Multiplierless Design of Linear DSP Transforms*

Levent Aksoy[1], Eduardo da Costa[2], Paulo Flores[3], and José Monteiro[3]

[1] INESC-ID, Lisbon, Portugal
[2] Universidade Católica de Pelotas, Pelotas-RS, Brazil
[3] INESC-ID/IST TU Lisbon, Lisbon, Portugal

Abstract. The last two decades have seen tremendous effort on the development of high-level algorithms for the multiplierless design of constant multiplications, *i.e.*, using only addition, subtraction, and shift operations. Among the different types of constant multiplications, the multiplication of a constant matrix by an input vector, *i.e.*, the constant matrix-vector multiplication (CMVM) operation, is the most general case and occurs in many digital signal processing (DSP) systems. This chapter addresses the problem of minimizing the number of addition and subtraction operations in a CMVM operation and introduces a hybrid algorithm that incorporates efficient techniques. This chapter also describes how the hybrid algorithm can be modified to handle a delay constraint. The experimental results on a comprehensive set of instances show the efficiency of the hybrid algorithms at both high-level and gate-level, in comparison to previously proposed methods.

Keywords: Constant matrix-vector multiplication, common subexpression elimination, difference method, area and delay optimization.

1 Introduction

The multiplication of data samples by constant coefficients is a ubiquitous operation and performance bottleneck in DSP systems, and can be categorized in four main classes:

1. *Single constant multiplication (SCM)*: The SCM operation realizes the multiplication of a single coefficient c by a single variable x, *i.e.*, $y = cx$. It is frequently used in the design of fast Fourier transforms (FFTs) [1] and fast discrete cosine transforms (DCTs) [2].
2. *Multiple constant multiplications (MCM)*: The MCM operation computes the multiplication of a set of m constants C by a single variable x, *i.e.*, $y_j = c_j x$ with $1 \leq j \leq m$. It occurs for instance, in the design of finite impulse response (FIR) filters in transposed form [3].

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

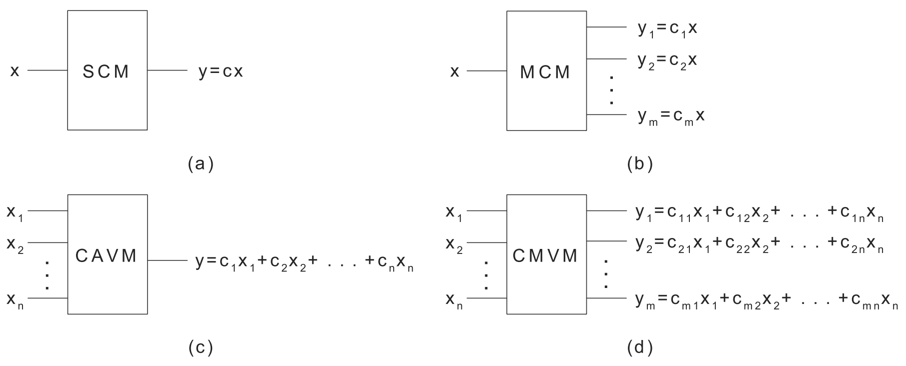


Fig. 1. Types of constant multiplications: (a) SCM; (b) MCM; (c) CAVM; (d) CMVM

3. *Constant array-vector multiplication (CAVM)*: The CAVM operation implements the multiplication of a $1 \times n$ constant array C by an $n \times 1$ input vector X, *i.e.*, $y = \sum_k c_k x_k$ with $1 \leq k \leq n$. It is used for instance, to compute the output of an infinite impulse response (IIR) filter and of an FIR filter in direct form [4].
4. *Constant matrix-vector multiplication (CMVM)*: The CMVM operation realizes the multiplication of an $m \times n$ constant matrix C by an $n \times 1$ input vector X, *i.e.*, $y_j = \sum_k c_{jk} x_k$ with $1 \leq j \leq m$ and $1 \leq k \leq n$. It occurs in the design of linear DSP transforms, such as DCTs and discrete sine transforms (DSTs), filter banks, and error correcting codes [5].

Figure 1 illustrates these four types of constant multiplications. Observe that the CMVM operation is the most general case of constant multiplications and it corresponds to an SCM operation when both m and n are equal to 1, to an MCM operation when n is 1, and to a CAVM operation when m is 1.

Although area-, delay-, and power-efficient multiplier architectures, such as Wallace [6] and modified Booth [7] multipliers, have been proposed, the full-flexibility of a multiplier is not necessary for the constant multiplications, since constant coefficients are fixed and determined beforehand by the DSP algorithms. Hence, constant multiplications are generally replaced with addition, subtraction and shift operations [8]. Note that shifts can be realized using only wires which represent no hardware cost. Thus, an important optimization problem is defined as finding the minimum number of addition and subtraction operations that implement the constant multiplications.

Over the years, many efficient high-level algorithms [2, 4, 5, 9–20] were introduced for the multiplierless design of constant multiplications. Most of these algorithms were designed for the MCM instances and little attention was given to the multiplierless design of the CMVM operation, although it occurs in many DSP systems. This is because a high-level algorithm designed for the MCM instances can be used for the implementation of a CMVM operation or can be modified to handle the CMVM instances. In the former, one can initially apply an MCM algorithm to the constants of each column of the matrix C and then, can utilize the sharing of the same constant multiplications in the rows of the matrix [9]. In the latter, each constant c_j and the variable x in an MCM instance can be replaced with a constant vector C_j and a variable vector X, respectively. While the former method yields poor results when compared to algorithms designed for the CMVM instances, as shown in Section 2.2, the efficient MCM algorithms [14–16] modified to handle the CMVM instances can only be applied to small size matrices with small constants, as indicated in [19, 20].

This chapter focuses on the multiplierless design of CMVM operations and introduces a high-level algorithm that targets the optimization of the number of addition and subtraction operations. Moreover, the proposed algorithm includes some hardware optimization techniques that take into account the type of the operation (addition or subtraction) and the size of input operands. Furthermore, it is modified to handle a delay constraint, which is given in terms of the number of adder-steps, *i.e*, the maximum number of operations in series. The experimental results indicate that the solutions of the proposed algorithms yield significant reductions in the number of operations, which consequently lead to CMVM designs with less hardware complexity, when compared to previously proposed algorithms.

The rest of this chapter proceeds as follows. Section 2 gives the background concepts and the proposed algorithms are introduced in Section 3. Experimental results are presented in Section 4, and, finally, Section 5 concludes the paper.

2 Background

This section presents the concepts related to the proposed algorithms, introduces the problem definitions, and gives an overview on the algorithms designed for the CMVM instances.

2.1 Number Representation

The *binary* representation decomposes a number in a set of additions of powers of two. The representation of numbers using a signed digit system makes the use of positive and negative digits. Thus, an integer number represented in the *binary signed digit* (BSD) system using n digits can be written as $\sum_{i=0}^{n-1} d_i 2^i$, where $d_i \in \{1, 0, \overline{1}\}$ and $\overline{1}$ denotes -1. The BSD system is a redundant number representation system, *e.g.*, both 0101 and $10\overline{1}\,\overline{1}$ correspond to 5.

The *canonical signed digit* (CSD) representation [21], a subset of BSD, has a unique representation for each number and verifies the following two main properties: i) two nonzero digits are not adjacent; ii) the number of nonzero digits is minimal. Any n digit number in CSD has at most $\lceil (n+1)/2 \rceil$ nonzero digits, and on average, the number of nonzero digits is reduced by 33% when compared to binary [22]. This representation is widely used in the multiplierless design of constant multiplications since it reduces the hardware requirements due to the minimum number of nonzero digits. An efficient conversion technique from binary to CSD can be found in [23].

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 3 & 11 \\ 5 & 13 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$y_1 = 3x_1 + 11x_2$$
$$y_2 = 5x_1 + 13x_2$$

(a)

$$\begin{aligned} y_1 &= 3x_1 + 11x_2 = (11)_{bin}x_1 + (1011)_{bin}x_2 \\ &= x_1 + x_1 \ll 1 + x_2 + x_2 \ll 1 + x_2 \ll 3 \\ &= x_1 + 2x_1 + x_2 + 2x_2 + 8x_2 \\ y_2 &= 5x_1 + 13x_2 = (101)_{bin}x_1 + (1101)_{bin}x_2 \\ &= x_1 + x_1 \ll 2 + x_2 + x_2 \ll 2 + x_2 \ll 3 \\ &= x_1 + 4x_1 + x_2 + 4x_2 + 8x_2 \end{aligned}$$

(b)

Fig. 2. (a) A CMVM realizing $y_1 = 3x_1 + 11x_2$ and $y_2 = 5x_1 + 13x_2$; (b) Decomposition of the constants in the linear transforms in binary

As an example, consider the constant 23 defined in six bits. The representation of 23 in binary, 010111, includes 4 nonzero digits. The constant is represented as $10\bar{1}00\bar{1}$ in CSD using 3 nonzero digits.

2.2 Problem Definitions

Here, we initially present the problem of optimizing the number of operations in a CMVM design and then, introduce the problem of optimizing the number of operations under a delay constraint.

Optimization of the Number of Operations. Given an $m \times n$ constant matrix C with $c_{jk} \in \mathbb{Z}$ and an $n \times 1$ variable vector X with $x_k \in \mathbb{Z}$, the multiplication of C by X is a linear transformation from $\mathbb{Z}^n$ to $\mathbb{Z}^m$ and each linear transform can be computed as

$$y_j = \sum_{k=1}^{n} c_{jk}\, x_k \tag{1}$$

where j and k range from 1 to m and n, respectively. Thus, the problem of optimizing the number of operations in linear transforms can be defined as follows:

Definition 1. CMVM PROBLEM. *Given $Y = \{y_1, \ldots, y_m\}$, a set of linear transforms, find the minimum number of addition and subtraction operations that generate the linear transforms.*

Note that the CMVM problem is an NP-complete problem due to the NP-completeness of the MCM problem proven in [24].

A straightforward way for the multiplierless realization of linear transforms, generally known as the digit-based recoding method [25], is to define the constants in binary, and for each 1 in the binary representation of the constant, is to shift the variable and add up the shifted variables. As a simple example, consider the multiplication of a constant matrix by a variable vector given in Figure 2(a).

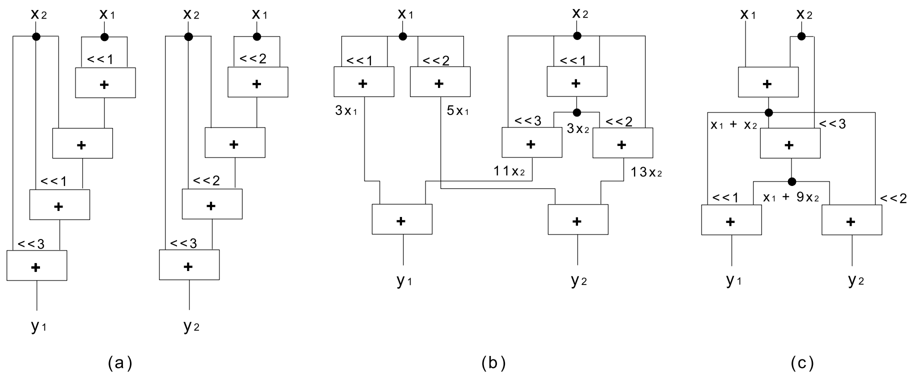


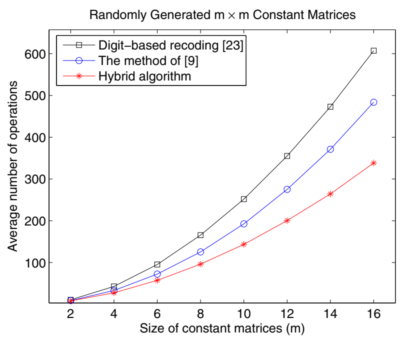
Fig. 3. Shift-adds implementations of $y_1 = 3x_1 + 11x_2$ and $y_2 = 5x_1 + 13x_2$: (a) with the technique of [25]; (b) with the method of [9]; (c) with our hybrid algorithm

The decomposed forms of linear transforms are given in Figure 2(b), where the computation of each y_1 and y_2 requires 4 operations, a total of 8 operations, as depicted in Figure 3(a). Note that a linear transform including n terms in its decomposed form requires $n-1$ operations in the digit-based recoding technique. For the linear transforms in Figure 2(a), the decomposition of constants in CSD also yields the same result in terms of the number of operations as in binary.

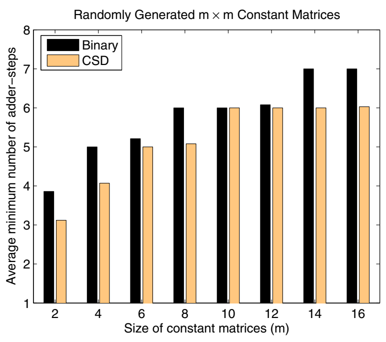
The multiplierless design of linear transforms can also be realized by applying an SCM algorithm to each element of the constant matrix or an MCM algorithm on the elements of each column of the constant matrix [9]. Then, the sharing of the same constant multiplications can be achieved in rows and columns of the matrix in the former method and the sharing of the same constant multiplications in rows of the matrix can be utilized in the latter technique. For our example in Figure 2(a), the latter method [9] obtains a solution with 7 operations as shown in Figure 3(b) when the exact algorithm of [18] is used as an MCM algorithm.

However, the sharing of partial products among the constant multiplications, that significantly reduces the required number of operations and, consequently, the area and power dissipation of the design at gate-level, is never utilized in the digit-based recoding technique [25], as can be observed in Figure 3(a). In the method of [9], this is achieved partially by an MCM algorithm on the CMVM operation, *i.e.*, only on the columns of the matrix, as can be seen in Figure 3(b). For our example, the hybrid algorithm introduced in Section 3, that fully exploits the partial product sharing, obtains a solution with 4 operations by finding the common partial products $x_1 + x_2$ and $x_1 + 9x_2$, as shown in Figure 3(c).

Figure 4(a) presents the effect of partial product sharing on the number of operations on $m \times m$ matrices, where m varies in between 2 and 16 in steps of 2. We used 100 instances for each matrix type, with a total of 800 instances. The constants were generated randomly from $[2^7+1, 2^8-1]$. In this experiment, the results of the hybrid algorithm are compared with those of the digit-based recoding technique [25] when constants are defined under CSD and with those of the technique [9] when the exact algorithm [18] designed for the MCM problem is applied to the elements of each column of a matrix. Observe from Figure 4(a) that



(a)



(b)

Fig. 4. (a) Effect of partial product sharing on the number of operations; (b) Effect of number representation on the minimum number of adder-steps

as the size of the matrix (m) increases, the impact of partial product sharing on the number of operations increases. The maximum gain in terms of the average number of operations between the hybrid algorithm and the digit-based recoding technique [25], which does not exploit any sharing, is 44.2% on 16×16 matrices. The maximum gain between the hybrid algorithm and the technique [9] is 30% on the same instances.

Optimization of the Number of Operations under a Delay Constraint. In many DSP systems, performance is also a crucial parameter and circuit area is generally expandable in order to achieve a given performance target. Although the delay parameter is dependent on several implementation issues, such as placement and routing, the delay of a CMVM operation is generally considered in terms of the number of adder-steps which denotes the maximal number of adders/subtracters in series [3]. As an example, the shift-adds designs of linear transforms in Figures 3(a)-(c) have 4, 3, and 3 adder-steps, respectively.

The minimum adder-steps of a linear transform y_j is computed by decomposing its constants c_{jk} under a number representation, finding the number of terms in its decomposed form, $S(y_j)$, and computing $\lceil log_2 S(y_j) \rceil$, as if all its terms in the decomposed form were realized in a binary tree. Returning to the linear transforms y_1 and y_2 in Figure 2(a), their decomposed forms under binary consist of 5 terms (Figure 2(b)) and hence, the minimum adder-steps of both linear transforms are computed as 3.

Given a set of linear transforms $Y = \{y_1, \ldots, y_m\}$, the minimum adder-steps of a CMVM operation [26] is computed as the maximum of the minimum adder-steps of each linear transform:

$$min\_delay_{\text{CMVM}} = \max_{y_j}\{\lceil log_2 S(y_j) \rceil\} \tag{2}$$

Thus, the minimum adder-steps of the CMVM operation realizing y_1 and y_2 in Figure 2(a) is computed as 3. Note that min_delay_{CMVM} of Eqn. 2 is generally determined when constants are defined under CSD since the CSD representation of a constant includes the minimum number of nonzero digits. However, in the case of high-level algorithms that extract the implementations of linear transforms when the constants are defined under a number representation, the given number representation determines the minimum adder-steps of the CMVM operation. Figure 4(b) presents the effect of a number representation on the minimum number of adder-steps on the previously used benchmark set. Observe that the use of binary representation may lead to greater minimum adder-steps with respect to CSD since a constant is generally represented with a larger number of nonzero digits in binary compared to CSD.

Thus, the problem of optimizing the number of operations in linear transforms under a delay constraint can be defined as:

Definition 2. CMVM PROBLEM UNDER A DELAY CONSTRAINT. *Given a set of linear transforms, $Y = \{y_1, \ldots, y_m\}$, and the delay constraint dc, where $dc \geq min\_delay_{\mathrm{CMVM}}$, find the minimum number of addition and subtraction operations that generate the linear transforms with a delay not exceeding dc.*

2.3 Related Work

The high-level algorithms designed for the multiplierless realization of constant multiplications are generally categorized in two classes: common subexpression elimination (CSE) [4, 9–13] and graph-based (GB) [2, 15, 17–20] techniques. Although both CSE and GB algorithms aim to maximize the sharing of partial products, they differ in the search space that they explore. The CSE algorithms initially define the constants under a number representation. Then, all possible subexpressions are extracted from the representations of constants and the "best" subexpression, generally, the most common, is chosen to be shared among the constant multiplications. The GB algorithms are not limited to any particular number representation and consider a larger number of alternative implementations of a constant multiplication, yielding better solutions than the CSE algorithms [17, 18]. Here, we only mention the algorithms applied to the CMVM problem. However, the readers are referred to [10, 17] for further details on the CSE and GB algorithms designed for the MCM instances.

The CMVM problem was formalized as a 0-1 integer linear programming (ILP) problem in [11], where the possible implementations of linear transforms are extracted after constants are defined under a number representation and the decomposed forms of linear transforms are obtained. However, the CSE algorithm [11] only considers the 2-term subexpressions due to the exponential growth in the size of 0-1 ILP problems. Furthermore, the exact CSE algorithm of [13] exploits all possible subexpressions and finds a solution with the minimum number of operations by representing the CMVM problem as a 0-1 ILP problem. It is shown in [13] that the exact CSE algorithm can be applied to small size constant matrices with small constants. On the other hand, the CSE heuristics of [5, 12, 27, 28] initially determine each linear transform by simply multiplying each row of the constant matrix with the input vector, as given in

Eqn 1, define the constants under CSD representation, and obtain the decomposed forms of linear transforms. Then, the sharing of common subexpressions is achieved based on heuristics. The algorithm of [12] selects the most common 2-term subexpression and eliminates its occurrences in the expressions in each iteration until there is no subexpression with a number of occurrences greater than 1. This algorithm is extended to handle a delay constraint in [27]. The algorithm of [28] chooses its subexpressions based on a cost value which is computed as the multiplication of the number of terms in the subexpression by the number of its occurrences in the linear transforms. The algorithm of [5] relies on an efficient CSE algorithm [29] that iteratively searches a subexpression with the maximal number of terms and with at least 2 occurrences. In [5], the selection of a subexpression is also modified by taking into account the conflicts between the possible subexpressions.

In [19], the efficient techniques of [14, 15] proposed for the MCM instances are modified to handle the CMVM problem. Keeping in mind that the input variables $x_1, x_2, \ldots, x_n$ and their shifted values are always available and the input variables are stored in a fundamental set, the algorithm of [19] iteratively finds all possible sums of elements of the fundamental set, chooses the one that is the closest to any linear linear transform in terms of the adder cost distance [19], and stores it in the fundamental set. If a possible sum of two elements in the fundamental set equals to a linear transform, the adder cost distance is 0 at this time, the linear transform is moved to the fundamental set. The algorithm of [19] continues until all the linear transforms are synthesized. However, it is computationally intensive for large size matrices and thus, can only be applied to small size matrices. Moreover, the algorithm of [20] is based on the algorithm of [16] designed for the MCM instances. It initially computes the differences between each two linear transforms and determines their implementation cost values. Then, it uses a minimum spanning tree (MST) algorithm to find the realizations of linear transforms with differences, that have the minimum cost, and replaces the linear transforms with these differences. The algorithm iterates until all the linear transforms are synthesized. As stated in [20], the algorithm is limited to the number of linear transforms and the bitwidth of constants due to the application of the MST algorithm in each iteration.

3 The Hybrid Algorithms

This section introduces the hybrid algorithm, called Hcmvm, designed for the multiplierless realization of linear transforms. The Hcmvm algorithm combines less-complex and time-efficient techniques from the CSE and GB algorithms to take the advantages of both techniques. It iteratively finds alternative realizations of linear transforms using the GB difference method and applies a CSE heuristic to further reduce the hardware complexity by sharing the common subexpressions. Hence, in the hybrid algorithm, the main drawback of a CSE algorithm, *i.e.*, its limitation to a number representation, is partially eliminated using a GB algorithm, and the main drawback of a GB algorithm, *i.e.*, its time-consuming search process, is partially decreased using a CSE heuristic. Although the hybrid algorithm finds solutions with the fewest number of operations, leading to low-complexity designs at gate-level, its solutions can be realized in a large number of adder-steps due to the partial product sharing, yielding CMVM designs with large delay at gate-level. To overcome this disadvantage, we also describe its modified version, called HCMVM-DC, that can find a solution under a delay constraint and enables us to find the optimal tradeoff between area and delay in a CMVM design.

3.1 The HCMVM Algorithm

The hybrid algorithm can handle the constants under binary and CSD where there is a unique representation for each constant. In its preprocessing phase, each linear transform is converted to an odd and positive expression, *i.e.*, it is divided by 2 until at least one of its constants is odd, and it is multiplied by -1 if the sign of the first nonzero constant in the expression is negative. Then, these expressions are stored in a set called *Eset* without repetition.

As done in GB algorithms designed for the MCM instances [15, 17, 18], the linear transforms, that can be synthesized using a single operation (the inputs of these linear transforms are: an element of the input vector; an implemented linear transform; or their shifted versions), are found iteratively and moved from *Eset* to *Iset* which will include the implemented expressions. As a simple example, consider the linear transforms $y_1 = x_1 + x_2$, $y_2 = x_1 + x_3$, and $y_3 = 3x_1 + x_2 + 2x_3$. Observe that y_1 and y_2 can be implemented using a single operation, whose inputs are the input variables, and y_3 can be synthesized as $y_1 + y_2 \ll 1$. This is the optimal part of the hybrid algorithm, meaning that, when all the linear transforms are realized in this part, the minimum solution is obtained.

If *Eset* is still not empty, HCMVM switches to its heuristic part given as:

1. Find a solution on the expressions of *Eset* with the CSE algorithm, called H2MC, that will be described next, and record its solution (considering also the number of elements in *Iset*) as the best solution found so far (*bs*).
2. Compute the cost of each expression in *Eset* as the total number of nonzero digits of each constant under the given number representation (binary or CSD) used in H2MC.
3. Sort the expressions in a descending order based on their cost values.
4. For each expression in *Eset*, $Eset_i$, with a cost value $cost_i$, where $i < m$ and m denotes the number of expressions in *Eset*,
   (a) Find all the implementations of $Eset_i$ including an expression in *Eset*, $Eset_j$, as $Eset_i = Eset_j \ll l_1 + d_{ij} \ll l_2$, where $i < j \leq m$ and $l_1, l_2 \geq 0$ denote the left shifts. Then, compute all the differences of $Eset_i$ as $d_{ij} \ll l_2 = Eset_i - Eset_j \ll l_1$.
   (b) Determine the cost of each difference in terms of the total number of nonzero digits of each constant under the given number representation and find the difference with the minimum cost value ($cost_d$).
   (c) If $cost_d < cost_i - 1$, then move $Eset_i$ from *Eset* to *Iset* and add the difference d_{ij} with the minimum cost value into *Eset*, in place of $Eset_i$.

5. If none of expressions is replaced with a difference (there are no promising differences for an expression in *Eset*), return the best solution found so far.
6. Otherwise, apply H_{2MC} to the expressions of *Eset* and obtain a set of operations realizing these expressions.
7. If the number of operations in the solution of H_{2MC} plus the number of elements in *Iset* is less than *bs*, update the *bs* value. Note that each element of *Iset* requires a single operation to be implemented.
8. Go to Step 2 and repeat the process.

The reason behind the application of H_{2MC} in Step 1 of the heuristic part is to determine an upper bound of the search space and also, to obtain a solution in case the difference method cannot achieve any promising difference on the initial *Eset*. This guarantees that a solution of HCMVM always includes a number of operations less than or equal to that of H_{2MC}. Note also that in Step 4(a) of the heuristic part, the left shifts are limited to the maximum bitwidth of constants in the expressions while searching for the implementations of an expression.

Figure 5 illustrates the procedure of HCMVM on the first example of [20] when H_{2MC} defines the constants under CSD. In this figure, the values between parenthesis next to the expressions denote the respective cost values. Initially, H_{2MC} is applied to the linear transforms and a solution with 19 operations is obtained. Then, in the first iteration, the linear transforms $Eset_1$, $Eset_2$, and $Eset_3$ are realized using a single operation whose inputs are an element of *Eset* and a difference with the minimum cost. These expressions are synthesized as $Eset_1 = Eset_2 + d_{12}$, $Eset_2 = Eset_3 + d_{23}$, $Eset_3 = Eset_4 + d_{34} \ll 1$. Then, these linear transforms are moved from *Eset* to *Iset* and the associated differences are added to *Eset*. In this case, H_{2MC} finds a solution with 10 operations on *Eset*. Thus, with 3 expressions in *Iset*, each requiring a single operation, a total of 13 operations are needed. Since this solution is better than the best solution found so far (*bs*), *i.e.*, 19, the *bs* value is updated. In the second iteration, HCMVM follows the same procedure, realizing $Eset_1$ as $Eset_4 + d_{14} \ll 1$ and finding a solution with a total of 13 operations again. The HCMVM algorithm takes only two iterations since there are no more promising differences for an expression. As reported in [20], the algorithms of [19, 20] find a solution with 14 operations on this instance.

H_{2MC} - The CSE Heuristic. The H_{2MC} algorithm is based on the CSE heuristics [4, 12] that iteratively compute the most common (MC) 2-term subexpressions. Furthermore, we improved their subexpression selection heuristic (that significantly affects the final solution due to the iterative decision making) by choosing an MC 2-term subexpression such that its selection leads to the least loss of subexpression sharing in the next iterations. These subexpressions are called the most common minimum conflicting (MCmc) 2-term subexpressions.

In HCMVM, H_{2MC} takes *Eset* as an input and returns *Sset*, that includes the subexpressions required to realize all the expressions of *Eset*, as an output. In H_{2MC}, for each element of *Eset*, the constants in expressions are defined under a given number representation, and the decompositions of expressions are obtained

Initial expressions:	
$y_1 = 7x_1 + 8x_2 + 2x_3 + 13x_4$	
$y_2 = 12x_1 + 11x_2 + 7x_3 + 13x_4$	
$y_3 = 5x_1 + 8x_2 + 2x_3 + 15x_4$	
$y_4 = 7x_1 + 11x_2 + 7x_3 + 11x_4$	
Solution of $\mathrm{H_{2MC}}$ on initial expressions: 19 operations, $bs = 19$	
Iteration 1	
Expressions of *Eset* and chosen differences:	
$Eset_1(10) : 12x_1 + 11x_2 + 7x_3 + 13x_4$	$d_{12}(3) : 5x_1 + 2x_4$
$Eset_2(10) : 7x_1 + 11x_2 + 7x_3 + 11x_4$	$d_{23}(5) : 3x_2 + 5x_3 - 2x_4$
$Eset_3(7) : 7x_1 + 8x_2 + 2x_3 + 13x_4$	$d_{34}(2) : x_1 - x_4$
$Eset_4(6) : 5x_1 + 8x_2 + 2x_3 + 15x_4$	
Expressions in *Eset*:	Expressions in *Iset*:
$5x_1 + 2x_4$	$12x_1 + 11x_2 + 7x_3 + 13x_4$
$3x_2 + 5x_3 - 2x_4$	$7x_1 + 11x_2 + 7x_3 + 11x_4$
$x_1 - x_4$	$7x_1 + 8x_2 + 2x_3 + 13x_4$
$5x_1 + 8x_2 + 2x_3 + 15x_4$	
Solution of $\mathrm{H_{2MC}}$ on *Eset*: 10 operations, Total: $10 + \lvert Iset \rvert = 13$, $bs = 13$	
Iteration 2	
Expressions of *Eset* and chosen differences:	
$Eset_1(6) : 5x_1 + 8x_2 + 2x_3 + 15x_4$	$d_{14}(4) : 2x_1 + 4x_2 + x_3 + 8x_4$
$Eset_2(5) : 3x_2 + 5x_3 - 2x_4$	
$Eset_3(3) : 5x_1 + 2x_4$	
$Eset_4(2) : x_1 - x_4$	
Expressions in *Eset*:	Expressions in *Iset*:
$2x_1 + 4x_2 + x_3 + 8x_4$	$12x_1 + 11x_2 + 7x_3 + 13x_4$
$3x_2 + 5x_3 - 2x_4$	$7x_1 + 11x_2 + 7x_3 + 11x_4$
$5x_1 + 2x_4$	$7x_1 + 8x_2 + 2x_3 + 13x_4$
$x_1 - x_4$	$5x_1 + 8x_2 + 2x_3 + 15x_4$
Solution of $\mathrm{H_{2MC}}$ on *Eset*: 9 operations, Total: $9 + \lvert Iset \rvert = 13$	

Fig. 5. Procedure of the HCMVM algorithm

and stored in a set called *Dset*. The part of $\mathrm{H_{2MC}}$, where the MCmc 2-term subexpressions are found and replaced in the decompositions of expressions, is given as follows:

1. Form a set, called *Sset*, that will store the selected 2-term subexpressions.
2. For each 2-term subexpression, that is extracted from the decompositions of expressions in *Dset*, convert the subexpression to positive and odd, find its occurrences in the elements of *Dset* considering its negated and shifted versions, and determine the MC 2-term subexpressions.
3. If the maximum number of occurrences of the MC 2-term subexpressions is 1, then return *Dset* and *Sset*.
4. Otherwise, find the minimum conflicting 2-term subexpressions in the MC 2-term subexpressions, *i.e.*, the MCmc 2-term subexpressions. In this case,

for each MC 2-term subexpression, we compute the number of MC 2-terms subexpressions it conflicts with.

5. Select one of the MCmc 2-term subexpressions, add it to *Sset* by labeling it with a variable, replace its occurrences in *Dset* with its label, go to Step 2.

Figure 6 illustrates the procedure of $H_{2\text{MC}}$ when constants are defined under CSD. In the first iteration, two MC 2-term subexpressions, that both occur in y_1 once and in y_2 twice, with a total of 3 occurrences, are obtained. Note that the occurrences can also be found in negated or shifted forms. However, the occurrences of the MC 2-term subexpressions in the linear transforms conflict with each other, indicating that selecting one of them will eliminate the other in the next iteration. The MCmc 2-term subexpressions are determined as MC 2-term subexpressions, and in this iteration, the subexpression $x_2 + 16x_2$ is chosen and replaced in the expressions. In Figure 6, the occurrences of selected MCmc 2-term subexpressions in *Dset* are shown in bold. In the second iteration, there are three MC 2-term subexpressions with two occurrences. The subexpressions $x_1 + 4x_2$ and $2x_1 + a$ occur only in y_1 and y_2, respectively. The subexpression $-x_1 + 16x_1$ occurs in both expressions and its occurrences conflict with the occurrences of both $x_1 + 4x_2$ and $2x_1 + a$. Thus, the MCmc 2-term subexpressions are determined as $x_1 + 4x_2$ and $2x_1 + a$, and in this iteration, $2x_1 + a$ is chosen and replaced in the expressions. Hence, in the third iteration, $x_1 + 4x_2$ is encountered again, is selected, and replaced in the expressions. The resulting expressions do not include 2-term subexpressions with a number of occurrence greater than 1. Thus, $H_{2\text{MC}}$ finds a solution with a total of 7 operations, 3 operations for the 2-term subexpressions selected in each iteration (the elements of *Sset*) and 4 operations for the elements of the final *Dset*.

To illustrate the impact of selecting an MCmc 2-term subexpression, observe that if the MC 2-term subexpression $-x_1 + 16x_1$ had been selected in the second iteration, there would not be any 2-term with a maximum occurrence greater than 1 in the next iteration since this subexpression would remove the occurrences of $x_1 + 4x_2$ and $2x_1 + a$. In this case, 8 operations would be required.

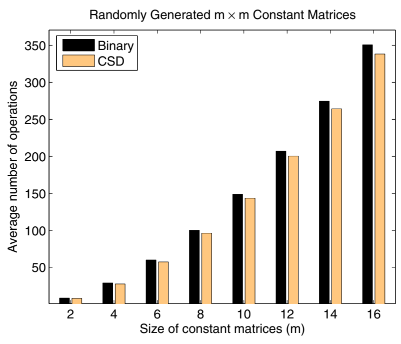
Note that the realization of expressions in the final *Dset* is a trivial process. However, as stated in [30], the high-level algorithms should also consider the gate-level implementation of operations to further reduce the hardware complexity. Hence, while synthesizing the expressions in the final *Dset*, we apply some hardware optimizations without changing the number of operations. For each expression in the final *Dset* including more than 2 terms, we separate the terms into two sets, *Pset* and *Mset*, considering their sign. This comes from the fact that although the cost of an adder and a subtracter is assumed to be equal in high-level algorithms, a subtracter occupies larger area than an adder at gate-level. Then, in each *Pset* and *Mset*, we iteratively select two terms, that have the smallest bitwidth, *i.e.*, the narrowest (require less hardware), to be the inputs of an adder, generate the output of the adder, label it with a variable, store it to *Sset* as a 2-term subexpression, remove these two inputs from the set, and add the output of the adder to the set. This process is iterated until the number of elements of each set is equal to 1. Note that if the initial *Mset* is not

Initial expressions:	
$y_1 = 15x_1 + 43x_2$	
$y_2 = 38x_1 + 51x_2$	
Initial expressions in *Dset*:	
$Dset_1 = -x_1 + 16x_1 - \mathbf{x_2} - 4x_2 - \mathbf{16x_2} + 64x_2$	
$Dset_2 = -2x_1 + 8x_1 + 32x_1 - \mathbf{x_2} + \mathbf{4x_2} - \mathbf{16x_2} + \mathbf{64x_2}$	
Iteration 1	
MC 2-terms: $x_2 + 16x_2$, $-x_2 + 4x_2$ (#occurrences = 3)	
MCmc 2-terms: $\mathbf{x_2 + 16x_2}$, $-x_2 + 4x_2$	
Current expressions in *Dset*:	Current expressions in *Sset*:
$Dset_1 = -x_1 + 16x_1 - 4x_2 + 64x_2 - a$	$Sset_1 = a = x_2 + 16x_2$
$Dset_2 = \mathbf{-2x_1} + \mathbf{8x_1} + 32x_1 \mathbf{-a} + \mathbf{4a}$	
Iteration 2	
MC 2-terms: $-x_1 + 16x_1$, $x_1 + 4x_2$, $2x_1 + a$ (#occurrences = 2)	
MCmc 2-terms: $x_1 + 4x_2$, $\mathbf{2x_1 + a}$	
Current expressions in *Dset*:	Current expressions in *Sset*:
$Dset_1 = \mathbf{-x_1} + \mathbf{16x_1} - \mathbf{4x_2} + \mathbf{64x_2} - a$	$Sset_1 = a = x_2 + 16x_2$
$Dset_2 = 32x_1 - b + 4b$	$Sset_2 = b = 2x_1 + a$
Iteration 3	
MC 2-terms: $x_1 + 4x_2$ (#occurrences = 2)	
MCmc 2-terms: $\mathbf{x_1 + 4x_2}$	
Current expressions in *Dset*:	Current expressions in *Sset*:
$Dset_1 = -a - c + 4c$	$Sset_1 = a = x_2 + 16x_2$
$Dset_2 = 32x_1 - b + 4b$	$Sset_2 = b = 2x_1 + a$
	$Sset_3 = c = x_1 + 4x_2$

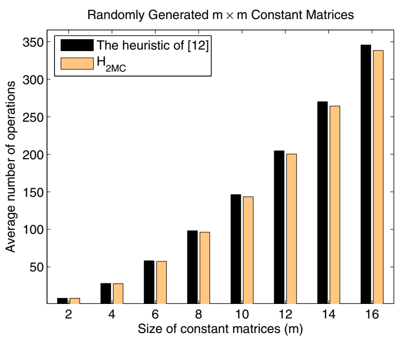
Fig. 6. Procedure of the H2MC algorithm

empty, the final operation will be a subtracter to realize the expression. Thus, in our example in Figure 6, $Dset_1$ and $Dset_2$ in the final *Dset* are respectively implemented as $4c - (a + c)$ and $(32x_1 + 4b) - b$, with a total of 4 operations.

Analysis of the Hybrid Algorithm. Although the HCMVM algorithm uses the GB difference technique, that is not limited to any number representation, it also includes a CSE heuristic whose solution depends on the number representation. Figure 7(a) illustrates the impact of a number representation (binary and CSD) on the solutions of HCMVM on the benchmark set introduced in Section 2.2. Observe from Figure 7(a) that as the size of the constant matrix (m) is increased, the difference on the average number of operations between the solutions of HCMVM obtained under binary and CSD increases and reaches up to 12.43 on 16×16 matrices. This is primarily because a constant is represented with the minimum number of nonzero digits under CSD, generating a linear transform with much less number of terms with respect to those decomposed under binary. We note that HCMVM generally obtains better solutions when constants are defined under CSD. However, there are solutions with less number of operations obtained by HCMVM under binary compared to those found under CSD.



(a)



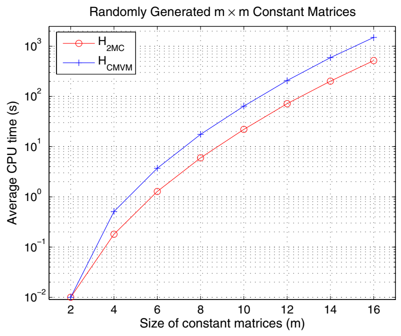
(b)

Fig. 7. Analysis of the Hcmvm algorithm: (a) Effect of number representations; (b) Effect of CSE heuristics

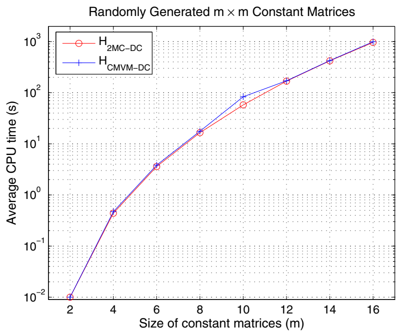
Moreover, the CSE heuristic used in Hcmvm has a significant impact on its solutions. Figure 7(b) presents the effect of a CSE heuristic (the heuristic of [12] and $\mathrm{H_{2MC}}$) on the solutions of Hcmvm on the benchmark set introduced in Section 2.2 when constants are defined under CSD. Note that this experiment explicitly presents the effect of using the MCmc 2-term subexpressions instead of the MC 2-term subexpressions which is the main difference between $\mathrm{H_{2MC}}$ and the heuristic of [12]. As can be observed from Figure 7(b), Hcmvm with $\mathrm{H_{2MC}}$ obtains better solutions in terms of the number of operations than those of Hcmvm including the heuristic of [12]. The maximum difference on the average number of operations between the solutions of Hcmvm with the heuristic of [12] and Hcmvm with $\mathrm{H_{2MC}}$ is 7.33 on 16×16 matrices. This is simply because the use of the MCmc 2-term subexpressions increases the possibility of subexpression sharing in later iterations with respect to the MC 2-term subexpressions. Note also that Hcmvm can incorporate any CSE algorithm and hence, as more efficient CSE heuristics are developed, they can be adapted to Hcmvm.

3.2 The Hcmvm-dc Algorithm

The preprocessing phase of Hcmvm-dc is similar to that of Hcmvm, but in Hcmvm-dc, we also compute the minimum adder-steps of each linear transform as described in Section 2.2. Then, given the delay constraint, dc, that is greater than or equal to the minimum adder-steps of the CMVM operation computed as in Eqn. 2, in its optimal part, we synthesize the linear transforms using a single operation if their realizations do not exceed dc. While searching for the promising differences, we compute the minimum adder-steps of each difference and accept the synthesis of an expression only if its realization does not violate dc. For our example in Figure 5, given $dc = 4$, that is the minimum adder-steps of the CMVM operation, the realization of $Eset_1 = Eset_2 + d_1$ in the first iteration is not possible in Hcmvm-dc because the realizations of both $Eset_1$ and $Eset_2$ require minimum 4 adder-steps. Thus, any implementation of $Eset_1$ with $Eset_2$ always violates $dc = 4$.



(a)



(b)

Fig. 8. Run time of algorithms: (a) H_{2MC} and HCMVM; (b) H_{2MC-DC} and HCMVM-DC

Moreover, we modified the H_{2MC} algorithm, called H_{2MC-DC}, to handle the delay constraint. In H_{2MC-DC}, we initially find the MC 2-term subexpressions, whose selections will not lead to a realization greater than *dc*, and then, we obtain the MCmc 2-term subexpressions among the MC 2-term subexpressions. In H_{2MC-DC}, we also apply the same hardware optimizations considered in H_{2MC} by taking into account *dc*.

Thus, with these modifications, HCMVM-DC can find a solution under a delay constraint. We note that HCMVM-DC presents a similar behavior as HCMVM under different number representations and CSE heuristics.

3.3 Time Complexity of the Hybrid Algorithms

The run time of the hybrid algorithms depends on the number of iterations they take and the performance of the CSE heuristics, H_{2MC} and H_{2MC-DC}, which are used in each iteration of HCMVM and HCMVM-DC, respectively. Hence, finding a solution with a hybrid algorithm will always take longer time than a CSE heuristic. Note that the number of iterations of HCMVM depends heavily on the linear transforms. Also, the computational complexity of the H_{2MC} algorithm is related to the number of expressions (m) and the number of terms in the decomposed form of each expression (t_i, where $1 \leq i \leq m$). The number of 2-term subexpressions to be considered in a decomposed form of an expression including t terms is $t(t-1)/2$. As a simple example, consider the decomposed form of an expression $x_1+x_2+x_3$. There exist three 2-term subexpressions, x_1+x_2, x_1+x_3, and x_2+x_3. Thus, the maximum number of 2-term subexpressions considered in one iteration of H_{2MC} is simply $\sum_{i=1}^{m} t_i(t_i-1)/2$.

Figure 8 presents the average run time of the hybrid and CSE algorithms in seconds on the benchmark set introduced in Section 2.2 when constants are defined under CSD. In H_{2MC-DC} and HCMVM-DC, the delay constraint was set to the minimum adder-steps of the CMVM operation. These algorithms were implemented in MATLAB and were run on a PC with Intel Xeon at 2.33GHz.

Table 1. Summary of results of high-level algorithms on linear DSP transforms in terms of the number of operations

Algorithms	H.264	DCT8	IDCT8	DHT	DST
[9]	–	227	222	211	252
[8]	–	202	183	209	238
[28]	53	161	140	161	181
[12] - CSD	51	147	138	159	176
$\text{H}_{2\text{MC}}$ - CSD	49	150	137	150	174
HCMVM - CSD	42	145	136	150	172

As can be observed from Figure 8, while HCMVM takes much longer time than $\text{H}_{2\text{MC}}$, HCMVM-DC and $\text{H}_{2\text{MC}-\text{DC}}$ have similar run time. In the former case, this is simply because HCMVM generally takes more than one iteration due to the difference method. In the latter case, the delay constraint with the minimum number of adder-steps highly restricts the ability of the difference method and HCMVM-DC generally takes few iterations. On the other hand, $\text{H}_{2\text{MC}}$ takes less time than $\text{H}_{2\text{MC}-\text{DC}}$. This is because $\text{H}_{2\text{MC}-\text{DC}}$ operates with the 2-term subexpressions, which are not the most common, due to the delay constraint. Hence, it requires more 2-term subexpressions and thus, it takes more iterations to obtain a solution than $\text{H}_{2\text{MC}}$ that considers the most common 2-term subexpressions. Overall, HCMVM-DC takes less time than HCMVM since it takes less number of iterations than HCMVM due to the delay constraint.

4 Experimental Results

This section presents the high-level results of previously proposed algorithms [5, 8, 9, 12, 27, 28] and of the algorithms introduced in this chapter on linear DSP transforms and random instances. We have also implemented the algorithms of [12, 27]. Moreover, we developed a computer-aided design (CAD) tool that automatically describes the solutions of high-level algorithms under the shift-adds architecture in VHDL. The CAD tool is also capable of describing the direct realizations of linear transforms in VHDL where they are defined as the summations of constant multiplications as given in Eqn. 1. Additionally, this section presents the gate-level results of 20×20 DCTs designed using the solutions of high-level algorithms. The DCTs were synthesized using the Cadence Encounter RTL Compiler with the Nangate 45nm Open Cell library [31].

The first experiment set [28] consists of a 7×3 H.264 video compression transform, an 8-point DCT, an 8-point inverse DCT (IDCT), an 8×8 discrete Hartley transform (DHT), and an 8×8 DST. The constants are defined using 14 bits. The solutions of the high-level algorithms in terms of the number of operations are given in Table 1 where the results of algorithms [8, 9, 28] were taken from [28]. In $\text{H}_{2\text{MC}}$ and HCMVM, the constants were defined under CSD. Observe that HCMVM finds better solutions than all algorithms in terms of the number of operations (except that both $\text{H}_{2\text{MC}}$ and HCMVM obtain the best solution on the DHT instance).

Table 2. Summary of results of high-level algorithms on $m \times m$ randomly generated matrices with 8-bit constants

m	CMVM problem							CMVM problem under a delay constraint					
	[12]		[5]	H2MC		Hcmvm		Hcmvm-dc*		[27]**		Hcmvm-dc**	
	adder	step	adder	adder	step	adder	step	adder	step	adder	step	adder	step
2	8.8	3.5	9.7	8.7	3.6	8.2	4.4	8.2	3.7	9.0	3.1	8.8	3.1
4	32.1	5.9	31.2	31.7	5.8	27.6	7.8	28.1	5.7	32.8	4.1	32.1	4.1
6	68.0	7.6	66.1	66.5	7.7	57.3	10.0	58.2	7.0	68.1	5.0	66.8	5.0
8	116.4	9.2	113.2	114.1	9.2	96.3	11.9	99.5	7.1	119.7	5.1	117.2	5.1
10	175.7	10.7	172.4	172.0	10.5	143.5	13.2	146.9	8.0	175.8	6.0	157.7	6.0
12	246.5	12.0	241.6	240.9	11.7	200.4	14.6	206.8	8.0	247.1	6.0	241.6	6.0
14	327.1	13.2	322.9	320.0	13.0	264.3	15.5	274.8	8.0	330.2	6.0	324.0	6.0
16	417.9	14.4	412.4	407.5	14.0	338.3	16.3	353.3	8.0	431.0	6.0	423.2	6.0

* the delay constraint was set to $min\_delay_{\text{CMVM}} + 2$

** the delay constraint was set to min_delay_{CMVM}

As the second experiment set, we used the benchmark set introduced in Section 2.2. Table 2 presents the results of high-level algorithms, where *adder* and *step* stand for the average number of operations and the average number of adder-steps, respectively. The results of the algorithm [5] were taken from its paper. The results of the algorithm of [27] were found when the delay constraint (*dc*) was set to the minimum delay of the CMVM operation (min_delay_{CMVM}), as computed in Eqn. 2. In Hcmvm-dc, *dc* was set to min_delay_{CMVM} and $min\_delay_{\text{CMVM}}+2$. In all algorithms, the constants were defined under CSD.

Observe from Table 2 that the Hcmvm algorithm finds significantly better solutions than the CSE algorithms [5, 12] and H2MC in terms of the number of operations due to the use of the difference method. However, its solutions lead to CMVM designs with a large number of adder-steps. On the other hand, when *dc* is $min\_delay_{\text{CMVM}} + 2$, the Hcmvm-dc algorithm still obtains better solutions than those of the CSE algorithms designed for the CMVM problem and finds solutions close to those of Hcmvm. Note that while the maximum gain on the number of operations between Hcmvm and Hcmvm-dc when *dc* is $min\_delay_{\text{CMVM}} + 2$ is 4.23% on 16×16 matrices, the maximum gain on the number of adder-steps between Hcmvm-dc when *dc* is $min\_delay_{\text{CMVM}} + 2$ and Hcmvm is 50.73% on the same instances. However, when *dc* is min_delay_{CMVM} in Hcmvm-dc, the effect of the difference method is diminished significantly. Note that the maximum gain on the number of operations obtained by Hcmvm-dc when *dc* is $min\_delay_{\text{CMVM}} + 2$ and min_delay_{CMVM} is 16.52% on 16×16 instances with a two adder-steps increase on average. However, it finds better solutions than the algorithm of [27] in terms of the number of operations on all matrix types. Moreover, there are instances, such as 10×10 matrices, where Hcmvm-dc under the minimum number of adder-steps delay constraint obtains better solutions than the CSE algorithms designed for the CMVM problem.

As the third experiment, we used 20×20 DCTs, where the bitwidth (*bw*) of the constants were defined from 2 to 16 with increments of 2. Table 3 presents the results of algorithms where *CPU* denotes their run time in seconds. The constants in DCTs were defined under CSD, and in the algorithm of [27] and Hcmvm-dc, *dc* was set to the minimum adder-steps of the DCT design.

Table 3. Summary of results of high-level algorithms on 20 × 20 DCTs

bw	CMVM problem						CMVM problem under a delay constraint					
	[12]			Hcmvm			[27]			Hcmvm-dc		
	adder	step	CPU	adder	step	CPU	adder	step	CPU	adder	step	CPU
2	118	6	6.5	98	5	16.0	118	5	15.5	98	5	35.8
4	156	7	46.7	156	8	193.3	156	6	115.1	156	6	412.9
6	192	8	105.3	189	8	392.8	194	6	245.2	191	6	250.7
8	232	11	250.4	232	11	905.4	232	7	573.3	232	7	1077.9
10	257	11	414.6	254	11	2135.8	260	7	938.8	258	7	3573.8
12	300	13	624.7	295	13	3808.9	303	7	1406.3	298	7	1437.0
14	323	13	962.7	319	13	4814.8	326	7	2107.2	323	7	2156.7
16	376	15	1556.9	357	15	9002.1	391	7	3362.4	379	7	3385.6
Total	1954	84	3967.8	1900	84	21269.1	1980	52	8763.8	1935	52	12330.4

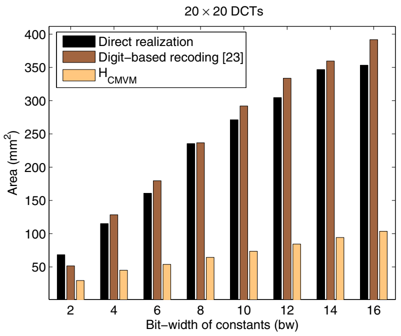
Table 4. Summary of gate-level results on 20 × 20 DCT designs

bw	CMVM problem						CMVM problem under a delay constraint					
	[12]			Hcmvm			[27]			Hcmvm-dc		
	area	delay	pd	area	delay	pd	area	delay	pd	area	delay	pd
2	36.5	3202	2.0	29.6	3066	1.7	35.7	2963	1.8	30.3	3019	1.7
4	49.3	4132	3.5	45.1	4015	3.3	48.5	3924	3.4	48.2	3911	3.5
6	60.7	4176	4.6	53.8	4237	4.2	61.0	4094	4.5	58.4	4060	4.5
8	76.3	5473	6.4	64.5	4971	5.6	71.7	4439	5.7	69.9	4539	5.6
10	87.7	5435	7.7	73.6	5325	6.5	84.5	4775	6.9	80.6	4799	6.8
12	101.9	5262	9.4	84.4	5704	7.6	99.7	4988	8.3	94.8	4966	8.1
14	112.7	5837	10.8	94.3	5724	9.2	111.9	5205	10.2	105.4	5480	9.5
16	122.5	5812	13.1	103.6	5846	11.0	126.0	5417	12.0	120.6	5683	11.5
Total	647.6	39329	57.5	548.7	38888	49.1	639.0	35805	52.9	608.2	36457	51.2

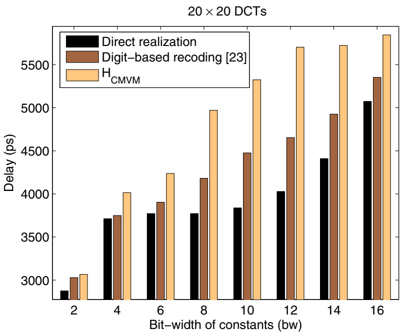
Observe from Table 3 that Hcmvm and Hcmvm-dc find respectively better solutions than the CSE heuristics [12] and [27], requiring 6.75 and 5.62 less operations on average. Note that all these algorithms obtain a solution with the same number of operations for DCTs when *bw* is equal to 4 and 8. However, the run times of Hcmvm and Hcmvm-dc are greater than those of [12, 27], since they may take more than one iteration due to the new realizations of linear transforms found by the difference method. Also, the run time of Hcmvm is longer than Hcmvm-dc on average since Hcmvm-dc may require fewer number of iterations than Hcmvm due to the delay constraint.

Table 4 presents the gate-level results of 20 × 20 DCTs synthesized based on the solutions of algorithms given in Table 3. In Table 4, *area* (mm^2), *delay* (*ps*), and *pd* (mW) stand for area, delay, and RTL power dissipation estimation, respectively. In this experiment, the bitwidths of input variables were taken as 16 and DCTs were synthesized under the minimum area design strategy.

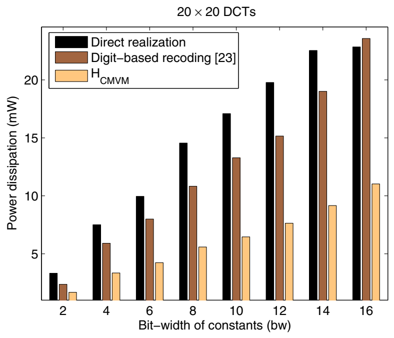
Observe from Table 4 that the solutions of Hcmvm yield low-complexity DCT designs (but with a large delay due to a large number of adder-steps) and high-speed DCT designs with low-complexity are obtained by the solutions of Hcmvm-dc with respect to designs obtained by the heuristics of [12] and [27], respectively. This is because they find a solution with less number of operations than the algorithms [12, 27] and they also consider some hardware optimizations. The impact of the latter fact can be easily observed on the results given



(a)



(b)



(c)

Fig. 9. Gate-level results of 20 × 20 DCTs: (a) Area; (b) Delay; (c) Power dissipation

in Table 4 when *bw* is 4 and 8, where all the algorithms find a solution with the same number of operations. The DCT designs obtained by the hybrid algorithms also consume less power on average with respect to those synthesized by the solutions of the CSE heuristics of [12, 27]. This is primarily because of less area in the DCTs designed using the solutions of the hybrid algorithms.

Figure 9 presents the gate-level results of direct realizations of 20 × 20 DCTs, where linear transforms are realized using additions and multipliers. The gate-level results of DCTs when they were synthesized under the shift-adds architecture using the digit-based recoding technique [25], that does not exploit any partial product sharing, are also given in this figure. These results are compared with those of DCTs obtained by Hcmvm. In the digit-based recoding method [25], the constants were defined under CSD and the linear transforms were realized with addition/subtraction operations as in a binary tree so that the minimum number of adder-steps of DCTs is achieved. A similar approach was also taken in the direct realizations of DCTs using multipliers and additions.

Observe from Figures 9(a) and (c) that the use of a high-level algorithm targeting the minimization of the number of operations in the multiplierless design of linear transforms leads to significant reductions in area and power dissipation when compared to the direct realizations of DCTs and the digit-based recoding technique [25]. However, as can be observed from Figure 9(b), the shift-adds designs insert larger delay than those of direct realizations due to a large number of addition/subtraction operations in series.

5 Conclusions

This chapter addressed the problem of minimizing the number of operations in the multiplierless design of linear transforms and introduced a hybrid algorithm, Hcmvm, that combines an efficient GB difference technique and an improved CSE algorithm. Since the proposed hybrid algorithm can lead to a solution with the fewest number of operations, but with a large number of adder-steps due to the sharing of partial products, this chapter also presented its modified version, Hcmvm-dc, that can handle the delay constraint. The experimental results on a comprehensive set of instances showed that the hybrid algorithms yield significantly better solutions than previously proposed algorithms at both high-level and gate-level. It was also indicated that the shift-adds design of linear transforms with the use of high-level algorithms lead to significant reductions in gate-level area compared to linear transforms designed using multipliers.

References

1. Quereshi, F., Gustafsson, O.: Low-Complexity Reconfigurable Complex Constant Multiplication for FFTs. In: Proc. of IEEE International Symposium on Circuits and Systems, pp. 24–27 (2009)
2. Thong, J., Nicolici, N.: A Novel Optimal Single Constant Multiplication Algorithm. In: Proc. of Design Automation Conference, pp. 613–616 (2010)
3. Kang, H.J., Park, I.C.: FIR Filter Synthesis Algorithms for Minimizing the Delay and the Number of Adders. IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing 48(8), 770–777 (2001)
4. Hartley, R.: Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers. IEEE Trans. on Circuits and Systems II 43(10), 677–688 (1996)
5. Boullis, N., Tisserand, A.: Some Optimizations of Hardware Multiplication by Constant Matrices. IEEE Trans. on Computers 54(10), 1271–1282 (2005)
6. Wallace, C.: A Suggestion for a Fast Multiplier. IEEE Trans. on Electronic Computers 13(1), 14–17 (1964)
7. Gallagher, W., Swartzlander, E.: High Radix Booth Multipliers Using Reduced Area Adder Trees. In: Proc. of Asilomar Conference on Signals, Systems and Computers, pp. 545–549 (1994)
8. Nguyen, H., Chatterjee, A.: Number-Splitting With Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis. IEEE Trans. on VLSI 8(4), 419–424 (2000)
9. Potkonjak, M., Srivastava, M., Chandrakasan, A.: Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination. IEEE Trans. on Computer-Aided Design of Integrated Circuits 15(2), 151–165 (1996)
10. Aksoy, L., Costa, E., Flores, P., Monteiro, J.: Exact and Approximate Algorithms for the Optimization of Area and Delay in Multiple Constant Multiplications. IEEE Trans. on Computer-Aided Design of Integrated Circuits 27(6), 1013–1026 (2008)

11. Yurdakul, A., Dündar, G.: Multiplierless Realization of Linear DSP Transforms by Using Common Two-Term Expressions. The Journal of VLSI Signal Processing 22(3), 163–172 (1999)
12. Hosangadi, A., Fallah, F., Kastner, R.: Reducing Hardware Complexity of Linear DSP Systems by Iteratively Eliminating Two-Term Common Subexpressions. In: Proc. of Asia and South Pacific Design Automation Conference, pp. 523–528 (2005)
13. Aksoy, L., Costa, E., Flores, P., Monteiro, J.: Optimization Algorithms for the Multiplierless Realization of Linear Transforms. ACM Trans. on Design Automation of Electronic Systems 17(1), Article 3 (2012)
14. Bull, D., Horrocks, D.: Primitive Operator Digital Filters. IEE Proc. G: Circuits, Devices and Systems 138(3), 401–412 (1991)
15. Dempster, A., Macleod, M.: Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters. IEEE Trans. on Circuits and Systems II 42(9), 569–577 (1995)
16. Gustafsson, O., Wanhammar, L.: A Novel Approach to Multiple Constant Multiplication Using Minimum Spanning Trees. In: Proc. of IEEE Midwest Symposium on Circuits and Systems, pp. 652–655 (2002)
17. Voronenko, Y., Püschel, M.: Multiplierless Multiple Constant Multiplication. ACM Trans. on Algorithms 3(2) (2007)
18. Aksoy, L., Gunes, E., Flores, P.: Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate. Elsevier Journal on Microprocessors and Microsystems 34(5), 151–162 (2010)
19. Dempster, A., Gustafsson, O., Coleman, J.: Towards an Algorithm for Matrix Multiplier Blocks. In: Proc. of IEEE European Conference on Circuit Theory and Design, pp. 1–4 (2003)
20. Gustafsson, O., Ohlsson, H., Wanhammar, L.: Low-Complexity Constant Coefficient Matrix Multiplication Using a Minimum Spanning Tree. In: Proc. of Nordic Signal Processing Symposium, pp. 141–144 (2004)
21. Avizienis, A.: Signed-digit Number Representation for Fast Parallel Arithmetic. IRE Trans. on Electronic Computers EC-10, 389–400 (1961)
22. Garner, H.: Number Systems and Arithmetic. Advances in Computers 6, 131–194 (1965)
23. Reitwiesner, G.: Binary Arithmetic. Advances in Computers 1, 261–265 (1960)
24. Cappello, P., Steiglitz, K.: Some Complexity Issues in Digital Signal Processing. IEEE Trans. on Acoustics, Speech, and Signal Processing 32(5), 1037–1041 (1984)
25. Ercegovac, M., Lang, T.: Digital Arithmetic. Morgan Kaufmann (2003)
26. Gustafsson, O.: Lower Bounds for Constant Multiplication Problems. IEEE Trans. on Circuits and Systems II 54(11), 974–978 (2007)
27. Hosangadi, A., Fallah, F., Kastner, R.: Simultaneous Optimization of Delay and Number of Operations in Multiplierless Implementation of Linear Systems. In: Proc. of International Workshop on Logic Synthesis (2005)
28. Arfaee, A., Irturk, A., Laptev, N., Fallah, F., Kastner, R.: Xquasher: A Tool for Efficient Computation of Multiple Linear Expressions. In: Proc. of Design Automation Conference, pp. 254–257 (2009)
29. Lefevre, V.: Multiplication by an Integer Constant. Technical report, Institut National de Recherche en Informatique et en Automatique (2001)
30. Aksoy, L., Costa, E., Flores, P., Monteiro, J.: Finding the Optimal Tradeoff Between Area and Delay in Multiple Constant Multiplications. Elsevier Journal on Microprocessors and Microsystems 35(8), 729–741 (2011)
31. Nangate website, `http://www.nangate.com/`