder-steps as computed by Eq. (4). The exact GB algorithm of [9] designed for the MCM problem finds a solution with 4 operations and 4 adder-steps without requiring any intermediate constant, as illustrated in Fig. 3a. On the other hand, the Hcub algorithm [14] gives a solution with 5 operations when the delay constraint was set to the minimum delay constraint 3, requiring the intermediate constant 127, as shown in Fig. 3b.

Although the cost of each addition and subtraction operation in hardware is assumed to be equal in the MCM problem, as can be easily observed from Figs. 1 and 3, a constant multiplication can be implemented with a number of different operations each having a different cost at gate-level. The area of an operation [10–12] depends on the type of the operation (addition or subtraction), the bit-width of an input of the operation, the number of shifts at the input or at the output of the operation (l_1, l_2, or r), the shifted input in a subtraction, and the type of the input variable (signed or unsigned). Hence, the gate-level area optimization problem is given 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 same conditions on the ready set given in Definition 1, the set of *A-operations* yields an MCM design with optimal area at gate-level.

Similar to the MCM problem under a delay constraint, the gate-level area optimization problem under a delay constraint can be given as:

Definition 4. THE GATE-LEVEL AREA OPTIMIZATION PROBLEM UNDER A DELAY CONSTRAINT.Given the target set $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$ and the delay constraint dc with $dc \geqslant min\_delay_{MCM}$, 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 with optimal area at gate-level and without exceeding dc.

Although Definitions 3 and 4 are concerned with the area and delay of the MCM design respectively, power dissipation is also highly related with the gate-level area and adder-step of each adder/subtracter as indicated in the switching activity model [16] and the Glitch Path Score (GPS) power dissipation estimation model [17]. This is simply because larger number of logic gates produce more transitions and the transitions generated at the output of an operation produce more glitching along the re-convergent paths. Hence, the solutions of the algorithms that take into account these two parameters will also lead to low-power MCM designs as shown in Section 7.

3. Related work

For the MCM problem, the 0–1 ILP formalization of the common subexpression sharing was introduced in [18,19]. In these algorithms, initially, the target constants are defined under a number

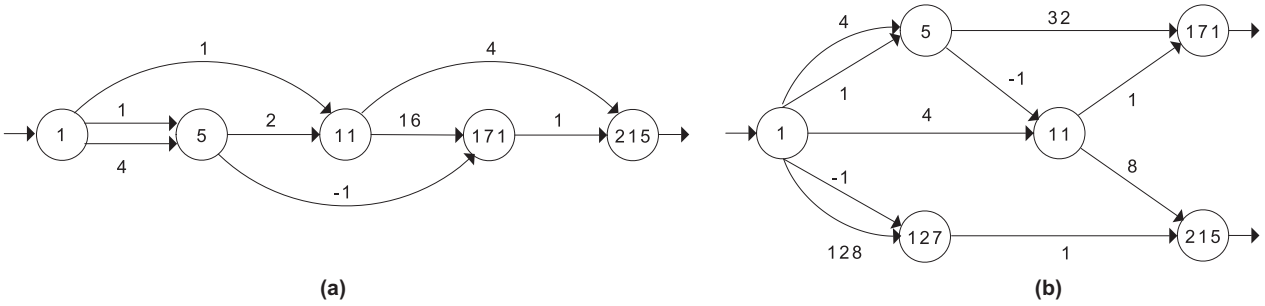

Fig. 3. Implementations of the target set {5,11,171,215}: (a) with the minimum number of operations; (b) with the minimum number of adder-steps.

representation and all possible implementations of constant multiplications that can be extracted from the representations of the constants are obtained. Then, the MCM problem is defined as a 0–1 ILP problem with constraints to be satisfied and a cost function to be minimized. Finally, the minimum number of operations solution is obtained using a generic 0–1 ILP solver. The problem reduction and model simplification techniques, that significantly reduce the size of the 0–1 ILP problem and, consequently, the required CPU time for the 0–1 ILP solver to find the minimum solution, were introduced in [6,20]. Prominent CSE heuristics can be found in [4,5,21,22].

The first GB methods, 'add-only', 'add/subtract', 'add/shift', and 'add/subtract/shift', were proposed in [23]. The 'add/subtract/shift' algorithm was modified in [7], called BHM, by extending the possible implementations of a constant, considering only odd numbers, and processing constants in order of increasing single constant cost that is evaluated by the algorithm of [24]. The RAG-n algorithm that includes two parts, optimal and heuristic, was also introduced in [7]. In its optimal part, each target constant that can be implemented with a single operation are synthesized. If there exist unimplemented elements left in the target set, RAG-n switches to its heuristic part. In this iterative part, RAG-n initially chooses a single unimplemented target constant with the smallest single constant cost evaluated by the algorithm of [24] and then, synthesizes it with a single operation including one (two) intermediate constant (s) that has (have) the smallest value among the possible constants. However, since the intermediate constants are selected for the implementation of a single target constant in each iteration, the intermediate constants chosen in previous iterations may not be shared for the implementation of the not-yet synthesized target constants in later iterations thus, may yield a local minimum solution. To overcome this limitation, the Hcub algorithm [8], which includes the same optimal part of RAG-n, uses a better heuristic that considers the impact of each possible intermediate constant on the not-yet synthesized target constants. In each iteration, for the implementation of a single target constant, Hcub chooses a single intermediate constant that yields the best cumulative benefit over all unimplemented target constants. The algorithm of [9], that includes the same optimal part as RAG-n and Hcub, computes all possible intermediate constants which can be synthesized with the current set of implemented constants using a single operation and chooses an intermediate constant that synthesizes the largest number of target constants in each iteration of its heuristic part. The exact GB algorithms that search a solution with the minimum number of operations in breadth-first and depth-first manners were also introduced in [9].

For the MCM problem under a delay constraint, the 0–1 ILP formalization when the constants are defined under a number representation was also given in [6,20]. In these algorithms, additional constraints were added to the 0–1 ILP problem to guarantee that the delay constraint is never violated. In the nonrecursive CSE algorithm of [25], contrary to the CSE heuristics of [4,26], the subexpressions extracted from a constant multiplication are not shared with those of different constant multiplications. This modification leads to independent structures of constant multiplications, where the relation between the number of operations and the number of adder-steps is compromised by the proposed method. In the GB heuristic of [13], three methods that reduce the number of adder-steps are applied in BHM and RAG-n algorithms designed for the MCM problem, leading to two algorithms called SLBHM and SLRAGn respectively. The modified version of Hcub introduced to handle the delay constraint, called Hcub-DC in this article, is available in [14]. In a different approach, the C1 algorithm [27] initially finds a solution using BHM or RAG-n including generally more number of operations but, with a small number of adder-steps. Then, it reduces the number of operations without increasing the delay in an iterative loop. We note that the C1 algorithm [27] cannot find a solution under a specific delay constraint. Furthermore, the MinLD algorithm [28] finds an MCM solution by synthesizing each constant at its minimum adder-step value. As the C1 algorithm, it cannot find a solution under a specific delay constraint, but its solution guarantees the minimum adder-steps of the MCM design.

For the gate-level area optimization problem, the exact CSE algorithm [10] formulates the optimization of area problem as a 0–1 ILP problem and finds the minimum area solution of the MCM operation when constants are defined under a particular number representation. Also, the GB algorithm [12], called RFAG-n, that is based on RAG-n, 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 leads to the smallest number of FAs overhead. The MINAS algorithm [11], presented in Section 6.1, is based on the approximate algorithm of [9] and in each iteration, finds an intermediate that has the smallest implementation cost and enables to implement the not-yet synthesized target constants using less hardware. These algorithms find the smallest area solution of the MCM operation while implementing each target and intermediate constant. In an interesting approach [29], initially, the solution of an MCM problem instance, i.e., the ready set consisting of the target and intermediate constants, is obtained by an MCM algorithm. Then, the interconnection strategies that focus on the reduction of area, delay, and power consumption in the MCM design were applied while finding the operations that generate the constants.

For the gate-level area optimization problem under a delay constraint, to the best our knowledge, only the MINAS-DC algorithm [11] has been proposed and will be described in Section 6.2.

4. Addition and subtraction architectures

This section presents architectures for all possible addition and subtraction operations encountered in the MCM design and gives

Table 1
Implementation cost of addition operations.

Operation	$u + 2^{l_2} v$		$(u + v)2^{-r}$	
Number	Unsigned	Signed	Unsigned	Signed
#FA	$n_m - l_2 - 1$	$n_M - l_2 - 1$	$n_m - r$	$n_M - r$
#HA	$n_M - n_m + 1$	1	$n_M - n_m$	0

the implementation 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 $t_i x$ is $\lceil log_2 t_i \rceil + N$, where N is the bit-width of the variable x that the constants are multiplied with. Hence, the area cost of an operation also depends on the bit-width of the input. Also, it depends on 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:

- 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: $max\ (n_u + l_1, n_v + l_2)$.

4.1. Addition operations

The implementation costs of addition operations are given in Table 1. As described in Section 2.2, there are only two types of addition operation that need to be analyzed due to the restriction on the left and right shifts of an *A-operation* which realizes a positive and odd constant.

Addition operation $u + 2^{l_2} v$: Observe from examples on the unsigned input model given in Fig. 4a and 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 1 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 Fig. 4b. 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 Fig. 4c that to determine the carry bit for the first FA, an OR gate whose inputs are the rth digits of operands u and v is required, although it is not listed in Table 1.

4.2. Subtraction operations

The implementation costs of subtraction operations are given in Table 2. Note that the subtraction operation is implemented using 2's complement, i.e., $u + \bar{v} + 1$. Hence, the number of inverter (inv) gates is included into the implementation cost of the subtraction operations. Also, 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 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 (c_{out}) are the functions of the input u_i and the carry input (c_{in}) given as $sum = \overline{c_{in} \oplus u_i}$ and $c_{out} = c_{in} + u_i$.

Subtraction operation $2^{l_1} u - v$: Observe from the example on unsigned input model given in Fig. 5a 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 2 also consider the case where the digits of operand u and v do not overlap, $l_1 \geqslant n_v$, that is not considered in [10].

Subtraction operation $u - 2^{l_2} v$: Observe from the example on unsigned input model presented in Fig. 5b 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 Fig. 5c that the operation output can be obtained by starting the addition from the $(r + 1)$th digit of the operands u and v, since the operation output is shifted right by r times.

5. The HCUB + ILP algorithms

This section presents the HCUB + ILP algorithms that target the optimization of area and the optimization of area under a delay constraint in MCM.

5.1. Optimization of gate-level area

The main idea behind the HCUB + ILP algorithm is to combine two efficient techniques of [8] and [10] to optimize area of the MCM design. In a preprocessing phase, 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 HCUB + ILP algorithm, whose pseudo-code is given in Fig. 6, takes these parameters as an input.

In the iterative loop of HCUB + ILP (lines 3–15), we initially find a set of operations, O, that implements the constant multiplications using Hcub [8] with a different seed at each time (line 5). Then, we determine the intermediate and target constants in the solution of Hcub 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, we add the depth-1 constants to R if they do not exist (line 7). The depth-1 constants have the adder-step value 1 and are in the form of $2^{i+1} - 1$ and $2^i + 1$, where i ranges in between 1 and bw. 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 described in Section 5.1.1 is applied on this ready set R to find a set of *A-operations* that realize the target and intermediate constants using optimal area at gate-level (line 10). It is based on the exact CSE algorithm of [10] and formalizes the optimization of gate-level area problem as a 0–1 ILP problem. After a solution is obtained by the *ILP* function, the implementation cost of the MCM design, $icost$, is computed as if all the *A-operations* were designed at gate-level, as given in Section 4 (line 11) and if its cost value is smaller than the best one ($icost_b$) found so far, then R_b,O_b, and $icost_b$ are updated (lines 12–14). 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 15).

5.1.1. The ILP technique

The *ILP* function used in HCUB + ILP consists of four main parts: (i) generation of constant implementations; (ii) construction of the

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

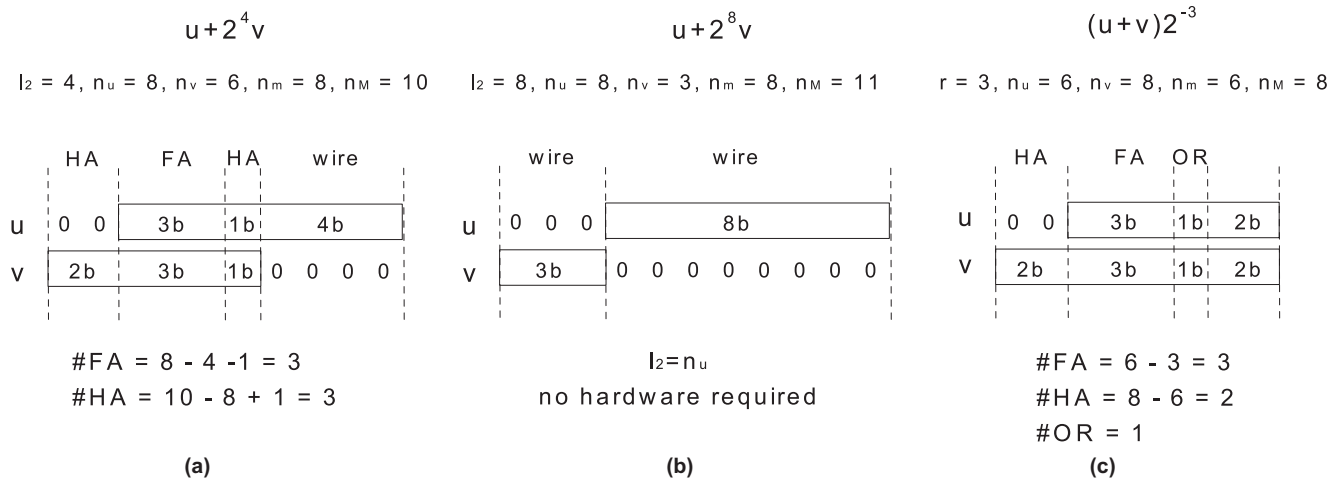

Fig. 4. Examples on addition operations under unsigned input: (a and b) $u + 2^{l_2} v$; (c) $(u + v)2^{-r}$.

Table 2
Implementation cost of subtraction operations.

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

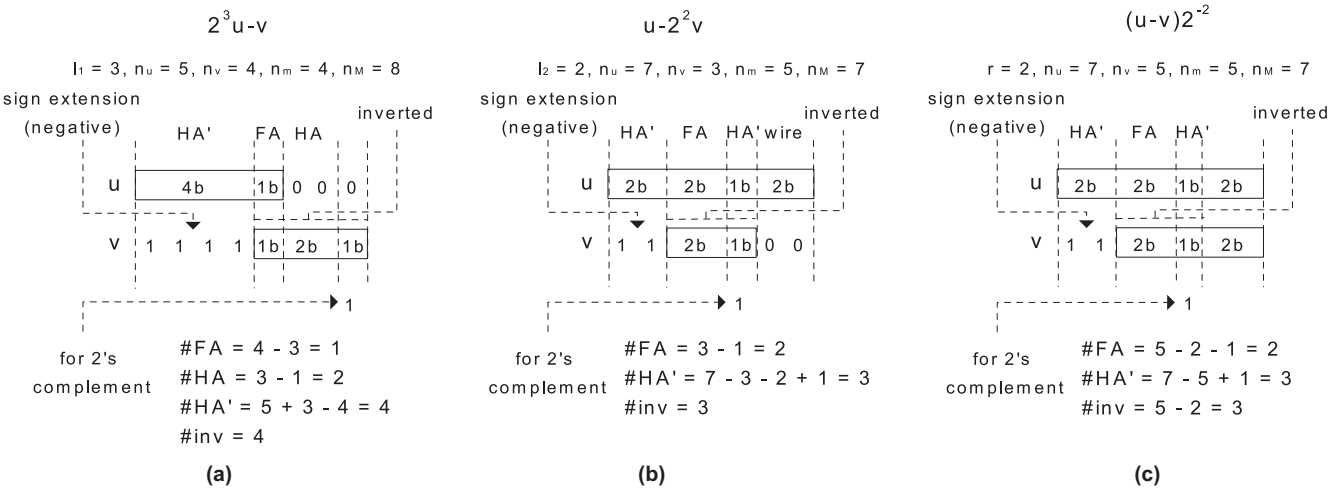

Fig. 5. Examples on subtraction operations under unsigned input: (a) $2^{l_1} u - v$; (b) $u - 2^{l_2} v$; (c) $(u - v)2^{-r}$.

Boolean network that represents the implementations of constants; (iii) formalization of the problem as a 0–1 ILP problem; and (iv) obtaining the minimum solution. In the following, these parts are described in detail.

Generation of constant implementations: After a set of operations O is found on an MCM instance by Hcub, while determining the ready set R, we also compute the adder-step values of each constant in R based on the set of operations O. After the depth-1 constants are included into R, we sort the constants in R according to their adder-step values in ascending order. The part of the algorithm, where the implementations of constants are found, is given as follows:

1. Take an element from R, r_i, except '1' that denotes the input 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, as a pair and its implementation cost.
2. For each *A-operation* that generates r_i,
   (a) Compute its implementation cost.
   (b) Make each of its inputs positive and odd.
   (c) Add its positive and odd inputs as a pair and the implementation cost of the operation to the set I_i.
3. Repeat Step 1 until all elements of R are considered.

HCUB-DC+ILP(**T**, **bw**)

$seed = 0, icost_b = \infty$
$R_{set} \leftarrow \{\}$, $R_b \leftarrow \{\}$, $O_b \leftarrow \{\}$
repeat
  $seed = seed + 1$
  $O = \text{Hcub}(T, seed)$
  $R = \text{GenerateReadySet}(O)$
  $R = \text{AddDepth1Constants}(R, bw)$
  if $R \notin R_{set}$ **then**
    $R_{set} \leftarrow R_{set} \cup R$
    $(R, O) = \text{ILP}(T, bw, R)$
    $icost = \text{ComputeImplementationCost}(O)$
    if $icost < icost_b$ **then**
      $icost_b = icost$
      $R_b \leftarrow R$, $O_b \leftarrow O$
until Termination conditions meet
return O_b

Fig. 6. The HCUB + ILP algorithm.

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 Eq. 3. Then, for each constant r_j, where $0 \leqslant j \leqslant i-1$, we assign r_j to the input u of the *A-operation* and by changing the shifts l_1, l_2, and r and the sign s values, we compute the input v of the *A-operation*. Note that shifts are restricted to $bw + 1$ and only one of the shifts 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 $0 \leqslant k \leqslant i-1$, 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 [30]. By doing so, we also 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 area as the MCM design obtained by Hcub.

As a simple example, consider one of the solutions of Hcub on the constant 11 with two operations, i.e., $3 = 1 \ll 2 - 1$ and $11 = 3 \ll 2 - 1$. After including the depth-1 constants, the ready set is formed as $R = \{1,3,5,7,9,15,17,31,11\}$. Thus, among many others, some implementations of the constant 11 including only the elements of R can be given as $11 = 5 \ll 1 + 1$, $11 = 1 \ll 4 - 5$, $11 = 9 \ll 1 - 7$, and $11 = (15 + 7) \gg 1$. We note that each of these operations has a different implementation cost.

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. The part of the algorithm where the network is constructed is given as follows:

1. Take an element from R, r_i, except '1'.
2. For each pair in I_i, generate a two-input AND gate. The inputs of the AND gate are the elements of the pair, i.e., '1' or the outputs of the OR gates representing the target and intermediate constants in the network.
3. Generate an OR gate associated with r_i where its inputs are the outputs of AND gates determined in Step 2.
4. If r_i is a target constant, assign the output of the corresponding OR gate as the output of the network.
5. Repeat Step 1 until all elements in R are considered.

After the network is constructed, we include the optimization variables into the network so that the cost function of the 0–1 ILP problem can be easily formed. To do this, we associate the optimization variables with the operations as done in [10]. Hence, for each AND gate in the network, we add a third input representing the optimization variable. Thus, the properties of the Boolean network can be given as follows:

1. The primary input of the network is the input to be multiplied with the constants denoted by '1'.
2. An AND gate in the network represents an addition or a subtraction operation and has three inputs including the optimization variable.
3. An OR gate in the network represents a target or an intermediate constant and combines all possible implementations of the constant.
4. The outputs of the network are the OR gate outputs associated with the target constants.

The 0–1 ILP formalization: The cost function of the 0–1 ILP problem is determined as the linear function of optimization variables representing operations, where the cost value of each optimization variable is the gate-level implementation cost of each operation determined as described in Section 4. 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 in CNF formulas as a linear inequality as described in [31]. For example, a 3-input AND gate, $d = a \wedge b \wedge c$, is translated to CNF as $(a + \bar{d})(b + \bar{d})(c + \bar{d})(\bar{a} + \bar{b} + \bar{c} + d)$ and converted to linear constraints as $a - d \geqslant 0$, $b - d \geqslant 0$, $c - d \geqslant 0$, $-a - b - c + d \geqslant -2$. 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. Thus, the generated model can serve as an input to a generic 0–1 ILP solver.

Finding the minimum solution: On the generated problem, a generic 0–1 ILP solver will search the minimum value of the cost function while satisfying the constraints that represent how target and intermediate constants are implemented. The solution of the 0–1 ILP solver, i.e., the operations whose optimization variables are set to 1, will directly determine the set of operations that yields the minimum area.

Interested readers may refer to [6,10] for more detailed explanations.

5.1.2. Properties of the HCUB + ILP algorithm

In this approach, in fact, any algorithm designed for the MCM problem can be applied. Although there exist many efficient algorithms, we preferred to use Hcub since it is one of the best algorithms designed for the MCM problem [8]. Moreover, Hcub can find more than one solution of an MCM problem (if there exists) with the use of randomness in the selection of intermediate constants. This enables the proposed approach to iteratively consider a local point in the search space of many possible MCM realizations. Furthermore, the *ILP* function enables the proposed algorithm to find an MCM operation with optimal area by exploring the region around this local point.

Although the approach of HCUB + ILP is similar to that of [29], there are three major differences between these methods. In HCUB + ILP, first, more than one realization of the MCM instance is considered if possible. Second, the minimum area solution is found using an exact method. Third, the depth-1 constants, which are less complex intermediate constants, are also included into the ready set to extend the possible implementations of intermediate and target constants. In addition to the depth-1 constants, we carried out experiments by including some depth-2 constants, *i.e.*, the constants that can be implemented using a single operation whose one input is always a depth-1 constant and the other is a depth-1 constant or '1'. It was observed that although better solutions in terms

of area can be found, the 0–1 ILP problem size increases significantly and the run-time of the 0–1 ILP solver.

5.2. Optimization of gate-level area under a delay constraint

The modified version of HCUB + ILP in order to handle the user-specified delay constraint *dc* given in terms of the number of adder-steps, called HCUB-DC + ILP, follows the same procedure of HCUB + ILP as given in Fig. 6. However, in HCUB-DC + ILP, Hcub is applied on the target set *T* with *dc*. Also, in the ILP function, while finding the possible implementations of an element in *R*, r_i with an adder-step value as_i, the *A-operations* whose inputs have less than as_i values are considered. Thus, in HCUB-DC + ILP, it is guaranteed that *dc* will never be exceeded when a set of operations, which leads to an MCM design with optimal area, is found.

6. The MINAS algorithms

This section presents the MINAS and MINAS-DC algorithms [11] designed for the gate-level area optimization problem and the gate-level area optimization problem under a delay constraint respectively.

6.1. Optimization of gate-level area

As opposed to the HCUB + ILP algorithm that iteratively finds a local minimum solution of the MCM problem and searches an MCM design with optimal area around this local minimum solution, the MINAS algorithm searches the "best" intermediate constants that lead to an MCM operation with optimal area while synthesizing all the target constants. During the selection of an intermediate constant for the implementation of the not-yet synthesized target constants in each iteration, MINAS favors the one that can be synthesized using the least hardware and will enable to implement the not-yet synthesized target constants in a smaller area with the available constants. After a 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 MCM.

The preprocessing phase of the MINAS algorithm is the same as that of the HCUB + ILP algorithm. The pseudo-codes of the algorithm and its functions are given in Figs. 7 and 8 respectively.

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 its iterative loop (lines 3–13), an intermediate constant is added to the ready set until all the target constants are synthesized. The intermediate constants considered in MINAS are 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). Note that the *ComputeCost* function (line 6) searches all *A-operations* that generate the constant with the elements of the current ready set. If there exist such *A-operations*, then it determines the cost of each operation as described in Section 4 and returns the minimum implementation cost of the constant. Otherwise, it returns 0 value indicating that the constant cannot be synthesized using an *A-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 (lines 8–9). 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 each of its digits is implemented using an FA, i.e., a total of $\lceil log_2 t_k \rceil$ 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 (line 10). After the cost value of each possible intermediate constant is found, the intermediate constant with the minimum cost is chosen to be added into the current ready set and its implications on the current target set are found using the *Synthesize* function (lines 11–13).

When there are no elements left in the target set, the *SynthesizeMinArea* function (line 14) is applied to find a set of *A-operations* that yields an MCM design with optimal 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 Section 5.1.1. This is simply because in each iteration of MINAS, the cost of an intermediate constant is determined by an operation whose inputs are available in the current ready set. However, the recently added intermediate constants may yield better realizations of previously added constants. In this case, the possible implementations of the constants are found by the *GenerateImp* function given in Fig. 8.

6.2. Optimization of gate-level 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

MINAS(T, bw)

```
 1: R ← {1}
 2: (R, T) = Synthesize(R, T)
 3: while T ≠ ∅ do
 4:     for j = 1 to 2^(bw+1) − 1 step 2 do
 5:         if j ∉ R and j ∉ T then
 6:             impcost_j = ComputeCost({j}, R)
 7:             if impcost_j ≠ 0 then
 8:                 A ← R ∪ {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 ← R ∪ {ic}
13:     (R, T) = Synthesize(R, T)
14: D = SynthesizeMinArea(R)
15: return D
```

Fig. 7. The main part of the MINAS algorithm.

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: $cost_{b_k} = \text{ComputeCost}(\{b_k\}, C)$
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: $(B, C) = \text{Synthesize}(A, \{r_k\})$
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** $R = \emptyset$

Fig. 8. The routines of the MINAS algorithm.

that of MINAS. Additionally, MINAS-DC takes the delay constraint *dc* as an input. It follows the similar procedure described in Figs. 7 and 8. However, in its *Synthesize* function, while finding a possible implementation of an intermediate or a target constant using an *A-operation*, it considers the operation whose implementation does not exceed *dc*. Also, in finding the implementation cost of an intermediate or a target constant in its *ComputeCost* and *ComputeTCost* functions, it considers the possible implementations that do not violate *dc*. After the set of target and intermediate constants is obtained, finding an MCM design with optimal 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 the *dc* value.

7. Experimental results

This section presents the high-level and gate-level results of algorithms presented in this article and compares with those of the algorithms designed for the MCM problem, the MCM problem under a delay constraint, and the gate-level area optimization problem. The gate-level results of an MCM operation are obtained in two phases. First, the solution of a high-level algorithm on an MCM instance, *i.e.*, a set of *A-operations* that generates MCM, is found. Second, the addition and subtraction operations that implement the MCM are described in a synthesizable format and then, the MCM design is mapped with the logic gates given in the UMC Logic 0.18μm Generic II library using the Sequential Interactive System (SIS) design tool.

As the first experiment set, we used uniformly distributed randomly generated instances where constants were defined under 14 bit-width. The number of constants (n) ranges between 10 and 100, and we generated 30 instances for each of them. Table 3 presents the results of the algorithms where *nop* (%) denotes the total number of operations normalized with respect to the solutions of the exact GB algorithm [9] and *narea* (%) stands for the total gate-level area normalized according to the solutions of the HCUB + ILP algorithm. In the exact CSE algorithm [10], the constants were defined under MSD. The solutions of Hcub [8] are the ones that have the best area found in the iterative loop of the HCUB + ILP algorithm. In HCUB + ILP-D1, the inclusion of the depth-1 constants to the ready set obtained by Hcub is not considered, actually line 7 of Fig. 6 is skipped. Finally, in this experiment, the unsigned input model was used and the bit-width of the input variable that the constants are multiplied with was taken as 16.

Table 3
Summary of algorithms on randomly generated instances.

n	Opt. of the number of operations				Opt. of gate-level area							
	Hcub [8]		Exact GB [9]		Exact CSE [10]		HCUB + ILP-D1		HCUB + ILP		MINAS	
	nop	narea	nop	narea	nop	narea	nop	narea	nop	narea	nop	narea
10	103.0	103.8	**100**	103.6	125.4	108.6	103.2	102.8	104.7	**100**	106.7	101.2
20	102.2	105.7	**100**	104.7	130.1	108.3	102.2	103.8	106.6	**100**	103.0	98.5
30	100.8	107.9	**100**	106.0	127.5	109.0	100.7	105.0	105.1	**100**	100.4	101.5
40	100.2	110.1	**100**	109.1	126.0	110.0	100.2	105.0	104.8	**100**	100.1	103.7
50	100.1	111.4	**100**	110.8	123.7	109.5	100.1	104.5	105.6	**100**	100.0	103.7
60	100.1	112.4	**100**	113.3	121.1	109.8	100.1	104.0	104.6	**100**	100.0	104.3
70	100.0	113.4	**100**	114.5	118.0	109.4	100.0	103.3	104.4	**100**	100.0	103.8
80	100.0	114.6	**100**	116.5	114.7	109.6	100.0	103.1	103.5	**100**	100.0	103.5
90	100.0	115.9	**100**	118.1	113.9	108.8	100.0	103.7	103.3	**100**	100.0	103.9
100	100.0	114.6	**100**	116.5	114.7	109.6	100.0	102.8	103.5	**100**	100.0	103.5

Table 4
Summary of high-level results of algorithms designed for the MCM and gate-level area optimization problems.

Fil.	Optimization of the number of operations						Optimization of gate-level area					
	Hcub [8]			Exact GB [9]			HCUB + ILP			MINAS		
	op	as	GPS	op	as	GPS	op	as	GPS	op	as	GPS
1	22	8	3597	22	10	5346	25	6	1824	23	8	2982
2	23	7	2987	23	7	3420	24	6	2295	23	5	2654
3	18	8	2715	17	11	4191	19	5	1762	18	8	3357
4	18	9	2982	17	8	2807	20	6	1878	18	7	2290
5	42	8	9651	41	10	9242	45	7	4747	41	11	9419
6	32	12	5989	31	11	7259	34	6	3567	33	8	5152
7	24	7	3071	23	10	4725	26	5	2172	25	6	2564
8	38	5	3842	37	6	5166	38	7	3811	37	9	6281
Avg.	27.1	8.0	4354	26.4	9.1	5270	28.9	6.0	2757	27.3	7.8	4337

Table 3 clearly indicates that a solution with the minimum number of operations does not always lead to an MCM design with optimal area at gate-level. It also shows that the exact CSE algorithm [10] does not yield better solutions in terms of area since its search space is significantly restricted with the particular number representation. The results on Hcub and HCUB + ILP-D1 point out that the *ILP* function of Fig. 6 significantly reduces the area of the MCM design found by Hcub. Also, the results on HCUB + ILP-D1 and HCUB + ILP show that further improvements can be achieved by including the depth-1 constants to the ready set obtained with Hcub. Moreover, HCUB + ILP leads to better results in terms of area than MINAS on almost all instances. One of the main reasons of this fact is that HCUB + ILP considers more than one possible realization of the MCM design with the fewest number of operations obtained by Hcub. Another one is that HCUB + ILP increases the possible implementations of a constant in its *ILP* function by including the depth-1 constants to the ready set obtained by Hcub. However, as can be observed when n is 20, there are still instances that MINAS can find better MCM designs than HCUB + ILP.

As the second experiment set, we used FIR filters[3] given in [11]. Table 4 presents the high-level results of algorithms, where *op* and *as* stand for the number of operations and the number of adder-steps respectively and *GPS* denotes the power dissipation estimation value obtained by the formulas given in [17]. In computation of the GPS value, the bit-width of the filter input was 16, as in the design of MCM operations at gate-level.

As can be observed from Table 4, although HCUB + ILP and MINAS find MCM designs including greater number of operations than those of the algorithms [8,9] designed for the MCM problem, their solutions lead to an MCM design with lower adder-step and *GPS* values on overall instances. We note that the average run-time of HCUB + ILP was 51.1s determined on a PC with Intel Xeon at 2.33 GHz and 4 GB of memory under Linux. The run-time of the 0–1 ILP solver SCIP [32] on the 0–1 ILP problems generated by its *ILP* function took maximum 0.86 s and less than 0.4 s on average. Also, the average run-time of MINAS was 412.2 s. Note that both algorithms were written in MATLAB and HCUB + ILP uses Hcub and SCIP that were written in C++.

The gate-level results of MCM designs that are obtained using the solutions of high-level synthesis algorithms given in Table 4 are presented in Table 5. In this table, *area* (mm^2), *delay* (ns), and *power* (mW) denote the area, delay, and power dissipation results of MCM designs at gate-level respectively. Note that power dissipation values were obtained using simulation results under 10,000 random input vectors with the SIS tool.

Observe from Tables 4 and 5 that although HCUB + ILP and MINAS find an MCM operation with greater number of operations than those obtained by the algorithms [8,9] designed for the MCM problem, the gate-level implementation of their solutions yields an MCM design with smaller area. Also, their solutions lead to high-speed and low-power MCM designs on overall instances. On the other hand, although HCUB + ILP obtains the smallest area MCM designs on average, there are MCM instances that MINAS finds the smallest area solution, e.g., Filters 4, 6, and 7. This is simply because the MINAS algorithm finds the "best" intermediate constants that require less hardware and enable to realize the not-yet synthesized target constants with less area. Hence, the intermediate constants chosen in MINAS, which may not be considered in Hcub, may yield less-complex MCM designs.

Table 6 presents the high-level results of algorithms Hcub-DC, HCUB-DC + ILP, and MINAS-DC algorithms, that are the modified versions of the algorithms presented in Tables 4 and 5 to handle the delay constraint. On these algorithms, the delay constraint was set to

[3] The FIR filter instances are available at http://algos.inesc-id.pt/multicon.

Table 5
Summary of gate-level results of algorithms designed for the MCM and gate-level area optimization problems.

Fil.	Optimization of the number of operations						Optimization of gate-level area					
	Hcub [8]			Exact GB [9]			HCUB + ILP			MINAS		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
1	29.2	60.4	513.7	29.4	60.2	802.3	27.9	49.0	149.2	28.1	57.4	437.3
2	31.1	58.8	334.3	31.2	63.0	365.0	29.0	54.3	194.4	29.5	57.8	226.7
3	23.6	64.2	440.4	23.3	61.2	607.4	21.8	58.3	136.6	22.1	60.7	397.8
4	23.1	54.0	302.9	22.5	60.8	368.3	22.2	37.7	147.7	21.6	52.8	206.5
5	59.2	70.5	1280.3	59.0	74.8	1373.5	54.1	56.3	407.3	55.4	73.8	1328.8
6	43.5	65.5	815.3	43.2	66.5	1139.9	41.0	55.2	320.8	40.2	63.0	664.4
7	31.9	61.5	330.4	32.1	65.9	852.5	30.2	47.0	170.5	29.9	50.7	239.8
8	45.1	61.4	465.5	45.8	58.3	642.7	40.2	45.5	341.0	42.9	56.7	930.4
Avg.	35.9	62.1	560.4	35.8	63.8	769.0	33.3	50.4	233.4	33.7	59.1	554.0

Table 6
Summary of high-level results of algorithms designed for the MCM and gate-level area optimization problems under the minimum delay constraint.

Fil.	Opt. of the number of operations under the minimum delay constraint						Opt. of gate-level area under the minimum delay constraint					
	Hcub-DC [14]			MinLD [28]			HCUB-DC + ILP			MINAS-DC		
	op	as	GPS	op	as	GPS	op	as	GPS	op	as	GPS
1	25	3	1875	26	3	1520	25	3	1575	26	3	1770
2	27	3	2115	27	3	1770	28	3	1878	28	3	1831
3	23	3	1663	21	3	1271	23	3	1442	22	3	1523
4	20	3	1540	22	3	1444	21	3	1501	22	3	1374
5	55	3	4610	51	3	3730	54	3	3994	51	3	3609
6	39	3	3088	37	3	2421	40	3	2788	38	3	2589
7	29	3	2138	28	3	2043	29	3	2102	30	3	1916
8	43	3	3274	44	3	2597	46	3	2876	43	3	2611
Avg.	32.6	3	2538	32	3	2100	33.3	3	2270	32.5	3	2153

the minimum delay of the MCM design computed as given in Eq. (4). In addition to these algorithms, this table shows the results of the MinLD algorithm [28] that also ensures an MCM design with the minimum number of adder-steps.

Observe from Tables 4 and 6 that in Hcub-DC, HCUB-DC + ILP, and MINAS-DC, a large number of operations is required to find a solution with the minimum number of adder-steps with respect to the solutions of Hcub, HCUB + ILP, and MINAS respectively. This is because the solutions obtained with these algorithms using the fewest number of operations lead to MCM designs with large number of adder-steps that is far away from their minimum. Also, observe that although finding an MCM design under a delay constraint generally increases the number of operations, it reduces the GPS value significantly. Also, observe from Table 6 that MinLD finds the best results in terms of the number of operations on average among these algorithms.

Table 7 presents the gate-level results of MCM designs that are obtained using the solutions of high-level algorithms given in Table 6. Observe that while HCUB-DC + ILP improves the area of the MCM operation obtained by Hcub-DC significantly, the solutions of MinLD lead to less complex MCM designs than those obtained by Hcub and HCUB-DC + ILP on average, since it obtains better solutions in terms of the number of operations on average. However, among these algorithms, MINAS-DC obtains the best results in terms of area on overall instances. Also, observe from Tables 5 and 7 that designing the MCM operation with the minimum logic depth using optimal area at gate-level decreases the delay and also the power dissipation of the MCM design significantly.

By changing the delay constraint on the algorithms that optimize the gate-level area under a delay constraint, one can easily find the optimal tradeoff between area and delay. In Table 8, we present the gate-level results of MCM designs obtained by Hcub-

Table 7
Summary of gate-level results of algorithms designed for the MCM and gate-level area optimization problems under the minimum delay constraint.

Fil.	Opt. of the number of operations under the minimum delay constraint						Opt. of gate-level area under the minimum delay constraint					
	Hcub-DC [14]			MinLD [28]			HCUB-DC + ILP			MINAS-DC		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
1	29.9	54.1	171.6	30.0	46.7	103.4	28.9	46.7	112.1	28.3	45.6	104.6
2	33.2	54.5	164.2	32.1	57.4	133.3	31.7	54.5	153.2	30.7	43.6	122.4
3	25.5	54.1	178.9	23.4	51.9	107.2	23.8	46.5	94.7	23.7	48.6	112.4
4	25.1	51.4	102.6	24.8	45.5	100.9	25.0	51.4	96.8	22.9	45.0	85.1
5	69.9	57.9	371.2	60.3	57.2	311.2	61.9	53.6	292.6	53.3	54.5	249.0
6	50.7	55.2	285.7	43.7	54.1	180.5	45.4	56.5	187.4	41.5	46.6	180.5
7	35.7	50.0	136.3	34.4	55.9	141.0	35.1	45.5	128.1	31.5	46.1	111.6
8	50.8	62.2	347.9	46.5	49.3	209.2	45.8	52.4	196.5	43.2	49.4	176.7
Avg.	40.1	54.9	219.8	36.9	52.2	160.8	37.2	50.9	157.7	34.4	47.4	142.8

Table 8
Summary of gate-level results of the Hcub-DC, HCUB-DC + ILP, and MINAS-DC algorithms when *dc* was set to 4.

Fil.	Hcub-DC [14]			HCUB-DC + ILP			MINAS-DC		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
1	30.8	52.6	181.5	28.6	50.1	133.9	29.8	51.4	180.4
2	31.8	55.5	208.0	30.9	55.8	207.4	29.2	50.9	161.8
3	23.5	56.5	241.0	23.0	47.3	115.7	21.7	56.3	183.2
4	22.5	46.2	104.4	21.7	36.4	94.2	21.6	35.8	100.0
5	63.0	61.1	509.0	57.6	51.3	336.7	54.1	54.1	402.9
6	44.6	58.9	339.4	41.8	53.5	229.8	39.5	50.4	195.2
7	34.4	51.1	217.9	32.0	44.3	164.3	30.6	48.5	160.3
8	46.5	64.0	452.4	41.9	54.7	211.1	43.2	53.9	253.5
Avg.	37.1	55.7	281.7	34.7	49.2	186.6	33.7	50.2	204.7

DC, HCUB-DC + ILP, and MINAS-DC when the delay constraint was set to 4 (one additional adder-step compared to the solutions of these algorithms in Table 7). Observe that when the delay constraint is increased, the area of the MCM design is reduced, and the delay and power dissipation are generally increased.

8. Conclusions

This article described two different approaches proposed for the gate-level area optimization problem. It is shown that taking into account the implementation cost of each addition/subtraction operation in the design of the MCM operation leads to significant improvements on the gate-level area with respect to those that are obtained by prominent algorithms which target the optimization of the number of addition/subtraction operations. Also, this article presented their modified versions for the gate-level area optimization problem under a delay constraint which enable us to find the optimal tradeoff between area and delay in the MCM design by changing the delay constraint. It is indicated that the design of the MCM operation under a smaller number of adder-steps yields high-speed and low-power MCM designs but with larger area. It is also shown that the reduction of the number of adder-steps and the hardware area in the MCM design should be realized simultaneously in order to minimize the power dissipation.

Acknowledgments

This work was partially supported by the *Portuguese Foundation for Science and Technology* (FCT) research project *Multicon – Architectural Optimization of DSP Systems with Multiple Constants Multiplications* PTDC/EIA-EIA/103532/2008 and by FCT through the PIDDAC Program funds.

References

[1] H. Nguyen, A. Chatterjee, Number-splitting with shift-and-add decomposition for power and hardware optimization in linear DSP synthesis, IEEE Transactions on VLSI 8 (4) (2000) 419–424.
[2] M. Ercegovac, T. Lang, Digital Arithmetic, Morgan Kaufman, 2003.
[3] P. Cappello, K. Steiglitz, Some complexity issues in digital signal processing, IEEE Transactions on Acoustics, Speech, and Signal Processing 32 (5) (1984) 1037–1041.
[4] R. Hartley, Subexpression sharing in filters using canonic signed digit multipliers, IEEE Transactions on Circuits and Systems II 43 (10) (1996) 677–688.
[5] I.-C. Park, H.-J. Kang, Digital filter synthesis based on minimal signed digit representation, in: Proceedings of Design Automation Conference, 2001, pp. 468–473.
[6] L. Aksoy, E. Costa, P. Flores, J. Monteiro, Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications, IEEE Transactions on Computer-Aided Design of Integrated Circuits 27 (6) (2008) 1013–1026.
[7] A. Dempster, M. Macleod, Use of minimum-adder multiplier blocks in FIR digital filters, IEEE Transactions on Circuits and Systems II 42 (9) (1995) 569–577.
[8] Y. Voronenko, M. Püschel, Multiplierless multiple constant multiplication, ACM Transactions on Algorithms 3 (2) (2007).
[9] L. Aksoy, E. Gunes, P. Flores, Search algorithms for the multiple constant multiplications problem: exact and approximate, Elsevier Journal on Microprocessors and Microsystems 34 (5) (2010) 151–162.
[10] L. Aksoy, E. Costa, P. Flores, J. Monteiro, Optimization of area in digital FIR filters using gate-level metrics, in: Proceedings of Design Automation Conference, 2007, pp. 420–423.
[11] L. Aksoy, E. Costa, P. Flores, J. Monteiro, Optimization of area and delay at gate-level in multiple constant multiplications, in: Proceedings of EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, 2010, pp. 3–10.
[12] K. Johansson, O. Gustafsson, L. Wanhammar, A detailed complexity model for multiple constant multiplication and an algorithm to minimize the complexity, in: Proceedings of IEEE European Conference on Circuit Theory and Design, 2005, pp. 465–468.
[13] H.-J. Kang, I.-C. Park, FIR filter synthesis algorithms for minimizing the delay and the number of adders, IEEE Transactions on Circuits and Systems II 48 (8) (2001) 770–777.
[14] Spiral website, <http://www.spiral.net>.
[15] A. Avizienis, Signed-digit number representation for fast parallel arithmetic, IRE Transactions on Electronic Computers EC-10 (1961) 389–400.
[16] J. Chen, C.-H. Chang, H. Qian, New power index model for switching power analysis from adder graph of FIR filter, in: Proceedings of IEEE International Symposium on Circuits and Systems, 2009, pp. 2197–2200.
[17] S. Demirsoy, A. Dempster, I. Kale, Power analysis of multiplier blocks, in: Proceedings of IEEE International Symposium on Circuits and Systems, 2002, pp. 297–300.
[18] P. Flores, J. Monteiro, E. Costa, An exact algorithm for the maximal sharing of partial terms in multiple constant multiplications, in: Proceedings of International Conference on Computer-Aided Design, 2005, pp. 13–16.
[19] O. Gustafsson, L. Wanhammar, ILP modelling of the common subexpression sharing problem, in: Proceedings of International Conference on Electronics, Circuits and Systems, 2002, pp. 1171–1174.
[20] Y.-H. Ho, C.-U. Lei, H.-K. Kwan, N. Wong, Global optimization of common subexpressions for multiplierless synthesis of multiple constant multiplications, in: Proceedings of Asia and South Pacific Design Automation Conference, 2008, pp. 119–124.
[21] M. Potkonjak, M. Srivastava, A. Chandrakasan, Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination, IEEE Transactions on Computer-Aided Design of Integrated Circuits 15 (2) (1996) 151–165.
[22] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, D. Durackova, A new algorithm for elimination of common subexpressions, IEEE Transactions on Computer-Aided Design of Integrated Circuits 18 (1) (1999) 58–68.
[23] D. Bull, D. Horrocks, Primitive operator digital filters, IEE Proceedings G: Circuits, Devices and Systems 138 (3) (1991) 401–412.
[24] A. Dempster, M. Macleod, Constant integer multiplication using minimum adders, IEE Proceedings – Circuits, Devices and Systems 141 (5) (1994) 407–413.
[25] M. Peiro, E. Boemo, L. Wanhammar, Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 49 (3) (2002) 196–203.
[26] M. Potkonjak, M. Srivastava, A. Chandrakasan, Efficient substitution of multiple constant multiplication by shifts and additions using iterative pairwise matching, in: Proceedings of Design Automation Conference, 1994, pp. 189–194.
[27] A. Dempster, S. Demirsoy, I. Kale, Designing multiplier blocks with low logic depth, in: Proceedings of IEEE International Symposium on Circuits and Systems, 2002, pp. 773–776.
[28] M. Faust, C.-H. Chang, Minimal logic depth adder tree optimization for multiple constant multiplication, in: Proceedings of IEEE International Symposium on Circuits and Systems, 2010, pp. 457–460.
[29] K. Johansson, O. Gustafsson, L. Wanhammar, Bit-level optimization of shift-and-add based FIR filters, in: Proceedings of International Conference on Electronics, Circuits and Systems, 2007, pp. 713–716.
[30] O. Gustafsson, Towards optimal constant multiplication: a hypergraph approach, in: Proceedings of Asilomar Conference on Signals, Systems and Computers, 2008, pp. 1805–1809.
[31] P. Barth, A Davis–Putnam based Enumeration Algorithm for Linear Pseudo-Boolean Optimization, Tech. Rep., Max-Planck-Institut Fur Informatik, 1995.
[32] Solving Constraint Integer Programs website, <http://scip.zib.de/>.

Levent Aksoy received the M.S. and Ph.D. degrees in electronics and communication engineering and electronics engineering from Istanbul Technical University (ITU) in 2003 and 2009 respectively. He was a Research Assistant with the Division of Circuits and Systems, Faculty of Electrical and Electronics Engineering, ITU from 2001 to 2009. Since November 2009, he has been with the Algorithms for Optimization and Simulation Research Unit, Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, Portugal, where he is currently a post-doctoral researcher. His research interests include satisfiability algorithms, pseudo-Boolean optimization, and electronic design automation problems.

Eduardo da Costa received the five-year engineering degree in electrical engineering from the University of Pernambuco, Recife, Brazil, in 1988, the M.Sc. degree in electrical engineering from the Federal University of Paraiba, Campina Grande, Paraíba, Brazil, in 1991, and the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2002. Part of his doctoral work was developed at the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, Portugal. He is currently a Professor with the Departments of Electrical Engineering and Informatics, Catholic University of Pelotas (UCPel), Pelotas, Brazil. He is also with the Master Degree Program in Computer Science, UCPel, as a Professor and a Researcher. His research interests are VLSI architectures and low-power design.

Paulo Flores received the five-year engineering degree, M.Sc., and Ph.D. degrees in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 1989, 1993, and 2001, respectively. Since 1990, he has been teaching at Instituto Superior Técnico, Technical University of Lisbon, where he is currently an Assistant Professor in the Department of Electrical and Computer Engineering. He has also been with the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, since 1988, where he is currently a Senior Researcher. His research interests are in the area of embedded systems, test and verification of digital systems, and computer algorithms, with particular emphasis on optimization of hardware/software problems using satisfiability (SAT) models. Dr. Flores is a member of the IEEE Circuit and Systems Society.

José Monteiro received the five-year engineering degree and the M.Sc. degree in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 1989 and 1992, respectively, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1996. Since 1996, he has been with Instituto Superior Técnico, Technical University of Lisbon, where he is currently an Associate Professor in the Department of Computer Science and Engineering. He is currently the Director of the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon. His main interests are computer architecture and CAD for VLSI circuits, with emphasis on synthesis, power analysis, and low-power and design validation. Dr. Monteiro received the Best Paper Award from the IEEE Transactions on Very Large Scale Integration (VLSI) Systems in 1995. He has served on the Technical Program Committees of several conferences and workshops.