EURASIP Journal on Embedded Systems

RESEARCH Open Access

A novel method for the approximation of multiplierless constant matrix vector multiplication

Levent Aksoy[1], Paulo Flores[2] and José Monteiro[2*]

Abstract

Since human beings have limited perceptual abilities, in many digital signal processing (DSP) applications, e.g., image and video processing, the outputs do not need to be computed accurately. Instead, their computation can be approximated so that the area, delay, and/or power dissipation of the design can be reduced. This paper presents an approximation algorithm, called AURA, for the multiplierless design of the constant matrix vector multiplication (CMVM) which is a ubiquitous operation in DSP systems. AURA aims to tune the constants such that the resulting matrix leads to a CMVM design which requires the fewest adders/subtractors, satisfying the given error constraints. This paper also introduces its modified version, called AURA-DC, which can reduce the delay of the CMVM operation with a small increase in the number of adders/subtractors. Experimental results show that the proposed algorithms yield significant reductions in the number of adders/subtractors with respect to the original realizations without violating the error constraints, and consequently lead to CMVM designs with less area, delay, and power dissipation. Moreover, they can generate alternative CMVM designs under different error constraints, enabling a designer to choose the one that fits best in an application.

Keywords: Approximate computing, constant matrix vector multiplication, Multiplierless design, 0–1 integer linear programming, Area and delay optimization, Discrete cosine transform

1 Introduction

The complexity of many digital signal processing (DSP) algorithms is dominated by repeated multiplications of input samples by constant values. In particular, the constant matrix vector multiplication (CMVM) operation realizes the multiplication of an $m \times n$ constant matrix C by an $n \times 1$ variable vector X, i.e., $y_j = \sum_k c_{jk} x_k$, with $0 \leq j \leq m-1$ and $0 \leq k \leq n-1$. It appears in hybrid form finite impulse response filters [1] and linear DSP transforms, such as discrete cosine transforms (DCTs) and Hadamard and Reed-Muller transforms [2].

Since the realization of a multiplier in hardware is expensive in terms of area, delay, and power dissipation and the constants of the matrix C are determined beforehand, the CMVM operation is generally implemented using only shifts and adders/subtractors [2]. Note that shifts by a constant value can be realized using only wires which represent no hardware cost. Thus, the CMVM problem is defined as finding a minimum number of adders/subtractors which realize the CMVM operation. Over the years, many efficient methods, whose main task is to maximize the sharing of common subexpressions, have been introduced for the CMVM problem [2–10].

Approximate computing refers to a class of methods that relax the requirement of exact equivalence between the specification and implementation of a computing system [11]. This relaxation allows trading the accuracy of numerical outputs for reductions in area, delay, or power dissipation of the design [12]. Research activities on approximate computing range from a transistor level to an algorithmic level [13–22].

In this paper, we propose a new method, AURA, for the approximate computation of the CMVM operation under the shift-adds architecture. Given the original constant matrix C and error constraints to be satisfied, AURA aims to find an optimized matrix C', where the total number

*Correspondence: jcm@inesc-id.pt
[2]INESC-ID/ Técnico, Universidade de Lisboa, Lisbon, Portugal
Full list of author information is available at the end of the article

of nonzero digits of constants is minimum. This objective is based on the observation that constants with fewer nonzero digits lead to a CMVM design with a smaller number of adders/subtractors [13]. The problem of finding the optimum C' matrix is formulated as a 0–1 integer linear programming (ILP) problem. Since there may exist many such matrices, each leading to a CMVM design with a different number of adders/subtractors, in each iteration, AURA finds one of them, computes the number of occurrences of all possible two-term subexpressions in the corresponding CMVM operation, and assigns the matrix with the highest value of this parameter to the optimized matrix C'. Finally, the shift-adds realization of the optimized CMVM operation based on C' is found using the state-of-the-art method of [10] which considers the sharing of the most common two-term subexpressions. This paper also presents a modified version of AURA, AURA-DC, proposed to find a solution with the fewest adders/subtractors under a delay constraint given in terms of the number of adder-steps which denotes the maximum number of operations in series. Experimental results indicate that significant reductions in the number of operations and adder-steps can be obtained without violating the error constraint. It is shown on the 8×8 DCT designs that the solutions of the proposed methods lead to significant reductions in area, delay, and power dissipation according to the original 8×8 DCT design with a slight decrease of performance in image compression.

2 Background

This section gives the background concepts related to the proposed algorithms and presents an overview on the algorithms used for the multiplierless design of the CMVM operation.

2.1 Matrix norms

The errors in the constant matrix C are measured using matrix norms. In AURA, both norm-one (maximum column sum) and norm-infinity (maximum row sum), which are defined as $\|C\|_1 = \max_k \sum_j |c_{jk}|$ and $\|C\|_\infty = \max_j \sum_k |c_{jk}|$, respectively, are used. Thus, the norm-one and norm-infinity error constraints are, respectively, formed as $\|C - C'\|_1 \leq \epsilon_1$ and $\|C - C'\|_\infty \leq \epsilon_\infty$, where $\epsilon_1, \epsilon_\infty > 0$ are tolerable error values. Other norms can be also considered and easily adapted in the proposed algorithms.

2.2 0-1 ILP problem

The 0–1 ILP problem is the optimization of a linear objective function subject to a set of linear constraints and is generally defined as follows:

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

$$\text{subject to} \;\; \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}, \;\; \mathbf{x} \in \{0,1\}^k \tag{2}$$

In the objective function of the 0–1 ILP problem given in Eq. 1, w_i in $\mathbf{w}$ is a weight value associated with each variable x_i, where $1 \leq i \leq k$ and $\mathbf{w} \in \mathbb{Z}^k$. In Eq. 2, $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ denotes a set of j linear constraints, where $\mathbf{b} \in \mathbb{Z}^j$ and $\mathbf{A} \in \mathbb{Z}^j \times \mathbb{Z}^k$.

The minimization objective can be converted to a maximization objective by negating the objective function. Less-than-or-equal and equality constraints are respectively accommodated by the equivalences, $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b} \Leftrightarrow -\mathbf{A} \cdot \mathbf{x} \geq -\mathbf{b}$ and $\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \Leftrightarrow (\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}) \wedge (\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b})$.

2.3 Multiplierless design of the CMVM operation

The linear transforms, which represent the CMVM operation, are obtained by multiplying each row of the constant matrix by the variable vector. A straightforward approach for the shift-adds design of the CMVM operation, called the digit-based recoding (DBR) technique [23], has two steps: (i) define the constants under a number representation, e.g., binary or canonical signed digit (CSD) (An integer can be written in CSD using j digits as $\sum_{i=0}^{j-1} d_i 2^i$, where $d_i \in \{1, 0, -1\}$. Under CSD, nonzero digits are not adjacent and a minimum number of nonzero digits is used.); (ii) for the nonzero digits in the representations of constants, shift the variables according to the digit positions and add/subtract the shifted variables with respect to the digit values.

Consider as a running example the constant matrix $C = [23\;\;37; 11\;\;25]$ and the corresponding linear transforms $y_0 = 23x_0 + 37x_1$ and $y_1 = 11x_0 + 25x_1$. Their decompositions under CSD are given as follows:

$$y_0 = 23x_0 + 37x_1 = 32x_0 - 8x_0 - x_0 + 32x_1 + 4x_1 + x_1$$
$$y_1 = 11x_0 + 25x_1 = 16x_0 - 4x_0 - x_0 + 32x_1 - 8x_1 + x_1$$

requiring 10 operations, as shown in Fig. 1a.

The number of operations can be further reduced by sharing the common subexpressions. The common subexpression elimination (CSE) method of [3] finds all possible implementations of linear transforms by extracting only the two-term subexpressions and formalizes the problem of maximizing the sharing of subexpressions as a 0–1 ILP problem. The exact CSE algorithm of [9] follows a similar approach but considers all possible realizations of linear transforms. However, these methods can only be applied to CMVM instances with a small size of constant matrices due to the exponential growth in the size of 0–1 ILP problems. The CSE heuristics of [4, 6] iteratively find the most common two-term subexpression and replace it within the linear transforms. They differ in the selection of subexpressions that have the same number of occurrences. The CSE algorithm [2] iteratively searches a subexpression with the maximal number of terms and with at least two occurrences. The CSE heuristic of [8]

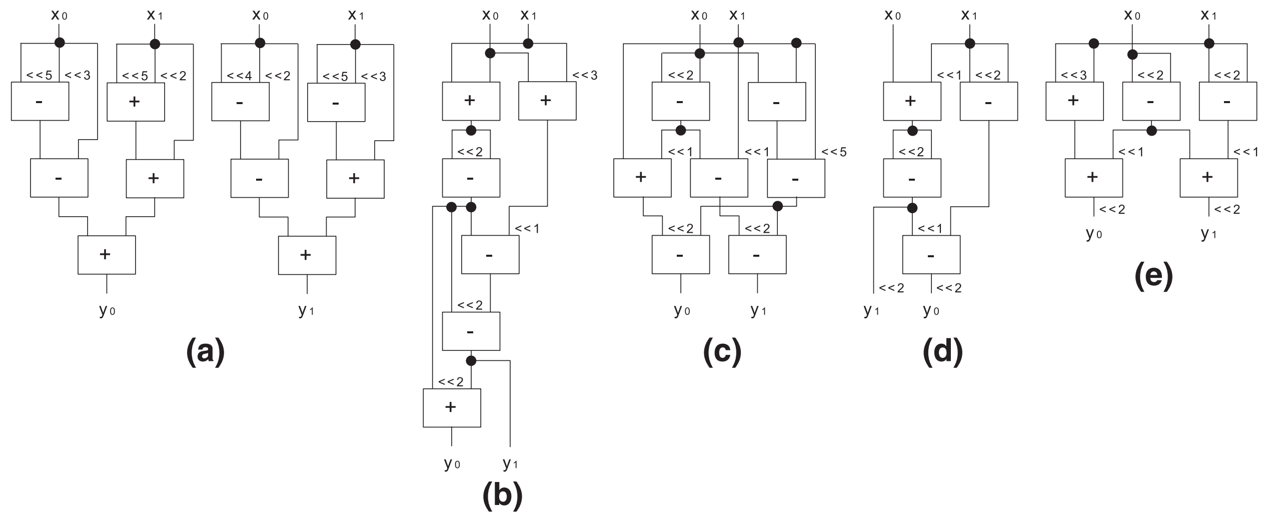

Fig. 1 Shift-adds designs of $y_0 = 23x_0 + 37x_1$ and $y_1 = 11x_0 + 25x_1$. **a** DBR method [23]. **b** HCMVM [10]. **c** HCMVM-DC [10]. Shift-adds designs of the solutions of the proposed approximation algorithms on these linear transforms: **d** AURA and **e** AURA-DC

chooses its subexpressions based on a cost value which is computed as the product of the number of terms in the subexpression and the number of its occurrences in the linear transforms. In turn, the algorithm of [5] initially computes the differences between each two linear transforms and determines their implementation cost values. Then, it uses a minimum spanning tree 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 hybrid algorithm of [10] iteratively finds the most promising differences of linear transforms and applies an improved CSE heuristic to further reduce the complexity of the CMVM operation.

Returning to our example, the hybrid algorithm of [10], HCMVM, finds a solution with six operations when the CSD representation is used in its CSE algorithm, sharing the common subexpressions x_0+x_1 and $3x_0+3x_1$ (Fig. 1b).

In many DSP systems, performance is a crucial parameter and circuit area is generally expendable 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. The minimum adder-steps of a linear transform y_j is found in three steps: (i) decompose its constants c_{jk} under a given number representation; (ii) find the total number of terms in its decomposed form $T(y_j)$, determined as $\sum_k S(c_{jk})$, where $S(c_{jk})$ denotes the number of nonzero digits of the constant c_{jk} under a given number representation; and (iii) compute $\lceil \log_2 T(y_j) \rceil$ as if all its terms in the decomposed form were realized in a binary tree. Thus, the minimum adder-step of the CMVM operation, MAS_{CMVM}, is the maximum of the minimum adder-step of its each linear transform, i.e., $\max_j\{\lceil \log_2 T(y_j) \rceil\}$ [24]. The methods of [7, 10] aim to find the fewest adders/subtractors realizing the CMVM operation and satisfying a delay constraint less than or equal to MAS_{CMVM}.

For our example, the minimum adder-steps for y_0 and y_1 under the CSD representation are 3, and thus, MAS_{CMVM} is found as 3. The hybrid algorithm of [10], HCMVM-DC, finds a solution with seven operations when the delay constraint is set to 3 and the CSD representation is used in its CSE algorithm, sharing the common subexpressions $3x_0$ and $x_0 - 33x_1$ (Fig. 1c). Observe that its solution has one more operation but two less adder-steps compared to the solution of HCMVM (Fig. 1b).

2.4 Exploring alternative sets of constants

The CMVM problem as enunciated above starts from a fixed set of constants in the matrix, and the best solution is found that maximizes the sharing of partial terms for that given input. There are problems however for which several sets of constants represent valid equivalent solutions. For these cases, we can run one CMVM minimization algorithm on each set of constants and select the set based on which one is more amenable for the sharing of partial terms.

The design of a finite input response (FIR) digital filter is a good example of these types of systems. Different algorithms can be used to determine the set of coefficients that verify a desired transfer function. With all the other filter characteristics the same, we can optimize the filter implementation by selecting the set of coefficients that maximize the sharing.

The work described in [25] goes one step further. Given the FIR desired filter characteristics (pass and stop frequencies, and allowed error for each band), the desired coefficients are determined so that the number of total partial terms required for the multiplications is minimized. The exact algorithm of [25] builds a search tree with clever pruning techniques the checks all valid constant combinations. To handle cases where the exact solution is not viable, a heuristic algorithm is also proposed that only makes a search around predefined values for the coefficients.

The technique we propose in the current paper follows a similar line of research. However, not only do we consider all possible accurate implementations, we also allow inaccurate computations in order to further explore the sharing of partial terms.

3 Approximate computing methods for DSP systems

Approximate computing has been motivated by the large and growing class of applications that demonstrate inherent error resilience, such as DSP, multimedia (images/audio/video), graphics, wireless communications, and emerging workloads such as recognition, mining, and synthesis [11]. The requirement for an exact computation is relaxed due to several factors: (1) the limited perceptual capability of humans (e.g., audio, video, graphics due to the ability of the human brain to “fill in” missing information and filter out high-frequency patterns), (2) a golden result is difficult to define or does not exist (e.g., web search, data analytics), (3) users are willing to accept approximate but good-enough results, and (4) noisy inputs [18].

The class of methods and techniques referred to as approximate computing usually relax the requirement of exact equivalence between the specification and implementation of a computing system [11]. This relaxation allows trading the accuracy of numerical outputs for reductions in area, delay, and/or power dissipation of the design [12].

In the last two decades, many design techniques, circuits, and algorithms have been introduced for approximate computing. The reader is referred to [18–20], for detailed surveys on approximate, stochastic, and probabilistic computing. The approaches for the design of approximate DSP systems can generally be grouped in three categories: (i) transistor level, (ii) gate level, and (iii) algorithmic level.

At the transistor level, the stochastic behavior of a binary switch under the influence of thermal noise is exploited and probabilistic CMOS (PCMOS) transistors are used to realize, first, the arithmetic circuits and then the DSP systems [20]. Other approaches consider scaling the voltage below the minimum voltage supply value required, so as to trade accuracy for power [26]. However, the observation that errors in bits of higher order affect the quality of the solution more than the lower order bits do led to operating adders of more significant bits with a higher voltage and downscale the voltages for lower bits [27].

At the gate level, approximate adders and multipliers are used to implement the DSP systems [21, 22]. In these cases, the circuits are designed to be error prone even in the absence of voltage over-scaling. The principle is to accept errors in rare cases for significant reductions in the logic complexity or length of the critical path. Changing the circuit truth table on selected positions results in a simplified implementation with lower power consumption, lower area, and higher performance at the cost of an approximation. Another kind of approximation technique is the reverse carry propagation [28] where approximations are introduced only in the least significant bits, which ensures that the error introduced is of small magnitude.

At the algorithmic level, design methodologies have been introduced for approximate DSP systems [12–15, 17, 29]. In most of these algorithms, a multiplier-free approximation of linear transforms was considered but the constants were approximated to the nearest integer power of 2 and the sharing of common subexpressions were not considered. This top-down approach of design, for approximating DSP systems, does not benefit from the low-level information about sharing of common subexpressions and how they are computed using approximate circuits. A solution to this problem was taken on a variant of the multiple constant multiplication (MCM) problem called multiple tunable constant multiplications (MTCM) problem, where each constant is not fixed as in the MCM problem but can be selected from a set of possible constants [30].

To the best of our knowledge, the algorithms we propose in this paper are the only methods that realize the approximation of the CMVM operation under the shift-adds architecture, considering the sharing of common subexpressions and satisfying the error constraints. Similar to the proposed algorithms, multiplier-free approximation of linear transforms was considered in [13, 14], but the constants are approximated to the nearest integer power of two without sharing of common subexpressions.

4 Proposed approximation algorithms

Under a tolerable error at the outputs of the CMVM operation, the constants of C can be changed such that the resulting (optimized) matrix C' leads to a CMVM design with the fewest adders/subtractors or adders-steps. Note that a matrix with constants including a small number of nonzero digits yields a CMVM design with a small number of operations [13]. Hence, the proposed algorithms find a constant matrix with the minimum total number of nonzero digits of constants under the given number representation, denoted as $N(C)$ which is computed as $\sum_j \sum_k S(c_{jk})$, satisfying the error constraints. In both AURA and AURA-DC, this problem is formulated as a 0–1 ILP problem. Note that there may exist many matrices with the minimum $N(C)$ value. Also, the hybrid algorithms of [10], which consider the sharing of the most common two-term subexpressions, are used to find the shift-adds design of the optimized CMVM operation in the proposed algorithms. Hence, both AURA and AURA-DC find more than one possible matrix with the minimum $N(C)$ value, for each matrix, compute the total number of occurrences of all possible tow-term subexpressions, denoted as $O(C)$, and return the one with the maximum $O(C)$ value as the optimized matrix C'. These steps are described in detail in the following two subsections using the constant matrix $C = [23\ 37; 11\ 25]$ as an example when ϵ_1 and ϵ_∞ are 2.

4.1 Details of AURA

While generating the $0-1$ ILP problem, possible values of each entry of the constant matrix C, c_{jk}, are determined to be in between $c_{jk} - r$ and $c_{jk} + r$ and stored in an array called R^{jk}, where $0 \le j \le m-1$, $0 \le k \le n-1$, and r denotes the range considered on each constant of C. Thus, for each constant in each R^{jk}, R_i^{jk} with $1 \le i \le |R^{jk}|$, a variable, $vR_i^{jk}@jk$, is generated (the "$@jk$" in the variable $vR_i^{jk}@jk$ is used to differentiate the same value of R_i^{jk} in different rows and columns of the constant matrix C). The objective function of the $0-1$ ILP problem is formed as a linear function of these variables whose weight values are $S(R_i^{jk})$ found under the given number representation, i.e., binary or CSD. The constraints of the $0-1$ ILP problem are obtained in three steps:

1. For each entry in C, the following constraint is generated to ensure that only one constant is chosen from R^{jk}.

$$\sum_i vR_i^{jk}@jk = 1, \ \forall j, k$$

2. The norm-one error constraint is formulated by generating the following constraint for each column of C.

$$\sum_j \sum_i |c_{jk} - R_i^{jk}| vR_i^{jk}@jk \le \epsilon_1, \ \forall k$$

3. The norm-infinity error constraint is formulated by generating the following constraint for each row of C.

$$\sum_k \sum_i |c_{jk} - R_i^{jk}| vR_i^{jk}@jk \le \epsilon_\infty, \ \forall j$$

After the 0–1 ILP problem is generated, a generic 0–1 ILP solver is applied to find a solution. Each entry of the new constant matrix is determined by finding the variables $vR_i^{jk}@jk$ set to 1 by the 0–1 ILP solver. For our example, the constant matrix [24 36; 10 24] is found when r is 4 and CSD representation is used. Note that while the $N(C)$ value of this matrix is 8, the $N(C)$ value is 12 for the original matrix.

Then, the corresponding linear transforms are obtained and their decomposed forms are found when constants are defined under a given number representation. The occurrences of each possible two-term subexpression are determined, considering its shifted and negative versions, and if its number of occurrences is greater than 1, the $O(C)$ value is updated. For our example, the decomposed forms of linear transforms under CSD are given as follows:

$$y_0 = 24x_0 + 36x_1 = 32x_0 - 8x_0 + 32x_1 + 4x_1$$
$$y_1 = 10x_0 + 24x_1 = 8x_0 + 2x_0 + 32x_1 - 8x_1$$

where only the two-term subexpression $x_0 - 4x_1$ appears twice. Thus, $O(C)$ is computed as 2.

In order to find another matrix with the minimum $N(C)$ value, the solution of the 0–1 ILP solver is turned into a constraint, indicating that it should not be found again. Suppose that the constants $R_i^{11}, R_i^{12}, \ldots, R_i^{mn}$ were chosen for each entry of the matrix. Thus, the following constraint is generated and added to the 0–1 ILP problem.

$$vR_i^{11}@11 + vR_i^{12}@12 + \ldots + vR_i^{mn}@mn \le m \cdot n - 1$$

After the augmented 0–1 ILP problem is solved, a different solution is obtained. For our example, [24 36; 12 24] is found in the second iteration. The decomposed forms of the corresponding linear transforms under CSD are found as follows:

$$y_0 = 24x_0 + 36x_1 = 32x_0 - 8x_0 + 32x_1 + 4x_1$$
$$y_1 = 12x_0 + 24x_1 = 16x_0 - 4x_0 + 32x_1 - 8x_1$$

where only the two-term subexpressions $4x_0 - x_0$ and $x_0 + 2x_1$ occur twice. Thus, $O(C)$ is 4.

This process iterates until a total of ni constant matrices are considered, where ni denotes the number of iterations, or a constant matrix with an $N(C)$ value greater than the minimum is obtained. The constant matrix with the maximum $O(C)$ value is determined to be the optimized matrix C'. For our example, C' is found as [24 36; 12 24].

Finally, the shift-adds design of the CMVM operation based on C' is found using HCMVM. For our example, the solution of AURA has four operations and three adder-steps, as shown in Fig. 1d. We note that the solution of HCMVM on [24 36; 10 24], i.e., the solution of AURA in the first iteration, has five operations.

4.2 Details of AURA-DC

AURA-DC is introduced to approximate the CMVM operation to have the fewest number of operations under the minimum number of adder-steps, satisfying the error constraints. To do so, whenever a constant matrix is found, its $\mathrm{MAS_{CMVM}}$ value is computed and the one with the minimum $\mathrm{MAS_{CMVM}}$ value is favored. In case of equality of the minimum $\mathrm{MAS_{CMVM}}$ values, the one with the maximum $O(C)$ value is preferred. Also, the shift-adds design of the CMVM operation based on C' is obtained using HCMVM-DC [10] when the delay constraint is set to its $\mathrm{MAS_{CMVM}}$ value. For our example, AURA-DC finds the optimized matrix as AURA, i.e., [24 36; 12 24]. But, its multiplierless design has five operations and two adder-steps as shown in Fig. 1e. Compared to the solution of AURA (Fig. 1d), it has one more operation but one less adder-step. Observe that AURA and AURA-DC may find the same optimized matrix but exploit different realizations.

4.3 Complexity of the proposed algorithms

Given the parameter r, the range of possible constants to be considered for each entry of the matrix is determined as $[c_{jk} - r,\ c_{jk} + r]$. Thus, the 0–1 ILP problem has $(2r+1)mn$ variables. In the proposed algorithms, r is set to 4. In turn, the number of constraints in the 0–1 ILP problem generated in the first iteration is $mn + m + n$. Given the number of iterations ni, the number of constraints in the 0–1 ILP problem generated in the last iteration is $mn + m + n + \mathrm{ni} - 1$, if the proposed algorithms were not terminated in an earlier iteration. In the proposed algorithms, ni is set to $m + n$. Observe that r and ni have an effect only on the number of variables and constraints, respectively. However, the length of a constraint, i.e., the number of terms, is directly proportional to r.

Table 1 presents the number of variables and constraints of the 0–1 ILP problem generated in the proposed algorithms for square matrices with different r and ni values. Note that the sizes of 0–1 ILP problems in Table 1 are in the reach of generic ILP solvers, and real-world DSP systems generally have m and n values less than or equal to 16.

Also, suppose that each linear transform y_j has t_j terms in its decomposed form. Thus, the number of the two-term subexpressions, to be searched for their occurrences, is $\sum_j t_j(t_j - 1)/2$ with $0 \le j \le m - 1$. For a two-term subexpression extracted from the jth transform y_j, its occurrences are searched in y_i with $j \le i \le m - 1$ and the subexpressions, whose occurrences have already been identified, do not need to be considered.

5 Experimental results

This section presents the optimized results of the proposed algorithms on randomly generated constant matrices under different error constraints and compares them against the original results of these instances. It also introduces the results of original and optimized 8 × 8 DCTs, their results on image compression, and their synthesis results on application specific integrated circuit (ASIC) and field programmable gate arrays (FPGA) design platforms. We note that AURA and AURA-DC were written in MATLAB and run on a PC with Intel Xeon at 2.4 GHz. Their solutions were found when the CSD representation was considered and SCIP2.0 [31] was used as a 0–1 ILP solver.

As the first experiment set, we used randomly generated $n \times n$ matrices with 8-b constants, where n ranges between 2 and 16, in steps of 2. For each group, there were 50 matrices, a total of 400. Figure 2 presents the results of HCMVM (denoted as original) and AURA (denoted as optimized) on these constant matrices when ϵ_1 and ϵ_∞ are $\frac{1}{2}n$, n, and $2n$. Note that the CPU time of AURA includes the CPU time of HCMVM to find the shift-adds design of the optimized CMVM operation.

Observe from Fig. 2 that as the tolerance on the output error increases, the number of adders/subtractors required to realize the CMVM operation decreases, since the number of alternative constants to be considered in an entry of the matrix increases. Note that the highest reduction in the number of operations between the original and optimized results are found as 10.7 %, 18.7 %, and 28.2 % when ϵ_1 and ϵ_∞ are $\frac{1}{2}n$, n, and $2n$, respectively. As the tolerance on the output error increases, the adder-step values of the CMVM operations also decrease. This is simply because the optimized matrix has the minimum $N(C)$ value under the given error constraints. For example, on

Table 1 Complexity of the 0–1 ILP problem

$m = n$	Number of variables			Number of constraints		
	$r = 4$	$r = 8$	$r = 16$	$\mathrm{ni} = m + n$	$\mathrm{ni} = 2(m + n)$	$\mathrm{ni} = 4(m + n)$
2	36	68	132	11	15	23
4	144	272	528	31	39	55
8	576	1088	2112	95	111	143
16	2304	4352	8448	319	351	415
32	9216	17,408	33,792	1151	1215	1343
64	36,864	69,632	135,168	4351	4479	4735

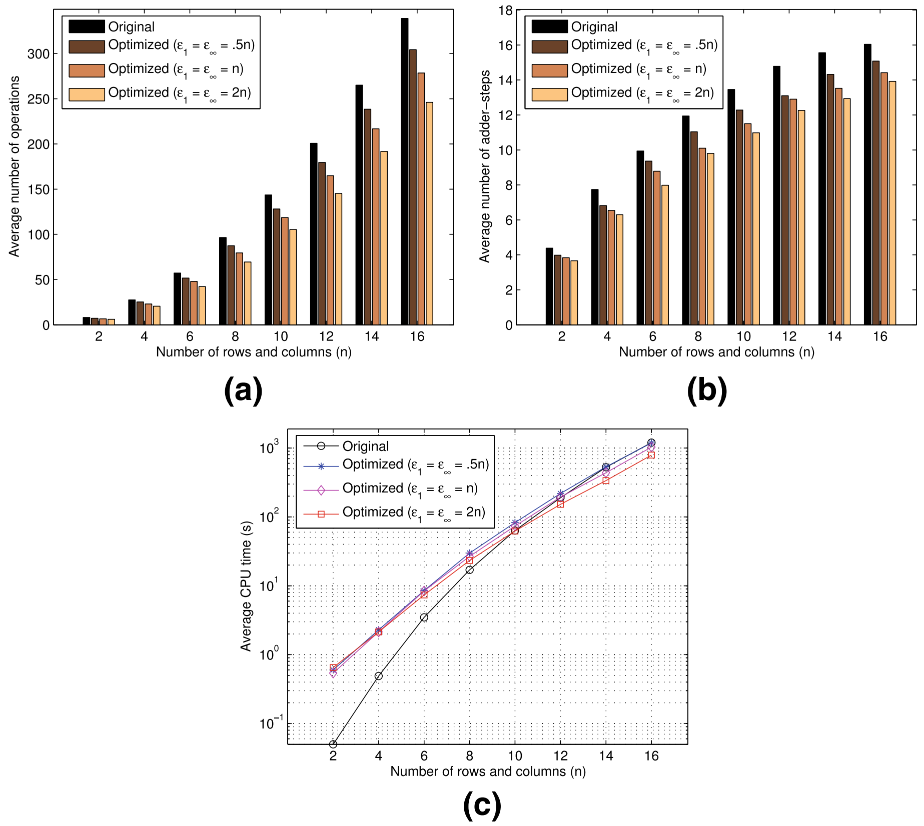

Fig. 2 Average results of HCMVM and AURA on $n \times n$ matrices. **a** Number of operations. **b** Number of adder-steps. **c** CPU time in log scale

16×16 instances, while the average $N(C)$ value of the original matrices is 877.9, this value for the optimized matrices is 758.8, 691.3, and 671.4 when ϵ_1 and ϵ_∞ are .5*n*, *n*, and 2*n*, respectively.

Observe also that for small instances, the execution time is much lower for the original matrices because the ILP problem dominates over the run time of HCMVM. For larger instances, it is the opposite, and since the problem sent to HCMVM is larger than for the optimized matrices, the original matrices actually take slightly longer to run. Moreover, since the number of two-term subexpressions to be considered in HCMVM decreases as the $N(C)$ value decreases, the run time of AURA decreases as the error tolerance increases.

Figure 3 presents the results of HCMVM-DC (denoted as original) and AURA-DC (denoted as optimized) on the same constant matrices with the same error tolerances used for HCMVM and AURA in Fig. 2. In HCVM-DC, the delay constraint was set to MAS_{CMVM}. Note that the CPU time of AURA-DC includes the CPU time of HCMVM-DC to find the shift-adds design of the optimized CMVM operation.

The observations given for the results in Fig. 2 are also valid for the results in Fig. 3. However, the solutions of HCMVM-DC and AURA-DC have adder-steps values significantly smaller than those of the solutions of HCMVM and AURA. Still, this comes with a penalty in the number of operations. Observe that the adder-step values of the original and optimized solutions are very close to each other, which is simply because both HCMVM-DC and AURA-DC target finding a solution with the minimum adder-step of the CMVM operation. It is also observed on 10 × 10 instances that some optimized matrices lead to designs with a smaller number of adder-steps but with a larger number of operations with respect to the designs found based on the original matrix. This is the reason why there is not-so-large reduction in the number of operations on these instances with respect to others.

Table 2 presents the results of AURAobtained using different *r* and *ni* values on 8×8 matrices when ϵ_1 and ϵ_∞ are 16. In this table, nzd denotes the average $N(C')$ value of the optimized matrices, oper and step stand, respectively, for the average number of operations and adder-steps of the CMVM designs, and cpu is the average run time of AURA in seconds.

Observe from Table 2 that as *r* increases, increasing the range of possible constants to be considered for each entry of the matrix, the number of required operations

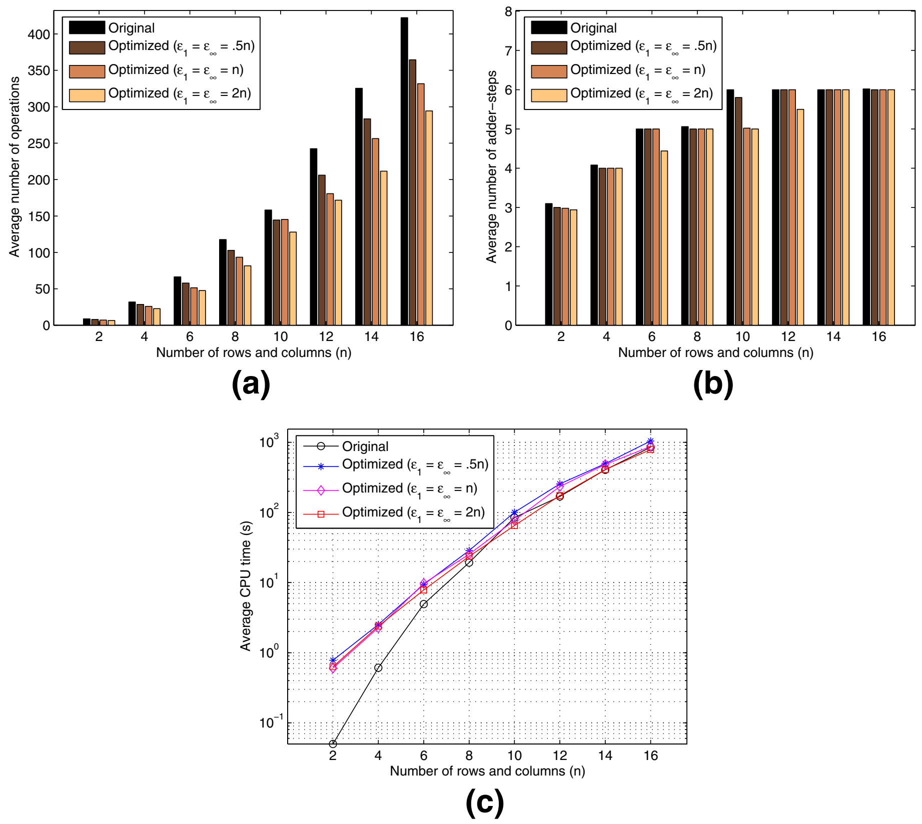

Fig. 3 Average results of HCMVM-DC and AURA-DC on $n \times n$ matrices. **a** Number of operations. **b** Number of adder-steps. **c** CPU time in log scale

decreases, except when r and ni are 16. This is simply because the number of alternative constants increases, as r increases, reducing the $N(C)$ value of the matrix. Note that the case when r and ni are 16 may occur, because there may exist many possible matrices with the minimum $N(C)$ value and all of them cannot be considered in a given number of iterations, ni. Also, as ni increases, the number of operations decreases, because more matrices are considered. Observe that 1.2 operations on average are saved when r is 16 and ni is 64 with respect to the case when r is 4 and ni is 16, which are the actual parameters of AURA for these 8 × 8 matrices. However, this reduction comes with an almost 20× increase in the run time of AURA, which is due to the increase in ni and in the complexity of 0–1 ILP problems generated by AURA as shown in Table 1. Since AURA targets the optimization of the number of operations, increasing the values of r and ni does not have a significant impact on the adder-step value of the CMVM design. Note that similar results were obtained when AURA-DC was used on the same instances with the same parameters.

Table 2 Results of AURA on 8 × 8 matrices

r	ni	nzd	oper	step	cpu
4		156.8	69.5	9.8	23.3
8	16	155.0	69.0	9.9	67.2
16		155.0	69.4	9.7	133.9
4		156.8	69.5	10.0	69.5
8	32	155.0	68.7	9.9	124.3
16		155.0	68.5	9.7	242.7
4		156.8	69.4	10.1	132.0
8	64	155.0	68.6	10.1	235.7
16		155.0	68.3	9.7	469.3

As the second experiment set, we used the 8 × 8 DCT matrix. Note that the $n \times n$ DCT matrix D is an orthonormal matrix, i.e., $DD^T = I$, where I is the identity matrix, and its entries d_{jk} with $0 \leq k \leq n-1$ are determined as follows:

$$d_{jk} = \begin{cases} \sqrt{1/n} & j = 0 \\ (\sqrt{2/n}) \cdot \cos(\pi(k+0.5)j/n) & 1 \leq j \leq n-1 \end{cases}$$

which leads to the following form [22]:

$$D = \begin{bmatrix} g & g & g & g & g & g & g & g \\ a & b & c & d & -d & -c & -b & -a \\ e & f & -f & -e & -e & -f & f & e \\ b & -d & -a & -c & c & a & d & -b \\ g & -g & -g & g & g & -g & -g & g \\ c & -a & d & b & -b & -d & a & -c \\ f & -e & e & -f & -f & e & -e & f \\ d & -c & b & -a & a & -b & c & -d \end{bmatrix}$$

The original DCT matrix was obtained when the floating-point constants are converted to integers using a quantization value of 8. Its coefficients are given in Table 3. The proposed algorithms are applied to the original DCT matrix when ϵ_1 and ϵ_∞ are 4, 8, 16. We note that the proposed methods were modified to respect the same change in the same constants of the DCT matrix to guarantee that the optimized DCT has the same matrix form as the original DCT. Table 3 also presents the values of optimized DCT coefficients. Note that AURA and AURA-DC obtain the same constant matrices under the same error constraint. However, their realizations vary since they have different objectives and use different CMVM methods as shown in Table 5.

These DCTs are applied to the compression of three well-known gray-scale images, namely, *cheese*, *cameraman*, and *lena* whose size is 128 × 128, 256 × 256, and 512 × 512, respectively. The peak signal to noise ratio (PSNR) values of the compressed images are given in Table 4. Observe from Table 4 that as the tolerable error is increased, the PSNR value of the compressed image tends to decrease. However, the PSNR values of the images compressed by the optimized DCT matrix are very close to or the same as the PSNR values of the images compressed by the original DCT matrix. Figure 4 shows the *cameraman* image and its compressed versions using the original and optimized DCT matrices.

Table 5 presents the synthesis results of these DCT matrices on ASIC and FPGA design platforms. The DCTs were described in VHDL when the bitwidth of the input variables was 8. For the ASIC design, the Synopsys Design Compiler was used with the UMCLogic 0.18-μm Generic II library. For the FPGA design, the Xilinx ISE Design Suite 13.1 was used with the Virtex 6 xc6vlx75T-2ff484 target device. The functionality of linear transforms was verified on 10,000 randomly generated input signals in simulation, from which we obtained the switching activity information that was used by the synthesis tool to compute the power dissipation. In Table 5, A, D, and P stand for the area in mm^2, the delay of the critical path in ns, and the total dynamic power dissipation in mW, respectively. Also, *LUTs* and *slices* denote the number of look-up tables and slices, respectively. Note that original shift-adds designs

Table 3 Values of DCT coefficients

Constant	Original	Optimized $\epsilon_1 = \epsilon_\infty = 4$	Optimized $\epsilon_1 = \epsilon_\infty = 8$	Optimized $\epsilon_1 = \epsilon_\infty = 16$
a	126	126	128	128
b	106	106	106	104
c	71	72	72	68
d	25	24	24	24
e	118	118	118	120
f	49	48	48	48
g	91	91	92	92

Table 4 PSNR values of compressed images using original and optimized DCTs

Image	Original	Optimized $\epsilon_1 = \epsilon_\infty = 4$	Optimized $\epsilon_1 = \epsilon_\infty = 8$	Optimized $\epsilon_1 = \epsilon_\infty = 16$
Cheese	25.5	25.5	24.7	24.7
Cameraman	25.12	25.12	23.77	23.77
Lena	30.92	30.92	27	27

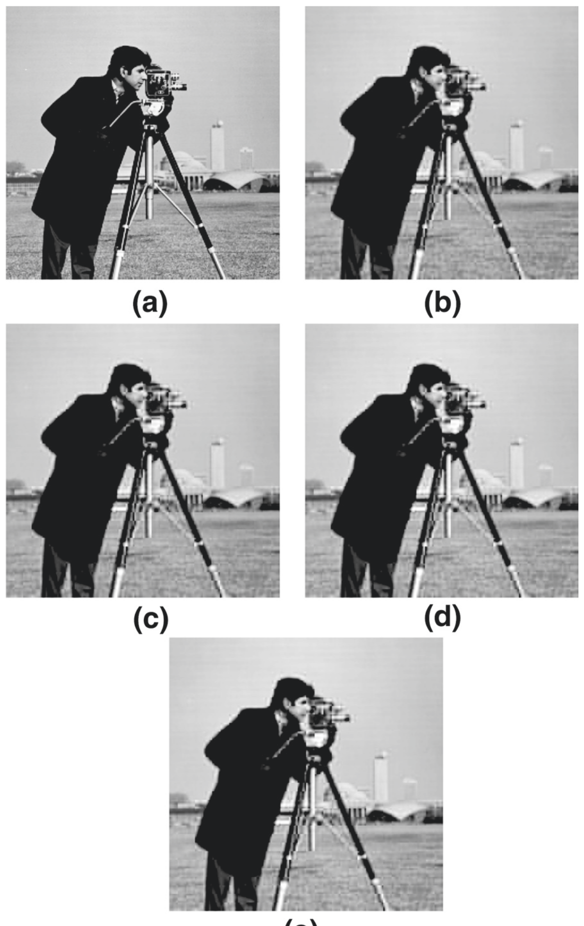

Fig. 4 **a** Original cameraman image. Its compressed versions using: **b** original DCT matrix, **c** optimized DCT matrix when $\epsilon_1 = \epsilon_\infty = 4$, **d** optimized DCT matrix when $\epsilon_1 = \epsilon_\infty = 8$, and **e** optimized DCT matrix when $\epsilon_1 = \epsilon_\infty = 16$

Table 5 Summary of synthesis results of original and optimized DCTs

Objective			Optimization of the number of operations						
DCTs	oper	step	ASIC			FPGA			
			A	D	P	LUTs	slices	D	P
Original	56	8	16.8	5.7	31.0	626	179	14.5	2201
Optimized $\epsilon_1 = \epsilon_\infty = 4$	54	6	16.0	5.4	27.6	595	170	14.1	2149
Optimized $\epsilon_1 = \epsilon_\infty = 8$	48	6	13.8	4.7	21.1	525	151	13.3	2107
Optimized $\epsilon_1 = \epsilon_\infty = 16$	46	6	13.1	4.7	19.5	504	144	13.1	1988
Objective			Optimization under a delay constraint						
DCTs	oper	step	ASIC			FPGA			
			A	D	P	LUTs	slices	D	P
Original	58	5	17.7	5.2	31.9	701	201	13.4	2079
Optimized $\epsilon_1 = \epsilon_\infty = 4$	54	5	16.4	5.1	28.8	612	177	13.3	2058
Optimized $\epsilon_1 = \epsilon_\infty = 8$	48	5	14.1	4.7	22.9	542	156	12.8	2088
Optimized $\epsilon_1 = \epsilon_\infty = 16$	46	5	13.1	4.5	19.5	508	145	12.2	1874

were found by applying HCMVM and HCMVM-DC to the original DCT matrix.

Observe from Table 5 that as the tolerable error increases, the number of required operations decreases, and thus, the CMVM designs require smaller area on the ASIC design and smaller number of LUTs and slices on the FPGA design than the DCT implementations based on original coefficients. We note that the reduction in area between the original and optimized synthesis results obtained while targeting the optimization of the number of operations are found as 5.0 %, 18.1 %, and 22.0 % when ϵ_1 and ϵ_∞ are 4, 8, and 16, respectively. These values are 7.4 %, 20.0 %, and 25.7 % when the optimization of the number of operations under a delay constraint is considered. Also, the reduction in the number of slices between the original and optimized synthesis results obtained while targeting the optimization of the number of operations are found as 5.0 %, 15.6 %, and 19.6 % when ϵ_1 and ϵ_∞ are 4, 8, and 16, respectively. These values are 11.9 %, 22.4 %, and 27.9 % when the optimization of the number of operations under a delay constraint is considered. Moreover, the delay and power dissipation of the design decrease as the tolerable error increases due to the decrease in the hardware complexity of the DCT design. For the ASIC design, the reduction in the delay and power dissipation of the optimized DCT design with respect to those of the original DCT design reaches up to 17.3 % (17.4 %) and 37.4 % (39.4 %) obtained while targeting the optimization of the number of operations (the optimization of the number of operations under a delay constraint), respectively. For the FPGA design, the reduction in the delay and power dissipation of the optimized DCT design with respect to those of the original DCT design reaches up to 15.2 % (9.0 %) and 9.7 % (10.2 %) obtained while targeting the optimization of the number of operations (the optimization of the number of operations under a delay constraint), respectively. However, the optimized DCT design may consume more power than the original DCT design as shown when ϵ_1 and ϵ_∞ are 8 and while targeting the optimization of the number of operations under a delay constraint on FPGA, since the primary objective of the proposed algorithms is not to optimize the power consumption. Observe from the results of DCT designs on FPGA that although finding a CMVM design with a smaller number of adder-step may increase the number of required operations and consequently the area and the number of LUTs and slices, it yields a CMVM design that has less delay and consume less power than that with a larger number of adder-steps. This fact is also valid for the DCT designs on ASIC for the delay parameter.

6 Conclusions

An efficient approximation algorithm was introduced for the multiplierless design of the CMVM operation, targeting the optimization of the number of operations without violating the error constraints. Its modified version, which can find an approximate CMVM design with a smaller number of adder-steps, was also presented. Experimental results showed that the proposed methods can significantly reduce the number of adders/subtractors and adder-steps in the CMVM operation, satisfying the error constraints. It was indicated that the solutions of the proposed algorithms lead to significant reductions in area, delay, and power dissipation of the CMVM designs. It was shown that another advantage of the proposed techniques is to offer a designer alternative CMVM circuits with different complexity and performance values, which can be found by changing the error constraints, so that the designer can choose the one which fits perfectly in an application.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

Author details
[1]Dialog Semiconductor, Istanbul, Turkey. [2]INESC-ID/ Técnico, Universidade de Lisboa, Lisbon, Portugal.

Received: 19 February 2016 Accepted: 14 April 2016
Published online: 20 May 2016

References

1. O Gustafsson, J Coleman, A Dempster, M Macleod, in *Proc. of Asilomar Conference on Signals, Systems and Computers*. Low-Complexity Hybrid Form FIR Filters Using Matrix Multiple Constant Multiplication, (2004), pp. 77–80
2. N Boullis, A Tisserand, Some optimizations of hardware multiplication by constant matrices. IEEE Trans. Comput. **54**(10), 1271–1282 (2005)
3. A Yurdakul, G Dündar, Multiplierless realization of linear DSP transforms by using common two-term expressions. J. VLSI Signal Process. **22**(3), 163–172 (1999)
4. M Macleod, A Dempster, A common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers.Electron. Lett. **40**(11), 651–652 (2004)
5. O Gustafsson, H Ohlsson, L Wanhammar, in *Proc. of Nordic Signal Processing Symposium*. Low-Complexity Constant Coefficient Matrix Multiplication Using a Minimum Spanning Tree, (2004), pp. 141–144
6. A Hosangadi, F Fallah, R Kastner, in *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC)*. Reducing Hardware Complexity of Linear DSP Systems by Iteratively Eliminating Two-Term Common Subexpressions, (2005), pp. 523–528
7. A Hosangadi, F Fallah, Kastner R, in *Proc. of International Workshop on Logic Synthesis (IWLS)*. Simultaneous Optimization of Delay and Number of Operations in Multiplierless Implementation of Linear Systems, (2005)
8. A Arfaee, A Irturk, N Laptev, F Fallah, R Kastner, in *Proc. of Design Automation Conference (DAC)*. Xquasher: A Tool for Efficient Computation of Multiple Linear Expressions, (2009), pp. 254–257
9. L Aksoy, E Costa, P Flores, J Monteiro, Optimization algorithms for the multiplierless realization of linear transforms. ACM Trans. Design Automation Electronic Syst. (TODAES). **17**(Issue 1) (2012). article no. 3
10. L Aksoy, E Costa, P Flores, J Monteiro, in *VLSI-SoC: Advanced Research for Systems on Chip, chap. 5*. Multiplierless Design of Linear DSP Transforms (Springer, 2012), pp. 73–93
11. R Venkatesan, A Agarwal, K Roy, A Raghunathan, in *Proc. of International Conference on Computer-Aided Design (ICCAD)*. MACACO: Modeling and Analysis of Circuits for Approximate Computing, (2011), pp. 667–673
12. M Schaffner, F Gürkaynak, A Smolic, H Kaeslin, L Benini, in *Proc. of Design Automation Conference (DAC)*. An Approximate Computing Technique for Reducing the Complexity of a Direct-Solver for Sparse Linear Systems in Real-Time Video Processing, (2014), pp. 1–6
13. N Merhav, Multiplication-free approximate algorithms for compressed-domain linear operations on images. IEEE Trans. Image Process. **8**(2), 247–254 (1999)
14. BK Natarajan, V Bhaskaran, in *Proc. of International Conference on Image Processing*. A fast approximate algorithm for scaling down digital images in the DCT domain, (1995), pp. 241–243
15. J Ludwig, S Nawab, A Chandrakasan, Low-power digital filtering using approximate processing. IEEE J. Solid-State Circ. **31**(3), 395–400 (1996)
16. S Nawab, A Oppenheim, A Chandrakasan, J Winograd, J Ludwig, Approximate signal processing. J. VLSI Signal Process. Syst. Signal Image Video Technol. **17**, 177–200 (1997)
17. S Venkataramani, A Sabne, V Kozhikkottu, K Roy, A Raghunathan, in *Proc. of Design Automation Conference (DAC)*. SALSA: Systematic Logic Synthesis of Approximate Circuits, (2012), pp. 796–801
18. J Han, M Orshansky, in *Proc. of European Test Symposium*. Approximate Computing: An Emerging Paradigm For Energy-Efficient Design, (2013), pp. 1–6
19. A Alaghi, JP Hayes, Survey of Stochastic Computing. ACM Trans. Embedded Comput Syst. **12**(Issue 2s) (2013). article no. 92
20. K Palem, A Lingamneni, Ten years of building broken chips: the physics and engineering of inexact computing. ACM Transa. Embedded Comput Syst. (TECS). **12**(Issue 2s) (2013). article no. 87
21. P Kulkarni, P Gupta, M Ercegovac, in *Proc. of International Conference on VLSI Design*. Trading Accuracy for Power with an Underdesigned Multiplier Architecture, (2011), pp. 346–351
22. V Gupta, D Mohapatra, A Raghunathan, K Roy, Low-Power Digital Signal Processing Using Approximate Adders. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **32**(1), 124–137 (2013)
23. M Ercegovac, T Lang, *Digital Arithmetic*. (Morgan Kaufmann, 2003)
24. O Gustafsson, Lower Bounds for Constant Multiplication Problems. IEEE Trans. Circ. Syst. II: Analog Digit. Signal Process. **54**(11), 974–978 (2007)
25. L Aksoy, P Flores, J Monteiro, Exact and approximate algorithms for the filter design optimizationproblem. IEEE Trans. Signal Process. **63**(1), 142–154 (2015). doi:10.1109/TSP.2014.2366713
26. R Hegde, N Shanbhag, in *International Symposium on Low Power Electronics and Design*. Energy-Efficient Signal Processing via Algorithmic Noise-Tolerance, (1999), pp. 30–35
27. K Palem, Energy aware computing through probabilistic switching: a study of limits. IEEE Trans. Comput. **54**(9), 1123–1137 (2005). doi:10.1109/TC.2005.145
28. N Zhu, WL Goh, W Zhang, KS Yeo, ZH Kong, Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. IEEE Trans. Very Large Scale Integration (VLSI) Syst. **18**(8), 1225–1229 (2010). doi:10.1109/TVLSI.2009.2020591
29. SH Nawab, AV Oppenheim, AP Chandrakasan, JM Winograd, JT Ludwig, Approximate Signal Processing. J. VLSI Signal Process. Syst. **15**(1/2), 177–200 (1997). doi:10.1023/A:1007986707921
30. L Aksoy, E Costa, P Flores, J Monteiro, in *Proc. of International Conference on Computer-Aided Design (ICCAD)*. Multiple Tunable Constant Multiplications: Algorithms and Applications, (2012), pp. 473–479
31. Solving Constraint Integer Programs website. http://scip.zib.de/. Accessed Feb 2016