A Hybrid Algorithm for the Optimization of Area and Delay in Linear DSP Transforms

Levent Aksoy
INESC-ID
Lisboa, Portugal

Eduardo Costa
Universidade Catolica de Pelotas
Pelotas-RS, Brazil

Paulo Flores
INESC-ID/IST TU Lisbon
Lisboa, Portugal

José Monteiro
INESC-ID/IST TU Lisbon
Lisboa, Portugal

***Abstract*—This paper addresses the problem of multiplierless realization of linear transforms using the fewest number of addition and subtraction operations and introduces a hybrid algorithm that incorporates a graph-based technique, called the difference method, and a Common Subexpression Elimination (CSE) algorithm. In the proposed algorithm, while the difference method extracts the most promising realizations of linear transforms in each iteration, the CSE algorithm achieves the most common minimum conflicting subexpressions in each solution of the difference method. This paper also describes how the hybrid algorithm can be modified in order to find a solution with the fewest number of operations under 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 algorithms.**

I. Introduction

The multiplication of data samples with constant coefficients is a ubiquitous operation in Digital Signal Processing (DSP) systems and can be categorized in four main classes: (a) the Single Constant Multiplication (SCM), where an integer coefficient c is multiplied by a single variable x, *i.e.*, cx. The SCM operation is frequently used in realizations of Fast Fourier Transforms (FFTs) [1] and fast Discrete Cosine Transforms (DCTs) [2]; (b) the Multiple Constant Multiplications (MCM), that is the multiplication of a set of constants C with a single variable x, *i.e.*, $y_j = c_j x$. Among many others, the MCM operation dominates the complexity of Finite Impulse Response (FIR) filters in transposed form [3]; (c) the multiplication of multiple constants with the input data x shifted in time, *i.e.*, $y = \sum_j c_j x(t-j)$, which actually computes the output of an FIR filter in direct form [4]; (d) and the Constant Matrix-Vector Multiplication (CMVM), where a constant matrix C is multiplied with a vector X including multiple variables, *i.e.*, $y_j = \sum_k c_{j,k} x_k$. The CMVM operation occurs in implementations of DCTs, Infinite Impulse Response (IIR) filters, filter banks, and error correcting codes [5]. Observe that the CMVM operation corresponds to an SCM operation when both j and k are set to 1, to an MCM operation when k is 1, and to an operation described in (c) when j is 1.

Since the realization of a multiplication operation in hardware is expensive in terms of area, delay, and power dissipation and the constant coefficients are determined beforehand by the DSP algorithms, the multiplication of constants with data samples are generally realized using only addition, subtraction, and shift operations [6]. Note also that shifts can be realized using only wires without representing any area cost. Hence, an important optimization problem is to realize the constant multiplications using the fewest number of addition/subtraction operations, which is an NP-complete problem [7].

This work was supported by the *Portuguese Foundation for Science and Technology* (FCT) research project Multicon - PTDC/EIA-EIA/103532/2008.

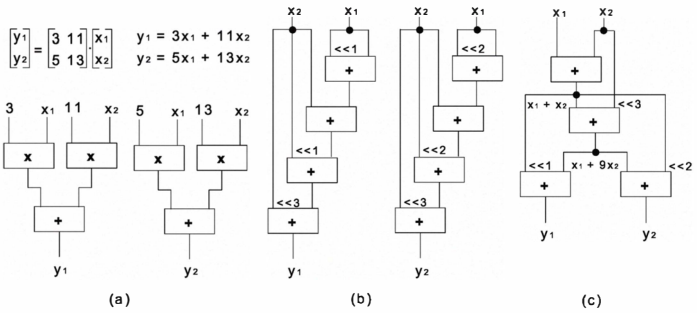

Fig. 1. (a) Direct realization of linear transforms $y_1 = 3x_1 + 11x_2$ and $y_2 = 5x_1 + 13x_2$; Shift-adds implementations of the linear transforms: (b) without partial product sharing; (c) with partial product sharing.

A straightforward way for the multiplierless realization of constant multiplications, generally known as the digit-based recoding method [8], 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, as illustrated in Figure 1(a). The decomposed forms of linear transforms $y_1 = 3x_1 + 11x_2$ and $y_2 = 5x_1 + 13x_2$ are given as follows:

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

where the computation of each y_1 and y_2 requires 4 operations, a total of 8 operations, as depicted in Figure 1(b).

Moreover, the shift-adds design of constant multiplications enables 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. Returning to our example given in Figure 1(a), finding the common subexpressions $x_1 + x_2$ and $x_1 + 9x_2$ yields a solution with 4 operations, as illustrated in Figure 1(c).

The last two decades have seen tremendous effort on the development of high-level algorithms for the multiplierless realization of constant multiplications. These algorithms can be categorized in two classes: Common Subexpression Elim-

978-1-4577-0170-2/11/$26.00 ©2011 IEEE

ination (CSE) [4], [5], [9], [10] and graph-based (GB) [11]–[14] 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 the 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 [12].

However, little attention has been given to the multiplierless realization of the CMVM operation compared to the MCM design. This is mainly because a high-level algorithm designed for the MCM problem can be used for the implementation of a CMVM operation or can be modified for the CMVM problem. In the former, one can apply an MCM algorithm on the constants of each column of the matrix C initially and then, utilize the sharing of constants in the rows of the matrix [9]. In the latter, each constant c_j and the variable x in MCM 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 problem as shown in Section IV, the efficient GB algorithms modified for the CMVM problem can be only applied to small size matrices with small constants as noted in [13], [14].

In this paper, we introduce a hybrid algorithm for the design of CMVM operations that incorporates less-complex and time-efficient CSE and GB algorithms to take the advantages of both techniques. The hybrid algorithm iteratively finds alternative realizations of linear transforms using the GB difference method and applies a CSE heuristic to further reduce the 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 by using a GB algorithm and the main drawback of a GB algorithm, *i.e.*, its time-consuming search process, is partially decreased by using a CSE heuristic. Although the proposed algorithm finds good solutions in terms of the number of operations, leading to low-complexity designs at gate-level, its solutions are generally realized in a large number of operations in series, yielding CMVM designs with large delay. To overcome this disadvantage, we also describe its modified version that can find a solution under a delay constraint and enables us to find the optimal tradeoff between area and delay in the design of linear transforms.

The rest of the paper proceeds as follows. Section II gives the background concepts and the hybrid algorithms are introduced in Section III. Experimental results are presented in Section IV and finally, Section V concludes the paper.

II. Background

This section presents the concepts related with the proposed algorithms and gives an overview on the algorithms designed for the CMVM problem.

A. Number Representations

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, $\{-1,0,1\}$. The *Canonical Signed Digit* (CSD) representation [4] is a signed digit system that has a unique representation for each number and verifies the following properties: i) two non-zero digits are not adjacent; ii) the number of non-zero digits is minimal. Any n digit number in CSD has at most $\lceil (n+1)/2 \rceil$ non-zero digits, and on average, the number of non-zero digits is reduced by 33% when compared to binary. Since hardware requirements are reduced due to the minimum number of non-zero digits, it is widely used in multiplierless realizations of constant multiplications.

B. Problem Definitions

Given an $m \times n$ constant matrix C with $c_{j,k} \in \mathbb{Z}$ 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

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

where j and k range from 1 to m and n respectively.

For the shift-adds realization of the CMVM operation, the fundamental optimization problem, which is called *the CMVM problem*, is defined as: Given the set of linear transforms $Y = \{y_1, \ldots, y_m\}$, find the minimum number of addition and subtraction operations that generate the linear transforms.

In many DSP systems, performance is also a crucial parameter. Hence, 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 the CMVM operation is generally considered as the number of adder-steps, which denotes the maximal number of adders/subtracters in series to produce any constant multiplication [3]. The minimum adder-step of a linear transform y_j is computed by decomposing the constants $c_{j,k}$ in y_j under a number representation and finding the number of terms in its decomposed form, $S(y_j)$. Thus, its minimum adder-step value is determined as $\lceil log_2 S(y_i) \rceil$, as if all its terms in the decomposed form were realized in a binary tree. For example, each linear transform given in Figure 1(a), *i.e.*, y_1 and y_2, has minimum 3 adder-step realizations when constants are defined under binary. Hence, for a set of linear transforms, $Y = \{y_1, \ldots, y_m\}$, the minimum adder-step of the CMVM operation [3] is computed as:

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

Note that, in general, min_delay_{CMVM} is determined when constants are defined under CSD, since the CSD representation of a constant includes the minimum number of non-zero digits. Thus, *the CMVM problem under a delay constraint* can be defined as: Given the set of linear transforms $Y = \{y_1, \ldots, y_m\}$ and the delay constraint dc with $dc \geq min\_delay_{CMVM}$, find

the minimum number of addition and subtraction operations that generate the linear transforms without exceeding dc.

C. Related Work

Although there are many efficient algorithms designed for the MCM problem, only a few methods have been proposed for the CMVM problem. Here, we mention only the algorithms applied to the CMVM problem, although any MCM algorithm can be easily modified for the CMVM problem.

The CMVM problem was formalized as a 0-1 Integer Linear Programming (ILP) problem in [15]. The possible implementations of linear transforms were found when constants are defined under a number representation in their decomposed forms. However, due to the exponential growth in the size of 0-1 ILP problems, the CSE algorithm [15] only considers the 2-term subexpressions. On the other hand, the CSE heuristic of [10] initially obtains the decomposed forms of linear transforms when constants are defined under binary or CSD. Then, in each iteration, it finds the most common 2-term subexpression and replaces it within the linear transforms. This algorithm was also extended to handle a delay constraint in [16]. The CSE algorithm [5] is similar to the algorithm [10] but, it relies on an efficient CSE algorithm [17] that iteratively searches a subexpression with the maximal number of terms and with at least 2 occurrences. In the heuristic of [5], the selection of a subexpression is also modified by taking into account the conflicts between the possible subexpressions. Furthermore, in [9], a CSE algorithm designed for the MCM problem is initially applied to find the common subexpressions in the columns of the constant matrix and then, in its rows.

In [13], an efficient GB algorithm [11] designed for the MCM problem is modified to handle the CMVM problem. As done in [11], the algorithm of [13] iteratively approaches to the linear transforms to be implemented with the available expressions by implementing subexpressions that require the minimum implementation cost. Note that in the beginning, the input variables $x_1, x_2, \ldots, x_n$ and their shifted values are available. However, the procedure in the GB algorithm of [13] is computationally intensive and thus, it can only be applied on small size matrices. Moreover, the algorithm of [14] 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 the required differences. The algorithm iterates until all the linear transforms are synthesized. However, as stated in [14], due to the application of the MST algorithm in each iteration, the algorithm is highly restricted in terms of the number of linear transforms and the bit-widths of constants.

III. The Hybrid Algorithms

This section presents the hybrid algorithm, called HCMVM, designed for the CMVM problem and its modified version, HCMVM-DC, for the CMVM problem under a delay constraint.

A. The HCMVM Algorithm

The HCMVM algorithm can handle the constants under binary and CSD representation 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.*, the expression is divided by 2 until one of its constants is odd and the expression is multiplied by -1, if the sign of the first non-zero constant in the expression is negative. The expressions are stored in a set called *Eset* without repetition.

Then, as done in the GB algorithms [11], [12], the linear transforms that can be synthesized using a single operation, whose inputs 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 includes 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 from the input variables and y_3 can be synthesized as $y_1 + y_2 \ll 1$. This is the optimal part, meaning that, when all the linear transforms are realized in this part, the minimum solution is obtained.

If there are still linear transforms in *Eset* after the optimal part, the algorithm switches to its heuristic part. In this part, initially it finds a solution on expressions in *Eset* with the CSE algorithm, H_{2MC}, that will be described in Section III-A1, and records its solution, considering also the number of elements in *Iset*, as the best solution found so far (*bs*). Then, the cost of each linear transform in *Eset* is computed as the total number of non-zero digits of each constant in a number representation, binary or CSD, that H_{2MC} uses in definition of constants. The linear transforms are sorted in a descending order based on their cost values. For each expression in *Eset*, $Eset_i$, with its cost value $cost_i$, where $i < m$ and m denotes the number of expressions in *Eset*, all the differences of $Eset_i$ with an expression in *Eset*, $Eset_j$, where $i < j \leq m$, are computed as $d_{ij} \ll l_1 = Eset_i - Eset_j \ll l_2$, where $l_1, l_2 \geq 0$ denote the left shifts. The cost of each difference is determined in terms of the total number of non-zero digits of each constant under the given number representation and a difference with the minimum cost value ($cost_d$) is determined. If $cost_d < cost_i - 1$, then $Eset_i$ is moved from *Eset* to *Iset* and the difference with the minimum cost is added into *Eset* in place of $Eset_i$. After all differences for each expression in *Eset* (except $Eset_m$) are explored, H_{2MC} is applied on the expressions in *Eset* and a set of operations realizing the expressions in *Eset* is obtained. If the number of operations in the solution of H_{2MC} plus the number of elements in *Iset* is less than *bs*, it is updated with this value. HCMVM iterates until there are no more differences that can be replaced with the expressions in *Eset*.

The procedure of HCMVM is illustrated on the first example of [14] when H_{2MC} defines the constants under CSD as given in Figure 2. In this figure, the values between parenthesis next to the expressions denote the respective cost values. Initially, H_{2MC} is applied on linear transforms and a solution with 19 operations is obtained. Then, in the first iteration

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 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 H_{2MC} on *Eset*:10 operations, Total:10+3=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 H_{2MC} on *Eset*:9 operations, Total:9+4=13

Fig. 2. The procedure of the HCMVM algorithm.

of HCMVM, 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. They 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, a total of 13 operations are required, considering that the expressions in *Iset* are synthesized using a single operation. 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. As reported in [14], the algorithms of [13], [14] find a solution with 14 operations on this instance.

1) The H_{2MC} algorithm: This previously mentioned method is based on the CSE heuristics [4], [10] that compute the most common (MC) 2-term subexpressions iteratively. 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 as the most common minimum conflicting (MCmc) 2-term subexpressions.

In HCMVM, the H_{2MC} algorithm takes *Eset* as an input and returns the *Sset* that includes the subexpressions required to realize all the expressions of *Eset*. 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 and stored in a set called *Dset*. The part of H_{2MC}, where the MCmc 2-term subexpressions are found and replaced in the decompositions of expressions, is as follows:

1) Form an empty 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 the occurrences of the subexpression in the elements of *Dset* considering its shifted and signed 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.*, MCmc 2-term subexpressions.
5) Choose 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, and go to Step 2.

After the sharing of the MCmc 2-term subexpressions is exploited, in each decomposed form of an expression in the final *Dset*, which includes greater than one term, in an iterative manner, two terms are selected and are realized by a single operation. The result is labeled with a variable, is stored in *Sset*, and these two terms are replaced with this variable in the decomposed form of the expression. This process continues until the number of terms is 1. However, as stated in [18], finding the fewest number of operations does not always lead to a design with optimal area at gate-level. Hence, to further reduce the area of a CMVM design, for an expression in the final *Dset*, we initially 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 set, we iteratively select two terms that have the smallest bit-width, *i.e.*, the narrowest, to be realized using an adder in order to reduce the size of the operation. Finally, if *Mset* is not empty, we use a subtracter to realize the expression. Consider the expression $y = a - b - c - d$ where each term is a subexpression and their bit-widths are 16, 8, 10, and 16 respectively. Thus, y is realized as $a - ((b + c) + d)$ in H_{2MC}.

B. The HCMVM-DC Algorithm

In the preprocessing phase of HCMVM-DC, we also compute the minimum adder-step of each linear transform as described in Section II-B. Then, given the delay constraint, dc, in the optimal part, we synthesize the linear transforms using a single operation if their realizations do not exceed dc considering their minimum adder-step values. While searching the promising differences, we compute the minimum adder-step of each difference and accept the implementation of an expression if its realization does not violate dc. Returning to our example in Figure 2, given the delay constraint, $dc = 4$, *i.e.*, the minimum adder-step of the CMVM operation, the realization of $Eset_1 = Eset_2 + d_1$ in the first iteration is not

TABLE I

SUMMARY OF RESULTS OF HIGH-LEVEL SYNTHESIS ALGORITHMS ON $m \times m$ RANDOMLY GENERATED MATRICES WITH 8-BIT CONSTANTS.

m	CMVM Problem									CMVM Problem under a Delay Constraint					
	[9]		[10]		[5]	H_{2MC}		HCMVM		[8]		[16]		HCMVM-DC	
	adder	step	adder	step	adder	adder	step	adder	step	adder	step	adder	step	adder	step
2	9.00	4.89	8.80	3.54	9.7	8.73	3.60	8.16	4.38	10.90	3.12	8.99	3.12	8.80	3.12
4	33.27	7.06	32.09	5.88	31.2	31.70	5.84	27.62	7.67	42.96	4.07	32.82	4.07	32.13	4.07
6	72.41	8.93	67.95	7.59	66.1	66.52	7.66	57.29	10.00	95.28	5.00	68.10	5.00	66.75	5.00
8	125.59	10.55	116.39	9.23	113.2	114.10	9.16	96.27	11.86	165.58	5.08	119.65	5.08	117.21	5.08
10	192.69	11.62	175.72	10.68	172.4	172.03	10.48	143.49	13.19	251.84	6.00	175.76	6.00	157.72	6.00
12	275.13	12.62	246.54	12.02	241.6	240.87	11.68	200.39	14.59	355.20	6.00	247.06	6.00	241.57	6.00
14	371.05	13.65	327.08	13.23	322.9	320.00	13.00	264.26	15.54	472.93	6.00	330.20	6.00	324.01	6.00
16	483.85	14.32	417.89	14.38	412.4	407.49	13.96	338.33	16.30	607.10	6.03	431.00	6.03	423.24	6.03

possible in HCMVM-DC, because the realizations of both $Eset_1$ and $Eset_2$ require minimum 4 adder-steps and hence, any implementation of $Eset_1$ with $Eset_2$ always violates $dc = 4$.

Moreover, in H_{2MC}, we initially find the most common 2-term subexpressions, whose selections will not lead to a realization greater than dc, and then, we obtain the minimum conflicting 2-term subexpression among these subexpressions. We also consider the hardware optimizations described in Section III-A1 taking into account dc. Thus, with these modifications, the HCMVM-DC algorithm can find a solution with the fewest number of operations under a delay constraint.

IV. EXPERIMENTAL RESULTS

This section presents the high-level results of hybrid algorithms on random instances and DCTs and compare them with those obtained by the previously proposed algorithms. It also introduces the gate-level results of DCTs designed using the solutions of high-level algorithms. To design linear transforms at gate-level, we also developed a tool that automatically describes the solutions of the high-level synthesis algorithms under the shift-adds architecture in VHDL and uses the Cadence Encounter RTL Compiler to synthesize the circuits with the Nangate 45nm Open Cell library [19].

As the first experiment set, we used $m \times m$ matrices, where m varies in between 2 and 16 in step of 2, consisting of randomly generated numbers in between $[2^7 + 1, 2^8 - 1]$, *i.e.*, 8-bit constants. There exist 100 instances for each matrix type, a total of 800 instances. Table I presents the results of algorithms, where *adder* and *step* denote the average number of operations and of adder-steps respectively. In the algorithm of [9], we used the exact GB algorithm [12] as an MCM algorithm to obtain the minimum number of operations for each column of a matrix. We also implemented the algorithms [10], [16] and the results of the algorithm [5] were taken from its paper. In HCMVM-DC and the algorithm [16], the delay constraint (dc) was set to the minimum delay of the CMVM operation as computed in Eqn. 2. In digit-based recoding technique [8], the constants were defined under CSD and the linear transforms were realized in a binary tree so that the minimum number of adder-step is achieved, as done in HCMVM-DC and the algorithm [16]. Note that its *adder* results represent the worst-case scenario for both CMVM problems. In CSE algorithms, the constants were defined under CSD.

Observe from Table I that the hybrid HCMVM algorithm finds significantly better solutions than the CSE algorithms [5], [10] and H_{2MC} in terms of the number of operations due to the use of the difference method. However, its solutions lead to CMVM designs with large number of adder-steps. On the other hand, the algorithm of [16] and HCMVM-DC find solutions with the minimum number of adder-steps, including greater number of operations than the algorithm of [10] and HCMVM respectively. Although the effect of the difference method is partially diminished due to the minimum delay constraint in HCMVM-DC, it finds better solutions in terms of the number of operations than [16] on all matrix types. Also, its *adder* results are similar to those of the algorithms [5], [10] and H_{2MC}, although they are not restricted to any delay constraint. Moreover, the results of [8] and [9] clearly show the importance of partial product sharing and using algorithms that target directly the CMVM problem respectively.

As the second experiment, we used 20×20 DCTs, where the bit-width (bw) of the constants were defined from 2 bits to 16 bits with an increment of 2. Table II presents the high-level results of the algorithms, all written in MATLAB, where *CPU* denotes their run time in seconds on a PC with Intel Xeon at 2.33GHz and 4GB memory. Again, the constants were defined under CSD, and in the algorithm of [16] and HCMVM-DC, dc was set to the minimum adder-step of the design.

Observe from Table II that HCMVM finds better solutions than the CSE heuristic [10], requiring 6.75 less operations on average, although they obtained a solution with the same number of operations for DCTs when bw is equal to 4 and 8. Also, HCMVM-DC obtains on average better solutions than [16] in terms of the number of operations. The run times of HCMVM and HCMVM-DC are greater than those of [10], [16], 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 larger than HCMVM-DC on average, since HCMVM-DC may require less iterations than HCMVM due to the delay constraint.

Table III presents the gate-level results of DCT designs synthesized based on the solutions of algorithms given in Table II. Also, it introduces the results on direct realizations of DCTs, where each linear transform was described in VHDL as the additions of constant multiplications by an input variable, as given in Eqn. 1. In this experiment, the bit-widths of input variables were taken as 16 and DCTs were synthesized under the minimum area design strategy in the synthesis tool, which includes advanced optimization techniques. In this table, *area* (mm^2), *delay* (ps), and *power* (mW) indicate area, delay, and RTL power estimation, respectively, as reported by the Cadence Encounter RTL Compiler after synthesis.

TABLE II
SUMMARY OF RESULTS OF HIGH-LEVEL SYNTHESIS ALGORITHMS ON 20×20 DCTS.

bw	CMVM Problem						CMVM Problem under a Delay Constraint					
	[10]			HCMVM			[16]			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
Tot.	1954	84	3967.8	1900	84	21269.1	1980	52	8763.8	1935	52	12330.4

TABLE III
SUMMARY OF GATE-LEVEL RESULTS ON 20×20 DCTS.

bw	Direct Realization			CMVM Problem						CMVM Problem under a Delay Constraint					
				[10]			HCMVM			[16]			HCMVM-DC		
	area	delay	power	area	delay	power	area	delay	power	area	delay	power	area	delay	power
2	68.4	2874	3.3	36.5	3202	2.0	29.6	3066	1.7	35.7	2963	1.8	30.3	3019	1.7
4	115.2	3712	7.5	49.3	4132	3.5	45.1	4015	3.3	48.5	3924	3.4	48.2	3911	3.5
6	160.8	3772	10.0	60.7	4176	4.6	53.8	4237	4.2	61.0	4094	4.5	58.4	4060	4.5
8	235.4	3772	14.6	76.3	5473	6.4	64.5	4971	5.6	71.7	4439	5.7	69.9	4539	5.6
10	271.2	3838	17.1	87.7	5435	7.7	73.6	5325	6.5	84.5	4775	6.9	80.6	4799	6.8
12	304.7	4029	19.8	101.9	5262	9.4	84.4	5704	7.6	99.7	4988	8.3	94.8	4966	8.1
14	346.8	4410	22.5	112.7	5837	10.8	94.3	5724	9.2	111.9	5205	10.2	105.4	5480	9.5
16	353.3	5074	22.8	122.5	5812	13.1	103.6	5846	11.0	126.0	5417	12.0	120.6	5683	11.5
Tot.	1855.8	31481	117.6	647.6	39329	57.5	548.7	38888	49.1	639.0	35805	52.9	608.2	36457	51.2

Observe from Table III that the use of high-level algorithms targeting a shift-adds architecture leads to significant improvements in terms of area and power dissipation when compared to the direct realizations of DCTs. Also, while the solutions of HCMVM yield low-complexity DCT designs (with great latency due to the large number of adder-steps), high-speed DCT designs with low-complexity are obtained by the solutions of HCMVM-DC with respect to designs obtained by [10] and [16] respectively. This is because, besides their better high-level results, HCMVM and HCMVM-DC also consider some hardware optimizations, whose impact can be easily observed on the results given in Table III when *bw* is 4 and 8, where all the algorithms find a solution with the same number of operations as can be seen in Table II.

V. CONCLUSIONS

We introduced a hybrid algorithm that includes an efficient GB difference technique and an improved CSE algorithm for the optimization of the number of operations in multiplierless realization of linear transforms. Since the proposed algorithm yields a solution with the fewest number of operations but with larger adder-steps due to the sharing of partial products, we also presented its modified version that can handle the delay constraint. The experimental results showed that they yield significantly better solutions than the previously proposed algorithms at both high-level and gate-level.

REFERENCES

[1] F. Quereshi and O. Gustafsson, "Low-Complexity Reconfigurable Complex Constant Multiplication for FFTs," in *ISCAS*, 2009, pp. 24–27.

[2] J. Thong and N. Nicolici, "A Novel Optimal Single Constant Multiplication Algorithm," in *DAC*, 2010, pp. 613–616.

[3] H.-J. Kang and I.-C. Park, "FIR Filter Synthesis Algorithms for Minimizing the Delay and the Number of Adders," *IEEE TCAS II*, vol. 48, no. 8, pp. 770–777, 2001.

[4] R. Hartley, "Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers," *IEEE TCAS II*, vol. 43, no. 10, pp. 677–688, 1996.

[5] N. Boullis and A. Tisserand, "Some Optimizations of Hardware Multiplication by Constant Matrices," *IEEE Transactions on Computers*, vol. 54, no. 10, pp. 1271–1282, 2005.

[6] H. Nguyen and A. Chatterjee, "Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis," *IEEE TVLSI*, vol. 8, no. 4, pp. 419–424, 2000.

[7] P. Cappello and K. Steiglitz, "Some Complexity Issues in Digital Signal Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 5, pp. 1037–1041, 1984.

[8] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.

[9] M. Potkonjak, M. Srivastava, and A. Chandrakasan, "Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination," *IEEE TCAD*, vol. 15, no. 2, pp. 151–165, 1996.

[10] A. Hosangadi, F. Fallah, and R. Kastner, "Reducing Hardware Complexity of Linear DSP Systems by Iteratively Eliminating Two-Term Common Subexpressions," in *ASPDAC*, 2005, pp. 523–528.

[11] A. Dempster and M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *IEEE TCAS II*, vol. 42, no. 9, pp. 569–577, 1995.

[12] L. Aksoy, E. Gunes, and P. Flores, "Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate," *Elsevier Journal on Embedded Hardware Design*, vol. 34, pp. 151–162, 2010.

[13] A. Dempster, O. Gustafsson, and J. Coleman, "Towards an Algorithm for Matrix Multiplier Blocks," in *ECCTD*, 2003, pp. 1–4.

[14] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Low-Complexity Constant Coefficient Matrix Multiplication Using a Minimum Spanning Tree," in *Nordic Signal Processing Symposium*, 2004, pp. 141–144.

[15] A. Yurdakul and G. Dündar, "Multiplierless Realization of Linear DSP Transforms by Using Common Two-Term Expressions," *The Journal of VLSI Signal Processing*, vol. 22, no. 3, pp. 163–172, 1999.

[16] A. Hosangadi, F. Fallah, and R. Kastner, "Simultaneous Optimization of Delay and Number of Operations in Multiplierless Implementation of Linear Systems," in *IWLS*, 2005.

[17] V. Lefevre, "Multiplication by an Integer Constant," Institut National de Recherche en Informatique et en Automatique, Tech. Rep., 2001.

[18] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Optimization of Area in Digital FIR Filters Using Gate-Level Metrics," in *DAC*, 2007, pp. 420–423.

[19] Nangate website, http://www.nangate.com/.