Design of Digit-Serial FIR Filters: Algorithms, Architectures, and a CAD Tool

Levent Aksoy, *Member, IEEE*, Cristiano Lazzari, *Member, IEEE*, Eduardo Costa, *Member, IEEE*, Paulo Flores, *Member, IEEE*, and José Monteiro, *Senior Member, IEEE*

***Abstract*— In the last two decades, many efficient algorithms and architectures have been introduced for the design of low-complexity bit-parallel multiple constant multiplications (MCM) operation which dominates the complexity of many digital signal processing systems. On the other hand, little attention has been given to the digit-serial MCM design that offers alternative low-complexity MCM operations albeit at the cost of an increased delay. In this paper, we address the problem of optimizing the gate-level area in digit-serial MCM designs and introduce high-level synthesis algorithms, design architectures, and a computer-aided design tool. Experimental results show the efficiency of the proposed optimization algorithms and of the digit-serial MCM architectures in the design of digit-serial MCM operations and finite impulse response filters.**

***Index Terms*— 0–1 integer linear programming (ILP), digit-serial arithmetic, finite impulse response (FIR) filters, gate-level area optimization, multiple constant multiplications.**

I. Introduction

FINITE impulse response (FIR) filters are of great importance in digital signal processing (DSP) systems since their characteristics in linear-phase and feed-forward implementations make them very useful for building stable high-performance filters. The direct and transposed-form FIR filter implementations are illustrated in Fig. 1(a) and (b), respectively. Although both architectures have similar complexity in hardware, the transposed form is generally preferred because of its higher performance and power efficiency [1].

The multiplier block of the digital FIR filter in its transposed form [Fig. 1(b)], where the multiplication of filter coefficients with the filter input is realized, has significant impact on the complexity and performance of the design because a large number of constant multiplications are required. This is generally known as the multiple constant multiplications (MCM) operation and is also a central operation and performance bottleneck in many other DSP systems such as fast Fourier transforms, discrete cosine transforms (DCTs), and error-correcting codes.

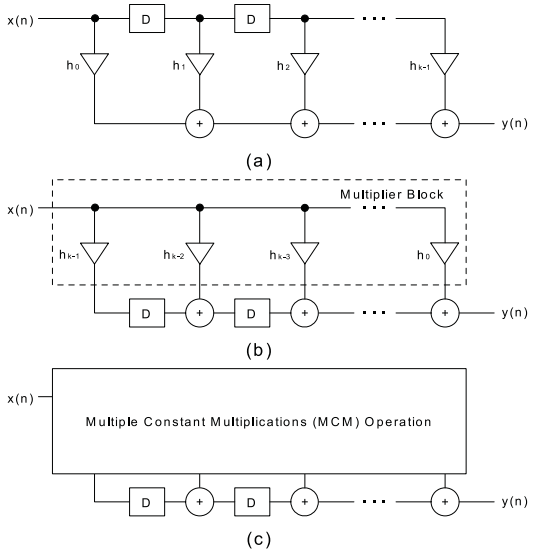


Fig. 1. FIR filter implementations. (a) Direct form. (b) Transposed form with generic multipliers. (c) Transposed form with an MCM block.

Although area-, delay-, and power-efficient multiplier architectures, such as Wallace [2] and modified Booth [3] multipliers, have been proposed, the full flexibility of a multiplier is not necessary for the constant multiplications, since filter coefficients are fixed and determined beforehand by the DSP algorithms [4]. Hence, the multiplication of filter coefficients with the input data is generally implemented under a shift-adds architecture [5], where each constant multiplication is realized using addition/subtraction and shift operations in an MCM operation [Fig. 1(c)].

For the shift-adds implementation of constant multiplications, a straightforward method, generally known as digit-based recoding [6], initially defines the constants 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:

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

which requires six addition operations as illustrated in Fig. 2(a).

Manuscript received May 25, 2011; revised November 17, 2011; accepted February 16, 2012. Date of publication March 16, 2012; date of current version February 20, 2013. This work was supported in part by the Portuguese Foundation for Science and Technology under the research project "Multicon - Architectural Optimization of DSP Systems with Multiple Constants Multiplications" PTDC/EIAEIA/ 103532/2008 and under INESC-ID multiannual funding through the PIDDAC Program funds.

L. Aksoy and C. Lazzari are with INESC-ID, Lisbon 1000-029, Portugal (e-mail: levent@algos.inesc-id.pt; lazzari@algos.inesc-id.pt).

E. Costa is with Universidade Católica de Pelotas, Pelotas 96010-000, Brazil (e-mail: ecosta@ucpel.tche.br).

P. Flores and J. Monteiro are with INESC-ID/IST TU Lisbon, Lisbon 1000-029, Portugal (e-mail: pff@inesc-id.pt; jcm@inesc-id.pt).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2012.2188917

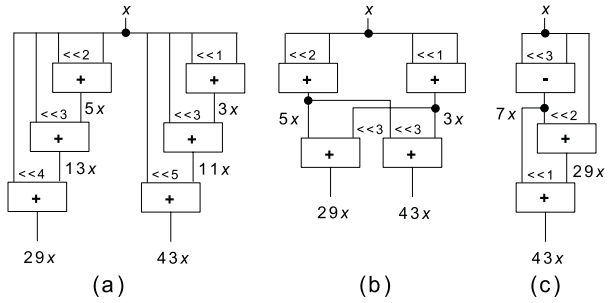


Fig. 2. Shift-adds implementations of $29x$ and $43x$. (a) Without partial product sharing [6] and with partial product sharing. (b) Exact CSE algorithm [9]. (c) Exact GB algorithm [12].

However, the digit-based recoding technique does not exploit the sharing of common partial products, which allows great reductions in the number of operations and, consequently, in area and power dissipation of the MCM design at the gate level. Hence, the fundamental optimization problem, called the MCM problem, is defined as finding the minimum number of addition and subtraction operations that implement the constant multiplications. Note that, in bit-parallel design of constant multiplications, shifts can be realized using only wires in hardware without representing any area cost.

The algorithms designed for the MCM problem can be categorized in two classes: common subexpression elimination (CSE) algorithms [7]–[9] and graph-based (GB) techniques [10]–[12]. The CSE algorithms initially extract all possible subexpressions from the representations of the constants when they are defined under binary, canonical signed digit (CSD) [7], or minimal signed digit (MSD) [8]. Then, they find the "best" subexpression, generally the most common, to be shared among the constant multiplications. The GB methods are not limited to any particular number representation and consider a larger number of alternative implementations of a constant, yielding better solutions than the CSE algorithms, as shown in [11] and [12].

Returning to our example in Fig. 2, the exact CSE algorithm of [9] gives a solution with four operations by finding the most common partial products $3x = (11)_{\text{bin}}x$ and $5x = (101)_{\text{bin}}x$ when constants are defined under binary, as illustrated in Fig. 2(b). On the other hand, the exact GB algorithm [12] finds a solution with the minimum number of operations by sharing the common partial product $7x$ in both multiplications, as shown in Fig. 2(c). Note that the partial product $7x = (111)_{\text{bin}}x$ cannot be extracted from the binary representation of $43x$ in the exact CSE algorithm [9].

However, all these algorithms assume that the input data x is processed in parallel. On the other hand, in digit-serial arithmetic, the data words are divided into digit sets, consisting of d bits, that are processed one at a time [13]. Since digit-serial operators occupy less area and are independent of the data wordlength, digit-serial architectures offer alternative low-complexity designs when compared to bit-parallel architectures. However, the shifts require the use of D flip-flops, as opposed to the bit-parallel MCM design where they are free in terms of hardware. Hence, the high-level algorithms should take into account the sharing of shift operations as well as the sharing of addition/subtraction operations in digit-serial MCM design. Furthermore, finding the minimum number of operations realizing an MCM operation does not always yield an MCM design with optimal area at the gate level [14]. Hence, the high-level algorithms should consider the implementation cost of each digit-serial operation at the gate level.

In this paper, we initially determine the gate-level implementation costs of digit-serial addition, subtraction, and left shift operations used in the shift-adds design of digit-serial MCM operations. Then, we introduce the exact CSE algorithm [15] that formalizes the gate-level area optimization problem as a 0–1 integer linear programming (ILP) problem when constants are defined under a particular number representation. We also present a new optimization model that reduces the 0–1 ILP problem size significantly and, consequently, the runtime of a generic 0–1 ILP solver. Since there are still instances which the exact CSE algorithm cannot handle, we describe the approximate GB algorithm [16] that iteratively finds the "best" partial product which leads to the optimal area in digit-serial MCM design at the gate level. This paper also introduces a computer-aided design (CAD) tool called SAFIR which generates the hardware descriptions of digit-serial MCM operations and FIR filters based on a design architecture and implements these circuits using a commercial logic synthesis tool. In SAFIR, the digit-serial constant multiplications can be implemented under the shift-adds architecture, and also can be designed using generic digit-serial constant multipliers [17].

Experimental results on a comprehensive set of instances show that the solutions of algorithms introduced in this paper lead to significant improvements in area of digit-serial MCM designs compared to those obtained using the algorithms designed for the MCM problem. The digit-serial FIR filter designs obtained by SAFIR also indicate that the realization of the multiplier block of a digit-serial FIR filter under the shift-adds architecture significantly reduces the area of digit-serial FIR filters with respect to those designed using digit-serial constant multipliers [17]. Additionally, it is observed that the optimal tradeoff between area and delay in digit-serial FIR filter designs can be explored by changing the digit size d.

The rest of this paper proceeds as follows. Section II gives the background concepts. The exact CSE and approximate GB algorithms are introduced in Sections III and IV, respectively. Section V describes the CAD tool. The experimental results are presented in Section VI and conclusions in Section VII.

II. Background

This section presents the main concepts related to the proposed algorithms, introduces the problem definitions, and gives an overview on previously proposed algorithms.

A. Number Representation

The *binary* representation decomposes a number in a set of additions of powers of 2. The representation of numbers using a signed digit system makes use of positive and negative digits, $\{1, 0, -1\}$. The CSD representation [18] is a signed

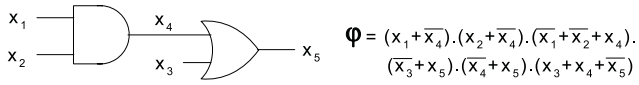


Fig. 3. Combinational circuit and its corresponding CNF formula.

digit system that has a unique representation for each number and verifies the following main properties: 1) two nonzero digits are not adjacent; and 2) the number of nonzero digits is minimum. Any n digit number in CSD has at most $\lceil (n+1)/2 \rceil$ nonzero digits and, on average, the number of nonzero digits is reduced by 33% when compared to binary. The MSD representation [8] is obtained by dropping the first property of the CSD representation. Thus, a constant may have several representations under MSD, including its CSD representation, but all with a minimum number of nonzero digits.

Consider the constant 23 defined in six bits. Its binary representation 010111 includes four nonzero digits. It is represented as $10\overline{1}00\overline{1}$ in CSD, and both $10\overline{1}00\overline{1}$ and $01100\overline{1}$ denote 23 in MSD using three nonzero digits (where $\overline{1}$ stands for -1).

B. Boolean Satisfiability

A Boolean function $\varphi : \{0,1\}^n \rightarrow \{0,1\}$ can be denoted by a *propositional formula*. The *conjunctive normal form* (CNF) is a representation of a propositional formula consisting of a conjunction of propositional clauses where each *clause* is a disjunction of literals and a *literal* l_j is either a variable x_j or its complement $\overline{x_j}$. Note that, if a literal of a clause assumes value 1, then the clause is satisfied. If all literals of a clause assume the value 0, then the clause is unsatisfied. The *satisfiability (SAT) problem* is to find an assignment on n variables of the Boolean formula in CNF that evaluates the formula to 1, or to prove that the formula is equal to the constant 0.

A *combinational circuit* is a directed acyclic graph with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates. Incoming edges of a node are called *fanins* and outgoing edges are called *fanouts*. The *primary inputs* of the network are the nodes without fanins. The *primary outputs* are the nodes without fanouts.

The CNF formula of a combinational circuit is the conjunction of the CNF formulas of each gate, where the CNF formula of each gate denotes the valid input–output assignments to the gate. The derivation of CNF formulas of basic logic gates can be found in [19]. As a simple example, consider the combinational circuit and its CNF formula given in Fig. 3. In this Boolean formula, the first three clauses represent the CNF formula of a two-input AND gate, and the last three clauses denote the CNF formula of a two-input OR gate. Observe from Fig. 3 that the assignment $x_1 = x_3 = x_4 = x_5 = 0$ and $x_2 = 1$ makes the formula φ equal to 1, indicating a valid assignment. However, the assignment $x_1 = x_3 = x_4 = 0$ and $x_2 = x_5 = 1$ makes the last clause of the formula equal to 0 and, consequently, the formula φ, indicating a conflict between the values of the inputs and output of the OR gate.

C. 0–1 ILP

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]

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

$$\text{s.t.} \quad \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}, \qquad \mathbf{x} \in \{0,1\}^n. \tag{2}$$

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 $\mathbf{A} \in \mathbb{Z}^m \times \mathbb{Z}^n$.

A clause $l_1 + \cdots + l_k$, where $k \leq n$, to be satisfied in a CNF formula can be interpreted as a linear inequality $l_1 + \cdots + l_k \geq 1$, where $\overline{x_j}$ is represented by $1 - x_j$, as shown in [20]. These linear inequalities are commonly referred to as CNF constraints, where $a_{ij} \in \{-1, 0, 1\}$ and b_i is equal to 1 minus the total number of complemented variables in its CNF formula. For instance, the set of clauses, $(x_1 + x_2)$, $(\overline{x_2} + x_3)$, and $(\overline{x_1} + \overline{x_3})$, has the equivalent linear inequalities given as $x_1 + x_2 \geq 1$, $-x_2 + x_3 \geq 0$, and $-x_1 - x_3 \geq -1$, respectively.

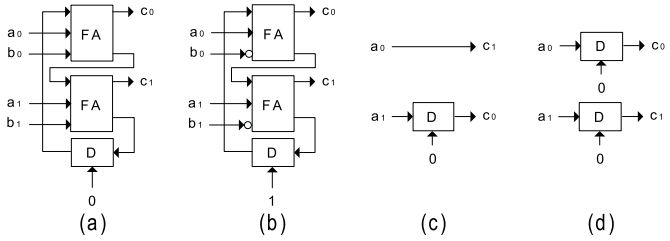


Fig. 4. Digit-serial operations when d is equal to 2. (a) Addition operation. (b) Subtraction operation. (c) Left shift by one time. (d) Left shift by two times.

D. Digit-Serial Arithmetic

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

The basic digit-serial arithmetic operations can be found in [21]. The digit-serial addition, subtraction, and left shift operations are depicted in Fig. 4 using a digit size d equal to 2, where the bits with index 0 and 1 denote the least significant bit (LSB) and the most significant bit (MSB), respectively. Notice from Fig. 4(a) that a digit-serial addition operation, in general, requires d full adders (FAs) and one D flip-flop. The subtraction operation [Fig. 4(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 left shift operation [Fig. 4(c) and (d)], the number of required D flip-flops is equal to the shift amount and is realized in d layers (one for each bit). The input–output correspondence and the number of flip-flops

[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.

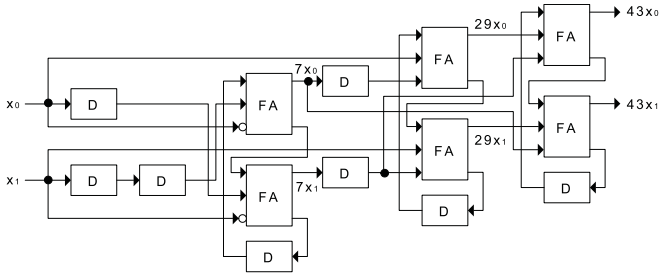


Fig. 5. Digit-serial design of shift-adds implementation of $29x$ and $43x$ given in Fig. 2(c) when d is 2.

cascaded serially for each input at each layer of the digit-serial left shift operation are given in (3) and (4), respectively, where i ranges from 0 to $d-1$ and ls denotes the amount of left shift

$$a_i \Rightarrow c_{(i\ +\ ls)\ mod\ d} \tag{3}$$

$$FF_{a_i} = \begin{cases} \lfloor \frac{ls}{d} \rfloor, & \text{if } i < d - (ls \bmod d) \\ \lceil \frac{ls}{d} \rceil, & \text{otherwise.} \end{cases} \tag{4}$$

As an example of digit-serial realization of constant multiplications under the shift-adds architecture, Fig. 5 presents the digit-serial implementation of $29x$ and $43x$ illustrated in Fig. 2(c) when d is 2. For the sake of clarity, the initializations of D flip-flops are omitted in this figure. As can be easily observed, the network includes two digit-serial additions, one digit-serial subtraction, and five D flip-flops for all the left shift operations. In this network, at each clock cycle, two bits of the input data x (x_1x_0) are applied to the network input, and at the outputs of each digit-serial addition/subtraction operation two bits of a constant multiplication are computed. In general, d bits are processed at each clock cycle.

The digit-serial design of the MCM operation occupies significantly less area when compared to its bit-parallel design since the area of the digit-serial design is not dependent on the bitwidth of the input data. However, the latency is determined in terms of clock cycles as

$$L_{MCM} = \left\lceil \frac{(bw + N)}{d} \right\rceil \tag{5}$$

where N is the bitwidth of the input variable x, bw is the maximum bitwidth of the constants to be implemented, and d is less than N. Thus, (5) does not apply to bit-parallel processing (when $d = N$). Note that in a bit-parallel design, the latency of the MCM computation is only one clock cycle. Returning to our example given in Fig. 5, suppose that x is a 16-bit input value. Thus, to obtain the actual output of $29x$ and $43x$ in the digit-serial network of Fig. 5, we need a total of 11 clock cycles. As a sign extension, $d \times L_{MCM} - N$ bits must be padded to the input data x, which are zeros if x is an unsigned input, or sign bits otherwise.

E. Problem Definitions

For the multiplierless realization of constant multiplications, the fundamental operation, called *A-operation* in [11], is an operation with two integer inputs and one integer output that performs a single addition or a subtraction, and an arbitrary number of shifts. It is defined as follows:

$$w = A(u, v) = |2^{l_1}u + (-1)^s 2^{l_2} v|2^{-r} \tag{6}$$

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

In the MCM problem, it is supposed that the input data is processed in parallel, and hence the shifting operation has no cost in hardware. It is also assumed that the sign of the constant can be adjusted at some part of the design, and the complexity of an adder and a subtracter is equal in hardware. Thus, only positive and odd constants are considered in the MCM problem. Observe from (6) that, in the implementation of an odd constant considering 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. Also, it is necessary to constrain the left shifts l_1 and l_2, otherwise, there exist infinite ways of implementing a constant. In the exact GB algorithm of [12], the number of shifts allowed is at most $bw + 1$. In CSE algorithms, the left shifts are already constrained to the bitwidth of the constant under the given number representation. Thus, the MCM problem [11] can be defined as follows.

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 [11]. Note that the MCM problem is an NP-complete problem [22].

Contrary to the bit-parallel MCM design, as described in Section II-D, shifts require D flip-flops in a digit-serial MCM design. Hence, the problem of optimizing the number of addition, subtraction, and shift operations is defined as follows.

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* include the minimum number of addition, subtraction, and shift operations.

In the digit-serial architecture, it is assumed that an *A-operation* that generates a constant multiplication has always a right shift equal to zero due to the excessive hardware required for the control logic. Note that a constant multiplication is rarely realized by such an *A-operation* in GB algorithms and it never occurs in CSE algorithms.

Note that, while the sharing of addition/subtraction operations reduces the number of required digit-serial addition/subtraction operations, the sharing of shift operations for a constant multiplication also reduces the number of required D flip-flops. Observe from Fig. 5 that the two D flip-flops used 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. Moreover, as described in Section II-D, the implementation costs of digit-serial addition, subtraction, and left shift operations are different at the 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 follows.

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.

F. Related Work

The exact CSE algorithms that formalize the MCM problem as a 0–1 ILP problem were introduced in [23] and [24]. In these algorithms, the target constants are defined under a number representation and all possible implementations of constant multiplications are extracted from the representations of constants. The problem reduction and model simplification techniques for the exact CSE algorithms were presented in [9] and [25]. Prominent CSE heuristics were proposed in [7], [8], and [26].

The exact GB algorithms that search for a solution with the minimum number of operations in breadth-first and depth-first manners were introduced in [12]. Efficient GB algorithms that includes two parts, i.e., optimal and heuristic, were introduced in [10]–[12]. In their optimal parts, each target constant that can be implemented with a single operation is synthesized. If there exist unimplemented elements left in the target set, then they switch to their heuristic parts where the required intermediate constants are found. The RAG-n algorithm [10] initially chooses a single unimplemented target constant with the smallest single coefficient cost evaluated by the algorithm of [27], and then synthesizes it with a single operation including one(two) intermediate constant(s) that has(have) the smallest value in its heuristic part. The Hcub algorithm [11] selects a single intermediate constant that yields the best cumulative benefit over all unimplemented target constants for the implementation of each target constant. The approximate algorithm [12] computes all possible intermediate constants that can be synthesized with the current set of implemented constants using a single operation and chooses the one that leads to the largest number of synthesized target constants.

For the MCM-DS problem, the GB algorithms based on RAG-n were introduced in [28] and [29]. The RSAG-n algorithm [28] chooses the intermediate constant(s) that require the minimum number of shifts. The RASG-n algorithm [29] 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 of gate-level area problem in digit-serial MCM operation, to the best of our knowledge, there are only the exact CSE [15] and approximate GB [16] algorithms, which will be described briefly in the following two sections.

III. Exact CSE Algorithm

The exact CSE algorithm consists of four main steps. First, all possible implementations of constants are extracted from the nonzero digits of the constants defined under a number representation: binary, CSD, or MSD. Then, the implementations of constants are represented in terms of a Boolean network. Third, the gate-level area optimization problem is formalized as a 0–1 ILP problem with a cost function to be minimized and a set of constraints to be satisfied. Finally, a set of operations that yields the minimum area solution is obtained using a generic 0–1 ILP solver. These four steps are described in detail next.

A. Finding the Partial Terms

In the 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 the target set T. Thus, T includes the minimum number of necessary constants to be implemented. The part of the algorithm where the implementations of the target constants and partial terms are found is as follows.

1) Take an element from T, t_i, find its representation(s) under the given number representation, and store it(them) in a set called S. Form an empty set O_i, associated with t_i, that will include the inputs and the amount of left shifts at the inputs of all addition/subtraction operations which generate t_i.
2) For each representation of t_i in the set S.
   a) Compute all nonsymmetric partial term pairs that cover the representation of t_i.
   b) In each pair, make each partial term positive and odd, and determine its amount of left shift.
   c) Add each pair to the set O_i with the amount of left shifts of partial terms.
   d) Add each partial term to T, if it does not represent the input that the constants are multiplied with, i.e., denoted by 1, and is not in T.
3) Repeat Step 1 until all elements of T are considered.

Observe that the target set T only includes the target constants to be implemented in the beginning of the iterative loop, and in later iterations it is augmented with the partial terms that are required for the implementation of target constants. All possible implementations of an element in the target set t_i are found by decomposing the nonzero digits in the representation of, t_i, into two partial terms. As an example, consider 25 as a target constant defined under MSD, which has two representations 011001 and $10\overline{1}001$. All possible implementations of 25 are given in Fig. 6.

Observe from Fig. 6 that the last implementations of 25 on both representations, i.e., $1 + 3 \ll 3$, are identical, therefore one of them can be eliminated. Also, the duplications of implementations, such as $1 \ll 4 + 9 = 9 + 1 \ll 4$, are not listed in Fig. 6. After the partial terms required for the implementation of 25 under MSD, i.e., 3, 7, 9, 17, and 33, are found, they are added to the target set T without repetition and their implementations are determined in a similar way.

B. Construction of the Boolean Network

After all possible implementations of target constants and partial terms are found, they are represented in a network that

$$25 = \begin{cases} 011001 = \begin{cases} 010000 + 001001 = 1 \ll 4 + 9 \\ 001000 + 010001 = 1 \ll 3 + 17 \\ 000001 + 011000 = 1 + 3 \ll 3 \end{cases} \\ 10\bar{1}001 = \begin{cases} 100000 + 00\bar{1}001 = 1 \ll 5 - 7 \\ 00\bar{1}000 + 100001 = -1 \ll 3 + 33 \\ 000001 + 10\bar{1}000 = 1 + 3 \ll 3 \end{cases} \end{cases}$$

Fig. 6. Possible implementations of 25 under MSD representation.

includes only AND and OR gates. Its properties are given as follows.

1) The primary input of the network is the input variable to be multiplied with the constants.
2) An AND gate in the network represents an addition/subtraction operation and has two inputs.
3) An OR gate in the network represents a target constant or a partial term and combines all its possible implementations.
4) The outputs of the network are the OR gate outputs associated with the target constants.

The Boolean network is constructed as follows.

1) Take an element from T, t_i.
2) For each pair in O_i, generate a two-input AND gate. The inputs of the AND gate are the elements of the pair, i.e., 1, denoting the input that the constants are multiplied with, or the outputs of OR gates representing target constants or partial terms in the network.
3) Generate an OR gate associated with t_i, where its inputs are the outputs of the AND gates determined in Step 2.
4) If t_i is a target constant, make the output of the corresponding OR gate an output of the network.
5) Repeat Step 1 until all elements in T are considered.

The network generated for the target constant 25 defined under MSD is given in Fig. 7, where one-input OR gates for the partial terms 7, 9, 17, and 33 are omitted and the type of each operation is shown inside of each AND gate.

C. Formalization of the 0–1 ILP Problem

We need to include optimization variables into the network, so that we can easily formalize the gate-level area optimization problem as a 0–1 ILP problem. The optimization variables are associated with two parameters that have different implementation costs at the gate level, i.e., addition/subtraction operations and left shifts of constants (partial terms and target 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., $\text{opt}_{a \pm b}$, where a and b denote the inputs of an operation, and we add this variable to the input of the AND gate. The cost value of this type of optimization variable in the cost function to be minimized is determined as the gate-level implementation cost of the digit-serial operation computed considering its type (addition or subtraction) and the digit size (d), as described in Section II-D.

In order to maximize the sharing of left shifts, i.e., the D flip-flops at the gate level, for each constant c in the network, we initially find the maximum amount of left shift mls_c that the constant c has. 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., $\text{opt}_{c \ll 1}, \text{opt}_{c \ll 2}, \ldots, \text{opt}_{c \ll mls_c}$. In the cost function to be minimized, the cost value of this type of optimization variable is determined as the gate-level cost of one D flip-flop, as described in Section II-D. The inclusion of these optimization variables into the network can be done in two ways.[2]

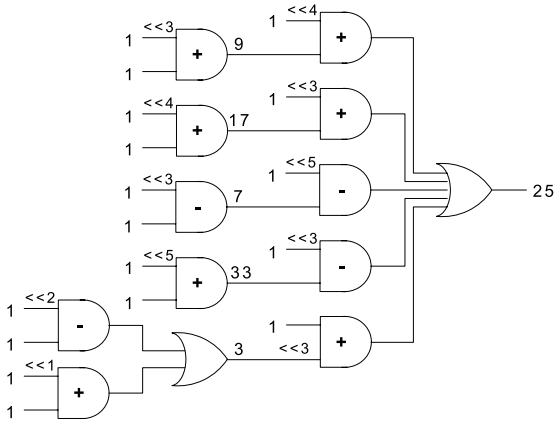


Fig. 7. Network constructed for the target constant 25 under MSD.

Model 1: For each AND gate in the network representing an addition/subtraction operation, if an input signal ins is shifted by $ls > 0$ times, then we include ls additional inputs standing for the optimization variables associated with the ls left shift of the input signal ins, i.e., $\text{opt}_{ins \ll 1}, \text{opt}_{ins \ll 2}, \ldots, \text{opt}_{ins \ll ls}$.

Model 2: Initially, for each constant c with mls_c greater than zero, we generate a chain of $mls_c - 1$ AND gates with two inputs, where the inputs of the first AND gate of the chain are $\text{opt}_{c \ll 1}$ and $\text{opt}_{c \ll 2}$, and the inputs of the ith AND gate are $\text{opt}_{c \ll i+1}$ and the output of the $(i-1)$th (previous) AND gate in the chain, where $2 \leq i \leq mls_c - 1$. Then, for each AND gate representing an addition/subtraction operation, if an input signal ins is shifted by $ls > 0$ times, we add a single input to the AND gate. This input is $\text{opt}_{ins \ll 1}$, if ls is equal to 1, or otherwise, the output of the $(ls-1)$th AND gate in the chain of AND gates including the optimization variables for the mls_{ins} left shift of the input signal ins.

Some simplifications in the network can be also achieved. The input variable x denoted by 1 can be eliminated from the inputs of the AND gates, because its logic value is always 1 (i.e., it is always available). Figs. 8(a) and (b) illustrate the networks generated for the target constant 25 under MSD after the simplifications are done and the optimization variables are added under *Models 1* and *2*, respectively.

After the optimization variables are added into the network, the 0–1 ILP problem is generated. The cost function of the 0–1 ILP problem is constructed as the linear function of optimization variables, where the cost value of each optimization variable is determined as described previously. The constraints of the 0–1 ILP problem are obtained by finding

[2]Recall that in digit-serial arithmetic, the left shift of a constant c by mls_c times requires a total of mls_c D flip-flops and any left shift of c less than mls_c can be obtained with this circuit, without requiring any additional hardware.

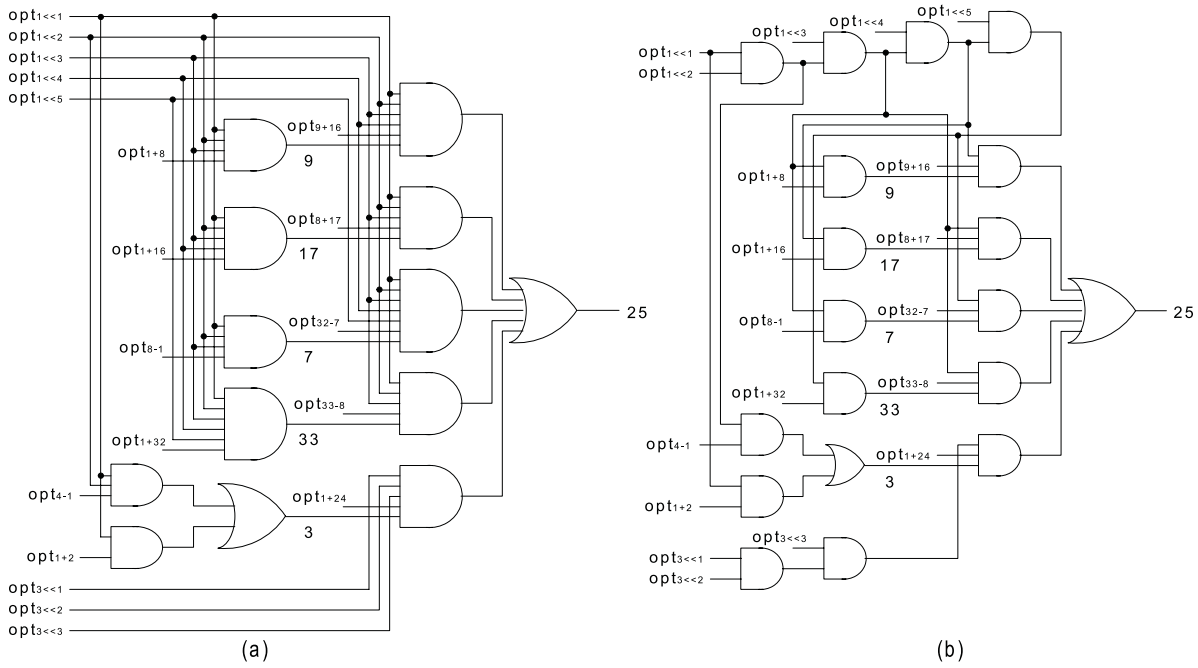


Fig. 8. Networks constructed for the target constant 25 under MSD after the optimization variables are added. (a) Under Model 1. (b) Under Model 2.

the CNF formulas of each gate in the network and expressing each clause of the CNF formulas as a linear inequality, as described in Section II-C. 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.

Note that, if the cost values of the optimization variables in the cost function to be minimized are set to 1, the 0–1 ILP formalization of the MCM-DS problem will be obtained.

Observe from Fig. 8(a) and (b) that Model 1 generates a 0–1 ILP problem including slightly less number of variables than Model 2 due to the chain of AND gates used in Model 2. However, the 0–1 ILP problem constructed under Model 2 has significantly less number of constraints than that of Model 1 since the number of inputs of an AND gate representing an addition/subtraction operation is increased only by 1 during the inclusion of the optimization variables denoting the left shift of a constant in Model 2. Note that the number of optimization variables under both models is the same.

D. Finding the Minimum Area Solution

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

IV. Approximate GB Algorithm

Note that the solution of an exact CSE algorithm described in Section III is not the global minimum since all possible implementations of a constant are found from its representation. Also, the optimization of gate-level area problem in digit-serial MCM design is an NP-complete problem due to the NP-completeness of the MCM problem. Thus, naturally, there will be always 0–1 ILP problems generated by the exact CSE algorithm that current 0–1 ILP solvers find difficult to handle. Hence, the GB heuristic algorithms, which obtain a good solution using less computational resources, are indispensable.

In our approximate algorithm called MINAS-DS, as done in algorithms designed for the MCM problem given in Definition 1, we find the fewest number of intermediate constants such that all the target and intermediate constants are synthesized using a single operation. However, while selecting an intermediate constant for the implementation of the not-yet synthesized target constants in each iteration, we favor the one among the possible intermediate constants that can be synthesized using the least hardware and will enable us to implement the not-yet synthesized target constants in a smaller area with the available constants. After the set of target and intermediate constants that realizes the MCM operation is found, each constant is synthesized using an *A-operation* that yields the minimum area in the digit-serial MCM design. In MINAS-DS, the area of the digit-serial MCM operation is determined as the total gate-level implementation cost of each digit-serial addition, subtraction, and shift operation under the digit size parameter d as described in Section II-D.

The preprocessing phase of the MINAS-DS algorithm is the same as that of the exact CSE algorithm, and its main part and routines are given in Figs. 9 and 10, respectively. Note that, as done in [29], the right shift of an *A-operation* is assumed to be zero in MINAS-DS.

In MINAS-DS, the ready set $R = \{1\}$ is formed initially, and then the target constants, which 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. If there exist unimplemented constants in the target set, then in its iterative loop (lines

```
MINAS-DS(T)
 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:          end if
12:       end if
13:    end for
14:    Find the intermediate constant, ic, with the minimum iccost_j
       cost among all possible constants, j
15:    R ← R ∪ {ic}
16:    (R, T) = Synthesize(R, T)
17: end while
18: D = SynthesizeMinArea(R)
19: return D
```

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

3–17 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 (lines 6 and 7) as possible intermediate constants. In MINAS-DS, the *Compute-Cost* 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 II-D 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 added 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 digit-serial 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 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 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 14–16).

When there are no elements left in the target set, the *SynthesizeMinArea* function (line 18) is applied on the final ready set to find the set of *A-operations* that yields a solution with the 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.

```
Synthesize(R, T)
 1: repeat
 2:    isadded = 0
 3:    for each t_k ∈ 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 ← R ∪ {t_k}
 7:          T ← T \ {t_k}
 8:       end if
 9:    end for
10: until isadded = 0
11: return (R, T)

ComputeCost({c}, C)
1: cost_c = 0
2: for all operations c = |2^l1 u + (−1)^s 2^l2 v|2^−r, where u, v ∈ C do
3:    Determine the cost of each operation under the digit-serial
      architecture, compute the minimum implementation cost of
      constant c, and assign it to cost_c
4: end for
5: return cost_c

ComputeTCost(B, C)
 1: cost_B = 0
 2: repeat
 3:    isadded = 0
 4:    for each b_k ∈ B do
 5:       cost_bk = ComputeCost({b_k}, C)
 6:       if cost_bk ≠ 0 then
 7:          isadded = 1
 8:          C ← C ∪ {b_k}
 9:          B ← B \ {b_k}
10:          cost_B = cost_B + cost_bk
11:       end if
12:    end for
13: until isadded = 0
14: for each b_k ∈ B do
15:    cost_B = cost_B + maxcost(b_k)
16: end for
17: 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 under
   the digit-serial architecture
4: return D

GenerateImp(R)
 1: A ← {1}, R ← R \ {1}, O ← {}
 2: repeat
 3:    for each r_k ∈ R do
 4:       (B, C) = Synthesize(A, {r_k})
 5:       if C = ∅ then
 6:          Find all operations, r_k = |2^l1 u + (−1)^s 2^l2 v|2^−r, where
             u, v ∈ A and determine their implementation costs under
             the digit-serial architecture and store them in O
 7:          A ← A ∪ {r_k}
 8:          R ← R \ {r_k}
 9:       end if
10:    end for
11: until R = ∅
12: return O
```

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

Hence, we formalize this problem as a 0–1 ILP problem, similar to the formalization described in Section III. In this case, the possible implementations of the constants are found by the *GenerateImp* function given in Fig. 10. Note that the size of the 0–1 ILP problem is much smaller than that of the 0–1 ILP problem generated by the exact CSE algorithm, and therefore finding the minimum solution of this 0–1 ILP problem is much simpler. This is because the possible implementations of a constant are limited to the elements in the final ready set.

V. Computer-Aided Design Tool

This section initially presents the design of a digit-serial MCM operation under the shift-adds architecture. Then, a generic digit-serial constant multiplier architecture adapted from [17], which is used for an alternative digit-serial realization of the MCM block and for comparison with the shift-adds architecture in Section VI, is introduced. Finally, the design process of a digit-serial FIR filter is presented. In SAFIR, all the circuits are described in VHDL, and a commercial synthesis tool is used to design these circuits.

A. Design of Digit-Serial MCM Operations Under the Shift-Adds Architecture

In this case, we use the solution of a high-level algorithm on an MCM instance that consists of *A-operations* implementing the constant multiplications. Initially, to design the necessary circuit for the implementation of left shift operations, for each constant c in the solution of a high-level algorithm we find the maximum amount of left shift that c has, i.e., mls_c. Then, we describe the d-layer cascaded D flip-flop circuit, where the number of required D flip-flops in each layer is determined by (4), as described in Section II-D. The d-bit inputs of this circuit are the d-bits of the input variable x if c is 1 or the d-bit outputs of a digit-serial addition/subtraction operation implementing cx, as described in the following.

For each addition/subtraction operation in the solution of a high-level algorithm, we describe a digit-serial addition/subtraction operation, as presented in Section II-D. If an input of the *A-operation* has a zero left shift, then the d bits of this input are the d-bits of the input variable x if this input is 1 or the d-bit outputs of the digit-serial operation implementing this input. The d-bit inputs of an operation with a left shift greater than zero are taken from the corresponding outputs of the cascaded D flip-flop circuit generated for the left shifts, as determined in (3).

When the conversion from digit-serial to bit-parallel is required, the d-bit outputs of the operations realizing the target constants generated at each clock cycle need to be stored. Note that the bitwidth 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. The circuits required to store the d-bits of the output of an operation generating cx in each clock cycle when d is 2 and 3 are shown in Figs. 11(a) and (b), respectively, where bw_{cx} is 8 and cx_0 is the LSB and cx_7 is the MSB.

Furthermore, the MCM operation generally includes constants with different bitwidths. Hence, we need a control logic to store the constant multiplications accurately. To do this, we used a $\lceil \log_2 L_{MCM} \rceil$-bit counter which increments by 1 in each clock cycle, where L_{MCM} is the latency of the MCM operation and is determined as given in (5). Then, we control the storage of the constant multiplication cx 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. 11.

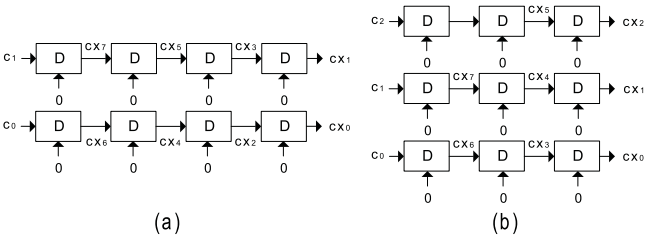


Fig. 11. Storage circuit for an 8-bit constant multiplication cx. (a) When d is 2. (b) When d is 3.

Recall that, in high-level algorithms, the constants to be realized are converted to positive and odd constants beforehand. Hence, when an output of the network is an even or negative version of a target constant, we realize this output by shifting the related target constant or taking its 2's complement.

B. Design of Digit-Serial MCM Operations Using Digit-Serial Constant Multipliers

Generic digit-serial multiplier architectures in which both operands are time-variant can be found in [30] and [31]. However, these architectures are not flexible enough to take the advantage of constant multiplications. On the other hand, bit-serial constant multiplier architectures in which one operand is constant (time-invariant) were presented in [32] and [33]. In these constant multiplier architectures, the hardware of the design is significantly reduced with respect to the generic digit-serial multipliers by utilizing the nonzero digits of the constant to be multiplied by the input variable x. Moreover, in [33], a CSE technique used to maximize the sharing of partial products among the constant multiplications was also proposed.

The digit-serial constant multiplier realized in SAFIR is based on the sequential multiplication algorithm presented in [17], which is illustrated on a simple example in Fig. 12(a). As can be easily seen, this method can be realized by iteratively generating the partial product, i.e., the multiplication of d-bit input data x with the constant c, shifting the partial product, and adding with the previous partial product sum. As depicted in Fig. 12(b), in our design, at each clock cycle the d-bit input data x (x_i) is applied to the select input of the $2^d - 1$ multiplexer and the partial product is generated at the multiplexer output based on the x_i value. Since the constant c is known, rather than using a multiplication operation the inputs of the multiplexer are assigned to the integer values of $0,\ c,\ 2c,\ 3c,\ \ldots,\ (2^d - 1)c$. Thus, the bitwidth of the multiplexer output, i.e., m, is $\lceil \log_2 (2^d - 1)c \rceil$. The partial product store (PPS) block is a shift register with a total of $\lceil \log_2 c \rceil + N$ D flip-flops and its leftmost m bits are assigned to the inputs of the addition operation. When the current partial product (the multiplexer output) is added with the leftmost m bits of the PPS block, the output of the adder is stored in the leftmost D flip-flops of the PPS block and is shifted right by d bits, filling zeros to the leftmost d bits. Note that the carry output bit of the addition operation is always zero due to the d

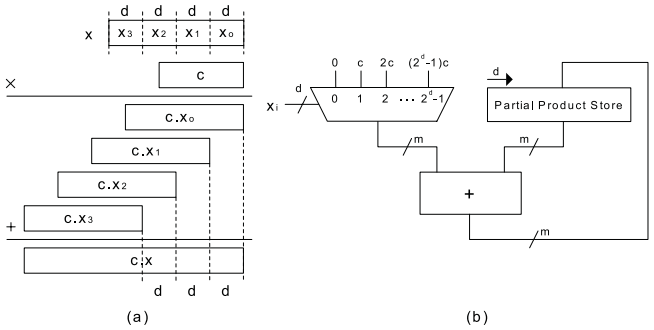


Fig. 12. Digit-serial constant multiplier using the sequential multiplication algorithm [17]. (a) Illustrative example. (b) Design architecture.

leftmost zero bits at the m-bit output of the PPS block. At the end of $\lceil N/d \rceil$ clock cycles, the constant multiplication cx is available at the outputs of D flip-flops in the PPS block. Note that the latency of this architecture is different from that of the shift-adds architecture given in (5). This is because in each clock cycle, except the last one, the d-bits of cx are obtained, and at the last clock cycle the most significant m-bits of cx are generated.

In this scheme, we only consider the positive and odd constants. If an even or negative version of a constant is required as an output, it is realized by shifting the related constant or taking its 2's complement.

C. Design of Digit-Serial FIR Filters

In the design of digit-serial FIR filters [Fig. 1(b) and (c)], we initially implement the multiplier block of the FIR filter based on the design architectures described in Sections V-A and V-B, i.e., using digit-serial addition, subtraction, and shift operations or using digit-serial constant multipliers, respectively. Then, we define the necessary logic required to compute the filter output, i.e., the adders and registers shown in Fig. 1(b) and (c). In this part, an addition operation is defined as a d-bit digit-serial addition/subtraction operation depending on the sign of the constant multiplication output. Also, a register consists of one D flip-flop on d layers, a total of d D flip-flops.

In order to convert the digit-serial filter output to a bit-parallel one, we again need a storage circuit. As can be observed from Fig. 1(b), the bitwidth of the filter output, i.e., bw_y, can be computed as $\lceil \log_2(\sum_{i=0}^{k-1} |h_i|) \rceil + N$, where h_i denotes a filter coefficient. Thus, we need $\lceil bw_y/d \rceil$ cascaded D flip-flops in d layers, a total of $d \times \lceil bw_y/d \rceil$ D flip-flops. Also, note that the latency to obtain the filter output in terms of clock cycles, i.e., L_{FIR}, is $\lceil bw_y/d \rceil$. Thus, in our control logic we require a $\lceil \log_2 L_{\text{FIR}} \rceil$-bit counter.

VI. Experimental Results

This section is divided in two parts: the first part presents the results of high-level algorithms on MCM blocks of FIR filters; the second part introduces the gate-level design results on MCM blocks and FIR filters.

A. Results on High-Level Implementations

As the first experiment set, we used the FIR filter instances given in Table I where the filter 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 bitwidth of the filter coefficients.

TABLE I
FIR Filter Specifications

Filter	Pass	Stop	#Tap	Width
1	0.20	0.25	120	8
2	0.10	0.25	100	10
3	0.15	0.25	40	12
4	0.20	0.25	80	12
5	0.24	0.25	120	12
6	0.15	0.25	60	14
7	0.15	0.20	60	14
8	0.10	0.15	60	14

Table II presents the size of 0–1 ILP problems generated by the exact CSE algorithm under CSD and MSD using Models 1 and 2 described in Section III-C for bit-serial design of multiplier blocks of the FIR filters. In this table, *vars*, *cons*, and *ovars* stand for the number of variables, constraints, and optimization variables, respectively. Also, CPU denotes the required CPU time in seconds of the 0–1 ILP solver SCIP 2.0 [34] to find the minimum solution on a PC with Intel Xeon at 2.33 GHz and 4 GB of memory.

Observe from Table II that the exact CSE algorithm generates a 0–1 ILP problem with higher complexity under MSD representation with respect to CSD. This is simply because a constant may have more than one possible representation under MSD and one of them is its CSD representation. Moreover, Model 2 generates a 0–1 ILP problem including significantly less number of constraints with a slight increase in the number of variables when compared to Model 1. Since the size of a 0–1 ILP problem has a significant impact on the runtime of the 0–1 ILP solver, the minimum solution is generally obtained in less time using Model 2.

Table III presents the results of the algorithms of [9] and [12] designed for the MCM problem and the algorithms designed for the optimization of gate-level area in digit-serial MCM design when d is equal to 1. In this table, *oper* and *shift* stand for the number of operations and shifts, respectively, and *icost* denotes the implementation cost of bit-serial MCM designs. The implementation costs of an FA, a D flip-flop, and an inverter were taken as 6, 52, and 90, respectively, the area (in μm^2) of these components in UMCLogic 0.18 μm Generic II library.

Observe from Table III that the high-level algorithms designed for the optimization of area lead to digit-serial MCM operations that require less hardware than those obtained by the algorithms designed for the MCM problem. Since the exact CSE algorithm considers alternative implementations of a constant under MSD representation when compared to CSD, it finds better solutions in terms of the implementation cost under MSD. Moreover, MINAS-DS gives better solutions than the exact CSE algorithm, since it considers more possible implementations of a constant yielding MCM designs with less number of operations. We note that the total runtime of MINAS-DS on these instances is 9.79 s, which is smaller than those of the exact CSE algorithms given in Table II.

TABLE II

SIZE OF 0–1 ILP PROBLEMS AND RUNTIME OF THE 0–1 ILP SOLVER [34] ON MCM BLOCKS OF THE FIR FILTERS IN TABLE I FOR $d = 1$

Fil.	Exact CSE - CSD								Exact CSE - MSD							
	Model 1				Model 2				Model 1				Model 2			
	vars	cons	ovars	CPU	vars	cons	ovars	CPU	vars	cons	ovars	CPU	vars	cons	ovars	CPU
1	163	423	94	0.01	186	322	94	0.01	367	995	195	0.24	394	724	195	0.06
2	462	1425	251	0.21	509	969	251	0.11	838	2679	440	0.41	897	1780	440	0.15
3	636	2041	343	6.08	704	1372	343	1.44	1599	5440	822	4.30	1690	3634	822	3.72
4	982	3321	512	2.41	1055	2110	512	2.93	2332	8041	1189	17.33	2432	5184	1189	9.13
5	843	2789	463	0.94	936	1767	463	0.27	1556	5265	820	0.17	1674	3288	820	0.13
6	1416	5100	734	9.15	1514	3208	734	16.65	4807	17 573	2442	77.18	4969	11 658	2442	35.85
7	1514	5347	807	24.34	1654	3452	807	9.26	2506	8874	1298	29.51	2653	5746	1298	13.00
8	1897	7180	995	42.32	2053	4397	995	18.34	4223	16 369	2149	71.09	4430	10 056	2149	44.21
Tot.	7913	27 626	4199	85.46	8611	17 597	4199	49.01	18 228	65 236	9355	200.23	19 139	42 070	9355	106.25

TABLE III

SUMMARY OF RESULTS OF ALGORITHMS ON MCM BLOCKS OF THE FIR FILTERS IN TABLE I FOR $d = 1$

Fil.	Optimization of the number of operations									Optimization of area in digit-serial MCM design								
	Exact CSE - CSD [9]			Exact CSE - MSD [9]			Exact GB [12]			Exact CSE - CSD			Exact CSE - MSD			MINAS-DS		
	oper	shift	icost	oper	shift	icost	oper	shift	icost	oper	shift	icost	oper	shift	icost	oper	shift	icost
1	10	14	2184	10	14	2184	10	13	2120	11	7	1974	11	7	1950	10	4	1640
2	18	27	4020	18	27	4020	17	30	4016	19	16	3584	20	9	3332	17	9	2894
3	16	40	4412	16	40	4412	15	35	3992	20	18	3842	19	15	3502	15	14	2876
4	29	37	6120	29	37	6120	28	35	5874	31	20	5520	32	12	5240	28	8	4398
5	34	49	7502	34	44	7236	34	43	7136	36	18	6138	35	16	5856	34	12	5488
6	23	44	5626	22	35	5004	20	37	4806	25	29	5130	24	23	4664	20	22	4020
7	35	56	7990	34	55	7802	29	40	6264	39	22	6814	38	18	6416	29	25	5478
8	35	62	8338	33	56	7718	28	43	6302	39	25	6958	35	23	6250	28	29	5544
Tot.	200	329	46 192	196	308	44 496	181	276	40 510	220	155	39 960	214	123	37 210	181	123	32 338

TABLE IV

FIR FILTER SPECIFICATIONS, SIZE OF 0–1 ILP PROBLEMS, AND RUNTIME OF THE 0–1 ILP SOLVER [34]

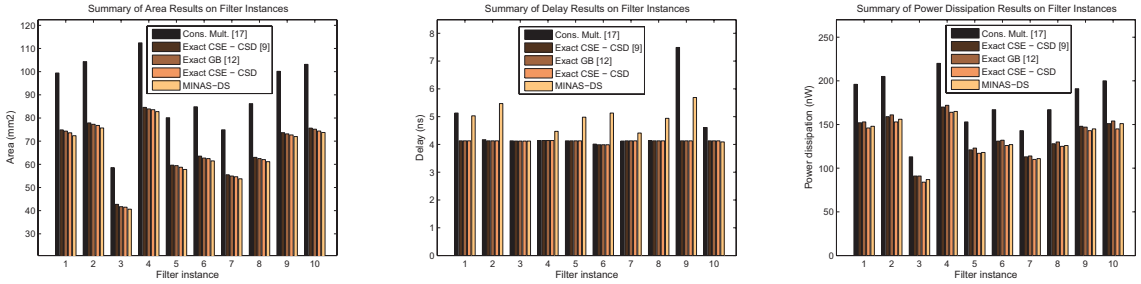
Filter	Filter specifications				Exact CSE - CSD							
					Model 1				Model 2			
	pass	stop	tap	width	vars	cons	ovars	CPU	vars	cons	ovars	CPU
1	0.10	0.15	200	16	8854	35 566	4559	1324	9359	20 626	4559	274
2	0.10	0.15	240	16	9788	39 721	5057	6970	10 358	23 057	5057	1839
3	0.10	0.25	180	16	10 847	45 002	5555	28 188	11 352	26 772	5555	5791
4	0.10	0.25	200	16	15 041	63 532	7715	19 231	15 737	36 699	7715	3849
5	0.10	0.20	240	16	9868	40 646	5091	32 616	10 410	23 796	5091	4970
6	0.10	0.20	300	16	9462	38 575	4890	95 402	10 014	22 563	4890	13 088
7	0.15	0.25	200	16	5786	22 795	2978	132	6118	13 486	2978	66
8	0.15	0.25	240	16	5947	22 875	3099	2344	6351	13 494	3099	312
9	0.20	0.25	240	16	4930	18 862	2593	399	5308	10 835	2593	234
10	0.20	0.25	300	16	3492	12 320	1820	15	3714	7542	1820	6
Total					84 015	339 894	43 357	186 621	88 721	198 870	43 357	30 429

As the second experiment set, we used the FIR filters given in Table IV to find out the limitations of the exact CSE algorithm introduced in this paper. These filters were chosen since they include a large number of coefficients defined under 16 bits. This table also presents the size of 0–1 ILP problems generated by the exact CSE algorithm under CSD using Models 1 and 2 when d is 1 and the runtime of the 0–1 ILP solver [34]. The results of the exact CSE algorithm under MSD are not given because of the space limitations. However, it generates larger size 0–1 ILP problems, and the 0–1 ILP solver [34] takes longer runtime to obtain the minimum solution when compared to those obtained under CSD. Observe from Tables II and IV that the size of a 0–1 ILP problem and, consequently, the runtime of the 0–1 ILP solver tend to increase as the bitwidth and the number of constants increase. This experiment shows that, although Model 2 reduces the number of constraints in the 0–1 ILP problem and the runtime of the 0–1 ILP solver with respect to Model 1, there are instances where minimum solutions cannot be obtained easily by the 0–1 ILP solvers.

Table V presents the results of high-level algorithms on FIR filters, where *mul* stands for the number of required bit-serial constant multipliers, which is actually the number of unrepeated positive and odd filter coefficients. Observe that MINAS-DS obtains the best solution in terms of the implementation cost on each instance among these algorithms. Also, the algorithms of [9] and [12] designed for the MCM problem require much less computational effort since the exact CSE algorithm [9] generates a 0–1 ILP problem with much less complexity and the exact GB algorithm [12] does

TABLE V
SUMMARY OF RESULTS OF ALGORITHMS ON MCM BLOCKS OF THE FIR FILTERS IN TABLE IV FOR $d = 1$

Filter	Optimization of the number of operations								Optimization of area in digit-serial MCM design							mul
	Exact CSE - CSD [9]				Exact GB [12]				Exact CSE - CSD			MINAS-DS				
	oper	shift	icost	CPU	oper	shift	icost	CPU	oper	shift	icost	oper	shift	icost	CPU	
1	83	156	20 126	0.07	79	130	18 164	1.69	96	41	16 028	79	31	12 908	317.62	78
2	88	144	20 254	0.26	83	126	18 554	1.83	97	49	16 604	83	32	13 540	758.39	82
3	56	118	14 280	5.20	47	93	11 642	1.46	62	36	10 814	47	34	8508	401.69	46
4	94	127	20 252	0.16	87	111	18 348	1.61	101	54	17 408	87	43	14 668	3.65	87
5	66	117	15 690	0.20	63	115	15 130	1.34	78	37	13 198	63	26	10 394	549.34	62
6	74	124	17 172	1.56	68	104	15 220	3.84	82	44	14 154	68	29	11272	68.36	67
7	65	105	14 882	0.08	59	92	13 270	0.73	74	32	12 364	59	24	9740	1.68	59
8	73	116	16 626	0.07	69	99	15 126	1.44	76	49	13 538	69	23	11 048	657.78	68
9	80	125	18 076	0.01	78	92	16 052	0.83	89	33	14 606	78	32	12 818	4.60	78
10	84	114	18 102	0.01	81	94	16 654	0.87	89	30	14 444	81	21	12 660	4.84	81
Total	763	1246	175 460	7.62	714	1056	158 160	15.64	844	405	143 158	714	295	117 556	2767.95	708



(a) (b) (c)

Fig. 13. Summary of gate-level results of MCM blocks of the FIR filters in Table IV for $d = 1$. (a) Area. (b) Delay. (c) Power dissipation.

not consider all possible realizations of a constant when it finds an operation implementing the constant as opposed to MINAS-DS.

B. Results on Gate-Level Implementations

In this second part, we present the gate-level results of digit-serial MCM operations and FIR filters implemented with the Synopsys design compiler using UMCLogic 0.18 μm Generic II library, where the wire load model is defined by the library based on the number of gates in the design under the *enclosed* mode. Fig. 13 presents the results of the bit-serial MCM designs obtained by the algorithms in Table V, and also the results of MCM blocks designed using bit-serial constant multipliers (*cons. mult.*) [17]. The bitwidth of the filter input (N) was taken as 16, the bit-serial MCM operations were synthesized under the minimum area design strategy during the technology mapping, and the power dissipation values were obtained at the maximum clock frequency of the circuits.

Observe from Fig. 13(a) that MINAS-DS yields bit-serial MCM designs with the least complexity among the high-level algorithms on each instance. However, the delay of an MCM design obtained by MINAS-DS is increased slightly. We also note that the bit-serial MCM operations obtained by the algorithms [9], [12] designed for the MCM problem consume more power on average than those obtained by the exact CSE and MINAS-DS algorithms introduced in this paper. Moreover, the shift-adds design of a bit-serial MCM block achieves major area improvements at the gate level when compared to designs realized with bit-serial constant multipliers.

TABLE VI
GATE-LEVEL RESULTS ON DIGIT-SERIAL MCM BLOCK OF FILTER 4

Algorithm	d	1	2	4	8
Exact CSE CSD	area (mm^2)	83.6	88.3	96.1	114.1
	delay (ns)	4.14	4.26	5.26	4.67
	latency (ns)	132.48	68.16	42.08	18.68
	power (nW)	164	197	237	316
	energy (aJ)	21 727	13 428	9973	5903
MINAS-DS	area (mm^2)	82.8	87	94.1	111.5
	delay (ns)	4.47	4.9	5.65	7.33
	latency (ns)	143.04	78.4	45.2	29.32
	power (nW)	165	196	235	318
	energy (aJ)	23 602	15 366	10 622	9324

Among the filters in Table IV, 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 V. Table VI presents the gate-level results on its multiplier block designed based on the solutions of exact CSE and MINAS-DS algorithms introduced in this paper when d is 1, 2, 4, and 8. We note that N was taken as 16 and the maximum bitwidth of the filter coefficients (bw) is 16. Thus, according to (5), 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. This table also presents the *latency* in ns computed as $clockcycles \times delay$ and the *energy* in aJ (10^{-18} W.s) determined as $clockcycles \times delay \times power$.

As can be observed from Table VI, as the digit size is decreased, the area, delay, and power dissipation values generally decrease. However, the latency and energy consumption increase in this case. Also, observe that, although the

TABLE VII
GATE-LEVEL RESULTS ON COMPLETE DESIGN OF FILTER 4

Arch.	DS	d	1	2	4	8	16
shift-adds	MA	area (mm^2)	201.7	214.8	228.9	281.1	322.9
		delay (ns)	5.45	6.16	6.94	7.70	9.90
		latency (ns)	190.75	110.88	62.46	38.50	9.90
		power (nW)	503	593	694	923	1060
		energy (aJ)	95 947	65 752	43 347	35 536	10 494
	MCF	area (mm^2)	220.6	222.8	233.2	291.1	490.8
		delay (ns)	1.78	2.21	3.04	3.89	3.88
		latency (ns)	62.30	39.78	27.36	19.45	3.88
		power (mW)	271	258	224	241	464
		energy (pJ)	27 412	17 145	10 287	5465	1901
cons. mult.	MA	area (mm^2)	252.0	264.8	269.7	377.9	439.0
		delay (ns)	3.97	5.79	6.92	12.00	9.00
		latency (ns)	138.95	104.22	62.28	60.00	9.00
		power (nW)	619	706	779	1023	1220
		energy (aJ)	86 010	73 579	48 516	61 380	10 980
	MCF	area (mm^2)	299.6	316.4	310.4	418.8	612.7
		delay (ns)	1.48	1.81	2.39	3.98	5.20
		latency (ns)	51.80	32.58	21.51	19.90	5.20
		power (mW)	440	431	376	281	490
		energy (pJ)	22 792	14 042	8088	5592	2548

solutions of MINAS-DS lead to less complex digit-serial MCM designs, since its designs have greater delay, they yield more latency and energy consumption with respect to MCM designs obtained using the solutions of the exact CSE algorithm. We note that, since the MCM designs include the control and storage logic required to convert digit-serial outputs to bit-parallel, the area ratio on different digit sizes is not equal to the digit size ratio. For example, the area ratio on MCM operations obtained by the solutions of MINAS-DS when d is 8 and 1 is 1.34.

The gate-level results of digit-serial designs of the complete Filter 4 are given in Table VII. The filter was designed using 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 by the solution of MINAS-DS under the *shift-adds* architecture and it is implemented using digit-serial constant multipliers described in Section V-B under the *cons. mult.* architecture. For bit-parallel processing ($d = 16$), the multiplier block is designed using addition and subtraction operations obtained by the solution of the exact GB algorithm [12] under the *shift-adds* architecture and it is designed by the logic synthesis tool after the multiplication of each filter coefficient by the filter input is described as constant multiplications in VHDL under the *cons. mult.* architecture. The FIR filters were synthesized under two design strategies (DS) using the logic synthesis tool, i.e., the minimum area (*MA*) and the minimum area under the maximum clock frequency (*MCF*) constraint. In the former, there was no constraint on the clock frequency. In the latter, we found the MCF that can be applied to the filter iteratively in a binary search manner with the use of a design script in SAFIR. Note that the bitwidth of the filter output is 35, and the number of clock cycles required to obtain the filter output is 35, 18, 9, and 5 when d is equal to 1, 2, 4, and 8, respectively. In bit-parallel designs, the filter output is obtained in 1 clock cycle.

As can be easily observed from Table VII, the design of FIR filters under the shift-adds architecture leads to significant savings in area and power dissipation with respect to those that include constant multipliers. The area reduction obtained under the *shift-adds* architecture with respect to the *cons. mult.* architecture reaches up to 34.1% and 43.6% with the *MA* and *MCF* DS, respectively, when d is equal to 8. Moreover, using the *MCF* design strategy, the delay of the designs can be reduced by accepting an increase in area and power dissipation. This design strategy diminishes the disadvantages of digit-serial designs on latency significantly.

VII. CONCLUSION

In this paper, we introduced the 0–1 ILP formalization for designing digit-serial MCM operation with optimal area at the gate level by considering the implementation costs of digit-serial addition, subtraction, and shift operations. Since there are still instances with which the exact CSE algorithm cannot cope, we also proposed an approximate GB algorithm that finds the best partial products in each iteration which yield the optimal gate-level area in digit-serial MCM design. This paper also introduced the design architectures for the digit-serial MCM operation and a CAD tool for the realization of digit-serial MCM operations and FIR filters.

The experimental results indicate that the complexity of digit-serial MCM designs can be further reduced using the high-level optimization algorithms proposed in this paper. It was shown that the realization of digit-serial FIR filters under the shift-adds architecture yields significant area reduction when compared to the filter designs whose multiplier blocks are implemented using digit-serial constant multipliers. It is observed that a designer can find the circuit that fits best in an application by changing the digit size.

REFERENCES

[1] L. Wanhammar, *DSP Integrated Circuits*. New York: Academic, 1999.
[2] C. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. 13, no. 1, pp. 14–17, Feb. 1964.
[3] W. Gallagher and E. Swartzlander, "High radix booth multipliers using reduced area adder trees," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, vol. 1. Pacific Grove, CA, Oct.–Nov. 1994, pp. 545–549.
[4] J. McClellan, T. Parks, and L. Rabiner, "A computer program for designing optimum FIR linear phase digital filters," *IEEE Trans. Audio Electroacoust.*, vol. 21, no. 6, pp. 506–526, Dec. 1973.
[5] H. Nguyen and A. Chatterjee, "Number-splitting with shift-and-add decomposition for power and hardware optimization in linear DSP synthesis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 4, pp. 419–424, Aug. 2000.
[6] M. Ercegovac and T. Lang, *Digital Arithmetic*. San Mateo, CA: Morgan Kaufmann, 2003.
[7] R. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 43, no. 10, pp. 677–688, Oct. 1996.
[8] I.-C. Park and H.-J. Kang, "Digital filter synthesis based on minimal signed digit representation," in *Proc. DAC*, 2001, pp. 468–473.
[9] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 6, pp. 1013–1026, Jun. 2008.
[10] A. Dempster and M. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 42, no. 9, pp. 569–577, Sep. 1995.
[11] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algor.*, vol. 3, no. 2, pp. 1–39, May 2007.
[12] L. Aksoy, E. Gunes, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *J. Microprocess. Microsyst.*, vol. 34, no. 5, pp. 151–162, Aug. 2010.
[13] R. Hartley and K. Parhi, *Digit-Serial Computation*. Norwell, MA: Kluwer, 1995.

[14] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Optimization of area in digital FIR filters using gate-level metrics," in *Proc. DAC*, 2007, pp. 420–423.
[15] L. Aksoy, C. Lazzari, E. Costa, P. Flores, and J. Monteiro, "Optimization of area in digit-serial multiple constant multiplications at gate-level," in *Proc. ISCAS*, 2011, pp. 2737–2740.
[16] L. Aksoy, C. Lazzari, E. Costa, P. Flores, and J. Monteiro, "Efficient shift-adds design of digit-serial multiple constant multiplications," in *Proc. Great Lakes Symp. VLSI*, 2011, pp. 61–66.
[17] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York: Oxford Univ. Press, 2000.
[18] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. 10, no. 3, pp. 389–400, Sep. 1961.
[19] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 1, pp. 4–15, Jan. 1992.
[20] P. Barth, "A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization," Max-Planck-Institut für Informatik, Saarbrücken, Germany, Tech. Rep. MPI-I-95-2-003, 1995.
[21] R. Hartley and P. Corbett, "Digit-serial processing techniques," *IEEE Trans. Circuits Syst.*, vol. 37, no. 6, pp. 707–719, Jun. 1990.
[22] P. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 5, pp. 1037–1041, Oct. 1984.
[23] P. Flores, J. Monteiro, and E. Costa, "An exact algorithm for the maximal sharing of partial terms in multiple constant multiplications," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2005, pp. 13–16.
[24] O. Gustafsson and L. Wanhammar, "ILP modelling of the common subexpression sharing problem," in *Proc. ICECS*, 2002, pp. 1171–1174.
[25] Y.-H. Ho, C.-U. Lei, H.-K. Kwan, and N. Wong, "Global optimization of common subexpressions for multiplierless synthesis of multiple constant multiplications," in *Proc. ASPDAC*, 2008, pp. 119–124.
[26] M. Potkonjak, M. Srivastava, and A. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Comput-Aided Design Integr. Circuits Syst.*, vol. 15, no. 2, pp. 151–165, Feb. 1996.
[27] A. Dempster and M. Macleod, "Constant integer multiplication using minimum adders," *IEE Proc.-Circuits, Devices, Syst.*, vol. 141, no. 5, pp. 407–413, Oct. 1994.
[28] K. Johansson, O. Gustafsson, A. Dempster, and L. Wanhammar, "Algorithm to reduce the number of shifts and additions in multiplier blocks using serial arithmetic," in *Proc. IEEE Medit. Electrotech. Conf.*, May 2004, pp. 197–200.
[29] K. Johansson, O. Gustafsson, and L. Wanhammar, "Multiple constant multiplication for digit-serial implementation of low power FIR filters," *WSEAS Trans. Circuits Syst.*, vol. 5, no. 7, pp. 1001–1008, 2006.
[30] Y.-N. Chang, J. Satyanarayana, and K. Parhi, "Low-power digit-serial multipliers," in *Proc. ISCAS*, 1997, pp. 2164–2167.
[31] H. Suzuki, Y.-N. Chang, and K. Parhi, "Performance tradeoffs in digit-serial DSP systems," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, 1998, pp. 1225–1229.
[32] K. Parhi, "A systematic approach for design of digit-serial signal processing architectures," *IEEE Trans. Circuits Syst.*, vol. 38, no. 4, pp. 358–375, Apr. 1991.
[33] F. Dittmann, B. Kleinjohann, and A. Rettberg, "Efficient bit-serial constant multiplication for FPGAs," in *Proc. NASA Symp. VLSI Design*, 2003, pp. 1–6.
[34] *Solving Constraint Integer Programs.* (2012) [Online]. Available: http://scip.zib.de/

Levent Aksoy (M'09) received the M.S. degree in electronics and communication engineering and the Ph.D. degree in electronics engineering from Istanbul Technical University (ITU), Istanbul, Turkey, 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, Lisbon, Portugal, where he is currently a Post-Doctoral Researcher. His current research interests include satisfiability algorithms, pseudo-Boolean optimization, and electronic design automation problems.

Cristiano Lazzari (M'08) received the M.S. degree in computer science from the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, in 2003, and the Ph.D. degree in microelectronics from the Universidade Federal do Rio Grande do Sul and the Institut National Polytechnique de Grenoble, Toulouse, France, in 2007.

He is a Researcher with the Algorithms for Optimization and Simulation Research Unit, Instituto de Engenharia de Sistemas e Computadores, Lisbon, Portugal. His current research interests include developing techniques for design and test of NoCs and developing algorithms for logic synthesis and technology mapping of multivalued circuits.

Eduardo Costa (M'10) 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, Brazil, in 1991, and the Ph.D. degree in computer science from the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, in 2002. Part of his doctoral work was carried out at the Instituto de Engenharia de Sistemas e Computadores, Lisbon, Portugal.

He is currently a Professor with the Departments of Electrical Engineering and Informatics, Catholic University of Pelotas (UCPel), Pelotas, Brazil. He is associated with the Master Degree Program in Computer Science, UCPel, as a Professor and Researcher. His current research interests include VLSI architectures and low-power designs.

Paulo Flores (M'92) received the five-year engineering, 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.

He has been with the Instituto Superior Técnico since 1990, where he is currently an Assistant Professor with the Department of Electrical and Computer Engineering. He has also been with the Instituto de Engenharia de Sistemas e Computadores, Lisbon, Portugal, since 1988, where he is currently a Senior Researcher. His current research interests include embedded systems, test and verification of digital systems, and computer algorithms, with particular emphasis on optimization of hardware and software problems using satisfiability models.

Dr. Flores is a member of the IEEE Circuits and Systems Society.

José Monteiro (SM'10) received the five-year engineering and M.Sc. degrees in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1989, 1992, and 1996, respectively.

He has been with the Instituto Superior Técnico since 1996, where he is currently an Associate Professor with the Department of Computer Science and Engineering. He is also the Director of the Instituto de Engenharia de Sistemas e Computadores, Lisbon. His current research interests include computer architecture and computer-aided design for VLSI circuits, with emphasis on synthesis, power analysis, and low-power and design validation.

Dr. Monteiro was a recipient of 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.