Contents lists available at SciVerse ScienceDirect

INTEGRATION, the VLSI journal

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

High-level algorithms for the optimization of gate-level area in digit-serial multiple constant multiplications

Levent Aksoy [a,*], Cristiano Lazzari [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

Available online 1 December 2011

Keywords:
Multiple constant multiplications
Digit-serial architectures
Gate-level area optimization
0–1 Integer linear programming
Graph-based algorithms

ABSTRACT

The last two decades have seen tremendous effort on the development of high-level synthesis algorithms for efficient realization of the multiplication of a variable by a set of constants using only addition, subtraction, and shift operations. These algorithms generally target the minimization of the number of adders and subtractors, assuming that shifts are realized using only wires due to the bit-parallel processing of the input data. On the other hand, digit-serial architectures offer alternative low-complexity designs since digit-serial operators occupy less area and are independent of the data wordlength. However, in this case, shifts are no longer free in terms of hardware and require D flip-flops. Moreover, each digit-serial addition, subtraction, and shift operation has different implementation cost at gate-level. Hence, this article introduces high-level algorithms that optimize the area of digit-serial constant multiplications under the shift-adds architecture by taking into account the implementation cost of each operation at gate-level. Experimental results indicate that our high-level algorithms obtain better solutions than prominent algorithms designed for the minimization of the number of operations in terms of gate-level area and their solutions lead to less complex digit-serial MCM designs. It is also shown that the use of shift-adds architecture yields significant area reductions when compared to the constant multiplications designed using generic digit-serial constant multipliers.

1. Introduction

In several computationally intensive Digital Signal Processing (DSP) algorithms, such as Fast Fourier Transforms (FFT) and Finite Impulse Response (FIR) filters (illustrated in Fig. 1), the same input is multiplied by a set of constant coefficients, an operation known as Multiple Constant Multiplications (MCM). In addition to its applications in DSP systems, the MCM operation frequently occurs in cryptography, compilers, and computer arithmetic. In many cases, hardwired dedicated architectures are the best option for maximum performance and minimum power consumption.

Although area, delay, and power efficient multiplier architectures, such as Wallace [1] and modified Booth [2] multipliers, have been proposed, the full-flexibility of a multiplier is not necessary for constant multiplications, since constant coefficients are fixed and determined beforehand. Hence, constant multiplications are generally replaced by addition/subtraction and shift operations [3].

Note that in bit-parallel MCM design, shifts can be realized using only wires without representing any hardware cost. Thus, a fundamental optimization problem, known as the MCM problem, is defined as finding the minimum number of addition/subtraction operations that implement the constant multiplications. Note also that the MCM problem is an NP-complete problem [4].

For the implementation of constant multiplications under the shift-adds architecture, a straightforward method, generally known as the digit-based recoding [5], initially defines the constants in multiplications under binary representation. Then, for each 1 in the representation of the constant, according to its bit position, it shifts the variable and adds up the shifted variables to obtain the result. As a simple example, consider the constant multiplications $29x$ and $43x$. Their decompositions in binary are listed as:

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

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

and require six addition operations as given in Fig. 2(a).

However, the digit-based recoding technique does not consider the sharing of common partial products among the constant multiplications that significantly reduces the area and power

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

0167-9260/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2011.11.008

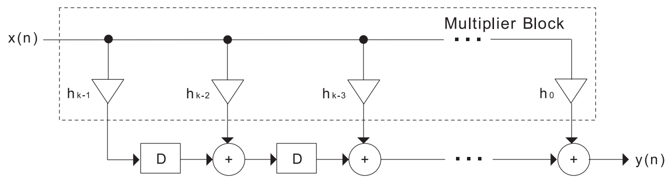

Fig. 1. Transposed form of a digital FIR filter.

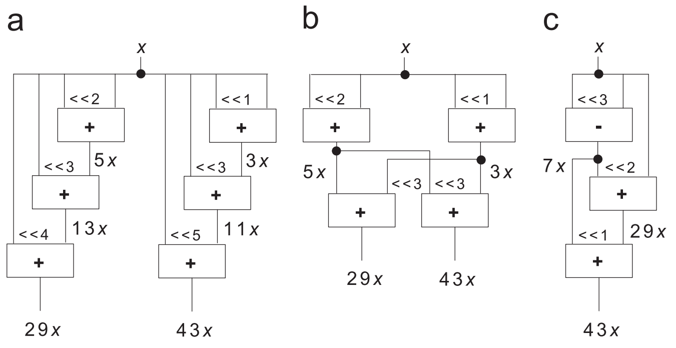

Fig. 2. Shift-adds implementations of 29x and 43x: (a) without partial product sharing [5]; with partial product sharing: (b) the algorithm of [8]; (c) the algorithm of [11].

dissipation of the MCM design. The algorithms that aim to maximize the sharing of partial products can be categorized in two classes: Common Subexpression Elimination (CSE) [6–8] techniques and graph-based (GB) [9–11] methods. The CSE algorithms first define the constants under a particular number representation namely, binary, Canonical Signed Digit (CSD) [6], or Minimal Signed Digit (MSD) [7], and then, find the "best" subexpression, generally the most common, among the constant multiplications. The GB algorithms are not restricted to any particular number representation and consider a large number of alternative implementations of a constant multiplication, yielding better solutions than the CSE algorithms as shown in [10,11]. Returning to our example in Fig. 2, the exact CSE algorithm [8] obtains a solution with four operations by finding the most common partial products $3x=(11)_{bin}x$ and $5x=(101)_{bin}x$ when constants are defined under binary (Fig. 2(b)). The exact GB algorithm [11] finds the minimum number of operations solution with three operations by sharing the common partial product 7x in both multiplications (Fig. 2(c)). Observe that the partial product $7x=(111)_{bin}x$ cannot be extracted from the binary representations of both multiplications 29x and 43x in the exact CSE algorithm [8].

In the MCM problem, it is assumed that the input data x is processed in parallel. Although shifts can be realized using only wires in this case, the implementation cost of an addition/subtraction operation realizing a constant multiplication depends on the bit-width of the input data and the constant [12]. On the other hand, in digit-serial arithmetic, the input data is divided into digit sets, consisting of d bits that are processed one at a time [13]. Thus, the implementation cost of a digit-serial addition/subtraction operation depends on the digit size d, requiring significantly less hardware when compared to its bit-parallel implementation. However, in this case, shifts require D flip-flops to be implemented due to the serial processing. Hence, the high-level algorithms should consider the sharing of addition/subtraction operations as well as the sharing of shifts with respect to the MCM problem. Moreover, as shown in [12], a solution with the minimum number of operations does not always yield a solution with minimum area at gate-level. In order to optimize the gate-level area of the design, the high-level algorithms should also take into account the implementation cost of each addition, subtraction, and shift operation.

Hence, this article introduces two high-level algorithms, called HCUB+ILP-DS and MINAS-DS that target the optimization of gate-level area in digit-serial MCM operation. In the HCUB+ILP-DS algorithm, we initially find a solution with the fewest number of operations that generate MCM, using the Hcub algorithm [10] designed for the MCM problem. Then, we obtain a set of operations that yields a digit-serial MCM design with optimal area by formalizing the optimization of gate-level area problem in digit-serial MCM design as a 0–1 Integer Linear Programming (ILP) problem on the solution of Hcub and finding the minimum solution using a generic 0-1 ILP solver. This article also describes the MINAS-DS algorithm [14] that iteratively finds the partial products which are required to realize the constant multiplications and which lead to digit-serial MCM designs with optimal area at gate-level.

The experimental results on a comprehensive set of instances indicate that the high-level algorithms introduced in this article yield digit-serial MCM designs using less area when compared to those obtained by prominent algorithms designed for the MCM problem. It is pointed out that the digit-serial realization of the MCM operation leads to alternative low-complexity designs with respect to its bit-parallel realization and a designer can easily find the optimal tradeoff between area and delay in the MCM design by changing the digit size d. The experimental results also show that the design of the digit-serial MCM operation under the shift-adds architecture with the use of high-level algorithms yields significant savings in area when compared to those designed using generic digit-serial constant multipliers [15].

The rest of the article proceeds as follows. Section 2 presents the background concepts and an overview on related work is given in Section 3. The HCUB+ILP-DS and MINAS-DS algorithms are introduced in Sections 4 and 5, respectively. Experimental results are presented in Section 6 and finally, Section 7 concludes the paper.

2. Background

This section presents the main concepts related with the proposed algorithms and gives 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 generally defined as follows[1]:

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

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

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

2.2. Multiplierless constant multiplications

Since the common input variable x is multiplied by multiple constants in MCM, the implementation of constant multiplications is in fact equal to the implementation of constants. For example, the implementation of 3x given as $3x=x\ll1+x=(1\ll1+1)x$ can be

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

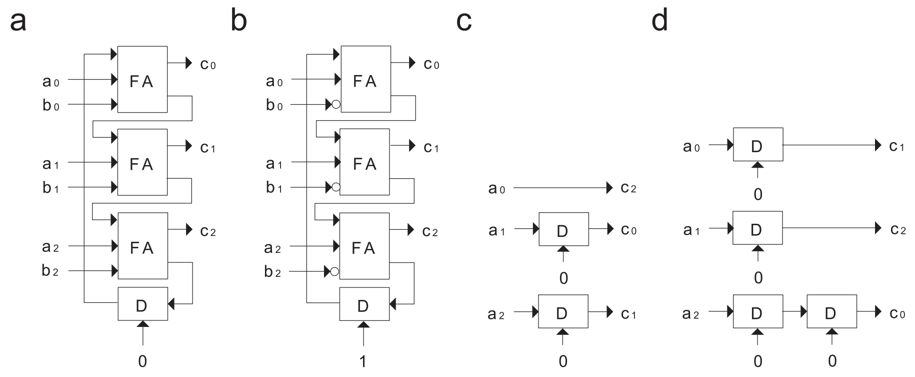

Fig. 3. The digit-serial operations when d is 3: (a) addition operation; (b) subtraction operation; (c) left shift by two times; (d) left shift by four times.

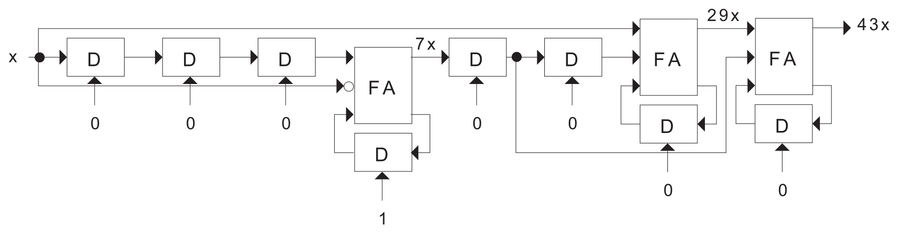

Fig. 4. Bit-serial design of shift-adds implementation of $29x$ and $43x$ given in Fig. 2(c).

rewritten as $3 = 1 \ll 1 + 1$ by eliminating the variable x from both sides. In the rest of the article, these notations will be used interchangeably.

In multiplierless constant multiplications, the fundamental operation, called *A-operation* in [10], 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:

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

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

In the MCM problem, the complexity of an adder and a subtracter in hardware is assumed to be equal. It is also assumed that the sign of the constant can be adjusted at some part of the design and the shifting operation has no cost in hardware due to the bit-parallel processing. Thus, only positive and odd constants are considered in the MCM problem. Observe from 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. 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 [11], 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 [10] is defined as:

Definition 1 (THE MCM PROBLEM). Given the target set composed of positive and odd unrepeated target constants to be implemented, $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$, find the smallest ready set, $R = \{r_0, r_1, \ldots, r_m\}$, with $T \subset R$, under the conditions of $r_0 = 1$ and for all r_k with $1 \leq k \leq m$, there exist r_i, r_j with $0 \leq i,j < k$ and an *A-operation* $r_k = A(r_i, r_j)$.

Hence, the number of operations required to be implemented for the MCM problem is $|R|-1$, as given in [10].

2.3. Implementation of digit-serial constant multiplications under the shift-adds architecture

In digit-serial processing, the input data x is divided into d bits and processed serially by applying each d-bit data in parallel. The special cases of the digit-serial computation, called bit-serial and bit-parallel processing, occur when the digit size d is equal to 1 and equal to the input data wordlength, respectively. The digit-serial computation plays an important role when the bit-serial implementations cannot meet delay requirements and the bit-parallel designs require excessive hardware. Thus, an optimal tradeoff between area and delay can be obtained by changing the digit size parameter (d).

The fundamental digit-serial operations can be found in [13]. The digit-serial addition, subtraction, and left shift operations are depicted in Fig. 3 using a digit size d equal to 3, where the bits with indices 0 and 2 denote the least and most significant bits, respectively. Notice from Fig. 3(a) that a digit-serial addition operation, in general, requires d full adders (FAs) and 1 D flip-flop. The subtraction operation (Fig. 3(b)) is implemented using 2's complement, requiring the initialization of the D flip-flop with 1 and additional d inverter gates with respect to the digit-serial addition operation. In a digit-serial left shift operation (Fig. 3(c,d), the number of required D flip-flops is equal to the shift amount and it is realized in d horizontal layers. The correspondence between the input (a_i) and the output (c_i) of the left shift operation and the number of flip-flops ($\#FF_{a_i}$) cascaded serially for each input at each layer (a_i) of the left shift operation are given in Eqs. (4) and (5), respectively. Note that i ranges from 0 to $d-1$ and ls denotes the amount of left shift:

$$a_i \Rightarrow c_{(i+ls) mod\ d} \tag{4}$$

$$\#FF_{a_i} = \begin{cases} \lfloor ls/d \rfloor & \text{if } i < d-(ls\ mod\ d) \\ \lceil ls/d \rceil & \text{otherwise} \end{cases} \tag{5}$$

As an example on digit-serial realization of constant multiplications under the shift-adds architecture, Fig. 4 illustrates the

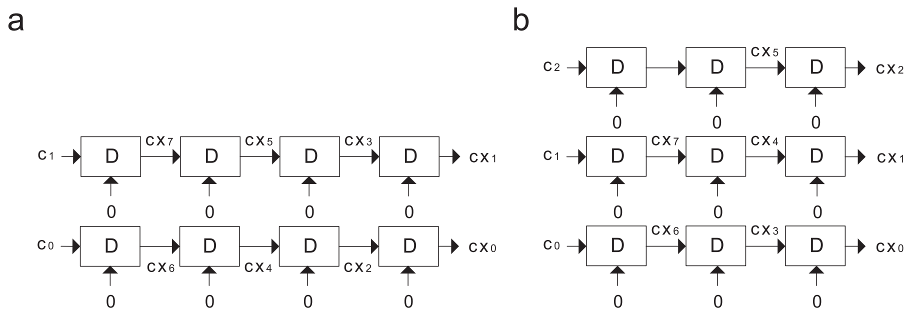

Fig. 5. The storage circuit for an 8-bit constant multiplication cx: (a) when d is 2; (b) when d is 3.

bit-serial ($d=1$) implementation of $29x$ and $43x$ obtained by the exact GB algorithm [11] given in Fig. 2(c). As can be easily observed, the network includes 2 bit-serial additions, 1 bit-serial subtraction, and 5 D flip-flops for all the left shift operations. In this network, at each clock cycle, one bit of the input data x is applied to the network input and one bit of the constant multiplication output is computed at the output of a bit-serial addition/subtraction operation. In general, d bits are processed at each clock cycle.

While the sharing of addition/subtraction operations reduces the complexity of the digit-serial MCM design (since each addition and subtraction operation requires a digit-serial operation), the sharing of shift operations for a constant multiplication also reduces the number of D flip-flops and, consequently, the area of the digit-serial MCM design. Observe from Fig. 4 that two D flip-flops cascaded serially to generate the left shift of $7x$ by two times can also generate the left shift of $7x$ by one time without adding any hardware cost. Hence, the problem of optimizing the number of addition, subtraction, and shift operations can be given as:

Definition 2 (THE MCM-DS PROBLEM). Given the target set $T=\{t_1,\ldots,t_n\}\subset\mathbb{N}$, find the ready set $R=\{r_0,r_1,\ldots,r_m\}$ such that under the same conditions on the ready set given in Definition 1, the set of *A-operations* includes the minimum number of addition, subtraction, and shift operations.

For the digit-serial MCM realization, it is assumed that an *A-operation* that generates a constant multiplication has a right shift which is always 0 [16,17]. This is simply because the complexity of the control logic is significantly increased to realize the MCM operation when there exists such an *A-operation*. Note that a constant multiplication is rarely realized by such an *A-operation* in GB algorithms and it never occurs in CSE algorithms.

As can be observed from Fig. 3, the implementation costs of digit-serial addition, subtraction, and left shift operations are different at gate-level. Thus, to optimize the area of a digit-serial MCM operation, one has to maximize the sharing of addition/subtraction and shift operations considering the implementation cost of each operation. Hence, the optimization of gate-level area problem in digit-serial MCM operation can be defined as:

Definition 3 (THE OPTIMIZATION OF GATE-LEVEL AREA PROBLEM IN DIGIT-SERIAL MCM OPERATION). Given the digit size d and the target set $T=\{t_1,\ldots,t_n\}\subset\mathbb{N}$, find the ready set $R=\{r_0,r_1,\ldots,r_m\}$ such that under the same conditions on the ready set given in Definition 1, the set of *A-operations* yields a digit-serial MCM design using optimal area at gate-level.

2.4. Design details in digit-serial MCM

Although the digit-serial design of the MCM operation occupies significantly less area when compared to its bit-parallel design, the latency of the MCM computation (L_{MCM}), which is dependent on the bit-width of the input data, is increased due to the serial processing. It can be determined in terms of clock cycles as:

$$L_{MCM}=\lceil (bw+N)/d\rceil \quad (6)$$

where bw is the maximum bit-width of the constants to be implemented, N is the bit-width of the input variable x, and d is less than N. Note that Eq. (6) does not apply to bit-parallel processing (when $d=N$), where the latency of the MCM computation is only one clock cycle. Returning to our bit-serial MCM design given in Fig. 4, suppose that x is a 16-bit input value. Thus, we need a total of 22 clock cycles to obtain the actual output of $29x$ and $43x$. Note also that, as a sign-extension, $d\times L_{MCM}-N$ bits must be padded to the input data x, which are 0s, if x is an unsigned input, or sign bits, otherwise.

When the conversion from digit-serial to bit-parallel is required in the digit-serial MCM design, the d-bit outputs of the operations realizing the target constants, generated at each clock cycle, need to be stored. Note that the bit-width of the constant multiplication cx, *i.e.*, bw_{cx}, is computed as $\lceil \log_2 c\rceil+N$. Thus, given the digit size d, we need $\lceil bw_{cx}/d\rceil$ cascaded D flip-flops (actually, a shift register) in d layers (a total of $d\times\lceil bw_{cx}/d\rceil$ D flip-flops) to store the digit-serial output produced in each clock cycle. As an example, suppose that bw_{cx} is 8, *i.e.*, cx is $(cx_7,cx_6,\ldots,cx_0)$. Fig. 5(a,b) illustrate the required circuits to store the d-bits of the output of the operation generating cx when d is 2 and 3, respectively.

Also, the MCM operation generally includes constants with different bit-widths. Hence, a control logic is needed to store the constant multiplications accurately. This process can be controlled using one counter and some constant comparators. In this case, the counter is a $\lceil \log_2 L_{MCM}\rceil$-bit counter, which increments by 1 in each clock cycle. Note that L_{MCM} is the latency of the MCM operation, determined as given in Eq. (6). Thus, the storage of the constant multiplication cx can be controlled by shifting the outputs of the operation implementing cx into the related storage block if $\lceil bw_{cx}/d\rceil$ is less than or equal to the value of the counter. Otherwise, the outputs of the operation are not shifted. Thus, at the end of the maximum clock cycle determined by L_{MCM}, the outputs of the constant multiplications are available at the outputs of the D flip-flops as illustrated in Fig. 5. Although positive and odd constants are realized in the MCM design, whenever the even or negative version of a constant multiplication is required, this output is obtained by shifting the related constant multiplication or taking its 2's complement.

3. Related work

For the MCM problem, the exact CSE algorithms were introduced in [18,19]. In these algorithms, initially the target constants are defined under a number representation and all possible implementations of constants are extracted from its representation. Then, the MCM problem is defined as a 0–1 ILP problem and a solution is obtained using a generic 0–1 ILP solver. The problem reduction and model simplification techniques, that significantly reduce the 0–1 ILP problem size and, consequently, the run-time of the 0–1 ILP solver, were introduced in [8,20]. Prominent CSE heuristics were also presented in [6,7,21].

The first GB methods for the MCM problem, 'add-only', 'add/subtract', 'add/shift', and 'add/subtract/shift', were proposed in [22]. The 'add/subtract/shift' algorithm was modified in [9], 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 [23]. In [9], the RAG-n algorithm that includes two parts, optimal and heuristic, was also introduced. In its optimal part, each target constant that can be implemented with a single operation is synthesized. If there exist unimplemented elements left in the target set, the algorithm switches to its heuristic part. In this iterative part of the algorithm, RAG-n initially chooses a single unimplemented target constant with the smallest single coefficient cost evaluated by the algorithm of [23] 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 or may not be shared for the implementation of not-yet synthesized target constants in later iterations, thus yielding a local minimum solution. The GB heuristic of [10], called Hcub, includes the same optimal part of RAG-n, but uses a better heuristic that considers the impact of each possible intermediate constant on the not-yet synthesized target constants. For the implementation of a single target constant, in each iteration of Hcub, a single intermediate constant that yields the best cumulative benefit over all unimplemented target constants is chosen. The approximate algorithm of [11], which includes the same optimal part as RAG-n and Hcub, computes all possible intermediate constants that can be synthesized with the current set of implemented constants using a single operation and chooses a single intermediate constant that synthesizes the largest number of target constants in each iteration of its heuristic part. The exact GB algorithms that search the minimum number of operations solution in breadth-first and depth-first manners were introduced in [11].

Although there exist many efficient algorithms designed for the MCM problem, there are only a few algorithms designed for the MCM-DS problem. The GB algorithms, called RSAG-n [16] and RASG-n [17], are based on the RAG-n algorithm designed for the MCM problem. In each iteration of RSAG-n, the intermediate constant(s) that require the minimum number of shifts, are chosen. On the other hand, RASG-n selects the intermediate constant(s) with the minimum cost value as done in RAG-n but, if there are more than one possible intermediate constant, it favors the one that requires the minimum number of shifts.

For the optimization gate-level area problem in digit-serial MCM operation, to the best of our knowledge, there are only the exact CSE [24] and approximate GB [14] algorithms. The exact CSE algorithm [24] formalizes this problem as a 0-1 ILP problem when constants are defined under a number representation. The GB algorithm of [14] is described briefly in Section 5.

4. The HCUB+ILP-DS algorithm

The HCUB+ILP-DS algorithm consists of two main parts. In the first part, a solution with the fewest number of operations, that generates MCM, is found by the Hcub algorithm [10]. In the second part, the gate-level area optimization problem in digit-serial MCM operation is formalized as a 0–1 ILP problem and a set of operations that yields a digit-serial MCM design with optimal area at gate-level, is obtained. In the first part of the algorithm, 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 [10]. Also, 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. However, while finding a solution with the fewest number of operations, Hcub does not consider the gate-level cost of each operation under the digit-serial architecture. Hence, the *ILP* technique used in the second part of the algorithm initially generates all possible implementations of constants based on the solution of Hcub and then, finds a set of operations that leads to optimal area by taking into account the implementation of each operation under the digit-serial architecture. The ILP formalization is based on the optimization model described in [24], but it is applied on the solution of Hcub in this case.

4.1. Implementation of the HCUB+ILP-DS algorithm

In the preprocessing phase of HCUB+ILP-DS, 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-DS algorithm, whose pseudo-code is given in Fig. 6, takes these parameters and also the digit size *d* as an input.

In the iterative loop of HCUB+ILP-DS (lines 3–16), we initially find a set of operations, *O*, that implements the constant multiplications using Hcub [10] with a different seed at each time (line 5). If the solution of Hcub includes an *A-operation* with a right shift greater than 0, then we apply the *Reconfigure* function, which replaces these operations by those including a right shift equal to zero (lines 6–7). These operations are found using a technique similar to the RAG-n algorithm [9]. Then, we determine the intermediate and target constants in *O* and store them in a set

```
HCUB+ILP-DS(T, bw, d)
1:  $seed = 0, R_{set} \leftarrow \{\}$
2:  $icost_b = \infty, O_b \leftarrow \{\}$
3:  repeat
4:      $seed = seed + 1$
5:      $O = \text{Hcub}(T, seed)$
6:      if there exist A-operations including a right shift greater
        than 0 then
7:          $O = \text{Reconfigure}(O)$
8:      $R = \text{GenerateReadySet}(O)$
9:      $R = \text{AddDepth1Constants}(R, bw)$
10:     if $R \notin R_{set}$ then
11:         $R_{set} \leftarrow R_{set} \cup R$
12:         $O = \text{ILP}(T, R, bw, d)$
13:         $icost = \text{ComputeImplementationCost}(O, d)$
14:         if $icost < icost_b$ then
15:             $icost_b = icost, O_b \leftarrow O$
16: until Termination conditions meet
17: return $O_b$
```

Fig. 6. The HCUB+ILP-DS algorithm.

called ready set, R (line 8). In order to increase the number of possible implementations of a constant in the *ILP* function, described in Section 4.2, we add the depth-1 constants to R if they do not exist (line 9). The depth-1 constants are in the form of $2^{i+1}-1$ and 2^i+1, where i ranges in between 1 and bw, which can be realized using a single *A-operation* whose inputs are 1. To avoid unnecessary computations, we check if R is already included in R_{set} which is a set that stores all the ready sets given to the *ILP* function (line 10). Then, the *ILP* function is applied on this ready set R to find a set of *A-operations* that yields a digit-serial MCM design with optimal area (line 12). After a solution is obtained by the *ILP* function, the implementation cost of the digit-serial MCM design, *icost*, is computed as if *A-operations* were designed using digit-serial operations, as described in Section 2.3 (line 13). If its cost value is smaller than the best one ($icost_b$) found so far, then O_b and $icost_b$ are updated (lines 14–15). 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 16).

4.2. Implementation of the ILP technique

The *ILP* function used in HCUB+ILP-DS consists of four main parts: (i) generation of constant implementations; (ii) construction of the Boolean network that represents the implementations of constants; (iii) formalization of the problem as a 0–1 ILP problem; (iv) obtaining the minimum solution. These parts are described in detail next.

4.2.1. Generation of constant implementations

After the set of operations (O) realizing the MCM instance is found by Hcub, in the *GenerateReadySet* function, we also compute the depth (adder-step [8]) of each intermediate and target constant of R in the network of addition and subtraction operations of O.[3] After the depth-1 constants are included into R, we sort the constants in R according to their depth values in ascending order. The part of the algorithm, where the implementations of constants are found, is given as follows:

1. Take an element from R, r_i, except ‘1’ that denotes the variable which the constants are multiplied with. Form an empty set, I_i, associated with r_i that will include the inputs and the amount of left shifts at the inputs of each *A-operation* which computes r_i.
2. For each *A-operation* that generates r_i,
   (a) Make each of its inputs positive and odd, and determine the amount of left shifts of the inputs.
   (b) Add its positive and odd inputs and their amount of left shifts to the set I_i.
3. Repeat step 1 until all elements of R are considered.

To find all possible *A-operations* that implement an element of R, r_i, we assign r_i to the output of an *A*-operation given in Eq. (3). Then, for each constant r_j, where $0 \le j \le i-1$, we assign r_j to the input u of the *A-operation* and by changing the left shifts l_1 and l_2 (the right shift r is set to 0), and the sign s values, we compute the input v of the *A-operation*. Note that left shifts are restricted to $bw+1$ and only one of them is set to a value greater than 0 and the other is set to 0, since only positive and odd constants are considered. If v is an element of R, r_k, where $0 \le k \le i-1$, 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 [25]. 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 digit-serial MCM design will always have less or equal area as the digit-serial MCM design obtained by Hcub.

As a simple example, consider one of the solutions of Hcub on a single constant 21 which requires two operations, *i.e*, $17=1 \ll 4+1$ and $21=1 \ll 2+17$. After the depth-1 constants are added to R, the ready set is formed as R={1, 3, 5, 7, 9, 15, 17, 31, 33, 63, 21}. Among many others, five implementations of the constant 21 including also the solution of Hcub can be given as $21=1 \ll 2+17$, $21=1 \ll 4+5$, $21=7 \ll 1+7$, $21=3 \ll 1+15$, and $21=3 \ll 3-3$. We note that each of these operations yields different implementation cost when they are realized under the digit-serial architecture.

4.2.2. Construction of the boolean network

After all possible implementations of constants in R are found, these implementations are represented in a Boolean network that includes only AND and OR gates. 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 two inputs.
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 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 input pair of an *A-operation* in I_i, generate a two-input AND gate. The inputs of the AND gate are the elements of the input pair, *i.e.*, ‘1’ or the outputs of the OR gates representing the target and intermediate constants 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.

The network generated for the target constant 21, which shows only its five realizations listed previously, is given in Fig. 7. In this figure, 1-input OR gates for the intermediate constants 5, 7, 15, and 17 are omitted and the type of each operation is shown inside of each AND gate.

4.2.3. The 0–1 ILP formalization

We need to include optimization variables into the network, so that the gate-level area optimization problem in digit-serial MCM design can be easily formed as a 0–1 ILP problem. To do this, we associate the optimization variables with two parameters that have different implementation costs at gate-level, *i.e.*, addition/subtraction operations and left shifts of constants.

For each AND gate that represents an addition/subtraction operation in the network, we introduce an optimization variable associated with the operation, *i.e.*, $opt_{a \pm b}$, where a and b denote the inputs of an operation, and we add this variable to the input of

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

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

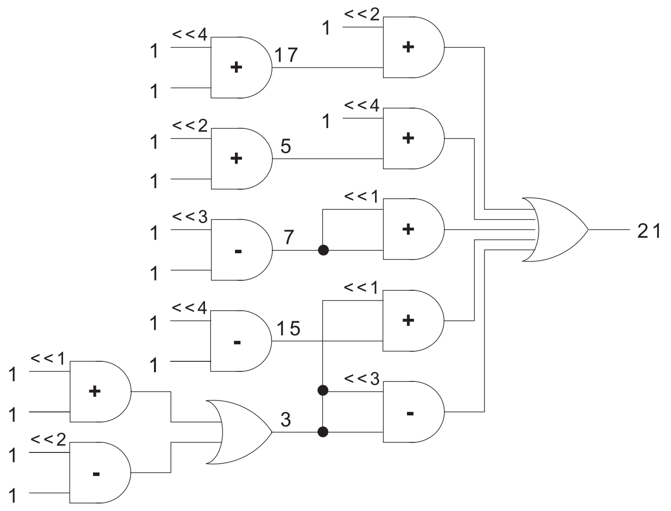

Fig. 7. Representation of limited implementations of 21 in a Boolean network.

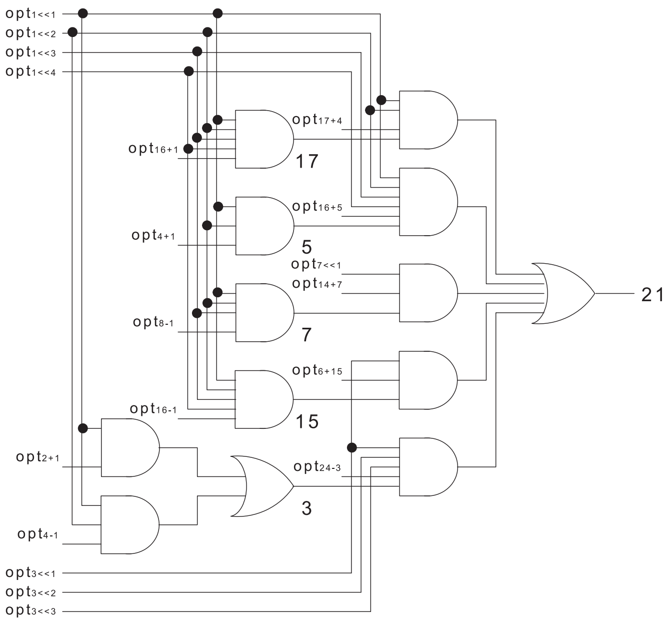

Fig. 8. Inclusion of optimization variables into the network constructed for the target constant 21 given in Fig. 7.

the AND gate. In the cost function to be minimized, the cost value of this type of optimization variable is determined as the implementation cost of the digit-serial operation at gate-level considering its type (addition or subtraction) and the digit size (d), as described in Section 2.3.

In order to maximize the sharing of left shifts, *i.e.*, the D flip-flops at gate-level, for each constant c in the network, we initially find the maximum amount of left shift that the constant c has, *i.e.*, mls_c. Then, for each constant c with mls_c greater than zero, we introduce mls_c optimization variables representing left shifts of c from 1 to mls_c, *i.e.*, $opt_{c \ll 1}, opt_{c \ll 2}, \ldots, opt_{c \ll mls_c}$. In the cost function to be minimized, the cost value of this type of optimization variable is determined as the implementation cost of one D flip-flop, as described in Section 2.3. The inclusion of these optimization variables into the network is done for each AND gate in the network representing an addition/subtraction operation. If an input signal of an operation c' (denoting the input variable, an intermediate constant, or a target constant) is shifted by $ls > 0$ times, then to the related AND gate, we include ls additional inputs standing for the optimization variables associated with the ls left shift of the input signal c', *i.e.*, $opt_{c' \ll 1}, opt_{c' \ll 2}, \ldots, opt_{c' \ll ls}$.

Some simplifications in the network can be also achieved. Note that the variable denoted by 1, which represents the input variable x, can be eliminated from the inputs of the AND gates, because its logic value is always one (*i.e.*, it is always available). Thus, Fig. 8 illustrates the network generated for the target constant 21 after the simplifications are done and the optimization variables are added.

After the optimization variables are added into the network, the generation of the 0–1 ILP problem is straightforward. The cost function of the 0–1 ILP problem is constructed as a linear function of optimization variables, where the cost value of each optimization variable is determined as described before. Also, the constraints of the 0–1 ILP problem are obtained by finding the Conjunctive Normal Form (CNF) formulas of each gate in the network and expressing each clause of the CNF formulas as a linear inequality, as described in [26]. For example, a 3-input AND gate, $d = a \wedge b \wedge c$, is translated to CNF as $(a+\overline{d})(b+\overline{d})(c+\overline{d})(\overline{a}+\overline{b}+\overline{c}+d)$ and converted to linear constraints as $a-d \geq 0$, $b-d \geq 0$, $c-d \geq 0$, $-a-b-c+d \geq -2$. The outputs of the network, *i.e.*, the outputs of OR gates associated with the target constants, are set to 1, since the implementation of target constants is aimed. Thus, the generated model can serve as an input to a generic 0–1 ILP solver.

4.2.4. *Finding the minimum solution*

A generic 0–1 ILP solver will search the minimum value of the cost function on the generated 0–1 ILP problem by satisfying the constraints that represent how the target and intermediate constants are implemented. The set of operations, which yields the minimum area solution, consists of the addition/subtraction operations whose optimization variables are set to 1 by the 0–1 ILP solver.

5. The MINAS-DS algorithm

As opposed to HCUB+ILP-DS that iteratively finds a local minimum solution of the MCM problem and searches a digit-serial MCM design with optimal area around this local minimum solution, MINAS-DS searches the "best" intermediate constants that lead to a digit-serial 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-DS 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 optimal area in digit-serial MCM design.

The preprocessing phase of the MINAS-DS algorithm is the same as that of the HCUB+ILP-DS algorithm. The main part of the algorithm and its routines are given in Figs. 9 and 10, respectively. In MINAS-DS, 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 presented in Fig. 10. Note that as done in the algorithms of [16,17], the right shift of an *A-operation* is always assumed to be zero in MINAS-DS. If there exist unimplemented constants in the target set, then in its iterative loop (lines 3–13 of Fig. 9), an intermediate constant is added to the ready set until there is no element left in the target set. The MINAS-DS algorithm considers the positive and odd constants that are not included in the current ready and target sets (lines 4 and 5) and that can be implemented with the elements of the current ready set using a single operation

```
MINAS-DS(T, bw, d)
1:  $R \leftarrow \{1\}$
2:  $(R, T) = \text{Synthesize}(R, T)$
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:        $impcost_j = \text{ComputeCost}(\{j\}, R, d)$
7:        if $impcost_j \neq 0$ then
8:          $A \leftarrow R \cup \{j\}$
9:          $impcost_T = \text{ComputeTCost}(T, A, d)$
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:   $(R, T) = \text{Synthesize}(R, T)$
14: $D = \text{SynthesizeMinArea}(R, d)$
15: return $D$
```

Fig. 9. Main part of the MINAS-DS algorithm.

(lines 6 and 7) as possible intermediate constants. In MINAS-DS, the *ComputeCost* function (line 6) searches all *A-operations* that compute the constant with the elements of the current ready set. If the implementations of the constant are found, it determines the cost of each operation under the digit-serial architecture (as described in Section 2.3) and returns its minimum implementation cost among possible operations. Otherwise, it returns a 0 value indicating that the constant cannot be synthesized using an operation with the elements of the current ready set. After a 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 and 9). In this function, similar to *ComputeCost*, the minimum implementation costs of the target constants that can be synthesized with the elements of the working ready set *A*, are determined under the digit-serial architecture. 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 it requires a digit-serial addition operation with digit size *d* and $\lceil log_2 t_k \rceil$ D flip-flops for the left shifts. 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 one 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 (lines 11–13).

When there are no elements left in the target set, the *SynthesizeMinArea* function (line 14) is applied on the final ready set to find the set of *A-operations* that yields a digit-serial MCM design with optimal area. Note that in each iteration of MINAS-DS, 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. Hence, we formalize this problem as a 0–1 ILP problem, similar to the formalization described in Section 4.2. In this case, the possible implementations of the constants are found by the *GenerateImp* function given in Fig. 10.

```
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:        $isadded = 1$
6:        $R \leftarrow R \cup \{t_k\}$
7:        $T \leftarrow T \setminus \{t_k\}$
8:  until $isadded = 0$
9:  return $(R, T)$

ComputeCost({c}, C, d)
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 under the digit-
      serial architecture, compute the minimum implementa-
      tion cost of constant $c$, and assign it to $cost_c$
4:  return $cost_c$

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

SynthesizeMinArea(R, d)
1:  Find all possible implementations of target and intermedi-
    ate 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
    under the digit-serial architecture
4:  return $D$

GenerateImp(R, d)
1:  $A \leftarrow \{1\}$, $R \leftarrow R \setminus \{1\}$, $O \leftarrow \{\}$
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 under the digit-serial architecture and store
          them in $O$
7:        $A \leftarrow A \cup \{r_k\}$
8:        $R \leftarrow R \setminus \{r_k\}$
9:  until $R = \emptyset$
10: return $O$
```

Fig. 10. Routines of the MINAS-DS algorithm.

6. Experimental results

This section presents the results of the HCUB+ILP-DS and MINAS-DS algorithms at both high-level and gate-level and compare them with those of prominent algorithms designed for the MCM problem and the optimization of gate-level area problem in digit-serial MCM operation. This section is divided into two parts. In the first part, we introduce the results of high-level algorithms on randomly generated instances and on the multiplier blocks of FIR filters. In the second part, we present the gate-level results of digit-serial MCM and FIR filter designs that are implemented based on the solutions of high-level algorithms and we compare them with those realized using generic digit-serial constant multipliers [15].

6.1. *Results on high-level implementations*

As the first experiment set, we used sets of instances that include a number of constants, *n*, ranging from 10 to 100, where each set includes 30 instances and the constants are 12-bit randomly generated numbers. Table 1 presents the high-level results of the algorithms [8,10,11] designed for the MCM problem and the exact CSE algorithm of [24], HCUB+ILP-DS, and MINAS-DS designed for the optimization of gate-level area problem in digit-serial MCM operation. In this experiment, the digit size *d* was taken as 1. In CSE algorithms [8,24], the constants were defined under MSD representation. In this table, *op* and *sh* stand for the average number of operations and shifts, respectively. Also, *hc* (in mm^2) denotes the average implementation cost of bit-serial MCM designs obtained by the high-level algorithms. In the computation of *hc*, the cost of an FA, a D flip-flop, and an inverter was taken as 90, 52, and 6, respectively, the area (in $\mu\mathrm{m}^2$) of these components in UMCLogic 0.18 μm Generic II design library.

Observe from Table 1 that the algorithms designed for the MCM problem find solutions with fewer number of operations than those designed for the optimization of gate-level area problem in digit-serial MCM operation. However, their solutions yield bit-serial MCM designs that occupy larger area than those obtained by the algorithms designed for the optimization of gate-level area problem in digit-serial MCM operation. This is simply because the algorithms designed for the MCM problem do not consider the sharing of shifts that require D flip-flops in digit-serial arithmetic. Note that while the average number of operations on solutions found by the exact GB algorithm [11] and MINAS-DS is the same on instances where *n* is larger than 20, the reduction on the average number of shift operations on solutions obtained by these algorithms reaches up to 77% when *n* is 100. A similar observation is also valid between Hcub and HCUB+ILP-DS. We also note that both exact CSE algorithms [8,24] obtain worse solutions than the GB algorithms under their optimization objectives. This is simply due to the smaller search space of a CSE algorithm, which is restricted by a particular number representation.

As the second experiment set, we again used randomly generated instances whose specifications are similar to the one given in the first experiment set, except 16-bit constants are used in this case. Table 2 presents the high-level results of only GB algorithms, since the exact CSE algorithm of [24] finds these instances hard to handle. This table introduces the results of the RAG-n [9] algorithm designed for the MCM problem and the RAG-N+ILP-DS algorithm, designed for the optimization of gate-level area problem in digit-serial MCM operation. The RAG-N+ILP-DS algorithm is similar to HCUB+ILP-DS given in Fig. 6, except RAG-n is used to obtain a solution on an MCM instance instead of Hcub on line 5 of Fig. 6. In the algorithms given in Table 2, *d* was taken as 1.

As can be observed from Table 2, RAG-N+ILP-DS and HCUB+ILP-DS significantly improve the solution of RAG-n and Hcub, respectively for the digit-serial MCM design. The maximum and minimum average area improvements between RAG-n and RAG-N+ILP-DS are computed as 25.4% and 13.7% when *n* is 20 and 90, respectively. These maximum and minimum values between Hcub and HCUB+ILP-DS are 25.4% and 14.4% when *n* is 10 and 70, respectively.

Table 1
Summary of results of high-level algorithms on randomly generated 12-bit constants when *d* is 1.

	Optimization of the number of operations									Optimization of area in digit-serial MCM design								
	Exact CSE [8]			Hcub [10]			Exact GB [11]			Exact CSE [24]			HCUB+ILP-DS			MINAS-DS		
n	op	sh	hc	op	sh	hc	op	sh	hc	op	sh	hc	op	sh	hc	op	sh	hc
10	15.5	28.6	3.7	13.9	26.9	3.4	12.8	25.0	3.1	17.1	15.7	3.2	13.7	17.7	2.8	13.1	18.9	2.8
20	26.5	42.5	6.0	22.5	35.3	5.0	21.4	33.2	4.8	29.2	19.7	5.2	22.7	23.3	4.4	21.5	26.5	4.4
30	36.7	53.3	8.0	30.4	39.4	6.4	30.1	39.3	6.4	39.9	23.1	6.9	31.0	25.2	5.7	30.1	28.8	5.8
40	46.0	60.6	9.8	39.4	47.5	8.1	39.4	47.9	8.1	50.2	24.9	8.5	40.2	24.8	7.0	39.4	28.5	7.1
50	55.9	68.7	11.6	49.0	52.9	9.8	49.0	53.2	9.8	59.8	27.6	10.0	49.5	23.2	8.3	49.0	26.5	8.4
60	65.4	77.0	13.4	59.0	58.6	11.5	59.0	58.7	11.5	69.8	29.3	11.5	59.1	21.9	9.5	59.0	23.9	9.6
70	75.1	83.3	15.2	68.2	60.7	13.0	68.2	61.8	13.1	79.5	31.0	13.0	68.4	21.0	10.8	68.2	23.1	10.9
80	83.2	89.5	16.7	77.7	73.9	15.1	77.7	74.5	15.1	87.6	31.3	14.2	77.7	19.0	12.0	77.7	20.7	12.2
90	92.7	99.9	18.6	86.8	73.9	16.4	86.8	73.8	16.4	97.2	32.0	15.6	86.9	19.1	13.4	86.8	20.9	13.5
100	101.7	104.7	20.2	96.5	83.9	18.3	96.5	86.0	18.4	106.4	32.4	17.0	96.5	18.0	14.7	96.5	19.8	14.8

Table 2
Summary of results of high-level algorithms on randomly generated 16-bit constants when *d* is 1.

	Optimization of the number of operations						Optimization of area in digit-serial MCM design								
	RAG-n [9]			Hcub [10]			RAG-N+ILP-DS			HCUB+ILP-DS			MINAS-DS		
n	op	sh	hc	op	sh	hc	op	sh	hc	op	sh	hc	op	sh	hc
10	28.7	87.3	8.6	22.2	53.4	5.9	25.7	65.0	7.0	19.9	36.7	4.7	19.7	51.2	5.4
20	48.3	133.7	13.9	35.2	74.1	8.9	42.8	95.4	11.1	32.8	52.5	7.4	32.8	66.0	8.1
30	62.1	162.8	17.4	47.5	96.9	11.9	55.2	117.8	14.0	44.8	65.8	9.9	43.4	79.2	10.4
40	74.2	184.9	20.3	59.0	111.5	14.3	67.4	131.7	16.5	55.8	78.4	12.1	53.9	91.0	12.5
50	82.9	189.5	21.8	70.0	124.3	16.6	77.0	139.5	18.3	66.1	87.0	14.1	64.0	98.4	14.4
60	93.5	205.0	24.2	79.2	134.8	18.5	87.4	151.0	20.4	76.4	96.5	16.0	73.7	108.9	16.3
70	100.3	201.3	25.0	88.1	147.5	20.4	95.1	147.8	21.3	86.3	103.5	17.8	82.4	120.0	18.1
80	106.4	206.7	26.1	98.3	165.5	22.8	103.6	151.1	22.8	95.4	114.2	19.7	91.1	124.6	19.6
90	114.5	206.7	27.3	107.5	173.9	24.6	112.5	150.1	24.0	104.8	119.3	21.3	99.8	132.6	21.3
100	123.7	219.2	29.3	117.2	190.6	26.9	121.3	155.9	25.6	114.7	129.6	23.3	109.3	142.5	23.2

Table 3
FIR filter specifications.

Filter	pass	stop	# tap	width
1	0.10	0.15	200	16
2	0.10	0.15	240	16
3	0.10	0.25	180	16
4	0.10	0.25	200	16
5	0.10	0.20	240	16
6	0.10	0.20	300	16
7	0.15	0.25	200	16
8	0.15	0.25	240	16
9	0.20	0.25	240	16
10	0.20	0.25	300	16

This is simply due to the *ILP* technique used in the algorithms of RAG-N+ILP-DS and HCUB+ILP-DS. Also, the solution of an MCM algorithm used in the approach of Fig. 6, *i.e.*, RAG-n or Hcub in this case, has a significant impact on the implementation cost of the bit-serial MCM design. While RAG-n finds a solution with larger number of operations when compared to Hcub on these instances, the solutions of RAG-N+ILP-DS also include larger number of bit-serial addition, subtraction, and left shift operations than HCUB+ILP-DS, increasing the complexity of the bit-serial MCM design. Observe from Tables 1 and 2 that HCUB+ILP-DS obtains the best solutions in terms of the implementation cost on most of the instances. Note that MINAS-DS finds solutions with less number of addition and subtraction operations and its solutions generally include greater number of left shift operations than HCUB+ILP-DS, yielding bit-serial MCM designs with larger implementation cost. However, as the number of constants (n) and the bit-width of the constants increase, *e.g*, when n is greater than 70 in Table 2, MINAS-DS finds significantly better solutions than HCUB+ILP-DS in terms of the number of addition and subtraction operations that makes its results on the implementation cost competitive with HCUB+ILP-DS.

As the third experiment set, we used the FIR filters[4] given in Table 3 where coefficients were computed with the *remez* algorithm in MATLAB. In this table, *pass* and *stop* are normalized frequencies that define the passband and stopband, respectively, *#tap* is the number of coefficients, and *width* is the bit-width of the filter coefficients.

The high-level results of algorithms on the multiplier blocks of these FIR filters when d is equal to 1 are presented in Table 4. In this table, *mul* stands for the number of required bit-serial constant multipliers, which is actually the number of unrepeated positive and odd filter coefficients. Also, *cpu* denotes the required CPU time in seconds for the high-level algorithms to find a solution on a PC with Intel Xeon at 2.33 GHz and 4 GB of memory running Linux. We note that although HCUB+ILP-DS and MINAS-DS were written in MATLAB, HCUB+ILP-DS uses Hcub written in C++ and both algorithms use SCIP 2.0 [27] as a 0–1 ILP solver, which is also implemented in C++.

As can be observed from Table 4, although HCUB+ILP-DS and MINAS-DS obtain similar results in terms of implementation cost, there are instances that MINAS-DS find better solutions than HCUB+ILP-DS, *e.g.*, Filters 1, 2, 6, and 8. Moreover, both HCUB+ILP-DS and MINAS-DS obtain better solutions than the algorithms designed for the MCM problem. However, their solutions require higher computational effort than these algorithms. For HCUB+ILP-DS, this depends heavily on the performance of the 0-1 ILP solver and on how many different MCM solutions that the algorithm considers. We note that the run-time of the SCIP 2.0 solver on the 0–1 ILP problems generated by *ILP* function took maximum 23.17 s and 4.86 s on average, on overall instances. For MINAS-DS, it depends on the number of intermediate constants considered in its loop given on lines 3–13 of Fig. 9. Hence, there are instances that MINAS-DS can find a solution in less run-time than HCUB+ILP-DS, *e.g.*, Filters 4, 7, 9, and 10. Note also that while HCUB+ILP-DS and MINAS-DS have to consider all possible implementations of constants in order to find the minimum gate-level cost of each operation under the digit-serial architecture, in the algorithms designed for the MCM problem, finding only one *A-operation* is enough to implement a constant.

6.2. Results on gate-level implementations

Table 5 presents the gate-level results of the bit-serial MCM designs obtained by the high-level algorithms in Table 4, which were obtained using the Synopsys Design Compiler with UMCLogic 0.18 μm Generic II library. In this table, A, D, and P denote, respectively, the area in μm^2, the delay of the critical path in ns, and the total dynamic power dissipation in nW. This table also presents the gate-level results of each MCM block designed using bit-serial constant multipliers, which is adapted from the sequential multiplier described in [15]. Note that the bit-width of the filter input (N) was taken as 16 and the necessary hardware to convert the bit-serial outputs to parallel, as described in Section 2.4, was also included in the MCM designs for both design architectures. During the technology mapping, the bit-serial MCM operations were synthesized under the minimum area design strategy without a constraint on the clock frequency.

Observe from Tables 4 and 5 that although the bit-serial MCM designs are synthesized using the Synopsys Design Compiler, which includes advanced optimization algorithms, there is a high correlation between the implementation costs (hc) at high-level and the gate-level area (A) results. As expected from high-level results, the solutions of HCUB+ILP-DS and MINAS-DS yield less complex bit-serial MCM designs when compared to those obtained by the algorithms designed for the MCM problem. Although the delay in the critical path is increased, power dissipation is decreased slightly in bit-serial MCM designs obtained by HCUB+ILP-DS and MINAS-DS. Also, the design of bit-serial MCM operations under the shift-adds architecture lead to significant area improvement with respect to those realized with bit-serial constant multipliers [15].

Among the filters in Table 3, we selected Filter 4 to further analyze our algorithms since the multiplier block of this filter requires the largest number of addition and subtraction operations as shown in Table 4. Table 6 presents the gate-level results on its multiplier block designed based on the solutions of HCUB+ILP-DS and MINAS-DS when d is 1, 2, 4, and 8. We note that N was again taken as 16 and the maximum bit-width of filter coefficients (bw) is 16. Thus, according to Eq. (6), the number of clock cycles required to obtain the results of all constant multiplications is 32, 16, 8, and 4 when d is equal to 1, 2, 4, and 8, respectively. Hence, in this table, L denotes the latency ($\#clockcycles \times D$) and E stands for the consumed energy ($\#clockcycles \times D \times P$) in fJ (10^{-15} W s) in the MCM design while obtaining the results of all constant multiplications. Also, *ALP* and *AEP* stand for the area–latency product and the area–energy product, respectively.

Observe from Table 6 that as the digit size is decreased, the area and delay of the MCM design is also decreased. However, as the digit-size is decreased, the number of clock cycles required to obtain the filter output is increased, that consequently increases the latency and the energy consumed by the design. This enables a designer to explore the optimal tradeoff between area, latency, and consumed energy by changing the digit size parameter (d).

[4] The filters are available at http://algos.inesc-id.pt/multicon

Table 4
Summary of results of algorithms on MCM blocks of the FIR filters in Table 3 for $d=1$.

	Optimization of the number of operations								Optimization of area in digit-serial MCM design								
	Hcub [10]				Exact GB [11]				HCUB+ILP-DS				MINAS-DS				
Fil.	op	sh	hc	cpu	op	sh	hc	cpu	op	sh	hc	cpu	op	sh	hc	cpu	mul
1	80	135	18.5	0.1	79	130	18.2	1.7	81	31	13.2	80.7	79	31	12.9	317.6	78
2	84	128	18.8	0.1	83	126	18.6	1.8	83	36	13.7	169.1	83	32	13.5	758.4	82
3	47	90	11.5	0.1	47	93	11.6	1.5	47	34	8.5	86.5	47	34	8.5	401.7	46
4	87	124	19.0	0.1	87	111	18.3	1.6	87	41	14.6	11.7	87	43	14.7	3.7	87
5	64	123	15.7	0.1	63	115	15.1	1.3	63	27	10.4	74.4	63	26	10.4	549.3	62
6	68	105	15.3	0.1	68	104	15.2	3.8	68	31	11.4	53.9	68	29	11.3	68.4	67
7	59	86	13.0	0.1	59	92	13.3	0.7	59	23	9.6	19.2	59	24	9.7	1.7	59
8	70	94	15.0	0.1	69	99	15.1	1.4	69	28	11.3	92.3	69	23	11.0	657.8	68
9	78	92	16.1	0.1	78	92	16.1	0.8	78	30	12.7	10.1	78	32	12.8	4.6	78
10	81	97	16.8	0.1	81	94	16.7	0.9	81	20	12.6	33.1	81	21	12.7	4.8	81
Tot.	718	1074	159.9	1.0	714	1056	158.2	15.6	716	301	118.0	631.0	714	295	117.6	2768.0	708

Table 5
Summary of gate-level results of MCM blocks of the FIR filters in Table 3 for $d=1$.

	Optimization of the number of operations						Optimization of area in digit-serial MCM design						Bit-serial constant multipliers [15]		
	Hcub [10]			Exact GB [11]			HCUB+ILP-DS			MINAS-DS					
Fil.	*A*	*D*	*P*	*A*	*D*	*P*	*A*	*D*	*P*	*A*	*D*	*P*	*A*	*D*	*P*
1	74 368	4.13	153	74 344	4.13	153	72 459	5.62	150	72 367	5.03	148	99 424	5.13	196
2	77 530	4.13	159	77 280	4.13	161	75 720	5.60	156	75 677	5.47	156	104 347	4.17	205
3	41 699	4.13	89	41 740	4.12	91	40 662	6.11	89	40 642	4.12	87	58 577	4.13	113
4	84 186	4.14	173	83 921	4.14	172	82 716	5.53	169	82 757	4.47	165	112 416	4.14	220
5	59 823	4.13	122	59 432	4.13	123	57 788	4.65	118	57 783	4.98	118	80 114	4.13	153
6	62 775	3.99	130	62 723	3.99	132	61 516	5.34	128	61 492	5.13	127	84 855	4.01	167
7	54 883	4.13	114	54 943	4.13	114	53 694	5.91	111	53 756	4.41	111	74 907	4.12	143
8	62 640	4.13	128	62 488	4.13	130	61 105	4.97	127	61 159	4.94	126	86 180	4.14	167
9	73 175	4.13	146	73 145	4.13	147	71 932	6.75	145	71 986	5.69	145	100 124	7.49	191
10	75 235	4.13	152	75 168	4.13	154	73 650	4.43	149	73 726	4.09	151	103 126	4.61	200
Tot.	666 314	41.16	1366	665 184	41.16	1377	651 242	54.91	1342	651 345	48.33	1334	904 070	46.08	1755

Table 6
Gate-level results of digit-serial multiplier block of Filter 4.

Algorithm	*d*	1	2	4	8
HCUB+ILP-DS	*A* (mm^2)	82.7	87.0	94.0	111.2
	D (ns)	5.53	6.87	7.51	8.52
	L (ns)	176.96	109.92	60.08	34.08
	ALP	14 635	9563	5648	3790
	P (μW)	0.169	0.200	0.238	0.321
	E (fJ)	29.85	21.98	14.27	10.93
	AEP	2469	1913	1341	1215
MINAS-DS	*A* (mm^2)	82.8	87.0	94.1	111.5
	D (ns)	4.47	4.90	5.65	7.33
	L (ns)	143.04	78.40	45.20	29.32
	ALP	11 844	6821	4253	3269
	P (μW)	0.165	0.196	0.235	0.318
	E (fJ)	23.60	15.37	10.62	9.32
	AEP	1954	1337	1000	1040

Moreover, although HCUB+ILP-DS and MINAS-DS obtain similar gate-level area results, the solutions of MINAS-DS yield high-speed and low-power digit-serial MCM designs and have lower *ALP* and *AEP* values when compared to those obtained by the solutions of HCUB+ILP-DS. This is simply due to the lower delay values of the MCM designs obtained by MINAS-DS. Thus, this observation suggests that in order to reduce the latency and consumed energy of digit-serial MCM design, the high-level algorithms should also take into account the delay of the design, which is generally defined as the maximum number of operations in series and commonly known as the number of adder-steps [8].

Table 7 presents the gate-level results of digit-serial designs of Filter 4, which is implemented in its transposed form as given in Fig. 1 under two architectures: shift-adds (*shift-adds*) and generic constant multipliers (*cons. mult.*). When *d* is 1, 2, 4, and 8, the multiplier block of the FIR filter is designed using digit-serial addition, subtraction, and shift operations determined by the solution of MINAS-DS under the *shift-adds* architecture and it is implemented using digit-serial constant multipliers [15] under the *cons. mult.* architecture. For bit-parallel processing (when $d=16$), under the *shift-adds* architecture, the multiplier block is designed using addition and subtraction operations obtained by the solution of the exact GB algorithm [11]. Under the *cons. mult.* architecture, the multiplication of each filter coefficient by the filter input in the multiplier block is initially described as constant multiplications in VHDL and then, the FIR filter with this multiplier block is designed by the logic synthesis tool. We note that the bit-width of the filter output is 35, hence, the number of clock cycles required to obtain the filter output is 35, 18, 9, and 5 when *d* is 1, 2, 4, and 8, respectively. When *d* is 16, the filter output is obtained in 1 clock cycle.

Observe from Table 7, although latency and consumed energy is slightly increased on some cases under the *shift-adds* architecture with respect to *cons. mult.*, the area is always smaller and the maximum area reduction between these architectures reaches up

Table 7
Gate-level results of complete digit-serial Filter 4.

Arch.	d	1	2	4	8	16
Shift-adds	A (mm^2)	201.7	214.8	228.9	281.1	322.9
	D (ns)	5.45	6.16	6.94	7.70	9.90
	L (ns)	190.75	110.88	62.46	38.50	9.90
	ALP	38 474	23 817	14 297	10 822	3197
	P (μW)	0.503	0.593	0.694	0.923	1.06
	E (fJ)	95.95	65.75	43.35	35.54	10.49
	AEP	19 353	14 123	9922	9989	3389
cons.mult.	A (mm^2)	252.0	264.8	269.7	377.9	439.0
	D (ns)	3.97	5.79	6.92	12.00	9.00
	L (ns)	138.95	104.22	62.28	60.00	9.00
	ALP	35 015	27 597	16 797	22 674	15 804
	P (μW)	0.619	0.706	0.779	1.023	1.22
	E (fJ)	86.01	73.58	48.52	61.38	43.92
	AEP	21 675	19 484	13 085	23 196	19 281

to 34.1% when d is equal to 8. Moreover, although the bit-parallel implementation of the FIR filter has the best *ALP* and *AEP* values, its digit-serial realizations yield less complex FIR filters albeit with a cost of an increased latency and energy consumption.

7. Conclusions

This article presented high-level algorithms for the optimization of gate-level area in digit-serial MCM design. The main advantage of these algorithms is that they consider the gate-level cost of each digit-serial addition, subtraction, and shift operation while synthesizing the constant multiplications. The experimental results showed that they yield less complex digit-serial MCM designs when compared to those obtained by prominent algorithms designed for the MCM problem. It was indicated that the use of shift-adds architecture leads to significant area improvement in digit-serial MCM designs with respect to those realized with digit-serial constant multipliers. It was also shown that the realization of digit-serial MCM operations and FIR filters under different digit sizes enables a designer to explore the optimal tradeoff between area and delay and to find a circuit that fits best on a specific application.

Acknowledgment

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

References

[1] C. Wallace, A suggestion for a fast multiplier, IEEE Transactions on Electronic Computers 13 (1) (1964) 14–17.
[2] W. Gallagher, E. Swartzlander, High radix booth multipliers using reduced area adder trees, in: Proceedings of Asilomar Conference on Signals, Systems and Computers, 1994, pp. 545–549.
[3] H. Nguyen, A. Chatterjee, Number-splitting with shift-and-add decomposition for power and hardware optimization in linear DSP synthesis, IEEE Transactions on Very Large Scale Integration Systems 8 (4) (2000) 419–424.
[4] P. Cappello, K. Steiglitz, Some complexity issues in digital signal processing, IEEE Transactions on Acoustics, Speech, and Signal Processing 32 (5) (1984) 1037–1041.
[5] M. Ercegovac, T. Lang, Digital Arithmetic, Morgan Kaufmann, 2003.
[6] R. Hartley, Subexpression sharing in filters using canonic signed digit multipliers, IEEE Transactions on Circuits and Systems II 43 (10) (1996) 677–688.
[7] 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.
[8] 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.
[9] 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.
[10] Y. Voronenko, M. Püschel, Multiplierless multiple constant multiplication, ACM Transactions on Algorithms 3 (2) (2007).
[11] L. Aksoy, E. Gunes, P. Flores, Search algorithms for the multiple constant multiplications problem: exact and approximate, Journal on Microprocessors and Microsystems 34 (5) (2010) 151–162.
[12] 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.
[13] R. Hartley, P. Corbett, Digit-serial processing techniques, IEEE Transactions on Circuits and Systems II 37 (6) (1990) 707–719.
[14] L. Aksoy, C. Lazzari, E. Costa, P. Flores, J. Monteiro, Efficient shift-adds design of digit-serial multiple constant multiplications, in: Proceedings of Great Lakes Symposium on VLSI, 2011, pp. 61–66.
[15] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, 2000.
[16] K. Johansson, O. Gustafsson, A. Dempster, L. Wanhammar, Algorithm to reduce the number of shifts and additions in multiplier blocks using serial arithmetic, in: Proceedings of IEEE Mediterranean Electrotechnical Conference, 2004, pp. 197–200.
[17] K. Johansson, O. Gustafsson, L. Wanhammar, Multiple constant multiplication for digit-serial implementation of low power FIR filters, WSEAS Transactions on Circuits and Systems 5 (7) (2006) 1001–1008.
[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] D. Bull, D. Horrocks, Primitive operator digital filters, IEE Proceedings G: Circuits, Devices and Systems 138 (3) (1991) 401–412.
[23] A. Dempster, M. Macleod, Constant integer multiplication using minimum adders, IEE Proceedings – Circuits, Devices and Systems 141 (5) (1994) 407–413.
[24] L. Aksoy, C. Lazzari, E. Costa, P. Flores, J. Monteiro, Optimization of area in digit-serial multiple constant multiplications at gate-level, in: Proceedings of IEEE International Symposium on Circuits and Systems, 2011, pp. 2737–2740.
[25] O. Gustafsson, Towards optimal constant multiplication: a hypergraph approach, in: Proceedings of Asilomar Conference on Signals, Systems and Computers, 2008, pp. 1805–1809.
[26] P. Barth, A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization, Technical Report, Max-Planck-Institut Fur Informatik, 1995.
[27] 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.

Cristiano Lazzari received the M.S. degree in Computer Science from the Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, in 2003, and the Ph.D. degree in Microelectronics from the Universidade Federal do Rio Grande do Sul (UFRGS) and the Institut National Polytechnique de Grenoble (INPG), France, in 2007. Cristiano Lazzari is a researcher in the Algorithms for Optimization and Simulation (ALGOS) research unit at Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, Portugal. His research interests are in developing techniques for design&test of NoCs and developing algorithms for logic synthesis and technology mapping of multi-valued circuits.

Eduardo 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 a 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 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.