A New Algorithm for Elimination of Common Subexpressions

R. Paško, P. Schaumont, V. Derudder, S. Vernalde, *Member, IEEE*, and D. Ďuračková

***Abstract*—The problem of an efficient hardware implementation of multiplications with one or more constants is encountered in many different digital signal-processing areas, such as image processing or digital filter optimization. In a more general form, this is a problem of common subexpression elimination, and as such it also occurs in compiler optimization and many high-level synthesis tasks. An efficient solution of this problem can yield significant improvements in important design parameters like implementation area or power consumption. In this paper, a new solution of the multiple constant multiplication problem based on the common subexpression elimination technique is presented. The performance of our method is demonstrated primarily on a finite-duration impulse response filter design. The idea is to implement a set of constant multiplications as a set of add-shift operations and to optimize these with respect to the common subexpressions afterwards. We show that the number of add/subtract operations can be reduced significantly this way. The applicability of the presented algorithm to the different high-level synthesis tasks is also indicated. Benchmarks demonstrating the algorithm's efficiency are included as well.**

***Index Terms*— Common subexpression elimination, DSP synthesis, optimization, resource sharing.**

I. Introduction

The advent of consumer applications demanding very high data throughputs like digital television requires high-speed components such as digital filters. Because of the speed, programmable solutions such as digital signal-processing (DSP) cores cannot be considered a satisfying solution in dealing with these problems. Rather, an application-specific approach in hardware is necessary, thus efficient very-large-scale integration (VLSI) synthesis methods are needed.

The core of many VLSI design tasks is the multiplication of a variable by a set of constants (digital filtering, image processing, linear transforms, etc.). The optimization of these multiplications can lead to important improvements in various design parameters like area or power consumption. In this paper, an algorithm for efficient solution of the multiple constant multiplication problem (MCM, as defined in [3]) is presented. Common subexpression elimination (CSE) as a way to tackle the MCM problem was already proposed by various authors [3]–[5], primarily as a possible method for the optimization of finite-duration impulse response (FIR) filter area through the reduction of the multiplier block logic. In [3] also, a number of other applications in which the MCM transformation can be successfully applied were proposed. In this work, we will introduce an algorithm able to solve the CSE problem in an efficient way.

Manuscript received March 31, 1998; revised August 31, 1998. This work was supported by COPERNICUS under Project SISPAS CP94-0223. This paper was recommended by Associate Editor G. Borriello.

R. Paško and D. Ďuračková are with the Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Bratislava, Slovakia.

P. Schaumont, V. Derudder, and S. Vernalde are with the Interuniversity Microelectronics Center (IMEC), Leuven 3001 Belgium.

Publisher Item Identifier S 0278-0070(99)00812-X.

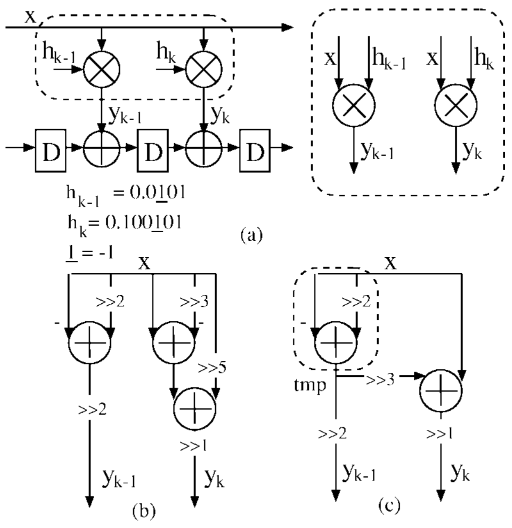


Fig. 1. CSE in FIR filter design.

The idea of CSE can be demonstrated on a FIR filter design example shown in Fig. 1. The optimization procedure targets the minimization of the multiplier block area [Fig. 1(a)]. After expressing the coefficients in a canonical signed digit (CSD) format [1], [2], in order to reduce the total number of nonzero bits (thus also the additions/subtractions necessary), an add-shift expansion is performed, as shown in Fig. 1(b). The goal of CSE is to identify the bit patterns that are present in the coefficient set more than once. Since it is sufficient to implement the calculation of the multiple identical expressions only once, the resources necessary for these operations can be shared. The pattern $\underline{1}01$ in the example in Fig. 1 is present twice, so an optimized structure shown in Fig. 1(c) can be implemented instead of the original one. The second occurrence of the pattern is removed, and only the result is used for the further calculation. In general, the goal of CSE can be defined as follows.

1) Identify multiple patterns in the coefficient set.
2) Remove these patterns and calculate them only once.

The problem to solve is how to identify the "proper" patterns for elimination so the optimization impact can be maximal. Our algorithm is based on a combination of an exhaustive search technique with a steepest descent (or greedy) approach in order to select the "proper" patterns for elimination. Thanks to an efficient implementation of the algorithm combined with some simplification techniques to speed up the processing of large tasks, very satisfactory runtimes are also achieved. The rest of this paper is structured as follows. In Section II, we discuss the related work. In the next section, we give a formal description of the problem and indicate our goal. Also, some considerations concerning the problem complexity are included. In Section IV, an in-depth discussion of our algorithm is given. In Section V, we indicate an implementation strategy of the algorithm for different design tasks (we concentrate on transposed- and direct-form FIR filters and matrix multiplication), followed by experimental results in Section VI. Section VII presents a comparison with the related work, and Section VIII states the conclusion.

II. Related Work

The idea of the consta+nt multiplications optimization (generally) or FIR filter area minimization (specifically) by common subexpressions sharing was already considered by several authors [3]–[7]. In this section, we will briefly introduce their approaches with an appropriate reference. In [3], a bipartite matching algorithm was used to identify the common subexpressions for elimination, and there were shown also numerous examples different from FIR filter optimization on which it can be successfully applied. In [4], an algorithm for the identification and elimination of only two-nonzero-bit subexpressions was proposed, but the mechanics was extended to direct-form FIR filters as well (the remaining papers consider only transposed-form FIR filters). In [5], an elimination of 2-bit subexpression was also proposed, but as an additional criterion in the subexpression identification process, an estimation of a latch-count improvement was used as well, which introduces the issues related with timing. Both [4] and [5] were specifically targeting the optimization of the FIR filters area. These three papers ([3]–[5]) used generally the same idea (common subexpression elimination) as a basic optimization strategy. References [7] and [6] applied a different approach. In these works, the whole multiplier block is synthesized using similar graph synthesis algorithms ([6] can be considered an extension of [7]). Despite this difference, the results obtained by these works are of course of interest for this paper in order to compare the effectiveness of both approaches.

III. Problem Analysis

In this section, we will define the goal of the CSE technique formally as a matrix transformation. Afterwards, a short discussion on behalf of the problem complexity will be given, and a simple heuristic to tackle the complexity issue will be proposed.

A. Problem Definition

The problem we are targeting can be formally described as a multiplication-free linear transform. In general, a multiplication-free linear transform is defined by the equation $Y = MX$, where Y and X are *n-dimensional* vectors and M is an $n \times n$ matrix containing only 1, -1, and 0. In this form, X represents the variable input while the matrix M indicates the set of constants. As will be shown later, this formalism can be extended to a number of different problems. Consider the example of a multiplication-free linear transform described in (1)

$$\mathbf{Y} = \mathbf{MX} = \begin{pmatrix} \mathbf{1001}0100 \\ \mathbf{1001}0000 \\ 01110010 \\ \mathbf{1001}0001 \end{pmatrix} \mathbf{X}. \tag{1}$$

The product $X_0 + X_3$ has to be calculated three times during the evaluation of Y. However, the splitting of M into two matrices M_1 and M_R, as shown in (2) and (3), groups the partial products in question into one matrix M_1, which can be decomposed as shown in (2)

$$\mathbf{M}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} (1\,001\,000) = \mathbf{M}_{1y}\mathbf{M}_{1x} \tag{2}$$

$$\begin{aligned} \mathbf{Y} &= \mathbf{M}_{1y}\mathbf{M}_{1x}\mathbf{X} + \underbrace{\begin{pmatrix} 00000100 \\ 00000000 \\ 01110010 \\ 00000001 \end{pmatrix}}_{M_R} \mathbf{X} \\ &= \mathbf{M}_{1y}(X_0 + X_3) + \mathbf{M}_R\mathbf{X}. \end{aligned} \tag{3}$$

The formulation (3) requires the evaluation of the partial product X_0+X_3 only once. This idea of a matrix splitting for multiple identified patterns can be expressed as follows:

$$\mathbf{M} = \sum_{i=1}^{K} \mathbf{M}_i + \mathbf{M}_R. \tag{4}$$

Concerning the matrices $\mathbf{M}_i$, any row in every matrix $\mathbf{M}_i$ must be either an all-zero vector or must be equal to any nonall-zero vector in the matrix as in (2). $\mathbf{M}_R$ is the remainder in which no more multiple subexpressions could be found. The final product Y can be computed as shown in (5)

$$\mathbf{Y} = \left(\sum_{i=1}^{K} \mathbf{M}_i + \mathbf{M}_R\right)\mathbf{X} = \sum_{i=1}^{K} \mathbf{Y}_i + \mathbf{Y}_R. \tag{5}$$

A subset of the previously defined problem deserving attention occurs in the case when the relative position of the pattern within the matrix is of no importance in the pattern identification process. This is a valid assumption in the case of the X vector's being defined as in (6) with $c = 2$ or $c = 2^{-1}$. Then the matrix splitting as shown in (7) can be performed

$$\mathbf{X} = (c^0 * x_0, c^1 * x_0, \cdots, c^n * x_0)^T \tag{6}$$

$$\mathbf{M} = \underbrace{\begin{pmatrix} \mathbf{1001}0000 \\ \mathbf{1001}0000 \\ 000\mathbf{1001}0 \\ \mathbf{1001}0000 \end{pmatrix}}_{\mathbf{M}_1} + \underbrace{\begin{pmatrix} 00000100 \\ 00000000 \\ 01100000 \\ 00000001 \end{pmatrix}}_{\mathbf{M}_R}. \tag{7}$$

In order to be able to perform a decomposition of the matrix M_1 in the same way as shown in (2), additional scaling of the M_{1ys} elements as shown in (8) must be performed. This results in the scaled Y_1 vector as shown in (9)

$$\mathbf{M}_1 = \begin{pmatrix} c^0 \\ c^0 \\ c^3 \\ c^0 \end{pmatrix} (1\,001\,000) = \mathbf{M}_{1ys}\mathbf{M}_{1x} \tag{8}$$

$$\begin{aligned}\mathbf{Y}_1 &= M_{1ys}\,\overbrace{(X_0 + X_3)}^{S_0} \\ &= (S_0, S_0, c^3 S_0, S_0)^T.\end{aligned} \tag{9}$$

Since $c = 2^{-1}$ or $c = 2$, the scaling of the Y_i vector elements is equivalent to the bit shifts, so it can usually be performed very efficiently in either software or hardware.

To demonstrate the matrix-splitting technique, we use the FIR filter subblock optimization example shown in Fig. 1(b) and (c). This can be described in terms of the matrix splitting as follows. The original structure is represented by (10)

$$\begin{aligned} y_{k-1} &= 0.01\underline{1}0100_{\text{CSD}} \times x \\ &= -x \gg 2 + x \gg 4 \\ &= (\mathbf{-x + x \gg 2}) \gg 2 \\ y_k &= 0.100\underline{1}01_{\text{CSD}} \times x \\ &= x \gg 1 - x \gg 4 + x \gg 6 \\ &= (x + (\mathbf{-x + x \gg 2}) \gg 3) \gg 1. \end{aligned} \tag{10}$$

The elimination of the pattern $\underline{1}01$ as shown in Fig. 1(c) is described in (11)

$$\begin{aligned} \text{tmp} &= -x + x \gg 2 \\ y_{k-1} &= \text{tmp} \gg 2 \\ y_k &= (x + \text{tmp} \gg 3) \gg 1. \end{aligned} \tag{11}$$

Equation (10) can be reformulated as a multiplication-free linear transform in the following way:

$$\begin{pmatrix} \vdots \\ y_{k-1} \\ y_k \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ 00\mathbf{\underline{1}01}00 \\ 0100\mathbf{\underline{1}01} \\ \vdots \end{pmatrix} \begin{pmatrix} x \gg 0 \\ x \gg 1 \\ \vdots \\ x \gg 6 \end{pmatrix}. \tag{12}$$

Then the pattern elimination performed in (11) corresponds to the matrix splitting shown in (13). Of course, the partial product Y_1 must be rescaled according to (14)

$$\underbrace{\begin{pmatrix} \vdots \\ 00\underline{1}0100 \\ 0100\underline{1}01 \\ \vdots \end{pmatrix}}_{\mathbf{M}} = \underbrace{\begin{pmatrix} \vdots \\ 00\underline{1}0100 \\ 0000\underline{1}01 \\ \vdots \end{pmatrix}}_{\mathbf{M}_1} + \underbrace{\begin{pmatrix} \vdots \\ 0\,000\,000 \\ 0\,100\,000 \\ \vdots \end{pmatrix}}_{\mathbf{M}_1} \tag{13}$$

$$\begin{aligned} \mathbf{Y}_1 \quad &= \begin{pmatrix} \vdots \\ 2^{-2} \\ 2^{-4} \\ \vdots \end{pmatrix} (\underline{1}010\,000) \\ &= M_{1ys}\,\overbrace{(-X + X \gg 2)}^{S_0} \\ &= (\cdots, S_0 * 2^{-2}, S_0 * 2^{-4}, \cdots)^T \\ &= (\cdots, S_0 \gg 2, S_0 \gg 4, \cdots)^T. \end{aligned} \tag{14}$$

Finally, there are some terms that need to be defined.

Definition 1 (Computational Effort): The computational effort $\mathbf{R}$ is equal to the number of additions/subtractions necessary to produce the final product $\mathbf{Y}$.

To estimate the computational effort, we consider only the necessary number adders and subtractors. Shifts can be implemented in the applications we are targeting (HW) almost for free (as hardwired with only some additional wiring effort). If this is not the case, the cost of shifts should be included in the computational effort estimation as well. Concerning the use of other criteria in the computational effort estimation process (like the timing related latch-count parameter used in [5]), we prefer to keep the method as general as possible. But as will be shown later, the modification of the computational effort cost function can be done easily without significant modification of the algorithm itself. Consequently, the goal of CSE defines the following.

Definition 2 (CSE Goal): The goal of the CSE technique is to find a splitting of the matrix $\mathbf{M}$ according to (2) and (4) such that the total computational effort to produce the product $\mathbf{Y}$ is minimal.

To measure the success of the CSE optimization, we use the following.

Definition 3 (Optimization Ratio): An *optimization ratio* $O = R_i/R_o$ where R_i and R_o are the *computational efforts* before and after CSE, respectively.

The optimization ratio can be used in an alternate definition of the CSE goal equivalent to Definition 1.

Definition 4 (CSE Goal): The goal of the CSE technique is to find a splitting of the matrix $\mathbf{M}$ that maximizes the optimization ratio O.

Last, the term *frequency* or *pattern frequency* will often be used in the text.

Definition 5 (Pattern Frequency): Pattern frequency (or just *frequency*) represents the number of occurrences of a pattern in a matrix.

For example, the frequency of the pattern 1001 in (1) is equal to three [or four if the relative position of a pattern is of no importance, as shown in (6)–(8)].

In this work, we propose an algorithm to solve both outlined problems efficiently. In order to be able to clearly distinguish between these two problems later, we will define the problem described in (1) as *Problem A* and the problem shown in (6) as *Problem B.*

Of course, multiplication-free linear transform is not the only application for the CSE technique. Similar problems occur in many different areas (e.g., in compiler design). The proposed algorithm might be capable of performing the common subexpression identification and elimination also for tasks that are quite different from multiplication-free linear transforms, but this is outside the scope of this paper.

B. Problem Complexity

In this section, a short discussion about the practical feasibility of the CSE goal is given. The problem in question is as follows: by each pattern elimination, we are likely to lose also other patterns due to the sharing of nonzero bits.

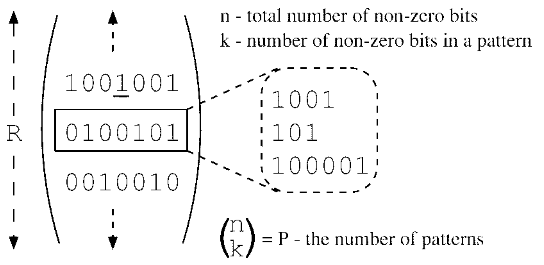

n - total number of non-zero bits
k - number of non-zero bits in a pattern
1001001
0100101
0010010
R
1001
101
100001
$\binom{n}{k}$ = P - the number of patterns

Fig. 2. Pattern statistic creation.

For example, during an elimination of the pattern **1001** from the row **1001**0100, also the pattern 101 is lost, so every pattern elimination can change the frequencies of other ones significantly. The graph synthesis problem is claimed to be NP-complete [7], but we are not aware of the existence of such a proof for CSE. As a consequence, since there is no known efficient algorithm to solve the problem exactly, we propose a simple heuristics for the CSE problem in this paper. It is based on a steepest descent approach, i.e., we choose in every matrix-splitting iteration a splitting such that the computational effort, minimization is maximal (for that iteration). This of course does not guarantee finding the optimal solution in a global sense, but the results have proven the viability of this approach. Another issue is the complexity of an algorithm creating the statistics of the available patterns for elimination, which is necessary in order to realize the proposed heuristics (see Fig. 2). The number of patterns with k ones in a single row is $P \leq \binom{n}{k}$, so the total effort to create the statistics containing R rows is equal to $O(\sum_{i=1}^{R}\binom{n_i}{k_i}) = \sum_{i=1}^{R} P_i \sim R * P_{\text{av}})$. It is obvious that the values n and k are the crucial factors in the complexity issue, since the combinatorial number can often rise significantly over the value R. Fortunately, for a number of problems, these values (n and k) are relatively small compared to R (e.g., in FIR filters, $n \sim$ the number of bits in the coefficients and $k \sim$ the number of nonzero bits), so it is possible to create the complete pattern statistics. An additional strategy to tackle this issue will be proposed in the next section for the cases when this does not apply.

IV. CSE Algorithm

In this section, we will give a detailed description of an algorithm able to solve Problem B (i.e., the elimination of patterns with arbitrary shifts within the input matrix). Afterwards, we will discuss the modifications necessary for the algorithm to be able to solve Problem A as well. As indicated in the previous section, the algorithm must accomplish the following tasks.

1) Identify the presence of multiple patterns in the input matrix.
2) Select one pattern for elimination.
3) Eliminate all occurrences of the selected pattern.

This should be iteratively repeated until there are no more multiple patterns present. The complete algorithm flowgraph is given in Fig. 3. The input parameter N represents the number of nonzero bits in the examined patterns. In the first step, an exhaustive search for all possible multiple N-bit patterns is

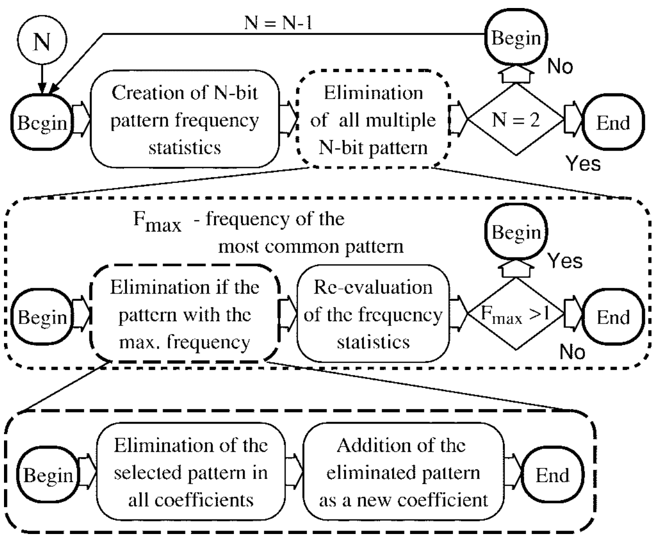

N = N-1
N
Begin
Creation of N-bit pattern frequency statistics
Elimination of all multiple N-bit pattern
N = 2
No
Yes
End
Fmax - frequency of the most common pattern
Elimination if the pattern with the max. frequency
Re-evaluation of the frequency statistics
Fmax >1
Elimination of the selected pattern in all coefficients
Addition of the eliminated pattern as a new coefficient

Fig. 3. CSE algorithm flowgraph.

performed and complete statistics of the pattern frequencies are created. Since many different patterns will occur more than once, some criterion must be used to select the one for elimination. We use the *steepest descent approach*, i.e., select always the pattern with the highest frequency. In the second step, all occurrences of the selected pattern are removed (i.e., the nonzero bits are replaced by zeros), and the pattern is added as a new line at the bottom of the matrix so it can be searched for the multiple patterns with smaller N later. Last, since the removal of a pattern must influence the total frequency statistics of the remaining ones, the global frequency statistic holding the complete information has to be adjusted to properly reflect the changes. After all multiple patterns with N nonzero bits are processed, the whole cycle is repeated for $N-1, N-2 \cdots 2$ nonzero bit patterns. A detailed discussion will be further concentrated on the following problems:

A) pattern identification;
B) pattern selection;
C) frequency statistics management;
D) adaptation of the algorithm for Problem A;
E) viability of the algorithm for large tasks;
F) applicability for similar CSE tasks.

A. Pattern Identification

Since an exhaustive search is performed, all possible combinations of N-bit patterns must be examined. The algorithm must also be able to detect a "collision" between two equal patterns, when these share at least one nonzero bit. Since such a pattern can be eliminated only once, it must be taken into account also during the frequency statistics creation phase (Fig. 4). For example, the pattern 1 010 101 in Fig. 4 has two valid 2-bit patterns, as shown in Table I, because only two patterns can be identified without conflicts [Fig. 4(a)]. The interleaving of two patterns without common nonzero bits, on the contrary, does not influence the statistics, as shown in Fig. 4(b).

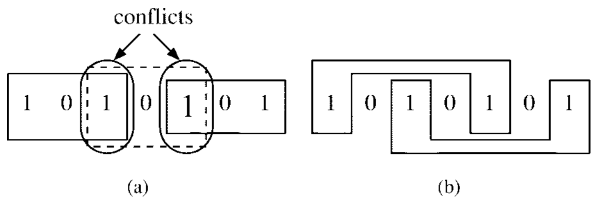


Fig. 4. Conflicts between patterns.

TABLE I
LEGAL 2-BIT PATTERNS IN "1010101"

Bit pattern	*Frequency*
101	2
10001	2
1000001	1

B. Pattern Selection

In case some patterns with the same frequency are present in the frequency statistics, a decision criterion must be provided to choose only one. The one we have chosen originated from the assumption that the optimized structure will be integrated on silicon. If two (or more) patterns with the same frequency occur, the shortest one is selected. Implementation of an addition/subtraction on silicon will result in an adder/subtractor with word length depending on the length of the pattern. Thus, the selection of the shorter one will result in a smaller adder/subtractor structure. In the case of 2-bit patterns, one additional criterion was introduced: the preference of adders over subtractors, i.e., the pattern 101 will be preferred over $10\underline{1}$. This can be justified by the same reasoning as before, since a subtractor structure is more expensive than an adder (in terms of the area). In a case where the algorithm would be used for tasks aiming at a different goal, another criterion might be preferred.

C. Frequency Statistics Management

Since an exhaustive search is performed, attention must be paid to its implementation strategy. For the N-bit statistics, a binary tree with the patterns as the keys is a perfectly suited structure. The complete statistics can be created simply by processing the input matrix row after row. One problem is caused by the fact that the same pattern can be present in a row multiple times, so the large binary tree must be searched for the same pattern multiple times. To avoid this, an alternative approach in the global statistics generation process was used as shown in Fig. 5. First, a local tree holding the frequency statistics of a single row is created [Fig. 5(b)], and this local tree is used to update the global statistics. This way, searching in the global statistics tree is minimized.

After pattern elimination, the frequencies of the other patterns can also change, and therefore the global frequency statistics must be reevaluated. Since the creation of a new global statistics after each pattern elimination is not a feasible solution, an alternative method of the global statistics adjustment must be found (Fig. 6). After each pattern elimination, a local statistics tree is created holding the information about the frequency changes of the remaining patterns in the processed

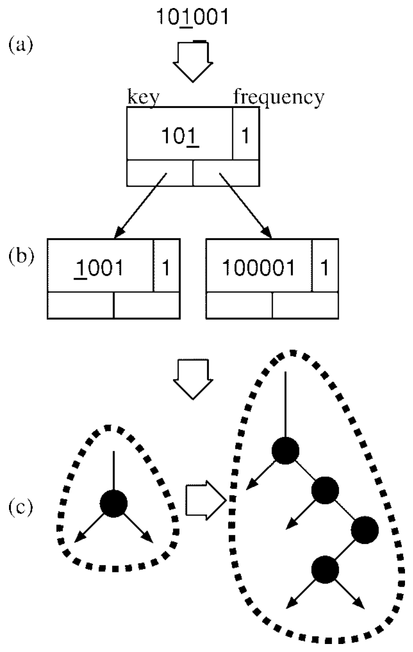


Fig. 5. Creation of 2-bit frequency statistics: (a) processed row, (b) local tree with the single-row frequency statistics, and (c) global statistics update.

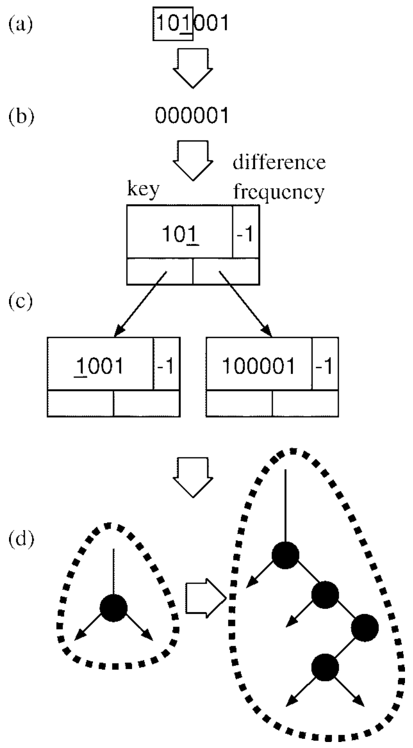


Fig. 6. Frequency statistics reevaluation: (a) original row, (b) row after elimination of pattern $10\underline{1}$, (c) difference statistics, and (d) global statistics update.

matrix row [Fig. 6(c)]. These difference statistics can be used to update the global statistics tree, which results in a much smaller number of operations on the large global frequency statistics since it has to be accessed only for the patterns that frequency actually changed. This way, the global tree has to be created only once at the beginning of each iteration (see Fig. 3).

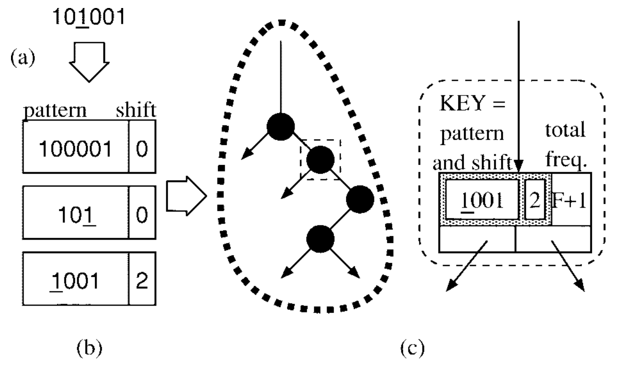


Fig. 7. Algorithm modification for Problem A.

D. Algorithm Modification for Problem A

It is also possible to use the previously described algorithm to perform a CSE optimization for the case when the position of a pattern within the matrix row is of importance (i.e., Problem A), but some changes must be made to take this into account. First, together with the pattern, also its position within the row must be used as a key during the construction of a binary tree. Second, since in this case all the patterns in a row are unique (at least their positions within a row must be different), it is not necessary to use the local statistics during the creation and maintaining of the global statistics (see Fig. 7). Apart from this, both algorithms can be identical. To distinguish between the two algorithms, we will mark them as *Algorithm I* (to solve Problem A) and *Algorithm II* (to solve Problem B).

E. Algorithm Modification for Large Tasks

To modify the algorithms to be able also to solve large tasks efficiently, first the bottlenecks must be identified. Let us assume that the processed matrix has the dimension $m \times n$. If the number of rows would be doubled to $2m$, then approximately a double amount of keys would have to be inserted into the binary tree during the k-bit statistics creation. On the contrary, if the number of columns would be doubled, then the number of patterns processed for each row would rise from $\binom{n}{k}$ to $\binom{2\times n}{k}$, which differs by a factor of approximately 2^k. Thus the total computing time in the second case can rise even in orders of magnitude. This limits the number of columns that can be processed by the algorithm. This analysis, however, also shows the possible recipe for tackling the problem of large inputs

$$\overbrace{\begin{pmatrix} h_{00} & \cdots & h_{0(n/2)} \\ & \vdots & \\ h_{m0} & \cdots & h_{m(n/2)} \end{pmatrix} \begin{pmatrix} h_{0(n/2+1)} & \cdots & h_{0n} \\ & \vdots & \\ h_{m(n/2+1)} & \cdots & h_{mn} \end{pmatrix}}^{\begin{pmatrix} h_{00} & \cdots & \cdots & h_{0n} \\ \vdots & & & \vdots \\ h_{m0} & \cdots & \cdots & h_{mn} \end{pmatrix}}. \tag{15}$$

If we would split the matrix as shown in (15) and both parts would be processed separately, we would gain an execution time at the cost of the detection of the patterns that cross the split boundary and the adders necessary to add both split parts together again. On the other hand, this process of matrix splitting can be applied iteratively again and again until acceptable runtimes can be achieved, so matrices of orders 1000×1000 or even higher can be optimized this way.

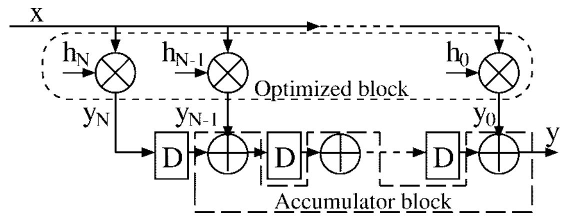


Fig. 8. Transposed-form FIR filter.

F. Applicability for Arbitrary CSE Tasks

Both of the previously defined algorithms can be applied (with some modification of the pattern generation function) to a matrix containing arbitrary elements as long as these are lexically ordered, since it is necessary to evaluate the relations $\langle\, , \,\rangle$ and $=$ during the binary-tree construction. No other restrictions are put on the matrix elements, so these can be numbers, algebraic elements, etc., which opens a whole new field of possible applications.

V. Application of CSE Algorithm

In this section, we will indicate several possible applications of the CSE algorithm for the optimization of some commonly faced design tasks. We will discuss the optimization of FIR filters (in both transposed and direct form) and linear transforms in general, as well as matrix multiplication. All those tasks are quite common in areas such as telecommunications and DSP (filters), image processing (matrix multiplication), etc.

A. FIR Filters—Transposed Form

Fig. 8 shows a transposed-form FIR filter. Let us express the coefficients $h_0 \cdots h_N$ in their binary (or CSD) form as shown in (16)

$$h_k = \sum_{i=0}^{B} h_{ki} \times 2^{-i}. \tag{16}$$

The multiplier block can be written in the form of a multiplication-free linear transform (17), since the elements of M consist of 1, 0 in the case of a binary representation or 1, 0, -1 in the case of CSD form

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} h_{00} & \cdots & h_{0B} \\ h_{10} & \cdots & h_{1B} \\ \vdots & & \vdots \\ h_{N0} & \cdots & h_{NB} \end{pmatrix} \begin{pmatrix} x \gg 0 \\ x \gg 1 \\ \vdots \\ x \gg N \end{pmatrix}. \tag{17}$$

In this case, (6) is satisfied with the constant $c = 2^{-1}$; thus Algorithm II can be used to optimize the multiplier block, and the scaling of the results can be implemented as hardwired shifts for free. An example of a transposed-form FIR filter optimization is given in Fig. 1 and (10)–(14).

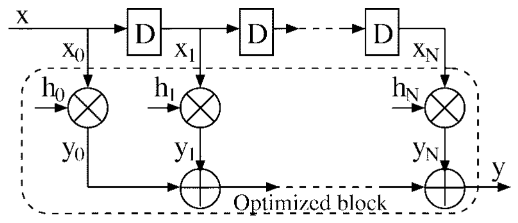


Fig. 9. Direct-form FIR filter.

B. FIR Filters—Direct Form

A block scheme of a direct-form FIR filter is shown in Fig. 9. After binary (CSD) expansion of coefficients (as in the previous section), the outputs y_i can be calculated according to (18) and the final result y according to (19)

$$\begin{aligned} y_0 &= h_{00}x_0 + \cdots + h_{0B}x_0 \gg B \\ y_1 &= h_{10}x_1 + \cdots + h_{1B}x_1 \gg B \\ \vdots\ &\vdots \qquad\qquad \vdots \\ y_N &= h_{N0}x_N + \cdots + h_{NB}x_N \gg B \end{aligned} \tag{18}$$

$$y = \sum_{i=0}^{N} y_i. \tag{19}$$

Equation (18) unfortunately cannot be represented as a multiplication-free linear transform, but it is possible to reorder it in such a way. Let us calculate the sums of columns instead of rows in (18). We obtain the set of equations in (20) and (21)

$$\begin{aligned} y'_0 &= (h_{00}x_0 + \cdots + h_{N0}x_N) \gg 0 \\ y'_1 &= (h_{01}x_0 + \cdots + h_{N1}x_N) \gg 1 \\ &\vdots \qquad\qquad \vdots \\ y'_B &= (h_{0B}x_0 + \cdots + h_{NB}x_N) \gg B \end{aligned} \tag{20}$$

$$y = \sum_{i=0}^{B} y'_i. \tag{21}$$

This set of equations can be expressed as a multiplication-free linear transform as shown in (22) and (23)

$$\begin{pmatrix} y''_0 \\ y''_1 \\ \vdots \\ y''_B \end{pmatrix} = \begin{pmatrix} h_{00} & \cdots & h_{N0} \\ h_{01} & \cdots & h_{N1} \\ \vdots & & \vdots \\ h_{0B} & \cdots & h_{NB} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{pmatrix} \tag{22}$$

$$y = \sum_{i=0}^{B} y''_i \gg i. \tag{23}$$

After this transform, it is possible to apply Algorithm I for the optimization. Unfortunately, the straightforward application of Algorithm I on (22) suffers from an important disadvantage. Since the order n in the resulting $m \times n$ matrix will be much larger than m, the matrix-splitting technique would be necessary. Therefore, we use an improved version of the described method. The idea is to perform the CSE on the original bit matrix (so the number of rows will be higher than the number of columns). Let us rewrite the filter output a last time as (24) and (25)

$$\begin{pmatrix} y'''_0 \\ y'''_1 \\ \vdots \\ y'''_N \end{pmatrix} = \begin{pmatrix} h_{00} & \cdots & h_{0B} \\ h_{10} & \cdots & h_{1B} \\ \vdots & & \vdots \\ h_{N0} & \cdots & h_{NB} \end{pmatrix} \begin{pmatrix} 2^0 \\ 2^{-1} \\ \vdots \\ 2^{-B} \end{pmatrix} \tag{24}$$

$$y = \sum_{i=0}^{N} y'''_i \times x_i. \tag{25}$$

In the next step, a matrix split satisfying (2) and (4) will be performed. Then the final sum can be rewritten as in (26)

$$y = \sum_{i=0}^{N} \left(\sum_{k=0}^{P} y'''_{ki} + y'''_{Ri} \right) \times x_i. \tag{26}$$

By reordering the sums, we can obtain (27) for the output of a direct-form FIR filter

$$y = \sum_{k=0}^{P} \left(y_{ki} \underbrace{\sum_{i=0}^{N} x_i}_{\text{tmp}_k} \right) + y'''_{Ri} \times x_i. \tag{27}$$

Furthermore, the intermediate results are again scalable by powers of two, so Algorithm II can be used for optimization. We will give a small example of direct-form FIR filter optimization as shown in Fig. 10

$$\begin{aligned} y &= x_0 * (0.1011)_{\text{BIN}} + x_1 * (0.1100)_{\text{BIN}} \\ &= (x_0 + (x_0 + x_0 \gg 1) \gg 2) \gg 1 + (x_1 + x_1 \gg 1) \gg 1. \end{aligned}$$

The pattern 11 is present twice, so the following optimization requiring one adder less for implementation can be performed:

$$\begin{aligned} \text{tmp}_0 &= x_0 \gg 2 + x_1 \\ y &= (x_0 + \text{tmp}_0 + \text{tmp}_0 \gg 1) \gg 1. \end{aligned}$$

This can be considered an improvement compared to the optimization proposed in [4], since in [4] a method equivalent to Algorithm I for only two-nonzero bit patterns was proposed.

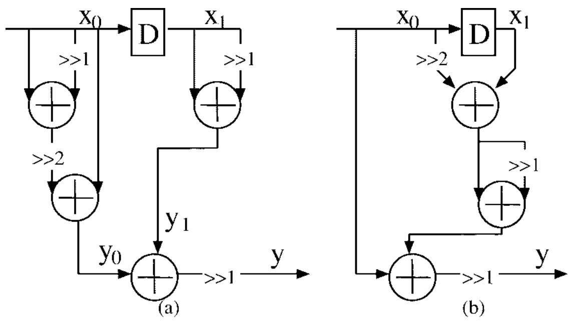


Fig. 10. Direct-form FIR filter optimization example.

C. Linear Transforms and Matrix Multiplication

Many operations in DSP, communications, or image processing can be expressed in the form of a multiplication of a matrix with either a vector or a matrix. A number of applications can be directly considered as a multiplication-free linear transform, where the application of Algorithm I is straightforward. These include some signal transforms (Walsh, Hadamard) or some error-correcting codes (Reed, BCH). In the case that the transform in question cannot be described as multiplication free [e.g., discrete Fourier transform (DFT)], an algorithm based on the multiple use of MCM was introduced in [3]. This enables one to transform any linear transform into a multiplication-free linear transform. The general linear transform can be described as in (28)

$$y_i = \sum_{j=0}^{N-1} c_{ij} x_j, \quad (i = 0, \cdots, N-1). \tag{28}$$

The conversion algorithm can be described by the following pseudocode.

1) Minimize the number of additions necessary to compute all products $c_{ij}x_j$.
2) Rebuild the input matrix using instances computed in the previous step (this will create a multiplication-free linear transform).
3) Apply Algorithm I to compute the CSE optimized structure.

If the optimized transform contains B different c_{ij} elements and we denote $C_k = c_{ij}x_j$ where $k = 1 \cdots B$, then the final transform is shown in (29). Note that values $h_{ij} \in 1, 0, -1$ (since the given C_k element is either not present in a certain row or is positive/negative). So (29) represents a multiplication-free linear transform and as such can be optimized using Algorithm I

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} h_{00} & \cdots & h_{0B} \\ h_{10} & \cdots & h_{1B} \\ \vdots & & \vdots \\ h_{N0} & \cdots & h_{NB} \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_B \end{pmatrix}. \tag{29}$$

To demonstrate the previously described technique, we will use an 8-point DFT. Since a DFT has a complex transform kernel $\mathcal{D}$, it must be split into its real and imaginary parts first [according to (30)]

$$\begin{aligned} \mathcal{Y} &= \mathcal{D}\mathcal{X} \\ &= (R + \jmath I)(X_R + \jmath X_I) \\ &= RX_R - IX_I + \jmath(RX_I + IX_R) \\ &= Y_{RR} - Y_{II} + \jmath(Y_{RI} + Y_{IR}). \end{aligned} \tag{30}$$

The resulting real and imaginary kernels R and I are shown in (31)

$$R = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & A & 0 & \underline{A} & \underline{1} & \underline{A} & 0 & A \\ 1 & 0 & \underline{1} & 0 & 1 & 0 & \underline{1} & 0 \\ 1 & \underline{A} & 0 & A & \underline{1} & A & 0 & \underline{A} \\ 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} & 1 & \underline{1} \\ 1 & \underline{A} & 0 & A & \underline{1} & A & 0 & \underline{A} \\ 1 & 0 & \underline{1} & 0 & 1 & 0 & \underline{1} & 0 \\ 1 & A & 0 & \underline{A} & \underline{1} & \underline{A} & 0 & A \end{pmatrix}$$

$$\begin{aligned} I &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & A & 1 & A & 0 & \underline{A} & \underline{1} & \underline{A} \\ 0 & 1 & 0 & \underline{1} & 0 & 1 & 0 & \underline{1} \\ 0 & A & \underline{1} & A & 0 & \underline{A} & 1 & \underline{A} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \underline{A} & 1 & \underline{A} & 0 & A & \underline{1} & A \\ 0 & \underline{1} & 0 & 1 & 0 & \underline{1} & 0 & 1 \\ 0 & \underline{A} & \underline{1} & \underline{A} & 0 & A & 1 & A \end{pmatrix} \\ A &= \sqrt{2}/2; \underline{A} = -A; \underline{1} = -1 \\ \mathcal{D} &= R + \jmath I. \end{aligned} \tag{31}$$

The transformation into a multiplication-free linear transform will be demonstrated only on the real-part kernel R, since its application on the imaginary part is equivalent. The resulting multiplication-free linear transform kernel H is shown in (32)

$$\begin{aligned} Y_{RR} &= HC \\ &= \begin{pmatrix} 110\,110\,110\,110 \\ 101\,001\,\underline{1}01\,001 \\ 100\underline{1}00\,100\underline{1}00 \\ 10\underline{1}00\underline{1}\underline{1}0\underline{1}00\underline{1} \\ 1\underline{1}01\underline{1}01\underline{1}01\underline{1}0 \\ 10\underline{1}00\underline{1}\underline{1}0\underline{1}00\underline{1} \\ 100\underline{1}00\,100\underline{1}00 \\ 101\,001\,\underline{1}01\,001 \end{pmatrix} \begin{pmatrix} x_{R0} \\ x_{R1} \\ Ax_{R1} \\ x_{R2} \\ x_{R3} \\ \underline{A}x_{R3} \\ x_{R4} \\ x_{R5} \\ \underline{A}x_{R5} \\ x_{R6} \\ x_{R7} \\ Ax_{R7} \end{pmatrix} \end{aligned} \tag{32}$$

where H and C are defined as in (29). The matrix H is a multiplication-free linear transform and as such can be subject to the CSE optimization by Algorithm I.

VI. Experimental Results

Both algorithms were developed in C and run on an HP-RISC workstation. First, the performance was tested on randomly generated data. In test I, Algorithm I was run on a set of matrices containing only zero and one with dimensions ranging from 16 × 8 up to 256 × 32. To evaluate the performance, we give the values of computational efforts before and after CSE (R_i and R_o, respectively) the optimization ratio O as well. Time values give the algorithm runtimes. The results are shown in Table II. The same type of tests was performed also for Algorithm II, with the results shown in Table III. For these tests, also the number of shifts before and after optimization are given (S_i and S_o), and the improvement ratio $O_S = S_i/S_o$ was calculated as well. The results indicate that the methods work better when more potential subexpressions are present (rise with the values m and n). The sudden increase of O_R in the cases of 128 × 8 and 256 × 8 can be explained by the fact that after raising the value m high enough $(m \sim 2^n)$, the whole rows start to repeat frequently, which leads to this abrupt increase of the optimization ratio. These tests also prove the statement that the increase of the value m is critical with respect to the algorithm runtimes.

To test the splitting technique for large matrices, we have run both algorithms again on random matrices with dimensions

TABLE II
ALGORITHM I PERFORMANCE TESTS

	$n = 8$			
m	R_i	R_o	O	t[s]
16	52	25	2.08	0:01
32	90	40	2.25	0:01
64	186	62	3.00	0:01
128	383	107	3.58	0:01
256	755	167	4.52	0:01
	$n = 16$			
m	R_i	R_o	O	t[s]
16	106	54	1.96	0:01
32	218	103	2.17	0:01
64	447	180	2.48	0:01
128	883	321	2.75	0:01
256	1757	550	3.19	0:01
	$n = 32$			
m	R_i	R_o	O	t[s]
16	324	126	1.86	1:42
32	479	223	2.15	6:11
64	947	418	2.27	8:47
128	1885	747	2.52	11:27
256	3792	1348	2.81	21:48

TABLE III
ALGORITHM II PERFORMANCE TESTS

	$n = 8$						
m	R_i	S_i	R_o	S_o	O_R	O_S	$t[s]$
16	52	57	21	26	2.48	2.19	0:01
32	90	103	32	45	2.81	2.29	0:01
64	186	213	50	77	3.72	2.77	0:01
128	383	442	74	133	5.18	3.32	0:01
256	755	879	103	227	7.33	3.87	0:01
	$n = 16$						
m	R_i	S_i	R_o	S_o	O_R	O_S	$t[s]$
16	106	111	41	46	2.59	2.41	0:01
32	218	230	78	90	2.59	2.56	0:01
64	447	477	127	154	3.52	3.10	0:01
128	883	941	222	280	3.98	3.36	0:01
256	1757	1887	412	542	4.26	3.48	0:01
	$n = 32$						
m	R_i	S_i	R_o	S_o	O_R	O_S	$t[s]$
16	324	238	83	87	2.82	2.73	1:00
32	479	491	150	162	3.19	3.03	4:56
64	947	976	285	314	3.32	3.11	7:23
128	1885	1943	504	562	3.74	3.46	8:13
256	3792	3905	941	1054	4.03	3.70	21:48

256 $\times$ 256, 512 $\times$ 512, and 1024 $\times$ 1024. The *Col* variable gives the number of columns the processed matrix was split into, and R_u gives the number of nonoptimizable adders due to the splitting. These are the adders used to add the submatrices together afterwards. The results are shown in Table IV and prove the viability of such an approach. It is also obvious that in the case of splitting into matrices with smaller n (such that $m \geq 2^n$), the effect described previously (the repeating of the rows) occurs again. This causes very high optimization ratios of the processed submatrices, so the adders that are unoptimizable due to splitting become a dominant part of the total number of adders after optimization ($R_u \rightarrow R_o$).

Last, the results of the optimization of the Hadamard matrix of order 1024 $\times$ 1024 with different numbers of split columns

TABLE IV
PERFORMANCE WITH MATRIX SPLITTING

Algorithm I performance with splitting						
Dim.	Col	R_i	R_u	R_o	O	t[s]
256	32	32666	7936	13037	2.5	0:08
	16		3840	12885	2.54	0:16
512	64	130742	32256	45805	2.85	0:23
	32		15872	47135	2.77	0:49
1024	128	522903	130048	160590	3.26	1:01
	64		64512	174490	2.99	3:11
Algorithm II performance with splitting						
Dim.	C	R_i	R_u	R_o	O	t[s]
256	32	32666	7936	11089	2.95	0:09
	16		3840	10473	3.12	0:15
512	64	130742	32256	39635	3.3	0:23
	32		15872	40116	3.26	0:43
1024	128	522903	130048	145640	3.59	1:07
	64		64512	152913	3.42	2:32

TABLE V
1024 $\times$ 1024 HADAMARD MATRIX OPTIMIZATION

Col.	R_i	R_u	R_o	O	t[m]
64	1047552	65512	68608	15.27	14:30
128	1047552	130048	133120	7.87	1:04

are given in Table V. The reason for the very high optimization ratios is the regular structure of the Hadamard matrix.

The second set of experiments consisted of the optimization of some real-life structures. First the transposed-form FIR filters were optimized by CSE and synthesized by means of the SYNOPSYS design compiler. In order to make the comparison of our results easier in the future, we have chosen the three filters published in [2] (examples 1 and 2—$S1$ and $S2$) and [8] (example 1—$L1$). The filters in [2] were subject to nonzero-bits minimization. We have optimized the coefficients of the filter $L1$ in the same way as in prior CSE processing, and the optimized coefficients are given in Table VI. This optimization results in a relatively small amount of adders in the multiplication block already before CSE (an average of 0.85 adders per tap coefficient in $S1$, 1.9 in $S2$, and 2.4 in $L1$). The optimization results are in Table VII. The structure was compiled into a MIETEC 0.5-μ CMOS library and optimized for area only. The area figures are divided in combinatorial area (C), part of which is subject to CSE, sequential area (S), and total area (T). The figures are given in equivalents of invertor gates. An interesting observation is that for a smaller filter, the area of the multiplication block becomes insignificant compared to the remaining registers and adders in the accumulator block (see Fig. 8), so the effect of the CSE optimization is not significant, especially if the optimized CSD coefficients are used (a similar conclusion is stated also in [6]). The experiments performed on a direct-form FIR filter showed a reduction of the multiplication block equivalent to the transposed-form FIR filter.

The third experiment was the optimization of a DFT to test the linear-transform optimization technique. We performed CSE optimization on real and imaginary kernels R and I, as defined in (30)–(32). The results are shown in Table VIII for DFT8, DFT16, and DFT32 (for DFT32, also matrix splitting into two columns was used).

TABLE VI
CSD OPTIMIZED COEFFICIENTS IN $L1$ FILTER
$h(n) = h(120 - n)$ FOR $61 \le n \le 120$

h(0) = $0.00000000000010\underline{1}0$	h(30) = $0.00000001010\underline{1}00\underline{1}0$
h(1) = $0.00000000000\underline{1}0010$	h(31) = $0.000000\underline{1}010\underline{1}01000$
h(2) = $0.000000000010\underline{1}00\underline{1}$	h(32) = $0.000000010100010\underline{1}$
h(3) = $0.0000000000\underline{1}00100$	h(33) = $0.000000000\underline{1}000101$
h(4) = $0.000000000010\underline{1}000$	h(34) = $0.0000000\text{T}000\underline{1}0\underline{1}0\underline{1}$
h(5) = $0.000000000000\underline{1}000$	h(35) = $0.0000001000010\underline{1}00$
h(6) = $0.00000000000\underline{1}00\underline{1}0$	h(36) = $0.000000\underline{1}000\underline{1}0010\underline{1}$
h(7) = $0.00000000010\underline{1}0\underline{1}01$	h(37) = $0.000000010010\underline{1}010$
h(8) = $0.000000000\underline{1}00010\underline{1}$	h(38) = 0.0000000010100101
h(9) = $0.000000000100\underline{1}00\underline{1}$	h(39) = $0.000000\underline{1}0\underline{1}0101000$
h(10) = $0.0000000000\underline{1}01000$	h(40) = $0.0000010\underline{1}001000\underline{1}0$
h(11) = $0.0000000000\underline{1}01000$	h(41) = $0.000000\underline{1}0\underline{1}010\underline{1}000$
h(12) = 0.0000000001000101	h(42) = 0.0000000001010100
h(13) = $0.00000000\underline{1}010100\underline{1}$	h(43) = 0.0000001001001001
h(14) = 0.0000000001000100	h(44) = $0.00000\underline{1}000\underline{1}010001$
h(15) = $0.000000000000\underline{1}000$	h(45) = 0.0000010001000010
h(16) = $0.000000000\underline{1}000\underline{1}00$	h(46) = $0.000000\underline{1}000\underline{1}000\underline{1}0$
h(17) = $0.0000000010000\underline{1}01$	h(47) = $0.000000\underline{1}01000\underline{1}010$
h(18) = $0.00000000\underline{1}00000\underline{1}0$	h(48) = 0.0000010100100000
h(19) = $0.00000000010010\underline{1}0$	h(49) = $0.0000\underline{1}001010\underline{1}0100$
h(20) = 0.0000000000100100	h(50) = $0.000001010100\underline{1}000$
h(21) = $0.00000000\underline{1}00\underline{1}0010$	h(51) = $0.00000000\underline{1}001010\underline{1}$
h(22) = $0.0000000010\underline{1}00001$	h(52) = $0.0000\underline{1}010000100\underline{1}0$
h(23) = $0.00000000\underline{1}0\underline{1}01000$	h(53) = $0.00010\underline{1}0\underline{1}0000100\underline{1}$
h(24) = $0.000000000010\underline{1}00\underline{1}$	h(54) = $0.000\underline{1}010001001000$
h(25) = $0.000000001001000\underline{1}$	h(55) = $0.000010\underline{1}0000\underline{1}0\underline{1}0\underline{1}$
h(26) = $0.0000000\underline{1}000\underline{1}0101$	h(56) = $0.000010\underline{1}01000\underline{1}0\underline{1}0$
h(27) = 0.0000000100010101	h(57) = $0.00\underline{1}0100100\underline{1}0\underline{1}000$
h(28) = $0.00000000\underline{1}00\underline{1}0\underline{1}00$	h(58) = $0.0010100000100\underline{1}00$
h(29) = $0.000000000\underline{1}0\underline{1}0010$	h(59) = $0.0\underline{1}010\underline{1}0\underline{1}01001000$
	h(60) = $0.0100\underline{1}010\underline{1}0\underline{1}0000\underline{1}$

TABLE VII
REAL FIR FILTER OPTIMIZATION

F	N	R_i	R_o	O_R		A_i	A_o	O_A
S1	25	11	6	1.83	C	2724	2425	1.12
					S	1861	1861	1
					T	4585	4286	1.06
S2	60	57	32	1.78	C	11475.4	9558.3	1.2
					S	3697.9	3702.1	1
					T	15173.9	13260.4	1.14
L1	121	145	58	2.5	C	29373.8	21125.5	1.39
					S	8456.5	8456.5	1
					T	37830.3	29582	1.28

TABLE VIII
DFT OPTIMIZATION RESULTS

	R_i		R_o			
	R	I	R	I	O_{re}	O_{im}
DFT8	40	24	18	8	2.2	3.0
DFT16	192	160	62	48	3.1	3.3
DFT32	864	800	280	262	3.08	3.05

VII. RELATED WORK COMPARISON

The optimization of the transposed-form FIR filters by means of CSE was already discussed by several authors. However, the exact comparison is not easy to make since all authors did make the experimental testing on different inputs. We tried to test our method on the same (or at least equivalent) data to obtain some estimate of the algorithm performance comparison.

TABLE IX
COMPARISON WITH [6]

			Dempster	Algorithm II
Filter	N	R_i	R_o	R_o
L2	63	49	22	23
L3	36	16	5	5

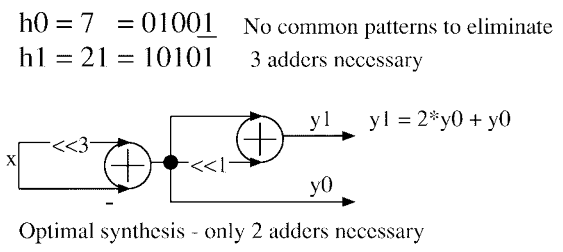

h0 = 7 = 0100$\underline{1}$
No common patterns to eliminate
h1 = 21 = 10101
3 adders necessary
x
<<3
-
<<1
y1
y1 = 2*y0 + y0
y0
Optimal synthesis - only 2 adders necessary

Fig. 11. Redundancy unidentifiable by CSE.

TABLE X
ALGORITHM II RESULTS WITH RESPECT TO THE N PARAMETER

$m \times n$	R_i	R_o				
		N=2	N=3	N=4	N=5	N=6
16×8	52	27	22	22	21	21
64×8	186	71	52	50	50	50
128×8	383	139	91	80	74	74

In [7] and [6], similar graph synthesis algorithms were used. Since [6] is an improvement over [7], we have made the comparison to [6], where experimental tests were performed on two FIR filters taken from [8] (examples 2 and 3). The results are shown in Table IX. The results are practically identical (there is one adder difference in the first filter). A possible explanation is that the presented graph synthesis algorithm is capable of finding redundancies unidentifiable to the common subexpression elimination technique (for example, see Fig. 11).

In [4], a method similar to ours based on the identification of 2-bit common subexpressions was proposed. Actually, this work can be considered an extension of [4]. However, starting an elimination of patterns with a higher number of nonzero bits than just two should give better results, as shown in Table X, where optimization results of the input set from Table II are used with respect to the input parameter N. For the FIR filters optimization, however, this is not a very important parameter, since the filter coefficients usually have only a small number of nonzero bits, which significantly reduces the chance that many N-nonzero bits expressions with $N > 2$ would be identified.

The work [5] was tested on 23 random coefficients quantized into 32 bits with an average improvement of the adder count by a factor of two. This seems to be a similar result to the one obtained in [4] or here in the case of $N = 2$, and the additional criterion based on the timing is of interest in the case where hardware (HW) implementation is targeted.

Last, in [3], the adder count improvement on the set of real-life filters of orders $N = 64, 100, 123$, and 126 ranged from 1.36 to 1.46. The values obtained for real filters by CSE were significantly higher (from 1.78 to 2.5). On the other hand, the number of shifts obtained by the bipartite matching algorithm used in [3] was much lower compared to the values from CSE optimization.

VIII. Conclusion

In this paper, a novel algorithm to solve the multiple constant multiplication problem, i.e., the optimization of the multiplication of a variable by a set of constants, was proposed. It is based on the common subexpression elimination technique and combines the exhaustive search for multiple pattern identification with a steepest descent approach for pattern selection. The results show a significant reduction in either arithmetic operations or hardware necessary to implement those operations combined with satisfactory runtimes. It can be considered an extension of the 2-bit pattern optimization technique presented in [4] since in the proposed method, no such restrictions on the patterns are given.

Comparison with related work based on the available data shows that our method yields comparable or better results in FIR filter optimization. Its major advantage is a general concept that does not restrict the use of the presented technique to the tasks proposed in this paper (FIR filter design and linear-transform optimization).

References

[1] K. Hwang, *Computer Arithmetic*. New York: Wiley, 1979.

[2] H. Samueli, “An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients,” *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1044–1057, July 1989.

[3] M. Potkonjak, M. B. Shrivasta, and P. A. Chandrakasan, “Multiple constant multiplication: Efficient and versatile framework and algorithms for exploring common subexpression elimination,” *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 151–161, Feb. 1996.

[4] M. Mehendale, S. D. Sherlekar, and G. Vekantesh, “Synthesis of multiplierless FIR filters with minimum number of additions,” in *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*. Los Alamitos, CA: IEEE Computer Society Press, 1995, pp. 668–671.

[5] R. I. Hartley, “Subexpression sharing in filters using canonic signed digit multipliers,” *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 677–688, Oct. 1996.

[6] A. G. Dempster and M. D. Mcleod, “Use of minimum-adder multiplier blocks in FIR digital filters,” *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 569–577, Sept. 1995.

[7] D. R. Bull and D. H. Horrocks, “Primitive operator digital filters,” *Proc. Inst. Elect. Eng.*, vol. 138, pt. G, no. 3, pp. 401–411, June 1991.

[8] Y. C. Lim and S. R. Parker, “Discrete coefficient fir digital filter design based upon an LMS criteria,” *IEEE Trans. Circuits Syst.*, vol. CAS-30, pp. 723–739, Oct. 1983.

R. Paško received the M.S. degree in electronics from the Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Bratislava, Slovakia, in 1994, where he is currently pursuing the Ph.D. degree in electronics.

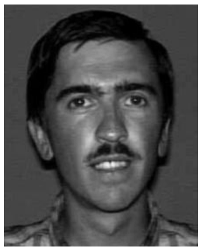
His main research interests are in the areas of VLSI design for digital signal processing and telecommunications.

P. Schaumont received the electronics and telecommunications engineering degree from the Industriele Hogeschool van het Rijk, Gent, Belgium, in 1988. He received the M.S. degree in informatics from the Rijksuniversiteit Gent, Belgium, in 1990.

Since 1992, he has been with the VLSI Systems Design Group, IMEC, Leuven, Belgium. His previous research activities included the development of code generators for the Cathedral 2/3 Silicon Compiler for medium- and high-throughput digital signal processing and the design of the Dolphin Silicon Compiler for accelerator processors. Currently, he is a Senior Research Engineer working on the development of and design automation for broadband access network modems. His research interests include the design of digital communication systems and their implementation as a system-on-chip.

V. Derudder received the degree in electrical engineering from the Katholieke Industriele Hogeschool West-Vlaanderen, Belgium, in 1990.

She is currently a Project Engineer in the VSDM Division of IMEC, Leuven, Belgium. Her professional interests are in design synthesis and design for testability.



S. Vernalde (S’88–M’90) received the electrical engineering degree from the Katholieke Universiteit Leuven, Belgium, in 1990.

He joined the IMEC Laboratory, Leuven, Belgium, in 1990, where he developed the Cathedral-2/3 datapath compiler for behavioral synthesis of high-speed DSP algorithms on multifunctional hardware accelerator processors. He is the author of the interprocess communication protocol for the communication between synchronous processors through encapsulation. He is a coauthor of *Accelerator Data-Path Synthesis for High-Throughput Signal Processing Applications* (Norwell, MA: Kluwer, 1997). Currently, he is heading the digital broadband transceivers group at IMEC and manages in this context several projects in the domain of VLSI implementation of complex digital telecommunication systems.

D. Ďuracǩová received the M.S. and Ph.D. degrees from the Faculty of Electrical Engineering, Slovak University of Technology, Bratislava, Slovakia, in 1974 and 1981, respectively.

Since 1991, she has been an Associate Professor in the Department of Microelectronics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology. The main areas of her research and teaching activities are the design of analog and digital circuits and neural networks.