Optimization of Constant Matrix Multiplication with Low Power and High Throughput

Martin Kumm, *Member, IEEE,* Martin Hardieck, and Peter Zipf, *Member, IEEE*

Abstract—Constant matrix multiplication (CMM), i. e., the multiplication of a constant matrix with a vector, is a common operation in digital signal processing. It is a generalization of multiple constant multiplication (MCM) where a single variable is multiplied by a constant vector. Like MCM, CMM can be reduced to additions/subtractions and bit shifts. Finding a circuit with minimal number of add/subtract operations is known as the CMM problem. While this leads to a reduction in circuit area it may be less efficient for power consumption or throughput. It is well studied for the MCM problem that a) reducing the adder depth (AD) leads to a reduced power consumption and b) pipeline resources have to be considered during optimization to enhance throughput without wasting area. This paper addresses the optimization of CMM circuits which considers both adder depth and pipelining for the first time. For that, a heuristic is proposed which evaluates the most attractive graph topologies. It is shown that the proposed method requires 16% less adders with min. AD and 40% less pipelined operations. Synthesis results for recent FPGAs show that these reductions also translate to superior results in terms of delay and power consumption compared to the state-of-the-art.

Index Terms—Constant matrix multiplication (CMM), multiple constant multiplication (MCM), pipelining, low power, optimization, FPGA

1 Introduction

The multiplication of a constant matrix with a vector, commonly referred to as constant matrix multiplication (CMM), is a frequently used operation in the domain of digital signal and image processing. It is a fundamental component that appears in linear systems like digital filters and controllers as well as discrete transforms. In most cases, their coefficients can be computed at design time which allows further optimizations to obtain compact multiplier-less CMM circuits. Several previous publications showed the advantages when using multiplier-less CMM circuits. Examples from image and video processing are digital 2-D filters (for image smoothing, edge detection, etc.) [1], the discrete cosine transform (DCT) [2] or color space conversion (from RGB to YCbCr and vice versa) [3]. Other examples from digital signal processing are polyphase filter banks [4] for decimation or interpolation which are building blocks in software defined radio or sampling rate converters [5]. The complex multiplication by constants is also a special case of CMM (for which a 2×2 constant matrix is multiplied by a 2-D vector). Its applications are, e. g., constant rotators as used in the butterflies of fast Fourier transform (FFT) implementations [6], [7] or low-latency implementations of the CORDIC algorithm [8]. All these different applications have in common that their complexity is dominated by the CMM operation. Hence, any improvment in the CMM directly translates to an improvment of these applications.

CMM is a generalization of the multiple constant multiplication (MCM) [9], [10] in which a *single* input variable has to be multiplied by several constants. An MCM operation can be expressed using additions, subtractions and bit shifts, which are organized in a so called adder graph. This adder graph is a directed acyclic graph (DAG) in which each node corresponds to an adder or subtractor and the edge weights correspond to bit shifts. Finding the minimal number of additions and subtractions, i. e., finding an adder graph with minimal number of nodes, is known as the MCM problem. The bit shifts are usually assumed without cost as they are hard-wired. An illustrating example of an MCM circuit is given in Fig. 1(a). The horizontal arrows denote left shifts and each add/subtract operation corresponds to an intermediate factor which is denoted in parenthesis. For example, node $7x$ is computed by left shifting the input x by three bits and subtracting the unshifted input: $2^3x - 2^0x = 7x$. Each node in an adder graph corresponds to an adder or subtractor. This is formally represented as $\mathcal{A}$-operation [10], which is defined as

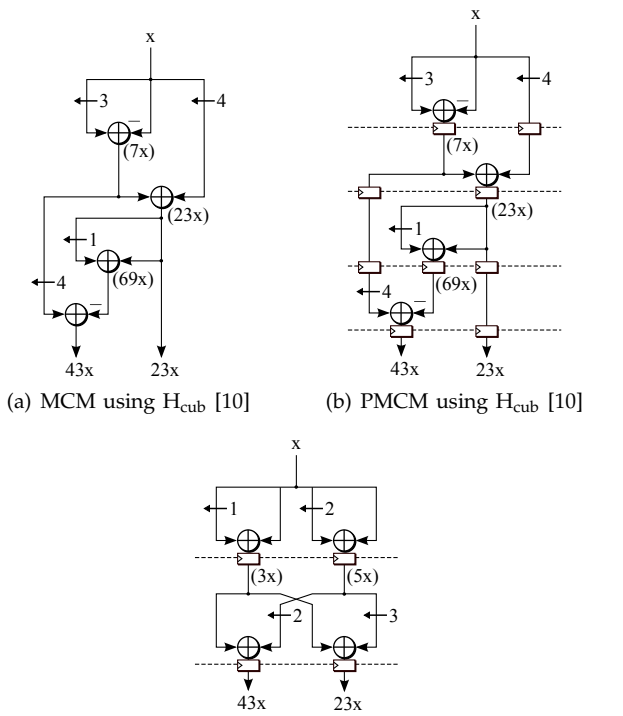
$$\mathcal{A}_q(u, v) = \left|2^{l_1}u + (-1)^{sg}2^{l_2}v\right| 2^{-r} \tag{1}$$

with $q = (l_1, l_2, r, sg)$, where u and v are the input arguments, $l_1, l_2, r \in \mathbb{N}_0$ are shift factors and the sign bit $sg \in \{0, 1\}$ denotes whether an addition or subtraction is performed.

Besides considering only the number of add and subtract operations as costs, two secondary metrics have been established in the last years.

The first metric respects the observation that the adder depth (AD), which is defined as the number of cascaded adders in an adder graph, is directly related to the critical path delay and has an indirect impact on the power consumption [12]. Thus, algorithms were proposed which reduce the AD in MCM [13], limit the global maximum AD in MCM [10] or FIR fiters [14],

M. Kumm, M. Hardieck and P. Zipf are with the Digital Technology Group, University of Kassel, 34121 Kassel, Germany (e-mail: kumm@uni-kassel.de, martin.hardieck@student.uni-kassel.de, zipf@uni-kassel.de).



(a) MCM using H_{cub} [10] (b) PMCM using H_{cub} [10]

(c) PMCM using RPAG [11]

Fig. 1. Example MCM operations for coefficients from the set $\{23, 43\}$

or guarantee a minimal AD for each MCM coefficient [15]–[17] in addition to the objective in minimizing the number of adders.

The second modified metric is related to pipelining. It has been observed that the pipelining of adder graphs may lead to an excessive increase in complexity for additional pipeline registers and some effort is necessary to find the optimal pipeline schedule for a given adder graph [18]. An MCM optimization algorithm in which the pipeline registers are considered in the cost metric – the pipelined MCM (PMCM) problem – was proposed by our group [11], [19]. The algorithm is called reduced pipelined adder graph (RPAG) algorithm and directly produces pipelined adder graphs (PAGs) with less complexity compared to solutions obtained by one of the best MCM heuristics H_{cub} [10] and applying an optimal pipeline schedule method [18] afterwards.

An example is given in Fig. 1(b), which is a PAG obtained by pipelining the MCM circuit of Fig. 1(a) using an as-soon-as-possible (ASAP) scheduling. Nine additional registers are needed from which five registers are used to balance the pipeline. The PAG solution obtained by RPAG is shown in Fig. 1(c). It uses the same number of adders but only four registers. Note that on FPGAs, the registers from Fig. 1(c) come for free as they otherwise remain unused. This is in contrast to the five pipeline balance registers of Fig. 1(b).

On ASICs, pipelined adder graphs are also beneficial for increased performance, but there, carry-save adders (CSAs) may be an alternative for increasing the speed. However, it was shown that there is no 1:1 mapping of CPA-based adder graphs to CSA adder graphs (some CPAs can be replaced by wires while others require two CSAs). Hence, specialized optimization algorithms for CSA-based adder graphs are required [20]. So far, to the best of our knowledge, no CSA-based CMM algorithm exists and, hence, no fast alternative to pipelining.

None of the previous methods for solving the CMM problem considers minimal adder depth solutions or pipelining, although it was shown that these are important metrics for low-power and high throughput implementations for MCM [1], [3], [11], [12], [14]–[18], [21]. Thus, the contribution of this paper is to close this gap by proposing an CMM algorithm that is able to either guarantee minimal AD in combinatorial circuits or optimizes the pipeline resources. The results are superior to previous CMM methods without pipelining as well as to PMCM methods used for pipelined CMM circuits.

2 CMM AND RELATED WORK

The CMM operation of a constant $M \times N$ matrix $\mathbf{C}$ with a vector $\vec{x}$ is represented in the following form:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,N} \\ c_{2,1} & c_{2,2} & \dots & c_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{M,1} & c_{M,2} & \dots & c_{M,N} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \quad (2)$$

Each output is computed by a sum-of-products (SOP) operation from the input vector:

$$y_1 = c_{1,1}x_1 + c_{1,2}x_2 + \ldots + c_{1,N}x_N \quad (3)$$

$$y_2 = c_{2,1}x_1 + c_{2,2}x_2 + \ldots + c_{2,N}x_N \quad (4)$$

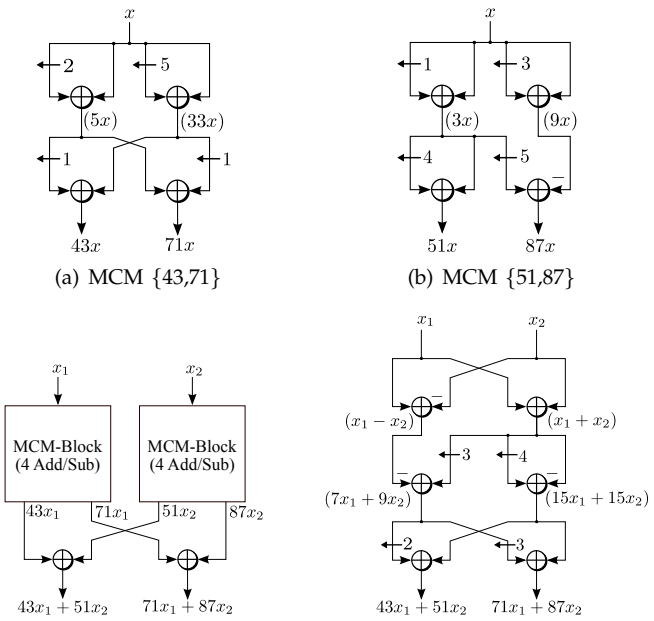
$$\vdots$$

$$y_M = c_{M,1}x_1 + c_{M,2}x_2 + \ldots + c_{M,N}x_N \quad (5)$$

Hence, it can be realized using M independent SOP operations, each computing one row, or by using N MCM operations, each computing the products of a column, followed by an adder tree. SOP and MCM are directly related by the transposition, i. e., transposing the single input N-output adder graph of an MCM operation results in an N-input single output SOP operation which processes the same coefficients [22]. To illustrate the CMM optimization, the matrix

$$\mathbf{C} = \begin{pmatrix} 43 & 51 \\ 71 & 87 \end{pmatrix} \quad (6)$$

is used as running example throughout the paper. The corresponding CMM circuit can be realized by using two MCM circuits from Fig. 2(a) and Fig. 2(b) and two additional adders as shown in Fig. 2(c). It requires 10 adders in total. An optimized CMM circuit uses only six add/subtract operations for the same computation as shown in Fig. 2(d).



(a) MCM {43,71} (b) MCM {51,87}

(c) CMM with MCM blocks (d) Direct CMM

Fig. 2. Example CMM operations

The advantage of optimizing CMM circuits instead of using an MCM optimization multiple times was first shown by Dempster et al. [23]. They extended their Bull Horrocks modified (BHM) MCM optimization algorithm [24] to solve the CMM problem. This inspired many other authors to develop CMM optimization methods. As it is a generalization of the NP-complete single constant multiplication (SCM) problem [25] it is also NP-complete and the algorithms so far are heuristics. Another CMM heuristic based on an MCM heuristic [26] was proposed by Gustafsson et al. [27]. They transferred the CMM problem to a graph representation in which a minimum spanning tree (MST) has to be found. This MST yields a new matrix with reduced complexity which is used for the next iteration until the matrix only contains ones and zeros. A CMM method based on common sub-expression elimination (CSE) was introduced by Macleod and Dempster [28]. They represent the CMM instance by using an $i \times j \times k$ 3D matrix which contains the signed digit (SD) value at bit position k for the constant of row i in column j. Then, two-term subexpressions that occur most often are searched and subsequently eliminated. Another method for optimizing CMM using a genetic algorithm (GA) was proposed by Kinane et al. [29], [30]. Permutations from different SD representations are taken to extract valid SOP subterms. These are then combined and selected by a GA.

3 Proposed CMM Algorithm

The proposed CMM algorithm is described in this section, starting with some preliminary considerations and a problem definition, followed by a detailed description of the optimization steps. The overall optimization algorithm follows a depth-first search (DFS) approach. As the depth search is based on ideas of the previously proposed PMCM algorithm RPAG [11] it is called RPAG-CMM.

3.1 Graph Representation

Each node in a CMM adder graph can be represented by a vector of input weights instead of a scalar weight like for MCM adder graphs [27]. This vector of length N corresponds to the sum of weighted inputs where element i corresponds to the (multiplicative) weight of input i, e. g., the vector $(43, 51)$ corresponds to $43x_1 + 51x_2$. As a consequence, the inputs are represented by their corresponding unit vector. Now, each node corresponds to an extended $\mathcal{A}$-operation which is defined for vectors as

$$\mathcal{A}_q(\vec{u}, \vec{v}) = |2^{l_1}\vec{u} + (-1)^{sg}2^{l_2}\vec{v}\,|\,2^{-r}\ , \tag{7}$$

i. e., all shift operations are performed element-wise. As there are infinite combinations possible for unbounded shifts the $\mathcal{A}$-operation is typically evaluated up to a limit $x_{\max}$. This limit is usually chosen as $x_{\max} = 2^{B+1}$ [10], where B is equal to the maximum bit width of the coefficients. In our case, it is the maximum of all bit widths of the matrix elements.

3.2 Adder Depth and Pipeline Depth

The minimal AD for a vector node in an adder graph can be obtained by evaluating the minimum signed digit (MSD) representations of the vector elements [31]. In the signed digit (SD) number system, the digits are out of the values $\{-1, 0, 1\}$, where digit -1 is usually denoted as $\bar{1}$. It is redundant, so an MSD number is defined to have a minimal number of non-zeros. In a constant multiplication, each non-zero digit of the constant corresponds to a partial product which has to be bit-shifted and added/subtracted to get the product. A minimal AD can then be obtained by using a binary adder tree. Hence, if constant c has $\mathrm{nz}(c)$ non-zero digits, there are $\mathrm{nz}(c) - 1$ adders in

$$\mathrm{AD}_{\min}(c) = \lceil \log_2(\mathrm{nz}(c)) \rceil \tag{8}$$

stages necessary. For a vector $\vec{c}$, the minimal AD is given by the sum over the non-zeros of all elements in the vector [31]:

$$\mathrm{AD}_{\min}(\vec{c}) = \left\lceil \log_2 \left(\sum_{n=1}^{N} \mathrm{nz}(c_n) \right) \right\rceil \tag{9}$$

Now, the total AD for the CMM instance can be obtained by finding the maximum AD of the rows of the coefficient matrix $\mathbf{C}$. If $\mathbf{C}$ is defined as a column vector (size M) of row vectors (each size N), i. e, $\mathbf{C} = (\vec{c}_1 \vec{c}_2 \dots \vec{c}_M)^T$ then, the total AD is [31]:

$$\mathrm{AD}_{\min}(\mathbf{C}) = \max_{\vec{c}_m \in \mathbf{C}} \mathrm{AD}_{\min}(\vec{c}_m)\ . \tag{10}$$

The pipeline depth has to be at least the adder depth if each adder is pipelined. As each extra pipeline stage introduces additional nodes in the PAG, it is unlikely that there exists a graph with higher depth but less cost. Therefore, we define the number of pipeline stages S to be as low as the minimal AD:

$$S := \mathrm{AD}_{\min}(\mathbf{C}) \tag{11}$$

3.3 Problem Formulation

Finding a CMM circuit with minimal complexity can now be seen as finding a minimal adder graph. The most complex part for that is to find the numeric values of the intermediate nodes (i. e., non-output and non-input nodes). The corresponding bit-shifts can be found easily by evaluating (7). Therefore, a set X_s is defined for each adder stage s. Stage X_0 contains the N different unit vectors of length N which correspond to the inputs. To have a unique representation all vectors are normalized which is denoted by $\mathrm{norm}(\vec{c}_m)$. For that, all elements are divided by two until at least one element is odd and the sign of all elements is changed such that the first non-zero element is positive [27]. This normalization can later be reversed by left shifting the result and/or changing the corresponding adder to a subtractor or swapping the inputs of a subtractor.

Now, the initialization of the sets X_s determines if the CMM optimization is done for pipelining or for minimal AD. For a PCMM optimization, only the set of the last stage X_S is initialized to the normalized row vectors of matrix $\mathbf{C}$:

$$X_S = \{\mathrm{norm}(\vec{c}_m)\} \text{ for } m = 1 \dots M \tag{12}$$

For a CMM optimization with minimal AD, set X_s is initialized to the normalized row vectors whose adder depth is equal to s:

$$X_s = \{\mathrm{norm}(\vec{c}_m) \mid \mathrm{AD}_{\min}(\vec{c}_m) = s\} \text{ for } m = 1 \dots M \tag{13}$$

All uninitialized sets remain empty except stage X_0, which contains all unit vectors of length N. With this notation, we can formally define the optimization problem as follows:

Definition 1 (CMM Problem):
Given the initial sets $X_{0,\dots,S}$ as described above, find the vectors with least total costs which have to be inserted in $X_{1,\dots,S-1}$ such that for all $\vec{x} \in X_s$ for $s = 1, \dots, S$, there exists an $\mathcal{A}$-configuration q such that $\vec{x} = \mathcal{A}_q(\vec{x}_1, \vec{x}_2)$ with $\vec{x}_1, \vec{x}_2 \in X_{s-1}$.

As both vectors of our running example have an adder depth of three, the initialization of the input sets would be $X_0 = \{(0,1),(1,0)\}$, $X_1 = \{\}$, $X_2 = \{\}$ and $X_3 = \{(43,51),(71,87)\}$ for both optimization goals. Thus, the minimal cost elements of X_1 and X_2 have to be found by the optimization which is further detailed in the following.

3.4 Depth Search

The overall algorithm performs a depth-first search while each depth search is a modified greedy search. This depth search is based on the previously proposed PMCM algorithm RPAG [11] which was extended to CMM. Each single depth search starts at the output adder stage $s = S$ and searches for elements in stage $s-1$ until all elements in X_s can be computed from X_{s-1}. Then, the next lower adder stage is optimized. The detailed steps are as follows:

1) The search starts at the last stage $s = S$.
2) If $s = 1$ terminate the actual depth search as all pipeline sets are complete.
3) The elements of the actual set are copied into a working set $W = X_s$ and the elements of the preceding stage, called predecessors, are copied into the predecessor set $P = X_{s-1}$.
4) All elements of W which can be directly computed from elements of P are removed. For this purpose the set $\mathcal{A}_*(X)$ is defined, which contains all possible vectors that can be computed from the elements of X with one $\mathcal{A}$-operation:

$$\mathcal{A}_*(X) = \bigcup_{\vec{x}_1, \vec{x}_2 \in X} \{\mathcal{A}_q(\vec{x}_1, \vec{x}_2) \mid q \text{ valid configuration}\} \tag{14}$$

   The working set is now reduced by all elements that can be realized from P:

$$W = W \setminus \mathcal{A}_*(P) \tag{15}$$

5) If W is empty, set $X_{s-1} = P$, decrement s and continue with step 2.
6) Now, valid predecessors are searched and evaluated for all remaining elements in W. The search is started by computing all possible single predecessors by evaluating the three possible topologies of Fig. 3 (a)-(c). If no single predecessor is found, a pair of predecessors is searched by evaluating the topologies of Fig. 3 (d) and (e) The best predecessor(s) found is (are) inserted in P and the algorithm continues with step 4. The predecessor computation and its evaluation is described in sections 3.6-3.8.

Note that the only heuristic part is the selection of predecessors in step 6. If the algorithm terminates without reaching step 6, then the solution is optimal.

3.5 Overall Algorithm

The evaluation and selection of elements in the above depth search can only be done from a local point of view, i. e., elements which are good for the actual stage may be hard to further optimize in earlier stages (or later optimization steps). Hence, several depth searches are performed in the overall algorithm. The first search corresponds to a pure greedy search as described above, i. e., the best element is selected from a local evaluation. In the next search, the first selection in stage 6 is forced

to the 2nd best element while the remaining elements are still selected in a greedy manner. This continues up to a certain limit L such that all of the L best elements in the first selection are evaluated. Now, the element of the first selection which leads to the best overall result is fixed and the same procedure is continued for the following selections in the greedy search until all selections are fixed. This can be interpreted as a greedy search (overall algorithm) on top of another greedy search (depth search).

3.6 Computation of Single Predecessors

The computation of all single predecessors in step 6 of the depth search is done by evaluating the graph topologies in Fig. 3 (a)-(c). Each predecessor $\vec{p}$ must have a lower AD than the actual stage, i. e., $\mathrm{AD}_{\min}(\vec{p}) < s$. Topology (a) occurs if W contains elements with a lower AD than s. These elements can be copied to P and are realized using pure registers (PCMM) or wires (CMM). In topology (b), an element of W is computed from a single predecessor by multiplying it by a number of the form $2^k \pm 1$. Hence, valid predecessors of that topology are all integer vectors obtained by the element-wise division of the working set vectors by $2^k \pm 1$. If P already contains some elements, more elements can be computed from an additional single predecessor using one $\mathcal{A}$-operation as shown in Fig. 3 (c). To obtain the corresponding predecessor $\vec{p}$, an important property of the $\mathcal{A}$-operation is used which says that if there exists a configuration q with $w = \mathcal{A}_q(u, v)$ then there also exists a configuration q' for which $u = \mathcal{A}_{q'}(w, v)$ [10]. This property also holds for w, u and v being vectors. Hence, predecessors of that topology can be computed from $\vec{p} = \mathcal{A}_q(\vec{w}, \vec{p}\,')$ for $\vec{w} \in W$ and $\vec{p}\,' \in P$ with $\mathrm{AD}_{\min}(\vec{p}) < s$. The computations can now be formally represented by the sets:

$$P_a = \{\vec{w} \in W \mid \mathrm{AD}_{\min}(\vec{w}) < s\} \tag{16}$$

$$P_b = \{\vec{w}/(2^k \pm 1) \mid \ \vec{w} \in W,\ k \in \mathbb{N}\} \cap \mathbb{N}^N \setminus \{\vec{w}\} \tag{17}$$

$$P_c = \bigcup_{\substack{\vec{w} \in W \\ \vec{p}\,' \in P}} \{\vec{p} = \mathcal{A}_q(\vec{w}, \vec{p}\,') \mid q \text{ valid},\ \mathrm{AD}_{\min}(\vec{p}) < s\} \tag{18}$$

3.7 Computation of Predecessor Pairs

If none of the working set elements can be computed from a single predecessor, a pair of predecessors is searched. However, the enumeration of all possible pairs is very time consuming as *any* vector $\vec{p}_1$ with lower adder depth has to be evaluated and for each of them there will be typically several valid $\vec{p}_2$ vectors to compute $\vec{w}$. Hence, the search space has to be pruned to make the search feasible.

For that, we first focus on finding the predecessor pairs from which two or more elements from the working set can be computed. These are covered by the graph topology (d) as shown in Fig. 3. The topology can be described as equation system of two equations which can be solved for the predecessor pair:

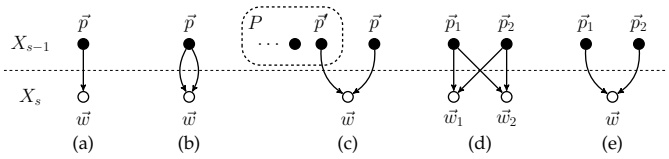


Fig. 3. Predecessor graph topologies for (a)-(c) a single predecessor $\vec{p}$, (d)-(e) a predecessor pair $(\vec{p}_1, \vec{p}_2)$

$$\vec{p}_1 = \left| \frac{\vec{w}_1 2^{r_1+l_{22}} - (-1)^{s_{12}}(-1)^{s_{22}}\vec{w}_2 2^{r_2+l_{21}}}{2^{l_{11}+l_{22}} - (-1)^{s_{22}} 2^{l_{12}+l_{21}}} \right| \tag{19}$$

$$\vec{p}_2 = \left| \frac{\vec{w}_1\, 2^{r_1+l_{12}} - (-1)^{s_{12}}\, \vec{w}_2\, 2^{r_2+l_{11}}}{2^{l_{21}+l_{12}} - (-1)^{s_{22}}\, 2^{l_{22}+l_{11}}} \right| \tag{20}$$

The computation of these predecessor pairs is performed element-wise (in fact only scalar multiplications or element-wise add/subtract operations occur).

Of course, there is no guarantee that any predecessor pair is found by topology (d). However, in case of failure, it is known that two predecessors are necessary to compute one working set element. Therefore, the pairs from combinations of the non-zero digits of the MSD representation of $\vec{w}$ are considered which have a high probability for resource reduction (topology (e)). Although this is still a large search space it turned out to be trackable for practical problems. The MSD representation has also the advantage that a reduced AD can be guaranteed by construction. Take, e. g., the vector $\vec{w} = (23, 42)$ which has an MSD representation of $\vec{w} = (10\bar{1}00\bar{1}, 10\bar{1}0\bar{1}0\bar{1})$. It consists of seven non-zeros which results in a minimal AD of three according to (9). To reduce the AD, each predecessor must have an AD of two or at most 2^2 non-zeros (see Section 3.2). Hence, all permutations of 4 non-zeros in $\vec{p}_1$ and $7 - 4 = 3$ non-zeros in $\vec{p}_2$ are evaluated. To give an example, the pair $\vec{p}_1 = (000000, 10\bar{1}0\bar{1}0\bar{1}) = (0, 43)$ and $\vec{p}_2 = (10\bar{1}00\bar{1}, 0000000) = (23, 0)$ would be a valid pair as well as $\vec{p}_1 = (10\bar{1}000, 10\bar{1}0000) = (24, 48)$ and $\vec{p}_2 = (00000\bar{1}, 0000\bar{1}0\bar{1}) = (-1, -5)$, which have the normalized representation of $\mathrm{norm}(\vec{p}_1) = (3, 6)$ and $\mathrm{norm}(\vec{p}_2) = (1, 5)$.

3.8 Predecessor Evaluation

The cost metric for selecting the best predecessors has to consider the number of predecessors, their complexity, the reduction of the working set as well as the cost of operations. The set of predecessors to be evaluated is denoted by P', which is set to $P' = \{\vec{p}\}$ if a single predecessor is evaluated and is set to $P' = \{\vec{p}_1, \vec{p}_2\}$ if a pair of predecessors is evaluated. The set of elements which can be removed from the working set is denoted as

$$W'(P') = W \cap \mathcal{A}_*(P \cup P')\ . \tag{21}$$

An operation can be either of type adder or of type wire (CMM) / register (PCMM). To distinguish between

these operations, we divide the working set into a set of elements which are realized by registers/wires (elements that are repeated from P' to W') and another set which are realized by adders (elements that have to be computed from P'):

$$W'_R(P') = W'(P') \cap P' \tag{22}$$
$$W'_A(P') = W'(P') \setminus P' \tag{23}$$

A similar division is done with the predecessor set P'. As the information on how the elements are computed is not known in the current stage, it is decided from the adder depth if the predecessor has to be computed by an adder or can potentially be realized by a register (if its adder depth is lower than necessary):

$$P'_R(P') = \{p \in P' \mid \mathrm{AD}_{\min}(p) < s-1\} \tag{24}$$
$$P'_A(P') = \{p \in P' \mid \mathrm{AD}_{\min}(p) = s-1\} \tag{25}$$

Now, a benefit-to-cost ratio is computed which is called the gain and is described by

$$\mathrm{gain}(P') = \frac{\sum_{\vec{w}\in W_R(P')} \mathrm{cost_R}(\vec{w}) + \sum_{\vec{w}\in W_A(P')} \mathrm{cost_A}(\vec{w})}{\sum_{\vec{p}\in P'_R(P')} \mathrm{cost_R}(\vec{p}) + \sum_{\vec{p}\in P'_A(P')} \mathrm{cost_A}(\vec{p})}. \tag{26}$$

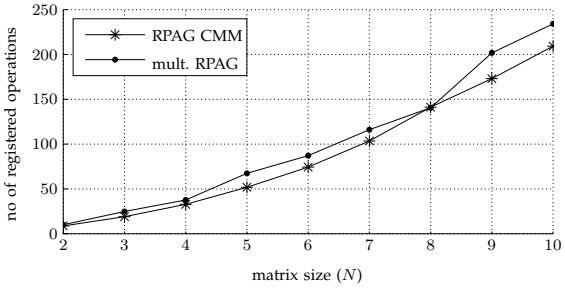
The cost functions $\mathrm{cost_R}()$ and $\mathrm{cost_A}()$ depend on the target architecture and optimization goal ("PCMM" or "CMM with min. AD"). For PCMM the register costs are included in both $\mathrm{cost_R}(\vec{x})$ and $\mathrm{cost_A}(\vec{x})$ while this is not the case for CMM. Here, $\mathrm{cost_R}(\vec{x})$ can be set to zero as wires are free. In the simplest case, the used add/subtract and register operations can be counted by setting the corresponding cost function(s) to one. A more detailed cost approximation is achieved by considering the exact number of full adders, half adders, simple gates and flip flops [32].

4 Results

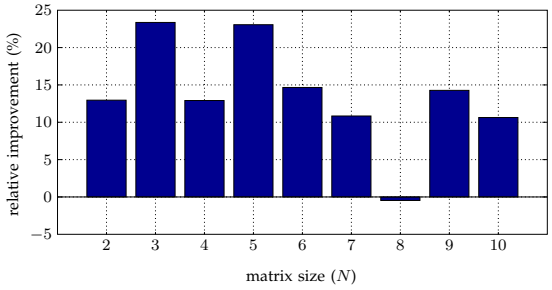
To evaluate the performance of the proposed algorithm, three experiments were made which are described in the following.

4.1 Comparison with multiple PMCM (RPAG)

As there is no CMM algorithm available that guarantees minimal AD or respects pipelining,the proposed method for the PCMM optimization goal is compared against the RPAG algorithm [11] in the first experiment. For that, RPAG is used multiple times to compute a PMCM circuit for each matrix column, denoted as "mult. RPAG", and a pipelined adder tree is used for each output. This was performed for a set of 100 instances of random square matrices $N \times N$ each with $N = 2 \ldots 10$ and 8 bit coefficients. The results in terms of average number of registered operations (over 100 instances) as well as the improvement of the proposed RPAG-CMM algorithm over RPAG are shown in Fig. 4. With *registered operations*, we mean the number of add/subtract operations



(a) Absolute number of registered operations



(b) Improvement of the proposed RPAG-CMM algorithm over multiple use of RPAG

Fig. 4. Benchmark of random 8 bit $N \times N$ matrices

including a pipeline register or pure registers. Significant reductions of pipelined operations can be achieved which are in a range of 10…22% except for 8×8 matrices. Here, a better solution is often found by RPAG. This can be explained by the fact that a pipelined adder tree with 2^k inputs can be realized without register overhead. However, in 49 out of 100 cases, a better solution is obtained by the proposed approach anyway. Note that this experiment is statistically significant but less realistic as most linear transforms are not random. In fact, they may contain several identical coefficients which appear periodically (e. g., in FFT/DCT).

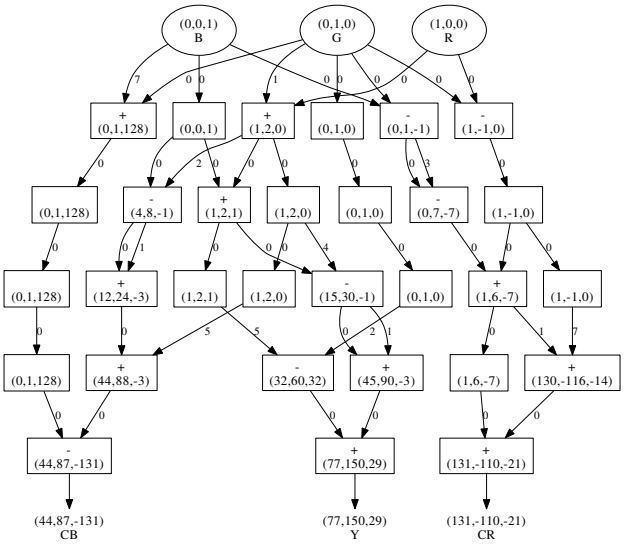
4.2 Comparison with Benchmarks from Literature

To get results for more realistic matrices, we collected matrices from several publications and took the best solutions that have been presented so far as the state-of-the-art. These are analyzed in the number of add/subtract operations, the adder depth and the number of pipelined operations after careful manual pipelining. For the proposed method, the number of adders is obtained for optimization goal "CMM with min. AD", the number of registered operations is obtained for optimization goal "PCMM". The results are listed in Table 1, unavailable data is marked with a '–'. In addition to the CMM results, the arithmetic complexity in terms of multipliers and adders is given. The number of multipliers of an $M \times N$ matrix is MN in general but we excluded all trivial multiplications (zero or power-of-two values), duplicate multiplications as well as addition with zeros from the resources in Table 1.

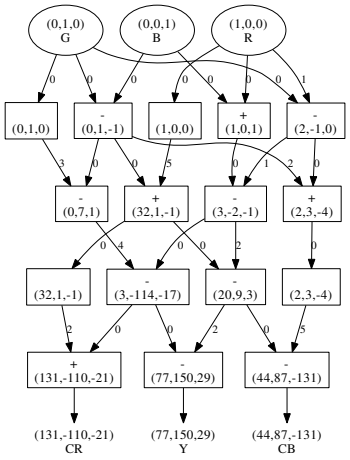
Matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ are color space conversion matrices for RGB to YCbCr and vice versa, taken from the

TABLE 1
Comparison of different CMM methods for benchmark matrices from the literature (best marked bold)

Matrix	Size	complexity		Method	Adders	AD_{max}	Reg. Ops
		#Mult.	#Add.				
$\mathbf{C}_1$	3×3	9	6	[3]	17	5	30
				[23]	18	9	–
				[33]	21	**4**	–
				[27]	18	6	–
				[28]	18	6	–
				[34]	18	6	–
				mult. RPAG	20	**4**	26
				prop.	**12**	**4**	**16**
$\mathbf{C}_2$	3×3	4	4	[3], [27]	**11**	5	22
				[23]	14	5	–
				[28]	12	5	–
				mult. RPAG	12	**4**	19
				prop.	12	**4**	**16**
$\mathbf{C}_3$	4×4	8	12	[27]	**14**	7	35
				mult. RPAG	21	**4**	30
				prop.	**14**	**4**	**19**
$\mathbf{C}_4$	2×2	4	2	[23]	**6**	4	12
				mult. RPAG	7	**3**	10
				prop.	**6**	**3**	**7**
$\mathbf{C}_5$	2×2	4	2	[35]	**5**	3	8
				mult. RPAG	**5**	**2**	**5**
				prop.	**5**	**2**	**5**
$\mathbf{C}_6$	4×4	0	12	[36]	**8**	**2**	**8**
				mult. RPAG	12	**2**	12
				prop.	**8**	**2**	**8**
$\mathbf{C}_7$	8×8	0	56	[37]	**24**	–	–
				mult. RPAG	61	**4**	77
				prop.	**24**	**4**	**24**
$\mathbf{C}_8$	11×16	0	61	[37]	43	–	–
				mult. RPAG	56	**3**	56
				prop.	**31**	**3**	**47**



(a) Solution from [3]



(b) Solution from proposed RPAG-CMM algorithm

Fig. 5. Adder graphs for RGB→YCbCr converters ($\mathbf{C}_1$)

corresponding ITU recommendation [38]. The previous results are taken from Gustafsson and Holm [3] and are also tabulated for the sake of comparison. The detailed solutions for matrix $\mathbf{C}_1$ are shown as pipelined adder graph in Fig. 5. Each node in the graph corresponds to an adder ('+') or subtractor ('−'), both including pipeline register or a pure register (without '+' or '−'). The edge weights correspond to left shifts. The minimal AD solution for this example is identical to the PCMM solution and can thus be obtained from Fig. 5(b) by simply replacing the registers by wires. Matrices $\mathbf{C}_3$-$\mathbf{C}_5$ are numeric examples taken from Figure 4 of [27], Figure 2 of [23] and Figure 1 of [35], respectively. Matrices $\mathbf{C}_6$-$\mathbf{C}_8$ are multiplication-free linear transforms as they contain only power-of-two elements of the form $\pm 2^k$ with $k \in \mathbb{N}_0$. Matrix $\mathbf{C}_6$ is used in H.264/AVC video coding and its CMM solution is taken from Figure 1 of [36]. Matrices $\mathbf{C}_7$ and $\mathbf{C}_8$ are used in an 8×8 Walsh-Hadamard transform (also known as the Hadamard transform or Walsh transform) and a (16,11) Reed-Muller error correcting code, respectively. Their CMM solutions are taken from Figures 5 and 6 of [37]. Note that the Walsh-Hadamard matrix provided in the paper (Fig. 5 in [37]) contains errors. The provided solutions were obtained with the correct matrix (as recursively defined as in [37]).

It can be observed that the proposed CMM optimization typically finds a lower or equal number of adders compared to the numerous previous methods although it provides minimal AD which typically increases the number of adders significantly. An average reduction of 12.5% of adders could be achieved compared to the best known CMM solution while providing a significantly lower AD (18.75% less on average). The number of registered operations could be reduced by 38.5% on average, compared to the best known solution. Interestingly, the result for the Walsh-Hadamard transform matrix $\mathbf{C}_7$ requires exactly the same number of additions (24) and registered operations (24) as the fast Walsh-Hadamard transform [39]. Hence, without knowing the iterative construction of the matrix – which was exploited to obtain the fast Walsh-Hadamard transform [39] – the proposed algorithm was able to find a solution with

identical complexity. In comparison with the arithmetic complexity, one can observe that many costly multiplications can be avoided by only a few additional adders. Even for the multiplier-less matrices $\mathbf{C}_6$-$\mathbf{C}_8$, the number of adders could be significantly reduced by sharing intermediate results.

4.3 Synthesis Results

The designs of the last section for which reference designs exist were synthesized and analyzed. For that, a VHDL code generator was implemented for the CMM and PCMM circuits as well as a 'straight-forward' VHDL description was implemented using '$*$' and '$+$' operators which is used as a baseline. The VHDL code generator was realized with the help of the FloPoCo core generator framework [40] which drastically simplified the generation of pipelines and testbenches. All CMM results were produced by using the same code generator and synthesis settings with an input word size of 12 bit. The synthesis results were performed for a Xilinx Virtex 6 FPGA (XC6VLX75T-FF784-2). All results are obtained after place&route using Xilinx ISE 13.4. All primary inputs and outputs are registered to enable a fair speed comparison. The power consumption was estimated using Xilinx XPower with 1000 random input values. The obtained dynamic power consists of clock, logic and signal power.

The results are shown in Table 2. They show that the proposed method for combinatorial CMM with minimal AD leads to a similar resource utilization (slices, LUTs) compared to the state-of-the-art (rows 'reference') but improves the average performance ($f_{\max}$) and power consumption by 15.4% and 21.0%, respectively (matrices $\mathbf{C}_7$ and $\mathbf{C}_8$ had to be skipped in this comparison as their solution was not provided in [37]).

The throughput can be further increased by using pipelining. This leads to speedups from 59% up to 312%. The price paid for these speedups is a 57% slice increase (compared to combinatorial) for the reference designs whereas it is only 32% for the proposed approach. While the performance is slightly reduced, a slice over $f_{\max}$ improvement of 10% is still achieved. Note that for the resource comparison of pipelined circuits, the slice count represents the most realistic metric as when comparing LUT and FF counts, the desirable case where LUTs and FFs share the same slice resource do not become visible.

A very interesting observation is that, although more resources are used for pipelining, the average dynamic power is reduced by 27% and 34% compared to the unpipelined reference, respectively. This result can be explained by the fact that the pipeline registers "isolate" power producing glitches from one adder stage to the next, similar to the operand isolation technique introduced by Correale [41]. Hence, pipelining is a method to improve both performance axes "performance" and "energy".

The comparison with the baseline shows that our method typically outperforms the synthesis tool in all metrics. The only exception is $\mathbf{C}_2$, where the slice count compared to our solution was reduced by 32 and 19 for combinatorial and pipelined, respectively, at the cost of three additional DSP blocks. Comparing slice and DSP blocks is difficult as their silicon area is unknown. However, die photo inspections yield an embedded multiplier to slice ratio of 1:18 for a Virtex II [42]. Although not directly comparable as the Virtex II contains 18×18 bit multipliers and slices with two 4-input LUTs while the Virtex 6 DSP contains a larger multiplier and pre/post-adders and the slice contains four 6-input LUTs, it shows the trend that a DSP is much more costly than a slice.

5 Conclusion

A novel method for optimizing constant matrix multiplications was proposed. Besides the number of add/subtract operations, either the adder depth of each output can be constrained to be minimal or the number of pipeline registers can be reduced. Significant improvements in resource usage (17% less slices for PCMM), throughput (15% higher clock for CMM min. AD) and power consumption (9% less power for PCMM and 21% for CMM min. AD) could be achieved compared to the state-of-the-art. The proposed algorithm was made available as open-source within the optimization suite for constant multiplication problems PAGsuite [43] (tool `rpag` with option `-cmm`). This provides a reference for further research as well as full reproducibility of the results provided in this paper.

References

[1] M. Kumm, D. Fanghänel, K. Möller, P. Zipf, and U. Meyer-Baese, "FIR Filter Optimization for Video Processing on FPGAs," *Springer EURASIP Journal on Advances in Signal Processing*, pp. 1–18, 2013.

[2] L. Aksoy, P. Flores, and J. Monteiro, "A Novel Method for the Approximation of Multiplierless Constant Matrix Vector Multiplication," *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 98–105, 2015.

[3] K. Holm and O. Gustafsson, "Low-Complexity and Low-Power Color Space Conversion for Digital Video," in *Norchip Conference*. IEEE, 2006, pp. 179–182.

[4] O. Gustafsson, K. Johansson, H. Johansson, and L. Wanhammar, "Implementation of Polyphase Decomposed FIR Filters for Interpolation and Decimation Using Multiple Constant Multiplication Techniques," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2006, pp. 924–927.

[5] O. Gustafsson and H. Johansson, "Efficient Implementation of FIR Filter Based Rational Sampling Rate Converters Using Constant Matrix Multiplication," in *Asilomar Conference on Signals, Systems and Computers (ACSSC)*, 2006, pp. 888–891.

[6] S. Ghissoni, E. Costa, C. Lazzari, J. Monteiro, L. Aksoy, and R. Reis, "Radix-2 Decimation in Time (DIT) FFT implementation based on a Matrix-Multiple Constant multiplication approach," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2010, pp. 859–862.

[7] M. Garrido, F. Qureshi, and O. Gustafsson, "Low-Complexity Multiplierless Constant Rotators Based on Combined Coefficient Selection and Shift-and-Add Implementation (CCSSI)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–11, Jan. 2014.

[8] M. Garrido, P. Källström, M. Kumm, and O. Gustafsson, "CORDIC II: A New Improved CORDIC Algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 2, pp. 186–190, 2016.

TABLE 2
Synthesis results for the straight-forward VHDL implementation (baseline), the best reference designs of Table 1 (ref) and the proposed method (prop.) for CMM min. AD and pipelined CMM (best results marked bold)

		Combinatorial (min AD)					Pipelined						
Matrix	Method	Slices	LUTs	DSPs	f_{max} [MHz]	Dyn. Pow. [mW]	Slices	LUTs	FFs	DSPs	f_{max} [MHz]	Slices/f_{max} [$\frac{1}{MHz}$]	Dyn. Pow. [mW]
	baseline	106	379	3	192.1	31.0	158	415	337	3	245.2	0.644	26.2
$\mathbf{C}_1$	ref [3]	74	230	0	217.3	27.2	122	306	445	0	**672.5**	0.181	17.4
	prop.	**60**	**189**	0	**256.5**	**18.1**	**80**	**222**	**286**	0	619.2	**0.129**	**14.0**
	baseline	**32**	**109**	3	214.7	**14.2**	**76**	**119**	**242**	3	255.8	0.297	**14.1**
$\mathbf{C}_2$	ref [3]	61	186	0	**218.4**	20.1	92	202	352	0	**702.3**	**0.131**	14.5
	prop.	64	190	0	215.5	17.1	95	224	315	0	592.8	0.160	15.2
	baseline	121	397	0	204.9	37.1	144	427	**333**	0	404.5	0.356	24.5
$\mathbf{C}_3$	ref [27]	**67**	**171**	0	167.7	28.6	122	314	402	0	**690.6**	0.177	18.8
	prop.	76	228	0	**244.6**	**21.0**	**106**	**257**	334	0	648.9	**0.163**	**16.1**
	baseline	46	177	0	204.9	13.1	47	176	**97**	0	331.2	0.142	11.4
$\mathbf{C}_4$	ref [23]	**25**	**61**	0	228.2	11.4	46	113	156	0	626.2	0.073	8.7
	prop.	27	88	0	**306.2**	**8.9**	**33**	**92**	121	0	**651.9**	**0.051**	**7.7**
	baseline	36	136	0	333.3	8.8	32	113	**77**	0	462.1	0.069	8.2
$\mathbf{C}_5$	ref [35]	**26**	**71**	0	367.0	7.6	41	74	140	0	**701.3**	0.058	8.0
	prop.	27	72	0	**494.3**	**7.2**	**30**	**71**	99	0	651.9	**0.046**	**7.2**
	baseline	48	153	0	278.2	16.2	82	202	250	0	636.5	0.129	16.0
$\mathbf{C}_6$	ref [36]	46	115	0	**443.3**	**11.4**	47	**108**	**155**	0	**718.9**	**0.065**	**10.4**
	prop.	**42**	**111**	0	424.1	11.7	**46**	**108**	**155**	0	704.2	**0.065**	**10.4**
	baseline	164	561	0	217.2	46.8	306	854	934	0	**610.9**	0.501	32.4
$\mathbf{C}_7$	prop.	**99**	**284**	0	**251.8**	**27.1**	**100**	**294**	**365**	0	484.5	**0.206**	**20.0**
	baseline	120	414	0	**223.2**	41.3	387	839	1246	0	**578.7**	0.669	42.7
$\mathbf{C}_8$	prop.	**90**	**157**	0	146.1	**32.3**	**237**	**532**	**846**	0	513.6	**0.461**	**38.9**
	baseline	64.8	225.2	1.0	238.0	20.1	89.8	242.0	222.7	1.0	389.2	0.3	16.7
Avg. ($\mathbf{C}_1$-$\mathbf{C}_6$)	ref	49.8	**139.0**	0.0	273.6	17.7	78.3	186.2	275.0	0.0	**685.3**	0.114	13.0
	prop.	**49.3**	146.3	0.0	**323.5**	**14.0**	**65.0**	**162.3**	**218.3**	0.0	644.8	**0.102**	**11.8**
Impr. ($\mathbf{C}_1$-$\mathbf{C}_6$)	baseline	23.9%	35.0%	100%	73.6%	30.3%	27.6%	32.9%	1.9%	100%	60.4%	62.5%	29.8%
	ref	1.00%	−5.3%	0%	15.4%	21.0%	17.0%	12.8%	20.6%	0%	−6.3%	10.4%	9.2%

[9] D. R. Bull and D. H. Horrocks, "Primitive Operator Digital Filters," *IEE Proceedings of Circuits, Devices and Systems*, vol. 138, no. 3, pp. 401–412, Jun. 1991.

[10] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, pp. 1–38, 2007.

[11] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined Adder Graph Optimization for High Speed Multiple Constant Multiplication," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2012, pp. 49–52.

[12] S. Demirsoy, A. Dempster, and I. Kale, "Transition Analysis on FPGA for Multiplier-Block Based FIR Filter Structures," *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, pp. 862–865 vol.2, 2000.

[13] A. G. Dempster, S. S. Demirsoy, and I. Kale, "Designing Multiplier Blocks with Low Logic Depth," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2002, pp. V–773–V–776.

[14] D. Shi and Y. J. Yu, "Design of Linear Phase FIR Filters With High Probability of Achieving Minimum Number of Adders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 1, pp. 126–136, 2011.

[15] K. Johansson, "Low Power and Low Complexity Shift-and-Add Based Computations," Ph.D. dissertation, Linköping Studies in Science and Technology, 2008.

[16] M. Faust and C.-H. Chang, "Minimal Logic Depth Adder Tree Optimization for Multiple Constant Multiplication," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 457–460, 2010.

[17] K. Johansson, O. Gustafsson, and L. S. DeBrunner, "Minimum Adder Depth Multiple Constant Multiplication Algorithm for Low Power FIR Filters," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2011, pp. 1439–1442.

[18] M. Kumm and P. Zipf, "High Speed Low Complexity FPGA-Based FIR Filters Using Pipelined Adder Graphs," in *IEEE International Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–4.

[19] M. Kumm, "Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays," Ph.D. dissertation, Springer Wiesbaden, Wiesbaden, Oct. 2015.

[20] O. Gustafsson and L. Wanhammar, "Low-Complexity and High-Speed Constant Multiplications for Digital Filters Using Carry-Save Arithmetic," in *Digital Filters*. InTech, Apr. 2011.

[21] U. Meyer-Baese, G. Botella, D. Romero, and M. Kumm, "Optimization of High Speed Pipelining in FPGA-Based FIR Filter Design Using Genetic Algorithm," in *Proceedings of SPIE, the International Society for Optical Engineering*, 2012, pp. 1–12.

[22] O. Gustafsson and A. Dempster, "On the Use of Multiple Constant Multiplication in Polyphase FIR Filters and Filter Banks," in *Nordic Signal Processing Symposium (NORSIG)*, 2004, pp. 53–56.

[23] A. Dempster, O. Gustafsson, and J. O. Coleman, "Towards an Algorithm for Matrix Multiplier Blocks," in *European Conference on Circuit Theory and Design (ECCTD)*, 2003, pp. 1–4.

[24] A. Dempster and M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 9, pp. 569–577, 1995.

[25] 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, Oct. 1984.

[26] O. Gustafsson and L. Wanhammar, "A Novel Approach to Multiple Constant Multiplication Using Minimum Spanning Trees," *IEEE Midwest Symposium on Circuits and Systems (MWSCAS)*, vol. 3, 2002.

[27] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Low-

Complexity Constant Coefficient Matrix Multiplication Using a Minimum Spanning Tree Approach," in *Proceedings of the 6th Nordic Signal Processing Symposium (NORSIG)*, 2004, pp. 141–144.
[28] M. D. Macleod and A. Dempster, "Common Subexpression Elimination Algorithm for Low-Cost Multiplierless Implementation of Matrix Multipliers," *Electronics Letters*, vol. 40, no. 11, pp. 651–652, 2004.
[29] A. Kinane, V. Muresan, and N. O'Connor, "Towards an Optimised VLSI Design Algorithm for the Constant Matrix Multiplication Problem," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2006, pp. 5111–5114.
[30] ——, "Optimisation of Constant Matrix Multiplication Operation Hardware Using a Genetic Algorithm," in *Lecture Notes in Computer Science*. Springer, 2006, pp. 296–307.
[31] O. Gustafsson, "Lower Bounds for Constant Multiplication Problems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 11, pp. 974–978, Nov. 2007.
[32] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Optimization of Area and Delay at Gate-Level in Multiple Constant Multiplications," in *Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. IEEE, 2010, pp. 3–10.
[33] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A New Algorithm for Elimination of Common Subexpressions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 1, pp. 58–68, 1999.
[34] N. Boullis and A. Tisserand, "Some Optimizations of Hardware Multiplication by Constant Matrices," in *IEEE Transactions on Computers*, Oct. 2005, pp. 1271–1282.
[35] S. Ghissoni, E. Costa, J. Monteiro, and R. Reis, "Combination of Constant Matrix Multiplication and Gate-Level Approaches for Area and Power Efficient Hybrid Radix-2 DIT FFT Realization," in *IEEE International Conference on Electronics, Circuits and Systems, (ICECS)*, 2011, pp. 567–570.
[36] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 598–603, 2003.
[37] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 2, pp. 151–165, 1996.
[38] International Telecomunication Union (ITU), "Encoding Parameters of Digital Television for Studios," *Recommendation ITU-R BT601-4*, pp. 1–13, 1994.
[39] J. L. Shanks, "Computation of the Fast Walsh-Fourier Transform," *IEEE Transactions on Computers*, vol. C-18, no. 5, pp. 457–459, May 1969.
[40] F. de Dinechin and B. Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2012.
[41] A. Correale Jr, "Overview of the Power Minimization Techniques Employed in the IBM PowerPC 4xx Embedded Controllers," in *International Symposium on Low Power Design ISLPED*, Apr. 1995.
[42] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, "Architectural Modifications to Enhance the Floating-Point Performance of FPGAs," *IEEE Transactions on Very Large Scale Integration Systems (VLSI)*, vol. 16, no. 2, pp. 177–187, 2008.
[43] M. Kumm, K. Möller, H. Martin, and P. Zipf. (2016) PAGsuite project website. [Online]. Available: http://www.uni-kassel.de/go/pagsuite

Martin Kumm (M'11) received the Dipl.-Ing. degree in electrical engineering from the University of Applied Sciences, Fulda, and the Technical University of Darmstadt, Germany, in 2003 and 2007, respectively. From 2003 to 2009, he was with GSI Darmstadt, working on digital RF control systems for particle accelerators. In 2015 he received his Ph.D. (Dr.-Ing.) degree from the University of Kassel, Germany. His current research interests in the Digital Technology Group of the University of Kassel are arithmetic circuits and their optimization in the context of reconfigurable systems.

Martin Hardieck Martin Hardieck received the Dipl.-Ing. degree in electrical engineering from the University of Kassel, Germany, in 2014, and the M.Sc. degree in electrical engineering from the University of Kassel, Germany, in 2016, where he is currently pursuing the Ph.D. degree with the Digital Technology Group. His current research interests include digital signal processing, model-based design and artificial neural network implementations all in the context of field-programmable gate arrays.

Peter Zipf (M'05) received the B.A. degree in computer science from the University of Kaiserslautern, Germany, in 1994 and the Ph.D. (Dr.-Ing.) degree from the University of Siegen, Germany, in 2002. He was a Postdoctoral Researcher with the Department of Electrical Engineering and Information Technology, Darmstadt University of Technology (Technische Universität Darmstadt), Germany, until 2009. He is currently the Chair for Digital Technology with the University of Kassel, Germany. His research interests include reconfigurable computing, embedded system and system-on-chip design, as well as design methodologies and CAD algorithms for circuit optimization and reconfigurable systems.