Exact and Approximate Algorithms for the Optimization of Area and Delay in Multiple Constant Multiplications

Levent Aksoy, *Student Member, IEEE*, Eduardo da Costa, Paulo Flores, *Member, IEEE*, and José Monteiro, *Member, IEEE*

***Abstract*—The main contribution of this paper is an exact common subexpression elimination algorithm for the optimum sharing of partial terms in multiple constant multiplications (MCMs). We model this problem as a Boolean network that covers all possible partial terms that may be used to generate the set of coefficients in the MCM instance. We cast this problem into a 0–1 integer linear programming (ILP) problem by requiring that the single output of this network is asserted while minimizing the number of gates representing operations in the MCM implementation that evaluate to one. A satisfiability (SAT)-based 0–1 ILP solver is used to obtain the exact solution. We argue that for many real problems, the size of the problem is within the capabilities of current SAT solvers. Because performance is often a primary design parameter, we describe how this algorithm can be modified to target the minimum area solution under a user-specified delay constraint. Additionally, we propose an approximate algorithm based on the exact approach with extremely competitive results. We have applied these algorithms on the design of digital filters and present a comprehensive set of results that evaluate ours and existing approximation schemes against exact solutions under different number representations and using different SAT solvers.**

***Index Terms*—Canonical signed digit (CSD), common subexpression elimination (CSE), delay constraints, minimal signed digit (MSD), multiple constant multiplications (MCMs), pseudo-Boolean optimization (PBO).**

I. Introduction

In SEVERAL computationally intensive operations such as finite impulse response (FIR) filters, as illustrated in Fig. 1, and fast Fourier transforms, the same input is to be multiplied by a set of coefficients—an operation known as multiple constant multiplications (MCMs). These operations are typical in digital signal processing (DSP) applications, and hardwired dedicated architectures are the best option for maximum performance and minimum power consumption.

Manuscript received February 23, 2007; revised July 6, 2007 and October 16, 2007. This work was conducted while the authors were researchers at the Instituto de Engenharia de Sistemas e Computadores (INESC-ID). This paper was recommended by Associate Editor S. Nowick.

L. Aksoy is with the Division of Circuits and Systems, Faculty of Electrical and Electronics Engineering, Istanbul Technical University, Istanbul 34469, Turkey (e-mail: levent@ehb.itu.edu.tr).

E. da Costa is with the Departments of Electrical Engineering and Informatics, Catholic University of Pelotas (UCPel), Pelotas 96010-000, Brazil (e-mail: ecosta@ucpel.tche.br).

P. Flores and J. Monteiro are with the Instituto Superior Técnico (IST), Technical University of Lisbon, 1000-029 Lisbon, Portugal (e-mail: pff@inesc-id.pt; jcm@inesc-id.pt).

Digital Object Identifier 10.1109/TCAD.2008.923242

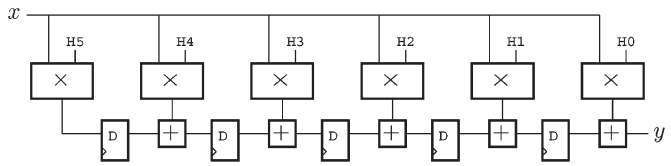

Fig. 1. Transposed form of a hardwired FIR filter implementation.

Constant coefficients allow for a great simplification of the multipliers, which can be reduced to a set of shift–adds [1]. When the same input is to be multiplied by a set of constant coefficients, significant reductions in hardware and, consequently, power, can be obtained by sharing the partial products of the input among the set of multiplications. We propose an algorithm that optimally solves the maximal sharing of partial terms. Although this problem has been proven to be NP-hard [2], we show that, for many practical instances, the size of the problem still allows for the computation of the optimum solution.

This maximal sharing problem has been the subject of extensive research in recent years. Several strategies have been proposed for the optimization of MCMs. One is to consider not only adders, but also subtracters to combine partial terms. A second approach is the use of the canonical signed digit (CSD) representation for the coefficients. This representation minimizes the number of nonzero digits, and hence, the maximal subexpression sharing search starts from a minimal level of complexity [3]. In a recent paper, Park and Kang [4] propose the use of the minimal signed digit (MSD) representation for the coefficients. Under the MSD representation, a given numerical value can have multiple representations. However, in all of them, the number of nonzero digits is minimal and, therefore, the same as the CSD representation.

To the best of our knowledge, all previous solutions to the maximal sharing problem have been heuristic, providing no indication as to how far from the optimum their solutions are. We propose an exact common subexpression elimination (CSE) algorithm that is feasible for many real situations [5]. The proposed algorithm can be applied to coefficients represented in binary, CSD, or MSD. We model this problem as a Boolean network that covers all possible partial terms, which may be used to generate the set of coefficients in the MCM instance. The inputs to this network are shifted versions of the value that serves as an input to the MCM operation. Each adder and subtracter used to generate a partial term is represented as an AND gate. All partial terms that result in the same numerical value are ORed together. There is a single output, which is an AND over all the partial terms that represent the coefficients in the

MCM instance. We cast this problem into a 0–1 integer linear programming (ILP) problem by requiring that the output is asserted, meaning that all coefficients are covered by the set of partial terms found, while minimizing the total number of AND gates that evaluate to one, i.e., the number of adders/subtracters that are effectively used. A generic satisfiability (SAT)-based 0–1 ILP solver is used to compute the exact solution.

In many designs, particularly in DSP systems, performance is a critical parameter. Hence, circuit area is generally expendable in order to achieve a given performance target. The exact algorithm that we propose is able to be parameterized with a delay constraint so that only solutions that meet the desired delay are considered [6]. Thus, the obtained solution is the minimum area solution under the specified maximum delay.

Although the exact algorithm can handle many real-sized designs, it naturally breaks down on some large instances. We have developed approximate algorithms, e.g., ASSUME, for both unconstrained and delay-constrained area minimization [7]. These algorithms are based on the same Boolean network model constructed for the exact algorithm. This network provides a top-down approach in the implementation of coefficients, whereas previous approaches use a bottom-up approach, combining simpler partial terms until the coefficients are implemented, thus more easily falling into local minima. We show that our approximate algorithms are extremely competitive, being able to find the exact solution in many cases.

We present results on experiments with randomly generated instances and with concrete filter instances. We compare solutions obtained with approximate algorithms, i.e., ours and those previously proposed, to the exact solutions for both unconstrained and delay-constrained area minimization. In this comparison, we use different coefficient representations, namely, binary, CSD, and MSD. Several conclusions can be drawn from the results. One is that our exact algorithm is able to handle very large problem instances. The other is that the heuristic algorithm, i.e., ASSUME, produces many times the optimum solutions, or a solution very close to the optimum, in a fraction of the CPU time. At a different level, we show that the CSD, while starting from a simpler coefficient representation, performs significantly worse than the binary representation. The redundancy of the MSD representation does provide the best solutions in most cases, but at the cost of a more complex model. On the other hand, the binary representation yields solutions with greater delay than the solutions obtained under CSD and MSD representations.

This paper is organized as follows. In Section II, we give the basic background concepts and an overview of relevant related work. The model developed for the exact algorithm is described in Section III. Section IV presents how this model can be extended to limit the search to solutions that meet a maximum delay constraint. The approximate algorithms, based on the same model, are described in Section V. Section VI presents and discusses a set of results on selected benchmarks. Finally, Section VII concludes the paper, summarizing the main contributions and giving directions for future work.

II. Definitions

In this section, we start by defining the problem for which we propose exact and approximate algorithms, followed by definitions of background concepts, and we end with an overview of related work.

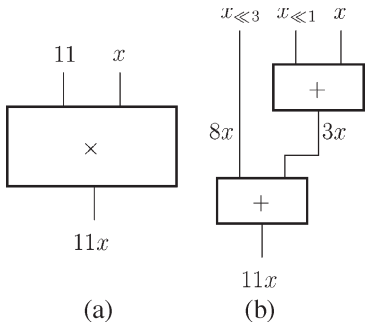

Fig. 2. Computation of $11x$ using: (a) multiplier and (b) shift–adds.

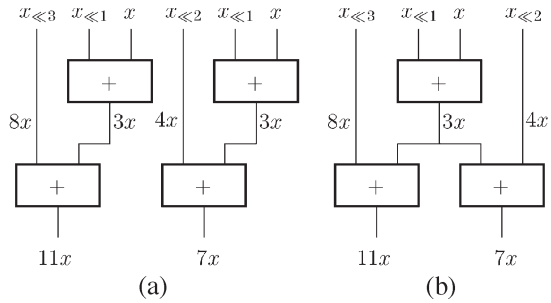

Fig. 3. Simultaneous computation of $7x$ and $11x$. (a) No sharing. (b) Sharing the partial term $3x$.

A. Problem Definition

We address the problem of minimizing the hardware required for a parallel multiplication of an input value over a set of constant coefficients (MCMs). A paradigmatic example of an application where MCMs are realized is the implementation of a digital FIR filter, as illustrated in Fig. 1.

Since each coefficient is constant, we can replace a full-fledged multiplier by a set of additions of shifted versions of the input [1]. A bit set to 1 in position m of the coefficient implies that the input x shifted left by m positions is to be added to the partial sum. Shifts are free in terms of hardware; hence, the hardware required for a multiplication with a constant with n bits set to 1 is simply $n-1$ adders. Fig. 2 presents an example of how $11x$ can be implemented using a shift–add approach.

Each addition generates a partial term. If the same input is to be multiplied by a set of constant coefficients, significant savings can be accomplished by sharing partial terms among the coefficient multiplications. To illustrate this point, consider that we need to implement both $7x$ and $11x$. Instead of using two adders per coefficient as in Fig. 3(a), we can share the adder that generates the value $3x$ to obtain an implementation with a total of three adders [Fig. 3(b)].

We make two immediate notes about the sharing of partial terms. The first is that all values obtained through a shift of any partial term can be considered. The second is that the sharing depends on how the coefficient is decomposed, because the partial terms depend on the sequence of additions. Returning to our example, the sharing exploited in Fig. 3(b) was possible, because we used the decomposition $11x = 2^3x + (2^1x + x)$. If, instead, we had used $11x = (2^3x + 2^1x) + x$, the same level of sharing could be obtained, albeit using the partial term

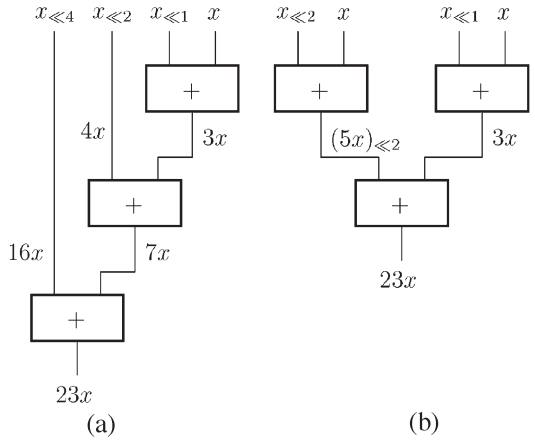

Fig. 4. Two implementations of $23x$: (a) $23x = 2^4x + (2^2x + (2^1x + x))$, with three adder-steps, and (b) $23x = (2^4x + 2^2x) + (2^1x + x)$, with two adder-steps.

$(2^3x + 2^1x)$, which is equivalent to $(2^2x + x)$ shifted to the left by one. However, if the decomposition is $11x = 2^1x + (2^3x + x)$, no sharing is possible with partial terms of $7x$.

This problem can be regarded as a particular case of a more general problem known as CSE [3].

Definition 1: We define the *unconstrained maximum sharing problem* as follows: Given a set of coefficients, find the minimum number of operations (additions or subtractions) required to implement the MCMs.

We extend this problem so that we can limit the maximum number of operations in series, which is generally called the number of *adder-steps*. Clearly, the maximum number of adder-steps over all coefficients defines the maximum delay of one computation. For example, as shown in Fig. 4, $23x$ can be implemented as $23x = 2^4x + (2^2x + (2^1x + x))$ with three adder-steps or as $23x = (2^4x + 2^2x) + (2^1x + x)$ with two.

Definition 2: We define the *maximum sharing problem under a delay constraint* as follows: Given a set of coefficients and a maximum number of adder-steps, find the minimum number of operations (additions or subtractions) required to implement the MCMs so that the user-specified maximum number of adder-steps is not exceeded.

B. Background

1) Number Representation: In the previous section, all examples use the binary representation for the numerical values, where a number is decomposed as a sum of powers of two. Although this is the numerical representation of choice for computer arithmetic, alternative representations can offer some advantages when implementing multiplications with known constants based on shift–adds.

The *canonical sign digit* (CSD) representation [3] is a signed-digit system with the digit set $\{1, 0, -1\}$ (we will be representing the digit -1 by $\bar{1}$). The CSD representation is unique and has the following two main properties: 1) the number of nonzero digits is minimal and 2) two nonzero digits are not adjacent. This representation is widely used in multiplierless implementations, because it reduces the hardware requirements due to the number of nonzero digits being reduced by 33% on the average when compared with the binary representation [8]. The *minimum signed digit* (MSD) representation [4] is obtained by dropping the second property of the CSD representation. Thus, a constant can have several representations under MSD, but all with a minimum number of nonzero digits. For example, suppose that the constant 23 is defined in six bits. The representation of 23 in binary, i.e., 010111, includes four nonzero digits. The constant is represented as $10\bar{1}00\bar{1}$ in CSD, and both $10\bar{1}00\bar{1}$ and $01100\bar{1}$ denote 23 in MSD with three nonzero digits.

The representation of constants in CSD yields a simpler optimization problem when compared with the binary and MSD representations, because the representation of a constant in binary includes more nonzero digits than CSD, and MSD includes several representations for a given constant.

2) Binate Covering Problem: The unconstrained maximum sharing problem that we are addressing can be seen as a binate covering problem (BCP), which is a special case of the 0–1 ILP problem, and can be represented as a Boolean network.

An instance P of a covering problem is defined as follows:

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

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

where c_j in $\mathbf{c}$ is a nonnegative integer cost associated with each of the n variables x_j, $1 \leq j \leq n$, in the cost function (1), and $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ denotes the set of m linear constraints (2). If every entry in the $m \times n$ matrix $\mathbf{A}$ is in the set $\{0, 1\}$ and $b_i = 1$, $1 \leq i \leq m$, then P is an instance of the *unate covering problem*. Moreover, if the entries a_{ij} of $\mathbf{A}$ belong to $\{-1, 0, 1\}$ and $b_i = 1 - |\{a_{ij} : a_{ij} = -1, 1 \leq j \leq n\}|$, then P is an instance of the *BCP*. Observe that, if P is an instance of the BCP, then each constraint can be interpreted as a propositional clause.

A *propositional formula* denotes a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$. A *conjunctive normal form* (CNF) is a representation of a propositional formula φ consisting of a conjunction of propositional clauses, where each clause ω is a disjunction of literals, and a literal l_j is either a variable x_j or its complement $\overline{x_j}$. If a literal assumes the value 1, then the clause is satisfied. If all literals of a clause assume the value 0, then the clause is unsatisfied. The derivation of CNF formulas of the basic gates can be found in [9], where the CNF formula of each gate denotes the valid input–output assignments to the gate. A clause ω to be satisfied in the formula $l_1 + \cdots + l_k$, $k \leq n$, can be interpreted as a linear inequality, $l_1 + \cdots + l_k \geq 1$, where the complement of the variable x_j is represented by $1 - x_j$.

3) SAT-Based 0–1 ILP Solvers: Recent advances in algorithms for Boolean SAT have led to a significant increase in the capacity and applicability of SAT solvers. One of these applications is the pseudo-Boolean optimization (PBO) that is a generalization of the BCP. In [10], a linear search is performed on the possible values of the cost function, starting from the highest, at each step requiring the next computed solution to have a cost that is lower than the most recently computed upper bound. Whenever a new solution is found that satisfies all the constraints, the value of the cost function is recorded as the current lowest computed upper bound. If the resulting instance of the SAT problem is unsatisfiable, then the solution to the instance of BCP is given by the last recorded solution. This approach is considered in the algorithm of Een and Sorensson [11] by converting PBO constraints to Boolean clauses efficiently and, then, calling a SAT solver [12] iteratively to find a minimal cost assignment. The algorithm of Manquinho and Marques-Silva [13] incorporates the most significant features from both approaches, namely, lower bound estimation methods such

as linear programming and Lagrangian relaxations and the reduction techniques from branch-and-bound algorithms and the search pruning techniques from SAT algorithms.

Although there have been additional SAT-based 0–1 ILP solvers [14], in this paper, we use and evaluate the algorithms of Een and Sorensson [11] and Manquinho and Marques-Silva [13], because they propose different approaches and obtain better solutions than other successful solvers.

C. Related Work

A large amount of work that considers the unconstrained maximum sharing problem has addressed the use of efficient implementations of multiplierless MCMs. The techniques include the use of different architectures, implementation styles, and coefficient optimization techniques, e.g., [15]–[17]. The methods restricted to a number representation of the constants basically find common nonzero digit combinations on the representations of the constants and are generally called CSE algorithms. In [18], the CSE method based on the CSD representation is introduced, and in [3], two algorithms—one considers all subexpressions and the other considers only the two most common subexpressions—are presented. The algorithm of Hosangadi *et al.* [19] applies two-term CSE technique iteratively while generating two-term divisors. In addition, the use of different selection criteria for the common subexpressions in CSE algorithms are described in [20] and [21]. In [4], it is shown that by properly exploiting the redundancy of the MSD representation, the hardware implementation can be significantly optimized with respect to the solutions obtained under the CSD representation. The effect of number representation on the achievable minimum number of operations is evaluated in [22], and it is shown that the use of binary representation achieves superior solutions than CSD and better results than MSD as the number of constants and bit-width increase. Furthermore, to extend the number of possible implementations of a constant, the algorithm of Dempster and Macleod [23] applies the CSE technique to all signed-digit representations of a constant, taking into account up to k additional signed digits to the CSD representation, i.e., for a constant including n signed digits in CSD, the constant is represented with up to $n + k$ signed digits. This approach is applied to multiple constants using exhaustive searches in [24].

The algorithms that are not restricted to a particular representation of a constant synthesize a constant iteratively by constructing a graph and are generally called graph-based algorithms. For a single constant multiplication problem, an exact algorithm that finds the minimum number of required operations for a constant up to 12 bit-width is introduced in [25], and it is extended up to 19 bit-width in [26]. Four algorithms, namely, "add-only," "add/subtract," "add/shift," and "add/subtract/shift," are proposed for multiple constants in [27]. The latter algorithm, i.e., "add/subtract/shift," is modified in [28] by extending the possible implementations of a constant, considering only odd numbers, and processing constants in the order of increasing single constant multiplication cost, which is evaluated by the algorithm of Dempster and Macleod [25]. It is shown that the modified algorithm gives much better results with these improvements. Furthermore, in this paper, a heuristic algorithm that uses the results of [25] in the selection of operations to be synthesized is introduced. In [29], another prominent algorithm that uses a better heuristic for synthesizing partial terms and explores a very large search space than existing graph-based algorithms is proposed.

It is shown in [29] that graph-based algorithms give better results than CSE algorithms, since they consider more possible implementations of a constant than CSE algorithms that are restricted to a number representation.

Despite the large number of techniques proposed for the optimization of the number of operations, there are not many methods that also consider the delay of the design, which is essential for high-speed systems. In [30] and [31], while minimizing area, delay is also considered in the selection criteria of the partial terms. In [32] and [33], initially, the number of addition/subtraction operations is reduced, and then, a set of transformations in an iterative loop is used to reduce the delay.

Although the described CSE algorithms obtain good solutions for an MCM problem, they are based on heuristics. In this paper, we introduce exact CSE algorithms for the unconstrained maximum sharing problem and the maximum sharing problem under a delay constraint where multiple constants are considered under a given number representation.

III. Exact Algorithm

In this section, we describe the proposed exact algorithm for the maximal sharing of partial terms. It consists of two steps: First, we construct a Boolean network that represents the computation of all the partial terms that may be used to generate the set of coefficients in the MCM instance; and second, we translate this network into a set of 0–1 ILP constraints and generate the cost function that serve as an input to a generic SAT-based 0–1 ILP solver.

A. Modeling the Problem as a Boolean Network

We model the maximal sharing of partial terms by a Boolean network consisting only of AND and OR gates. Each AND gate represents an operation (addition or subtraction) that produces a partial term value. Each OR gate representing a partial term combines all operations that yield the same value. This model readily lends itself to different number representations, as partial terms are simply the decompositions of the coefficients in the given representation. In the presence of a redundant number representation, such as MSD, all operations that produce the same value are ORed together.

The Boolean network that models the computation of all possible partial terms presents the following characteristics.

1) The primary inputs of the network are the input value (the value to which we are applying the MCM operation) or its shifted versions.
2) There is a two-input AND gate to represent a simple *operation* (addition or subtraction) that generates a given partial term. Since shifts are free, the output of an AND gate can be used for any power of two times the partial term value. An AND gate evaluating to 1 indicates that this operation is available.
3) There is an OR gate to assemble all the different operations that yield a given *partial term* value. An OR gate evaluating to 1 indicates that this value is available.
4) The primary outputs (POs) of the network are the outputs of the OR gates associated with the coefficients in the

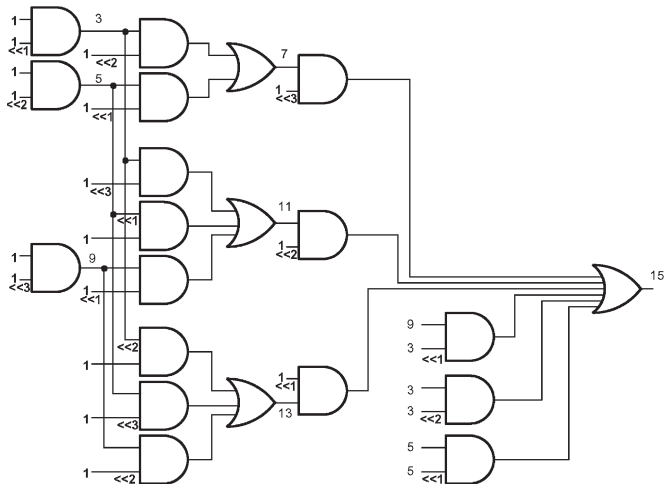

Fig. 5. Boolean network representing the coverage of coefficient 15.

MCM problem. By forcing that all POs evaluate to 1, we ensure that all the coefficients are covered.

Given this model, the SAT-based 0–1 ILP solver has to search for a combination of variables that sets all POs to 1, while minimizing the cost function, defined as the number of AND gates that are selected.

As an illustrative example, consider a single 4-bit coefficient, 15 (in binary, 1111). The value can be obtained as

$$8 + 7(1\boxed{111}), 11 + 4(\overbrace{\boxed{1}1\boxed{11}}), 13 + 2(\overbrace{\boxed{11}1\boxed{1}})$$

or

$$14 + 1(\boxed{111}1)$$

by adding the input to a partial sum, or as

$$9 + 6(\overbrace{\boxed{1}11\boxed{1}}), 12 + 3(11\boxed{11})$$

or

$$10 + 5(\overbrace{\boxed{1}1\boxed{1}}1)$$

by adding two partial sums. In turn, $8 + 7$, for instance, requires 7 to be obtained either as

$$6 + 1(0\boxed{11}1), 5 + 2(0\overbrace{\boxed{1}1\boxed{1}})$$

or

$$4 + 3(01\boxed{11}).$$

The same analysis applies to all the remaining partial sums.

In general, a coefficient with a value v can be obtained with, at most, $\lceil v/2 \rceil$ partial sums. However, we can create equivalent classes from cases that can be computed from each other by a shifting operation, thus significantly reducing the total number of cases. From the example above, $14 + 1$ and $7 + 8$ are equivalent because 14 and 7 are partial sums that differ only on a shift. The same is valid for 1 and 8 and, similarly, for $6 + 1$ and $3 + 4$. The complete Boolean network for this example is presented in Fig. 5, where equivalent cases are omitted.

When the coefficients are represented in CSD or MSD, the model generates a similar network. However, an AND in the network may represent either an adder or a subtracter. Consider, for example, a single 3-bit coefficient with the value 3. The CSD representation of the coefficient is $10\bar{1}$ ($\bar{1}$ stands for -1). Therefore, this value can be obtained with a single subtracter as $4 - 1$. In MSD, the value 3 can be represented both by 011 and $10\bar{1}$ that can be obtained with an adder as $2 + 1$ and with a subtracter as $4 - 1$ respectively.

B. Boolean Network Generation

The implemented algorithm that generates the above optimization model can be used for any type of coefficient representation: binary, CSD, or MSD. However, using the MSD representation results in a more elaborated algorithm, because several representations may exist for the same value. We describe the MSD implementation of the algorithm and, then, summarize the changes for binary and CSD representations.

In a preprocessing phase, all coefficients are converted to positive, and then, they are made odd by successive divisions by 2, i.e., we shift all coefficients to the right so that zero bits on the right are eliminated. Each new resulting coefficient is added to the set of coefficients to be synthesized—*Iset*. This set represents the minimum set of values necessary to synthesize the MCM implementation.

For each element i in *Iset*, all MSD representations are determined using $\lceil \log_2(i) \rceil + 1$ bits and inserted in *Cset*. Therefore, *Cset* begins with all the MSD coefficient representations as in [4]. However, during the execution of our algorithm, *Cset* will be augmented with MSD representations of partial terms.

Then, we enter in the main algorithm loop where an element c, which is removed from *Cset* and represents a number i, is processed to determine its covers.

1) Compute all partial term pairs that cover the element c.
2) For each of these cover pairs, make each element of the pair positive and odd.
3) Ignore cover pairs that are equivalent to a previously generated pair by simply checking for equality of an already existing pair (after the conversion in step 2).
4) Add each cover pair to the corresponding set of covers of the value being processed, i.e., $Aset_i$.
5) Add the MSD representations of each element of the cover pair to *Cset* if the representation has not been processed yet and if it is not in the set. Elements with only one nonzero digit are discarded.

This loop is repeated until there are no more elements in *Cset*. The pair of elements in each $Aset_i$ represents all possible implementations of partial terms for a value i based on its MSD representations. The generation of the Boolean network model is then straightforward.

1) For each element pair in $Aset_i$, generate the corresponding AND gate.
2) Generate an OR gate for the value i with the outputs of all the ANDs resulting from $Aset_i$.
3) Identify all the OR outputs that represent a coefficient (values belonging to *Iset*) and make them POs.

This algorithm can easily be adapted to obtain the network using different coefficient representations. In the procedure above, instead of starting and generating MSD representations,

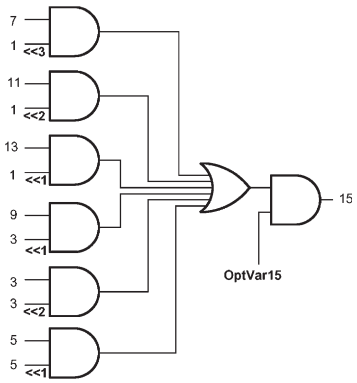

Fig. 6. Inclusion of an AND gate that creates an optimization variable used to minimize the overall number of partial terms.

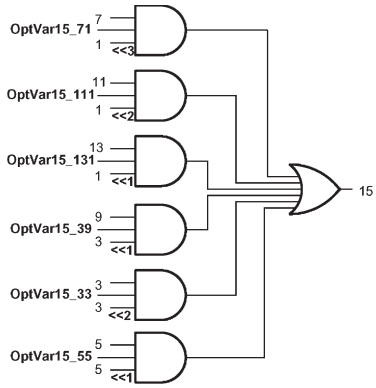

Fig. 7. Addition of an extra input per AND gate to create an optimization variable associated with each possible operation.

we perform this decomposition on the binary or CSD representations instead.

C. Addition of Optimization Variables for the Cost Function

In the generated Boolean network model, we need to include free variables to be used in the cost function. There are basically two variations on how to create these variables: We either associate them with the use of a partial term or with the implementation of a particular operation. As we will see, both these metrics lead to the same optimum solution.

1) Minimizing Partial Terms: The minimization of the total number of partial terms is equivalent to minimizing the number of OR gates in the Boolean network that evaluate to one. Under our model, we can achieve this objective by adding a two-input AND gate for each OR gate in the network, where one input is the output of the OR gate, and the other is the optimization variable. This is illustrated in Fig. 6.

Lemma 1: If the optimization variable evaluates to 1 in the optimum solution, then the output of the corresponding OR *gate evaluates to 1.*

Since the cost function that we are minimizing is the summation of the optimization variables, if the optimization variable evaluates to 1, then the output of the corresponding OR gate is required in the optimum solution. Otherwise, the optimization variable could be set to 0, and we would have a better solution, which is a contradiction. □

Note that the converse is not true. If a pair of partial terms that can be combined to generate a partial term with a single operation is available, then the output of the OR gate will evaluate to 1. For the example in Fig. 6, even if 15 is not required as a partial term, if the partial term 3 is available, then the output of the OR gate will automatically be 1. The meaning of this is that the partial term could be computed using a single operation with available partial terms. However, this does not mean that this particular partial term is going to be computed, i.e., that operation will actually be implemented.

We select the operations that we need to compute by choosing one, from possibly several, of the AND gates of each OR gate with an optimization variable set to 1. Hence, the number of operations will be the same as the optimum value of the cost function.

2) Minimizing Operations: The alternative approach is to associate the optimization variables with the operations themselves. For this, we add a third input to each AND gate, as exemplified in Fig. 7. The solution to the minimization of the sum of the optimization variables will directly indicate which operations are required for the optimum solution.

We make two simple observations.

Lemma 2: There is only one optimization variable set to 1 among the AND *gates that feed the same* OR *gate.*

We first note that any optimization variable in an AND gate with one other input set to 0 will necessarily be 0. Otherwise, we have a contradiction, as setting it to 0 would be a solution with a lower cost function.

For the remaining AND gates, one suffices to set the output of the OR gate to 1. Hence, only one optimization variable over those gates will be 1 in order to minimize the cost function. □

3) Lemma 3: Minimizing the number of operations is equivalent to minimizing the number of partial terms.

In the minimization of the number of partial terms, if the optimization variable at the output of an OR gate evaluates to 1, then we will select, arbitrarily, one of the AND gates at its inputs that evaluate to 1. Thus, we obtain one operation per required partial term (which, by Lemma 1, is the same as optimization variables set to 1).

In the minimization of the number of operations, Lemma 2 shows that we also obtain one operation per partial term.

In both approaches, since we have a one-to-one correspondence between operation and partial term, and since these both solutions are optimum, they have to yield the same cost. □

One advantage of this approach is that the result directly indicates which operations to use. For the unconstrained minimization, this is not very relevant, because it is indifferent which of the available operations is used to compute a partial term. However, as we will discuss in Section IV, it is essential for the delay-constrained optimization, where each operation will correspond to a given level in terms of adder-steps.

One potential downside of this approach is that the number of optimization variables is increased with respect to the approach based on partial terms. As we will show, while this may signify an increased difficulty for some SAT-based 0–1 ILP solvers, others do perform better with a larger number of optimization variables.

4) Network Simplification: Once we have added the optimization variables to the Boolean network (using either of the previous approaches), we use the following rules to simplify the model. While these rules can be safely applied to the unconstrained maximum sharing problem, only rules 1 and 2

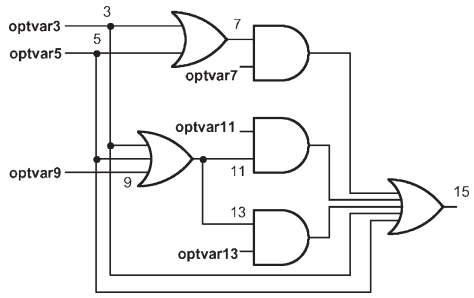

Fig. 8. Simplification of the network of Fig. 5 after optimization variables for minimizing partial terms were added.

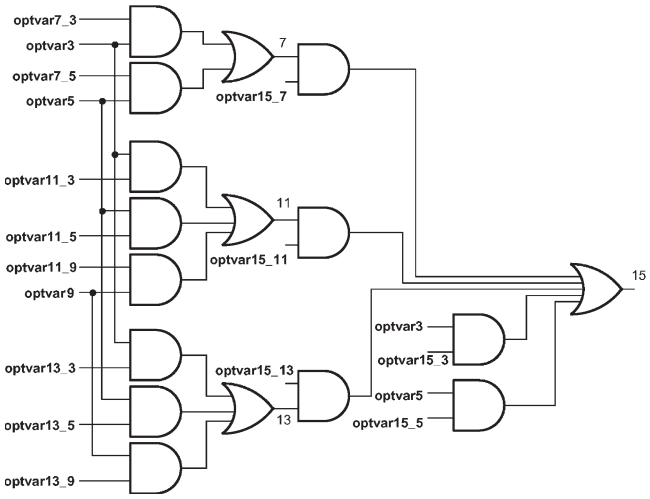

Fig. 9. Simplification of the network of Fig. 5 after optimization variables for minimizing operations were added.

can be applied to the maximum sharing problem under a delay constraint.

1) Shifted versions of the input value are freely available; hence, we set these inputs to 1 in the Boolean network and propagate this value to remove unnecessary gates.
2) If the requirements of an operation are more stringent than another operation that generates the same partial term, we may remove it. For example, $15 = 9 + 3_{\ll 1}$ requires partial terms 9 and 3, whereas $15 = 3_{\ll 2} + 3$ only requires partial term 3; thus, we may eliminate the former, because if partial term 3 is available, we can always use the latter.
3) If a coefficient can be implemented with a single operation whose inputs are the primary inputs and/or other coefficients, then we do not need to represent this filter coefficient in the Boolean network.

The impact of these simplifications heavily depends on the particular instance. They may yield few simplifications in the network or an immediate solution, hence avoiding the 0–1 ILP solver altogether. To exemplify the impact of these simplifications, we present in Fig. 8 the network of Fig. 5, where optimization variables for minimizing the number of partial terms (i.e., an extra AND gate at the output of each OR gate) were added and the described simplifications were applied. Fig. 9 shows the simplified network of Fig. 5 when using optimization variables that minimize the number of operations (i.e., adding an extra input to each AND gate).

TABLE I
UPPER BOUNDS ON THE SIZE OF THE NETWORK AND THE 0–1 ILP PROBLEM

n	#OR	#AND	#variables	#constraints	#opt. variables
8	120	2,059	2,187	10,415	2,059
10	502	19,171	19,683	96,357	19,171
12	2,036	175,099	177,147	877,531	175,099
14	8,178	1,586,131	1,594,323	7,938,833	1,586,131
16	32,752	14,316,139	14,348,907	71,613,447	14,316,119

Additionally, during the construction of the network and the translation of the network into CNF for both problems, the issues described in [34] that speed up a generic SAT-based 0–1 ILP solver are also considered.

D. Mapping Into a 0–1 ILP Optimization Model

We construct the cost function to be minimized as the linear function of the optimization variables, where the cost value of each optimization variable is set to 1. Then, we map the Boolean network into a 0–1 ILP optimization model by representing each gate in CNF format [35]. For example, a two-input AND gate $c = a \wedge b$ is translated to CNF as $(a + \bar{c})(b + \bar{c})(\bar{a} + \bar{b} + c)$. Each clause is converted into a 0–1 ILP constraint using the straightforward mapping presented in [10]. The two-input AND gate would be described by the following set of restrictions:

$$\begin{aligned} a - c &\geq 0 \\ b - c &\geq 0 \\ -a - b + c &\geq -1 \\ a, b, c &\in \{0, 1\}. \end{aligned}$$

To guarantee that all coefficients are covered, we add a constraint that all POs must evaluate to 1. Thus, the obtained model can serve as an input to a generic SAT-based 0–1 ILP solver.

E. Analysis of 0–1 ILP Problem Complexity

Consider the coefficient represented in binary with n bits all set to 1. In this case, the Boolean network includes all partial terms, with b bits, $b \leq n$, set to 1. Thus, all coefficients that include the number of 1 bit less than n are considered in the network. Hence, for n-bit coefficients in any representation, the complexity of the problem is bounded above by the case of a single coefficient with all the n bits set to 1. Table I gives the size of the Boolean network in terms of the number of AND and OR gates, and the size of the 0–1 ILP problem in terms of the number of variables, constraints, and optimization variables for a single coefficient with different values of n bits, all set to 1. We note that these results are obtained, when the most complex case, i.e., the minimization of operations model, is considered without taking into account the network simplifications described in Section III-C3. Hence, an upper bound on the size of the 0–1 ILP problem is found.

Although we can observe the exponential growth in complexity, the size of the 0–1 ILP problem for up to $n = 12$ is within the reach of current SAT-based 0–1 ILP solvers. In practice, coefficients with 12 bits set to 1 may suffice for many real problems. Observe that the exact algorithm can be efficiently applied to larger coefficients, when they are defined in CSD or MSD. We also note that the network simplifications described in Section III-C3 significantly reduce the problem size,

particularly for the model of minimizing partial terms, hence allowing the exact algorithm to be applied to larger designs.

IV. Minimizing Area Under a Delay Constraint

In this section, we describe the exact algorithm designed for the problem of maximum sharing under a delay constraint. We use the Boolean network model described in Section III-C2 and consider the delay as the number of adder-steps, which denotes the maximal number of adders/subtracters in series to produce any multiplication. Since the definition of adder-steps is identical to the definition of level in combinational circuits, in the following, we use both definitions interchangeably.

The exact algorithm can find a solution with either the minimum delay that the network can have, i.e., *min_delay*, or a user-specified maximum delay constraint, i.e., *user_delay*.

A. Computing the Levels of the Operations

In general, a partial term can be implemented with operations that have different adder-steps. Therefore, we can define a range of levels for each partial term, and consequently, a range of levels for the operations that use this partial term.

For a partial term with n nonzero digits, the minimum latency implementation has $\lceil \log_2 n \rceil$ adder-steps and the maximum latency implementation of a partial term has $n-1$ adder-steps. In the network, an OR gate associated with the partial term gathers all of these operations. Therefore, a partial term can be generated with the number of adder-steps ranging from its minimum to maximum latency implementations. As can be seen in Fig. 5, the coefficient 15 can be implemented with a minimum of two and a maximum of three adders-steps, determined, for instance, by $15 = 3_{\ll 2} + 3$, which has a minimum and a maximum of two adder-steps, and by $15 = 1_{\ll 3} + 7$, which has a minimum and a maximum of three adder-steps.

After the Boolean network has been constructed, we compute the minimum level (*min_level*) and maximum level (*max_level*) values of each operation and partial term by traversing the network from primary inputs to POs. Then, we find the *min_delay* value by computing the maximum of the *min_level* values of the POs. By setting *user_delay* = *min_delay* as the maximum delay constraint, the algorithm that we propose is an exact algorithm for the minimum area design that achieves minimum delay. Naturally, if the user sets *user_delay* < *min_delay*, no solution is possible.

B. Incorporating the Delay Information in the 0–1 ILP Model

Using the information on minimum and maximum levels, we compute the paths in the network that exceed the maximum delay constraint (overdelay paths). For each path, we add a delay constraint to the 0–1 ILP problem to prevent the set of operations in the path from being selected in the final solution.

Our algorithm starts by determining the POs of the network with *max_level* values higher than the *user_delay* and storing these outputs in a set called *Pset*. The elements of *Pset* are the filter coefficients that can be implemented in a greater delay than the *user_delay*. Then, for each element $Pset_i$ in *Pset*, if an operation that implements it has a *min_level* value higher than *user_delay*, this operation is deleted from the network, since it can never be used in order to meet the *user_delay*.

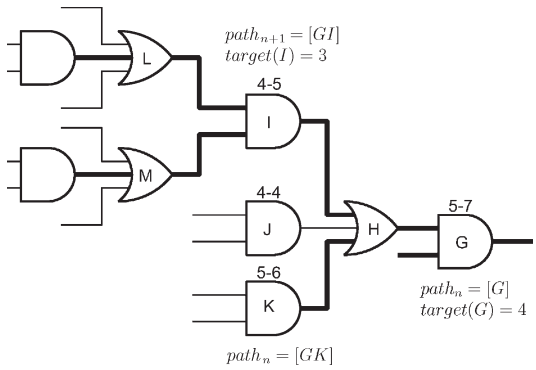

Fig. 10. Illustrative example of determining the paths that exceed the maximum delay.

Otherwise, if an operation has a *max_level* value higher than *user_delay*, then this operation is added to a set called $path_j$ as an initial node. Additionally, this operation is added to a set called *Oset* with a *target* level, *user_delay* $-$ 1, and the associated path identifier j. When all elements in *Pset* have been considered, the initial nodes of the paths that violate the *user_delay* constraint are found. In the following iterative loop, all these paths are constructed in a breadth-first manner.

1) Remove an operation from *Oset* with its target level *target* and the associated path identifier j. For each input of the operation P_i, i.e., a partial term, do the following.
   a) If an operation that implements P_i has a *min_level* value higher than *target*, then add this operation to $path_j$ as a terminal node, i.e., identify the complete overdelay path.
   b) Otherwise, if an operation has a *max_level* value higher than *target*, then create an extended path by adding this operation, as a nonterminal node, to $path_j$. Additionally, insert this operation into *Oset* with its target level, $target - 1$, and a path identifier.
2) Repeat step 1) until there is no element left in *Oset*.

We note that *Oset* includes the last added operations with their *target* level values of the associated paths that have not been constructed yet.

As an example, suppose that a situation, as illustrated in Fig. 10, is encountered while finding the paths that exceed the *user_delay* $= 5$. In this figure, optimization variables are omitted, and the relevant paths are highlighted for the sake of clarity. The operations and partial terms are labeled with letters inside the gates and the *min_level* and *max_level* values are given with a *min–max* pair above the gates. *path* includes the operations that exceed the *user_delay*, which is determined when traversing the network from the outputs to the inputs.

Suppose that the operation G with a target level, $target(G) = 4$, and an associated path identifier n is removed from *Oset*. Suppose also that the partial term H is considered as the input of G. The operation K is added to $path_n$ as a terminal node, and the path is constructed, since the operation K can be implemented in a minimum of five adder-steps that exceeds *target(G)*. Furthermore, a new path, namely, $path_{n+1}$, is formed by inserting the operation I to $path_n$, since the *max_level* value of the operation I is higher than *target(G)*, indicating that there is (are) operation(s) that cause greater delay than the *user_delay* with the operations in this path. Therefore, the operation I with its target level $target(I) =$

$target(G) - 1$ and associated path identifier $n + 1$ is added to *Oset*. We note that the operation J is not considered to be added to $path_n$, because it can be implemented in a maximum of four adder-steps that does not exceed the *target(G)* value.

After all paths that violate *user_delay* have been found, a single additional constraint for each complete overdelay path is added to the 0–1 ILP problem: $-optvar_1 - optvar_2 - \cdots - optvar_m \geq 1 - m$, where $optvar_j$, $1 \leq j \leq m$, denotes the optimization variable of an operation in the path, and m is the number of operations in the path. The delay constraints express that the operations in the path must not be included together in the solution. This guarantees that the solution to be found by the 0–1 ILP solver respects the delay constraints and allows for the possible sharing of partial terms in the paths with other partial terms not in the critical paths. Finally, using the same cost function, the constraints obtained from the Boolean network, together with these delay constraints, are given to the 0–1 ILP solver to find a solution with the minimum area.

V. Approximate Algorithms

Although the exact algorithms presented in the two previous sections can be applied effectively to relatively large MCM problems, the execution time does tend to grow exponentially, limiting its application to more complex instances.

The heuristic algorithms that we propose use as the underlying model the Boolean network generated by the exact algorithm, as described in Section III. In these heuristics, each coefficient is synthesized one at a time by selecting an operation among the set of possible operations, rather than finding the minimum solution of the BCP that considers all coefficients, as done in the proposed exact algorithms. In the selection of an operation, initially, the implementation costs of all operations are found by considering not-yet synthesized coefficients, and then, the operation that has the minimum implementation cost is chosen to implement the coefficient. The advantages of the proposed heuristics are the use of the network that has the view of all the possible manners a coefficient can be synthesized and the use of a selection criteria that also considers not-yet synthesized coefficients while choosing an operation to implement a coefficient. The given properties make these heuristics quite different from the heuristics that find pairs of the most common nonzero digits [3] or the two-term common subexpressions [19]. Since the heuristics of Hartley [3] and Hosangadi *et al.* [19] build coefficients starting at the most simple (in the number of nonzero digits) to the most complex by combining existing partial terms, this bottom-up approach yields a much more limited view of the search space.

In this section, initially, we describe the heuristic called ASSUME-A, which is designed for unconstrained area optimization, and, then, the heuristic called ASSUME-D, which is designed for delay-constrained area optimization. We note that the definitions given in Section IV are also used in the description of these algorithms.

A. Unconstrained Area Optimization: ASSUME-A

In a preprocessing phase, by traversing the Boolean network from primary inputs to POs, the *min_adder* and *max_level* values of each operation and partial term are computed. The *min_adder* is the minimum number of operations that are required to implement an operation or a partial term. The *min_adder* value of a partial term (OR gate) is determined by finding the minimum of the *min_adder* values of operations (AND gates) that implement the partial term. The *min_adder* value of an operation (AND gate) is the sum of the *min_adder* values of its inputs plus 1, if the inputs are different; otherwise, it is the *min_adder* value of an input plus 1. The *min_adder* value of a primary input is assigned to 0. As an example, consider again the network given in Fig. 5, with the coefficient 15. The *min_adder* value of the coefficient 15 is 2, which is determined, for instance, by $15 = 3_{\ll 2} + 3$ and $3 = 1_{\ll 2} - 1$ operations.

In a manner similar to the algorithm of Dempster and Macleod [28], ASSUME-A has two main parts: 1) optimal and 2) heuristic. The algorithm is given as follows.

1) Store the preprocessed coefficients of the filter (POs of the network, all made positive and odd) in a set called *Aset*, and label them as unimplemented.
2) *Optimal* part: For each element labeled as unimplemented in *Aset*, if the element is implemented in the network with an operation whose inputs are either primary inputs or are in *Aset*, then synthesize the element with this operation, and label it as implemented.
3) If there are not more elements labeled as unimplemented in *Aset*, return the solution and stop.
4) *Heuristic* part: Take an unimplemented element from *Aset*, i.e., *Aset(i)*, that has the lowest *max_level* value.
5) For each operation $O(j)$ that implements *Aset(i)*, set its cost value $C(j)$ to its *min_adder* value, as determined in the preprocessing phase and for each unimplemented element in *Aset*, i.e., *Aset(k)*, with $i \neq k$.
   a) Determine $C_{\text{before}}(k)$ by finding the *min_adder* value of *Aset(k)*, when the *min_adder* values of the elements in *Aset* are assigned to 0. [$C_{\text{before}}(k)$ is the cost of implementation of *Aset(k)* at this phase of the algorithm, since all elements in *Aset* will be implemented at the end of the algorithm.]
   b) Determine $C_{\text{after}}(k)$ as done in a), but *assume* also that the inputs of $O(j)$ are in *Aset*. [$C_{\text{after}}(k)$ is the cost of implementing *Aset(k)* if *Aset(i)* is synthesized with $O(j)$ at this phase of the algorithm.]
   c) Update the cost value, $C(j)$, as $C(j) = C(j) - (C_{\text{before}}(k) - C_{\text{after}}(k))$.
6) After the cost value of each operation $C(j)$ has been computed, select the operation to synthesize *Aset(i)* that has the minimum cost. If there are operations that have the same minimum cost, select the operation that has the minimum *min_adder* value among these operations. Label *Aset(i)* as implemented.
7) Add each input of the selected operation to *Aset*, provided that they do not already exist in *Aset*, and label them as unimplemented. Go to step 2).

We note that in the first iteration, the elements of *Aset* are the filter coefficients, and in later iterations, *Aset* may include the partial terms needed for the synthesized operations. Observe that all elements of *Aset* are implemented at the end of the algorithm. We also note that if all elements of *Aset* are implemented in the optimal part, then the global minimum solution is obtained. If an element of *Aset* is implemented in the heuristic part, the local minimum solution is obtained.

B. Area Optimization Under a Delay Constraint: ASSUME-D

Just as the exact version, ASSUME-D can find a solution with either the minimum delay of the network, i.e., *min_delay*, or a maximum user-specified delay constraint, i.e., *user_delay*.

Again, we start by traversing the Boolean network to obtain the *min_adder*, *min_level*, and *max_level* values of each operation and partial term. As defined in Section IV, the *min_delay* is determined as the maximum of the *min_level* values of the POs. A minimum delay solution can be obtained when the *user_delay* is assigned to the *min_delay*.

ASSUME-D synthesizes the coefficients of the filter one at a time in a top-down approach that yields more possible implementations of a partial term while controlling the delay. The algorithm is given as follows.

1) Store the preprocessed coefficients of the filter (POs of the network, all made positive and odd) in a set called *Dset*, and label them as unimplemented. Assign the *delay_limit* value of each element in *Dset* to *user_delay*.
2) Take an element labeled as unimplemented from *Dset*, i.e., *Dset(i)*, that has the highest *max_level* value. Store the operations that implement *Dset(i)* and whose *min_level* value does not exceed *delay_limit(i)* in an empty set called *Oset*.
3) if *Dset(i)* can be implemented with an operation in *Oset* whose inputs are primary inputs or are in *Dset*, then synthesize *Dset(i)* with the operation, and label it as implemented. Assign the delay limit of each input of the operation, i.e., *delay_limit(j)*, to $\min(delay\_limit(j), delay\_limit(i) - 1)$.
4) Otherwise, choose an operation from *Oset* to synthesize *Dset(i)* as done in steps 5) and 6) of ASSUME-A, and label it as implemented. If the input(s) of the operation is in *Dset*, then assign the delay limit of the input $delay\_limit(j) = \min(delay\_limit(j), delay\_limit(i) - 1)$. If not, add this element to *Dset*, label it as unimplemented, and assign its delay limit value to $delay\_limit(i) - 1$.
5) If there is an element left labeled as unimplemented in *Dset*, go to step 2); otherwise, return the solution.

VI. Experimental Results

In this section, we present the results obtained with the exact and heuristic algorithms proposed for the unconstrained maximum sharing problem, as well as the same problem under a delay constraint. The benchmarks used in our experiments include randomly generated and filter instances. In the algorithms designed for maximum sharing problem under a delay constraint, we set the *user_delay* to the *min_delay*, i.e., we find the minimum area under minimum delay solutions. We compare our results with several previously proposed CSE heuristics, namely, with the heuristics of Hartley [3] and Park and Kang [4], which we have implemented, and the heuristic of Hosangadi *et al.* [19], [30], whose results were provided by A. Hosangadi.

As the first experimental set, we used randomly generated instances where constants are defined in 12 bit-width. The number of constants ranges between 10 and 100, and for each of them, we generated 30 instances. We compare the effect of different number representations, i.e., binary, CSD, and MSD, on the minimum number of operations and delay solutions. The results of the exact algorithms on unconstrained maximum sharing problem and maximum sharing problem under a delay constraint are given in Figs. 11 and 12, respectively.

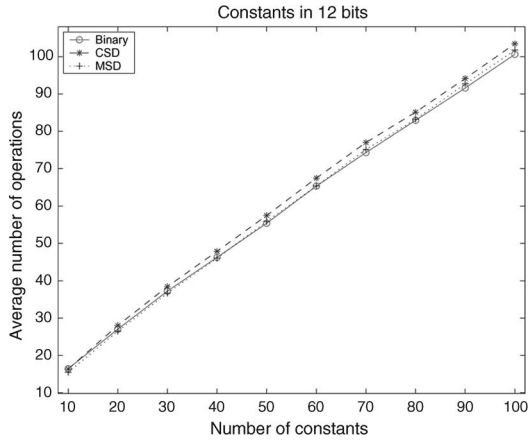

Fig. 11. Comparison of the number representations on the unconstrained maximum sharing problem.

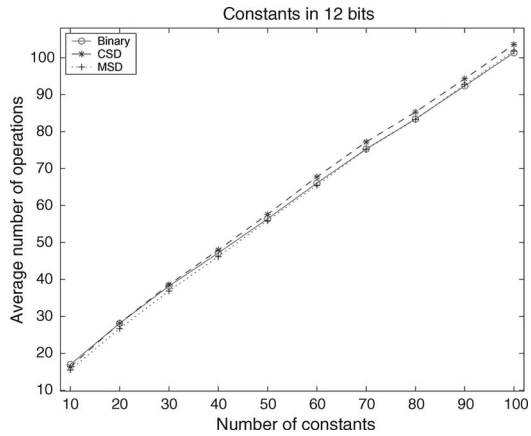

Fig. 12. Comparison of the number representations on the maximum sharing problem under a delay constraint.

We can observe that these three representations yield about the same solutions for instances with few constants. For instances with a larger number of constants, the CSD representation achieves worse solutions than the binary and MSD representations, requiring more than two additional operations on the average. Binary and MSD representations yield very similar results, with the binary performing better as the number of constants increases. This demonstrates that having a third digit, i.e., the signed digit, while desirable in representing one or a few constants, creates a more varied set of patterns that limits the amount of sharing for a larger number of constants. This is partially overcome by the redundancy in the MSD representation.

We compare the minimum delay solutions achievable with the different number representations for the maximum sharing problem under a delay constraint in Fig. 13. We observe that the CSD and MSD representations provide solutions with, at most, three operations in series, while the binary representation, on the average, requires more operations in series, and this number increases with the number of constants. Hence, the minimum

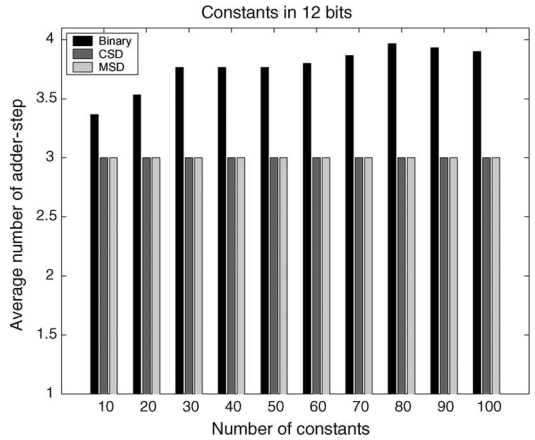

Fig. 13. Comparison of the average number of minimum delay under binary, CSD, and MSD representations.

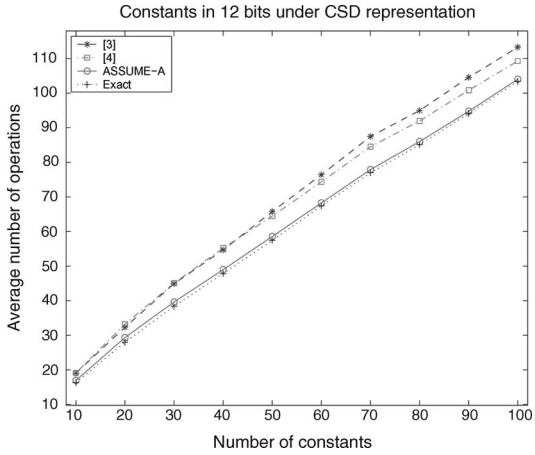

Fig. 14. Comparison of the exact and heuristic algorithms for the unconstrained maximum sharing problem.

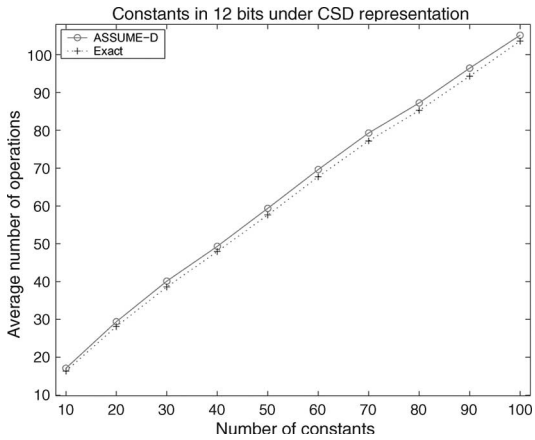

Fig. 15. Comparison of the exact and heuristic algorithms for the maximum sharing problem under a delay constraint.

delay solutions presented in Fig. 12, while similar in area, have a much smaller delay in the cases of the CSD and MSD.

We also compare the exact solutions with the heuristics [3], [4] and ASSUME-A for unconstrained maximum sharing problem, and with ASSUME-D for the maximum sharing problem under a delay constraint on randomly generated instances where constants are represented in CSD. The results are given in Figs. 14 and 15.

TABLE II
CHARACTERISTICS OF THE FIR FILTERS

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
9	0.10	0.15	100	16

In this experiment, we observe that for the unconstrained maximum sharing problem, while the average number of operations between the ASSUME-A and the exact algorithm is almost 1 on all instances, the average number of operations between the heuristic of Park and Kang [4] and the exact algorithm reaches up to 7.4 operations. Furthermore, since the heuristic of Hartley [3] is a greedy algorithm that finds the most common subexpression in each iteration of the algorithm, it is easily trapped to the local minima on instances that include more than 40 constants. On the instances with 100 constants, the average number of operations between this heuristic and the exact algorithm is almost 10. For the maximum sharing problem under a delay constraint, ASSUME-D finds solutions with almost two additional operations on the average, compared to the exact solutions. This clearly shows that exact algorithms find better solutions than the heuristic algorithms, and among the heuristics, ASSUME-A finds much better solutions than the heuristics of Hartley [3] and Park and Kang [4].

As the second experimental set, we used FIR filters where filter coefficients were computed with the *remez* algorithm in MATLAB. The specifications of filters are presented in Table II, where *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 coefficients.

We compare our exact and heuristic algorithms with other heuristic algorithms under binary, CSD, and MSD representations. The results are given in Tables III and IV for unconstrained maximum sharing problem and maximum sharing problem under a delay constraint, respectively. In these tables, *adder* stands for the number of operations, and *stp* denotes the maximum number of operations in series.

We note that the proposed exact algorithms can find minimum solutions for real-sized filter instances. As can be observed in Tables III and IV, while ASSUME-A and ASSUME-D find similar solutions to the exact algorithms, they find better solutions than other heuristics on overall filter instances. While the average difference of the number of operations between the heuristic of Park and Kang [4] and the exact algorithm is almost 1, the average difference of the number of operations between the heuristics of Hartley [3] and Hosangadi *et al.* [19], [30] and the exact algorithms is greater than 1.

The 0–1 ILP problem size of the proposed models (the minimization of partial terms and minimization of operations models for the unconstrained maximum sharing problem and the minimization of operations with a delay constraint model for the maximum sharing problem under a delay constraint) for the filter coefficients defined under MSD representation are given in Table V, where *vars*, *cons*, *delay cons*, and *optvars*

TABLE III
SUMMARY OF THE RESULTS FOR THE UNCONSTRAINED MAXIMUM SHARING PROBLEM

Filter	Binary				CSD										MSD					
	ASSUME-A		Exact		[19]		[3]		[4]		ASSUME-A		Exact		[4]		ASSUME-A		Exact	
	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp
1	10	3	10	3	10	3	10	3	10	3	10	2	10	2	10	3	10	3	10	2
2	18	3	18	4	18	3	18	3	18	3	18	3	18	3	18	3	18	3	18	3
3	17	5	17	5	18	3	19	3	18	4	16	3	16	3	18	4	16	3	16	3
4	29	4	29	4	30	3	30	3	29	4	29	3	29	4	29	4	29	4	29	3
5	35	4	35	4	35	3	36	3	34	3	34	3	34	3	34	3	34	3	34	3
6	24	4	23	4	25	3	25	3	24	3	23	3	23	4	22	4	22	4	22	4
7	33	5	32	5	35	3	35	3	36	4	35	3	35	3	35	3	34	3	34	3
8	35	5	34	4	37	3	37	4	36	4	35	3	35	3	36	4	34	4	33	3
9	52	5	51	5	58	4	55	4	53	5	52	4	51	4	51	5	49	4	49	4
Total	253	38	249	38	266	28	265	29	258	33	252	27	251	29	253	33	246	31	245	28

TABLE IV
SUMMARY OF THE RESULTS FOR THE MAXIMUM SHARING PROBLEM UNDER A DELAY CONSTRAINT

Filter	Binary				CSD						MSD			
	ASSUME-D		Exact		[30]		ASSUME-D		Exact		ASSUME-D		Exact	
	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp
1	10	3	10	3	11	2	10	2	10	2	10	2	10	2
2	18	3	18	3	18	3	18	3	18	3	18	3	18	3
3	18	3	18	3	18	3	16	3	16	3	16	3	16	3
4	29	3	29	3	30	3	29	3	29	3	29	3	29	3
5	36	3	35	3	35	3	34	3	34	3	34	3	34	3
6	25	3	25	3	26	3	23	3	23	3	22	3	22	3
7	33	4	32	4	36	3	35	3	35	3	34	3	34	3
8	36	4	34	4	37	3	35	3	35	3	35	3	33	3
9	53	4	51	4	58	3	52	3	52	3	49	3	49	3
Total	258	30	252	30	269	26	252	26	252	26	247	26	245	26

TABLE V
SIZES OF THE 0–1 ILP PROBLEM FOR THE PROPOSED MODELS UNDER MSD REPRESENTATION

Filter	Minimization of Partial Terms			Minimization of Operations			Minimization of Operations under a Delay Constraint			
	vars	cons	optvars	vars	cons	optvars	vars	cons	delay cons	optvars
1	10	10	10	247	347	144	247	372	25	144
2	76	97	56	635	1027	345	635	1075	48	345
3	151	298	80	1327	2387	677	1327	2546	159	677
4	93	139	64	1926	3331	1023	1926	3616	285	1023
5	34	34	34	1142	1769	651	1142	1897	128	651
6	107	144	74	4324	8547	2153	4324	10127	1580	2153
7	205	455	93	2250	4828	1062	2250	5081	253	1062
8	546	1405	200	3915	8542	1856	3915	9230	688	1856
9	4010	14880	779	26778	55489	13329	26778	71670	16181	13329

TABLE VI
RUN TIME COMPARISON OF THE SAT-BASED 0_1 ILP SOLVERS

Filter	Minimization of Partial Terms				Minimization of Operations				Minimization of Operations under a Delay Constraint			
	Bsolo		MiniSat+		Bsolo		MiniSat+		Bsolo		MiniSat+	
	adder	CPU	adder	CPU	adder	CPU	adder	CPU	adder	CPU	adder	CPU
1	10	0.1	10	0	10	0.2	10	0.1	10	0.2	10	0
2	18	0	18	0	18	0.2	18	0.1	18	0.2	18	0.6
3	16	0.1	16	0	16	0.5	16	0.7	16	0.4	16	2.1
4	29	0	29	0	29	1.4	29	0.5	29	2.7	29	0.7
5	34	0.1	34	0	34	0.3	34	0.3	34	0.3	34	0.3
6	22	0.1	22	0	22	6.8	*22*	*3600.1*	22	25.4	*22*	*3600.1*
7	34	0.1	34	0.1	34	4	34	19.5	34	5.2	34	8.9
8	33	9.8	33	0.1	33	27.4	33	60,8	33	41.6	33	29.2
9	49	380.6	49	4.3	49	1974.8	*59*	*3600.1*	49	1332.2	*53*	*3600.1*

denote total number of variables, constraints, delay constraints, and optimization variables, respectively.

As can be seen in Table V, when the unconstrained maximum sharing problem is defined using the model that considers the minimization of partial terms, a smaller 0–1 ILP problem can be obtained than is defined by the minimization of operations model due to the network simplifications. On these problem instances, we compare SAT-based 0–1 ILP solvers, i.e., Bsolo [13] and MiniSat+ [11], in terms of CPU time required to find a solution. The results are given in Table VI, where *CPU* denotes the CPU time (in seconds) of a personal computer with dual Pentium Xeon at 2.4 GHz and 4 GB of main memory, running Linux. The allowed CPU time for the algorithms was 3600 s. In this table, the italic results indicate that a satisfiable, rather than the minimum, solution is obtained within the given CPU time limit.

TABLE VII
CHARACTERISTICS OF FILTER INSTANCES

Filter	Type	pass	stop	#tap	width
1	Butterworth	0.25	0.3	20	24
2	Elliptical	0.25	0.3	6	24
3	Least Square	0.25	0.3	41	24
4	Park Mc-Clennan	0.25	0.3	28	24
5	Butterworth	0.27	0.2875	71	24
6	Elliptical	0.27	0.2875	8	24
7	Least Square	0.27	0.2875	172	24
8	Park Mc-Clennan	0.27	0.2875	119	24
9	Elliptical	0.27	0.29	13	24
10	Least Square	0.27	0.29	326	24
11	Park Mc-Clennan	0.27	0.29	189	24

As can be seen in Table VI, Bsolo finds the minimum solutions for all instances under all models, where MiniSat+ cannot conclude with the minimum solution for filters 6 and 9 under the minimization of operations model in an hour. We note that even if the minimum solution is obtained for filter 6 by MiniSat+, it

TABLE VIII
SUMMARY OF THE RESULTS OF THE EXACT AND HEURISTIC ALGORITHMS AND THE 0–1 ILP PROBLEM SIZES

	[19]		[3]		[4]		ASSUME-A		Exact			0-1 ILP Problem Size		
Filter	adder	stp	adder	stp	adder	stp	adder	stp	adder	stp	CPU	vars	cons	optvars
1	26	4	26	7	31	5	24	4	21	5	6244.2	50732	202698	2158
2	10	3	11	7	11	4	11	5	10	4	27.7	3410	12303	350
3	58	4	61	7	67	6	52	4	*77*	*4*	*86400.1*	58652	230136	3269
4	45	4	46	7	48	6	43	4	*45*	*4*	*86400.1*	20572	77736	1703
5	61	4	57	6	61	6	54	4	*63*	*4*	*86400.1*	81641	324765	3984
6	14	4	15	7	16	5	16	5	12	5	2387.7	27614	108062	1266
7	178	4	167	5	203	6	156	5	*228*	*5*	*86400.1*	46959	183081	4037
8	136	4	137	6	158	6	124	5	*192*	*4*	*86400.1*	74334	294575	4905
9	24	4	24	6	27	6	23	4	*23*	*4*	*86400.1*	34969	137129	1746
10	266	4	238	5	240	6	211	5	*249*	*5*	*86400.1*	38742	150786	3802
11	199	4	204	6	223	5	176	4	*247*	*5*	*86400.1*	55816	218351	4820
Total	1017	43	986	69	1085	61	890	49	1167	49	$>$ 8 days	493441	1939622	32040

could not prove that the found solution is the minimum solution. However, we note that the minimization of partial terms model is more appropriate for MiniSat+, since this model includes fewer optimization variables with respect to the minimization of operations model. In addition, the minimization of operations model is more appropriate for Bsolo than MiniSat+, since Bsolo incorporates problem reduction techniques from both sides, i.e., SAT algorithms and branch-and-bound algorithms.

As the third experimental set, we used filter instances introduced in [30] to find out the limitations of the exact algorithm. In Table VII, the filter instances where coefficients are defined in 24 bit-width are given. We compare the results of the exact algorithm with heuristics for the unconstrained maximum sharing problem where filter coefficients are defined under CSD representation in Table VIII. In this table, the 0–1 ILP problem size of each filter is given under the problem sizes columns. MiniSat+ was used to obtain the minimum solutions, and the allowed CPU time was determined as 1 day. Again, the italic results indicate that an optimal, rather than the minimum, solution is obtained within the given CPU time limit. We note that the results of heuristic algorithms are obtained with very low computational effort.

In this experiment, we observe that the minimum solutions of three out of 11 filters, i.e., filters 1, 2, and 6, are obtained within the CPU time limit. However, the minimum solutions of eight filters could not be found in 1 day. We note that even if the problem size of filter 1 is greater than the problem size of filter 4, a minimum solution could not be obtained for filter 4. This shows that the size of the 0–1 ILP problem and the hardness of the problem heavily depend on the filter coefficients. We observe that for the filter instances where the minimum solutions are not obtained, the found solution by the exact algorithm can be far from the solutions that are obtained using a heuristic, such as filters 7 and 8. On overall instances, ASSUME-A finds the best optimum solutions among these algorithms. This experiment also shows that the use of a heuristic algorithm is indispensable when an exact algorithm could not conclude to obtain the minimum solution.

VII. CONCLUSION

We have described an exact algorithm that computes the minimum number of adder/subtracter modules in the implementation of MCM structures by maximizing the sharing of common subexpressions. The algorithm can handle binary, CSD, and MSD representations for the coefficients. Delay constraints can be included in the model so that a user-specified delay can be accommodated. A heuristic variation of this algorithm is presented and shown to be extremely competitive. We presented results for digital filter synthesis, where we demonstrate that the exact algorithm can be applied to real-sized problems. We compare our heuristic algorithm with previously proposed heuristics and showed that, although these algorithms perform reasonably well, our heuristic based on the exact model is significantly superior.

An interesting result demonstrated in this paper is that the binary representation allows for a greater amount of sharing, hence producing more area-efficient implementations for MCM problems than the CSD and MSD representations. However, when seeking minimum delay solutions, the MSD representation should be used.

The algorithms proposed in this paper can be extended to handle general number representation of constants by using the techniques described in [36] and [37] to be competitive with graph-based algorithms. As future work, we are currently working on the implementation of an exact graph-based algorithm.

REFERENCES

[1] 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.

[2] P. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-32, no. 5, pp. 1037–1041, Oct. 1984.

[3] R. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 43, no. 10, pp. 677–688, Oct. 1996.

[4] I.-C. Park and H.-J. Kang, "Digital filter synthesis based on minimal signed digit representation," in *Proc. Des. Autom. Conf.*, 2001, pp. 468–473.

[5] 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 Des.*, 2005, pp. 13–16.

[6] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Optimization of area under a delay constraint in digital filter synthesis using SAT-based integer linear programming," in *Proc. Des. Autom. Conf.*, 2006, pp. 669–674.

[7] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "ASSUMEs: Heuristic algorithms for optimization of area and delay in digital filter synthesis," in *Proc. Int. Conf. Electron., Circuits Syst.*, 2006, pp. 748–751.

[8] H. Garner, "Number systems and arithmetic," *Adv. Comput.*, vol. 6, pp. 131–194, 1965.

[9] 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.

[10] 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, Jan. 1995.

[11] N. Een and N. Sorensson, "Translating pseudo-Boolean constraints into SAT," *J. Satisfiability, Boolean Model. Comput.*, vol. 2, pp. 1–26, 2006.

[12] N. Een and N. Sorensson, "An extensible SAT-solver," in *Proc. Theory Appl. Satisfiability Testing*, 2004, vol. 2919, pp. 502–518.
[13] V. Manquinho and J. Marques-Silva, "Effective lower bounding techniques for pseudo-Boolean optimization," in *Proc. IEEE/ACM Des., Autom. Test Eur. Conf.*, Mar. 2005, pp. 660–665.
[14] *Pseudo-Boolean Evaluation PB'06*. [Online]. Available: http://www.cril.univ-artois.fr/pb06/
[15] M. Mehendale, S. Sherlekar, and G. Venkatesh, "Techniques for low power realization of FIR filters," in *Proc. Des. Autom. Conf.*, 1995, pp. 404–416.
[16] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with power-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. 36, no. 7, pp. 1044–1047, Jul. 1989.
[17] A. Nannarelli, M. Re, and G. Cardarilli, "Tradeoffs between residue number system and traditional FIR filters," in *Proc. Int. Symp. Circuits Syst.*, May 2001, pp. 305–308.
[18] R. Hartley, "Optimization of canonic signed digit multipliers for filter design," in *Proc. Int. Symp. Circuits Syst.*, 1991, pp. 1992–1995.
[19] A. Hosangadi, F. Fallah, and R. Kastner, "Reducing hardware complexity of linear DSP systems by iteratively eliminating two-term common subexpressions," in *Proc. IEEE Asia South Pacific Des. Autom.*, Jan. 2005, pp. 523–528.
[20] 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.
[21] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 1, pp. 58–68, Jan. 1999.
[22] L. Aksoy, E. O. Gunes, E. Costa, P. Flores, and J. Monteiro, "Effect of number representation on the achievable minimum number of operations in multiple constant multiplications," in *Proc. IEEE Workshop Signal Process. Syst.*, 2007, pp. 424–429.
[23] A. Dempster and M. Macleod, "Using all signed-digit representations to design single integer multipliers using subexpression elimination," in *Proc. Int. Symp. Circuits Syst.*, 2004, pp. 165–178.
[24] A. Dempster and M. Macleod, "Digital filter design using subexpression elimination and all signed-digit representations," in *Proc. Int. Symp. Circuits Syst.*, 2004, pp. 169–172.
[25] A. Dempster and M. Macleod, "Constant integer multiplication using minimum adders," *Proc. Inst. Electr. Eng.—Circuits, Devices Systems*, vol. 141, no. 5, pp. 407–413, Oct. 1994.
[26] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circuits, Syst. and Signal Process.*, vol. 25, no. 2, pp. 225–251, Apr. 2006.
[27] D. Bull and D. Horrocks, "Primitive operator digital filters," *Proc. Inst. Electr. Eng G—Circuits, Devices Systems*, vol. 138, no. 3, pp. 401–412, Jun. 1991.
[28] A. Dempster and M. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 9, pp. 569–577, Sep. 1995.
[29] Y. Voronenko and M. Pschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, May 2007.
[30] A. Hosangadi, F. Fallah, and R. Kastner, "Simultaneous optimization of delay and number of operations in multiplierless implementation of linear systems," in *Proc. Int. Workshop Logic Synthesis*, 2005.
[31] E. Costa, P. Flores, and J. Monteiro, "Maximal sharing of partial terms in MCM under minimal signed digit representation," in *Proc. IEEE Eur. Conf. Circuit Theory Des.*, 2005, pp. 221–224.
[32] A. Dempster, S. Demirsoy, and I. Kale, "Designing multiplier blocks with low logic depth," in *Proc. Int. Symp. Circuits Syst.*, 2002, pp. 773–776.
[33] H.-J. Kang, H. Kim, and I.-C. Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders," in *Proc. Int. Conf. Comput.-Aided Des.*, 2000, pp. 51–54.
[34] M. Velev, "Efficient translation of Boolean formulas to CNF in formal verification of microprocessors," in *Proc. IEEE Asia South Pacific Des. Autom.*, 2004, pp. 310–315.
[35] P. Flores, H. Neto, and J. Marques-Silva, "An exact solution to the minimum size test pattern problem," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 6, no. 4, pp. 629–644, Oct. 2001.
[36] O. Gustafsson and L. Wanhammar, "ILP modelling of the common subexpression sharing problem," in *Proc. Int. Conf. Electron., Circuits Syst.*, 2002, pp. 1171–1174.
[37] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Minimum number of operations under a general number representation for digital filter synthesis," in *Proc. IEEE Eur. Conf. Circuit Theory Des.*, 2007, pp. 252–255.

Levent Aksoy (S'06) was born in Istanbul, Turkey, on September 19, 1976. He received the M.S. degree in electronics and communication engineering from Istanbul Technical University (ITU) in 2003. He is currently working toward the Ph.D. degree in electronics at ITU.

Since 2001, he has been a Research Assistant with the Division of Circuits and Systems, Faculty of Electrical and Electronics Engineering, ITU. During 2005–2006, he was a Visiting Researcher with the Algorithms for Optimization and Simulation Research Unit, Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, Portugal. His research interests include satisfiability algorithms, pseudo-Boolean optimization, and electronic design automation problems.

Eduardo da Costa received the five-year engineering degree in electrical engineering from the University of Pernambuco, Recife, Brazil, in 1988, the M.Sc. degree in electrical engineering from the Federal University of Paraiba, Campina Grande, Paraíba, Brazil, in 1991, and the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2002. Part of his doctoral work was developed at the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, Portugal.

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

Paulo Flores (S'92–M'00) 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 (S'93–M'96) received the five-year engineering degree and the M.Sc. degree in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 1989 and 1992, respectively, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1996.

Since 1996, he has been with Instituto Superior Técnico, Technical University of Lisbon, where he is currently an Associate Professor in the Department of Computer Science and Engineering. He is currently the Director of the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon. His main interests are computer architecture and CAD for VLSI circuits, with emphasis on synthesis, power analysis, and low-power and design validation.

Dr. Monteiro received the Best Paper Award from the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS in 1995. He has served on the Technical Program Committees of several conferences and workshops.