Multiplierless Implementation of 2-D FIR Filters

Arda Yurdakul

Boğaziçi University, Computer Engineering Department, Bebek, 34342, İstanbul, Turkey

Abstract

In this paper, the multiplierless design of two-dimensional FIR filters is studied. The adders that are used in place of multiplications are reduced by using the 2-D CSE (*C*ommon*S*ubexpression*E*limination) method which is developed in this paper. This method decomposes the original filter into smaller filters and the output of each filter is scaled with an appropriate coefficient to form the original filter output. It is also shown in this paper that each step of this procedure is an NP-complete problem. Hence, 0-1 integer programming model for each step is formed. Since the solution of these models are time and memory consuming, heuristic algorithms have also been developed. The heuristic 2-D CSE method proves to be better than traditional implementations of 2-D filters in terms of adder count. Though power and performance analysis of the produced filters have not been made, it is estimated that power consumption of these filters will be low due to low interconnect density. Also, concurrent processing of data with small filters increase the overall filtering performance filter.

Key words: common subexpression elimination, integer programming, multiplierless, 2-D FIR filter architecture.

1 Introduction

Two-dimensional FIR filters are vastly used in image and video processing systems [1]. However, by definition, 2-D FIR filtering requires a huge amount of computations and memory. Hence, the computational complexity must be reduced so as to meet real-time requirements of today's commercial applications with minimum implementation cost. The most common solution is the replacement of each constant multiplication with a network of adders and shifters [4]-[7]. This is called the *multiplierless* implementation. A 2-D FIR filter can also be implemented as a delayed-sum of 1-D multiplierless FIR filters as shown in 1(a). Here, sharing the common terms between the coefficients in each 1-D filter reduces the number of computations. A number of algorithms have been developed for generating cost-effective multiplierless 1-D FIR filters [2]-[3].

In the literature, there has not been a study how an efficient multiplierless implementation of a 2-D FIR filter whose coefficients are obtained by quantizing the infinite-precision coefficients. Also there has not been a study how common sub-expressions can be exploited in 2-D FIR filters. This paper is the first study that addresses the efficient multiplierless implementation of 2-D filters with quantized coefficients. Here, both a method and an architecture are presented to solve this problem. In the architecture, the filter is decomposed into *binary* filters, H_i, whose outputs are scaled and summed up to form the original filter output (1(b)). A binary filter consists of only one-bit signed or unsigned coefficients. This means that the adders in these filters are the smallest-width adders because there does not exist any shifting operation. The method presented in this study is developed to realize the architecture in 1(b) with minimum number of adders by exploiting the common sub-expressions between binary filters. In the following section, the problem will be stated. It will be shown that the problem consists of three sub-problems that are NP-complete. The first problem is the decomposition of the original filter into binary filters. Note that the number of binary filters has a direct impact in the number of adders. The second problem is the extraction of the common patterns in the binary filters so as to reduce the adder cost by sharing these common terms. The final problem is the extraction of common partial sums, because the outputs of binary filters obtained in the second problem must be added up to form the binary filters of the first problem and during this process, some additions might be common and can be shared. In Section 3, a heuristic algorithm is presented to solve the problem. The experimental results (Section 4) reinforce the initial claim that this algorithm produces more efficient architectures than traditional ones. Though power and performance analysis of the produced filters have not been made, it is estimated that power consumption of these filters will be low due to low interconnect density. Also, concurrent processing of data with small filters increase the overall filtering performance filter.

2 Theory

Throughout this section, it will be assumed that H is a filter whose coefficients are real numbers in $[-1, 1]$.

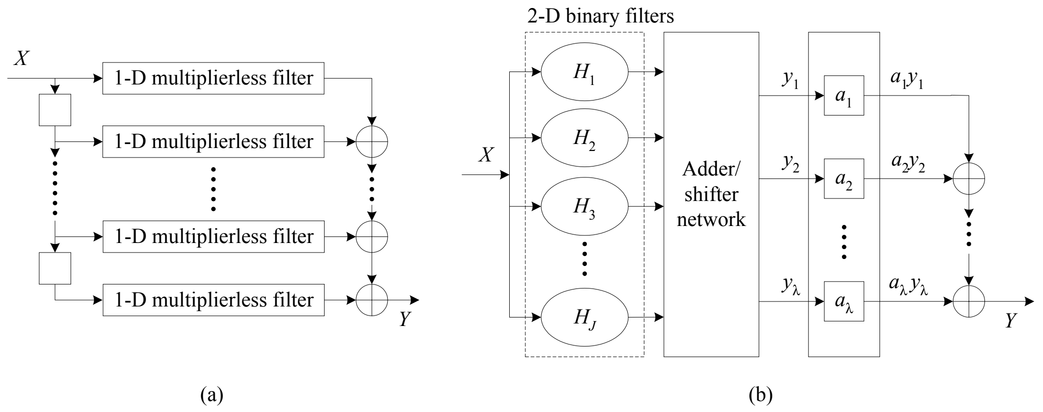


Fig. 1. Implementation of a multiplierless 2-D filter, (a)with multiplierless 1-D FIR filters, (b)with binary 2-D filters.

2.1 Decomposition into Sub-filters

In multiplierless realizations, a constant multiplicand c can be written, without loss of generality, as

$$c = \sum_{b=0}^{L-1} c_b 2^{-b}, \tag{1}$$

Obviously, if two's complement is used in the representation of c_b, then $c_b \in B$. Signed representations allow also the usage of -1 in the representation of c_b. Here, L can be regarded as the quantization wordlength. Using this equation, a nonseparable two-dimensional filter, H can be decomposed into L binary filters in a similar manner:

$$H = \sum_{b=0}^{L-1} H_b 2^{-b}, \tag{2}$$

where each sub-filter, H_b, consists of only $-1, 0$ and 1. Recall that the number of sub-filters directly affects the total number of additions in the system. Hence, Eq 2 hints that the nonseparable two-dimensional filter H can also be written as a linear combination of binary filters:

$$H = \sum_{l=1}^{\lambda} H_l a_l. \tag{3}$$

where a_l is an L-bit fractional scalar.

Theorem 1 *"Decomposition of a two-dimensional filter into scaled-sum of binary filters" problem is NP-complete.*

Proof: Let $H_{int} = \text{round}\left(H2^L\right)$ and $\overline{a}_l = \text{round}\left(a_l 2^L\right)$. Since $\overline{a}_l$ is a bounded integer variable, $\overline{a}_l$ take any value between 0 and $v_{\max}$. Then Eq. 3 can be generalized as

$$H_{int} = \sum_{l=0}^{v_{\max}} l H_l. \tag{4}$$

Note that the coefficients of these sub-filters can be either signed-binary or 0. Binary filters can be written as $H_l = H_l^+ - H_l^-$ such that both H_l^+ and H_l^- contains only 1 or 0. Then,

$$H_{int} = \sum_{l=-v_{\max}}^{v_{\max}} l H_l^s. \tag{5}$$

such that H_l^s is either H_l^+ or H_l^-. Based on this reformulation, an instance of the problem can be defined as "Given a two-dimensional filter H_{int} with integer coefficients, is $\left\{H_l^s \in B : \sum_{l=-v_{\max}}^{v_{\max}} l H_l^s = H_{int}\right\} \neq \emptyset$?" The proof is as follows: Membership in NP is shown by guessing binary coefficients for H_l^s. Membership in NPC will be demonstrated by polynomially transforming an instance of the subset sum problem to the instance of this problem. An instance of the subset problem is defined as the decision problem in the form "Given η_l and b as integers, is $\{x_l \in B : \sum_{l \in S} \eta_l x_l = b\} \neq \emptyset$?". Firstly note that η_l can be made lower than $v_{\max}$ by selecting the quantization wordlength L appropriately. Define an impulse filter H_b such that only one of its coefficients is nonzero. Let this nonzero coefficient be at the (i, j)'th entry of this filter and let its value be b. Also define S as a set of impulse filters such that each filter in S, i.e. H_l, has $\eta_l x_l$ at its (i, j)'th entry and the remaining entries are 0. From each filter in S, η_l can be factored out to form the binary filters whose (i, j)'th entries are x_l. Since $\sum_l \eta_l x_l = b$ from the definition of the subset sum problem, then the impulse filter H_b can also be given as a linear combination of impulse filters in S: $\sum_l \eta_l H_l = H_b$. Recall that impulse filter is a special filter such that all filter types can be formed by using appropriate impulse filters. Q.E.D.

Since the aim is to minimize the number of adders in the overall system, the 0-1 integer programming model will be formed. Firstly the binary variables of the model must be defined:

$$q_l = \begin{cases} 1 \text{ , if } l \text{ is used as a scalar.} \\ 0 \text{ , otherwise} \end{cases} \tag{6}$$

$$p_l = \begin{cases} 1 \text{ , if } q_l \text{ or } q_{-l} \text{ or both are selected.} \\ 0 \text{ , otherwise} \end{cases}$$

$$(h_{ij})_l : \quad (i,j)\text{'th entry of } H_l^s.$$

If the dimension of the filter is $M \times N$, then the constraints of the problem are as follows

Constraint 1 *Eq 5 is rewritten for each entry h_{ij} of H_{int}.*

$$\textstyle\sum_{l=-v_{\max}}^{v_{\max}} l\,(h_{ij})_l = h_{ij} \ , 0 \le i \le M-1, 0 \le j \le N-1 \tag{7}$$

Constraint 2 *If at least one entry of H_l^s is nonzero, then l will be used as a scalar in the implementation.*

$$MNq_l - \textstyle\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(h_{ij})_l \ge 0,\ -v_{\max} \le l \le v_{\max} \tag{8}$$

Constraint 3 *If both q_l and q_{-l} are selected, then the filters must be combined to reduce the cost:*

$$2p_l - q_l - q_{-l} \ge 0,\ 1 \le l \le v_{\max} \tag{9}$$

The objective function will consist of three items. Firstly, each filter will be implemented using adders only. Secondly, adders and shifters will be also used to implement each scalar because each scalar is an integer number which can be represented with L-bit signed binary string. Let σ_l be the minimum number of adders required in the representation of l and $-l$. Finally, the number of selected scalars will determine the number of additions to sum the scaled sub-filter outputs. Then the total number of adders A is

$$A = \sum_{l=-v_{\max}}^{v_{\max}} \left(p_l + \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} \left(|h_{ij}|_l - 1 \right) \right) + \sum_{l=0}^{v_{\max}} p_l \sigma_l - 1 \tag{10}$$

The decomposition of the filter with minimum number of adders, A^*, is the solution of the model defined as "$A^* = \min A$ such that equations 7, 8 and 9 are satisfied".

The number of adders required for the implementation of the binary filters of Eq. 3 can be reduced further by generating a set of binary filters that will be summed up.

Theorem 2 *"Implementation of a binary filter as a sum of binary filters" problem is NP-complete.*

Proof: An instance of this new problem can be defined as follows: "Given a set of binary filters H_n where $1 \leq n \leq N$ and a binary filter H_l, is $\{q_{nl} \in B : \sum_n H_n q_{nl} = H_l\} \neq \emptyset$?" The membership in NP can be easily shown by guessing the binary filters. The membership in NPC can be easily shown by polynomially transforming an instance of set-partitioning feasibility problem ("Given an $M \times N$ 0-1 matrix $\mathbf{A}$, is $\left\{x \in B^N : \mathbf{Ax} = \mathbf{1}\right\} \neq \emptyset$?") to an instance of this problem. Let the output vector $\mathbf{1}$ represent the nonzero entries of H_l such that H_l has M nonzero entries. Arbitrarily assign i and j values for each entry in $\mathbf{1}$ such that $i + j = m$. Using this assignment, form a binary filter H_l such that for assigned values of (i, j), the filter coefficient $\left(h_{ij}\right)_l = (-1)^m$ and the remaining coefficients are zero. Similarly, let the column vector $\mathbf{A}_n$ represent the coefficients of H_n such that for assigned values of (i, j), the filter coefficient $\left(h_{ij}\right)_n = a_{mn}.\left(h_{ij}\right)_l$ and the remaining coefficients are zero. In this way, n binary filters have been formed from matrix $\mathbf{A}$. So an instance of the set partitioning feasibility problem is polynomially transformed to an instance of the new problem. The set-partitioning problem has a solution if and only if there exists at least a 1 at each row, $\sum_n a_{mn} \geq 1$. Similarly, the new problem has a solution only when there exists at least one sub-filter that has the nonzero coefficient of the binary filter, $\sum_n \left|\left(h_{ij}\right)_n\right| \geq \left|\left(h_{ij}\right)_l\right|$. Q.E.D.

Now, a method will be defined to extract the binary filters that will be used to implement the sub-filters in Eq. 3. Let $\Lambda^{(0)}$ denote the ordered-set of sub-filters satisfying Eq. 3. The sub-filters in $\Lambda^{(0)}$ can be ordered in a completely arbitrary manner but once they are ordered, the order must not change. Let $Q_{x,y}$ denote the ordered subset that contains the sub-filters of $\Lambda^{(0)}$. Here, x is the number of sub-filters in the subset and y is the order of the subset among the subsets having the same number of elements. Let H_m and H_n be the elements of a subset that contains only these two sub-filters, say $Q_{2,y}$ and let $m < n$. The common filters that contain common terms of both H_m and H_n, can be found by positive and negative intersection operations whose definitions are given as follows:

Definition 1 *(*Positive intersection*) The positive intersection of H_m and H_n is $H^1_{2,y}$ and it is defined as $H^1_{2,y} = H_m \bigcap^1 H_n$. Each coefficient $\left(h_{ij}\right)^1_{2,y}$ is given* *by*

$$(h_{ij})^1_{2,y} = \begin{cases} (h_{ij})_m \text{ , } \mathit{if} \ (h_{ij})_m = (h_{ij})_n \\ 0 \qquad\quad \text{ , } \mathit{otherwise} \end{cases} , m < n \tag{11}$$

Definition 2 *(*Negative intersection*) The negative intersection of H_m and H_n is $H^0_{2,y}$ and it is defined as $H^0_{2,y} = H_m \bigcap^0 H_n$. Each coefficient $(h_{ij})^0_{2,y}$ is given by*

$$(h_{ij})^0_{2,y} = \begin{cases} (h_{ij})_m \text{ , } \mathit{if} \ (h_{ij})_m = -(h_{ij})_n \\ 0 \qquad\quad \text{ , } \mathit{otherwise} \end{cases} , m < n \tag{12}$$

Using these definitions, H_m and H_n can be written as

$$\begin{aligned} H_m &= H^1_{2,y} + H^0_{2,y} + H_{m,(2,y)} \\ H_n &= H^1_{2,y} - H^0_{2,y} + H_{n,(2,y)} \end{aligned} \tag{13}$$

where $H_{m,(2,y)}$ and $H_{n,(2,y)}$ are the *remainders* of H_m and H_n respectively. These filters contain the terms that are covered neither by $H^1_{2,y}$ nor $H^0_{2,y}$,

Lemma 1 *If at least two nonzero terms exist in an intersection, then there exists a common filter which is the intersection itself.*

There are 2^λ subsets of $\Lambda^{(0)}$. Therefore, there are $2^\lambda - \lambda - 1$ subsets that contain two or more sub-filters. According to Lemma 1, this is an upper bound on the number of common filters. The common filter of a subset can be found by using Lemma 1 with the Definitions 1 and 2 recursively as shown in 2. The requirement for calling the *find_common* routine is that each sub-filter in $Q_{x,y}$ has at least two nonzero terms. Each sub-filter in the subset $Q_{x,y}$ is accessed by the index of $Q_{x,y}$. The term r_0 is the root vertex of the tree and its assignment with the first sub-filter in $Q_{x,y}$ is only for the initialization of the tree. Note that this algorithm generates a binary tree of sub-filters and the generation order is important to reconstruct the filters in $\Lambda^{(0)}$ (3). The values on the arcs of the tree determines the intersection type such that a 1 stands for the positive and a 0 stands for the negative intersection. Also the superscripts on the leaves of the tree show the intersection orders. The reconstruction equation for each sub-filter can be obtained by using these leaves and the intersection order.

Let $\Lambda^{(1)}$ denote the set of common filters and remainders that are derived from $\Lambda^{(0)}$ such that the common filters due to subsets with one sub-filter, i.e. $Q_{1,y}, \forall y$, do not appear in $\Lambda^{(1)}$. Note that each filter appears only once in

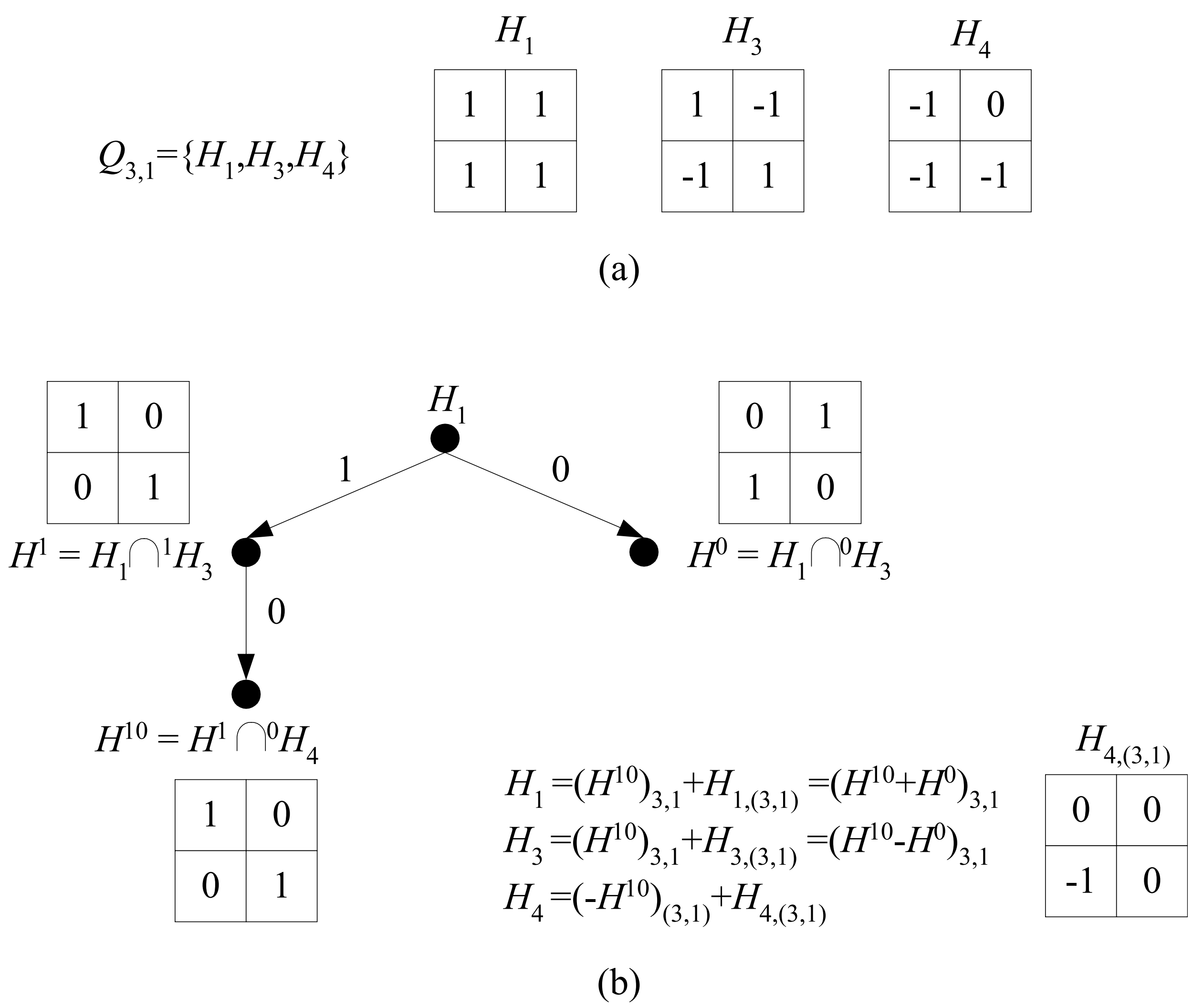
```
find_common$\left(Q_{x,y}, \Lambda^{(0)}\right)$
{   $l = Q_{x,y}[1]$;
    $p = 2$;
    $t^c_{x,y}[r_0] = H_l$;
    common $\left(H_l, Q_{x,y}, p, x, \Lambda^{(0)}, t^c_{x,y}\right)$;
    return $\left(t^c_{x,y}\right)$;
}
common $\left(H_m, Q_{x,y}, p, x, \Lambda^{(0)}, t^c_{x,y}\right)$
{   $l = Q_{x,y}[p]$;
    $H^1 = H_m \bigcap^1 H_l$;
    $H^0 = H_m \bigcap^0 H_l$;
    if $(|H^1| > 1)$ $t^c_{x,y}[H_m] \rightarrow left = H^1$;
    if $(|H^0| > 1)$ $t^c_{x,y}[H_m] \rightarrow right = H^0$;
    if $(p < x)$
    {       if $(|H^1| > 1)$ common $\left(H^1, Q_{x,y}, p+1, x, \Lambda^{(0)}, t^c_{x,y}\right)$;
            if $(|H^0| > 1)$ common $\left(H^0, Q_{x,y}, p+1, x, \Lambda^{(0)}, t^c_{x,y}\right)$;
    }
    return $\left(t^c_{x,y}\right)$;
}
```

Fig. 2. Algorithm that generates possible common terms in $Q_{x,y}$

$\Lambda^{(1)}$. Then $H_l \in \Lambda^{(0)}$ can be written by using the filters in $\Lambda^{(1)}$. If $\Lambda^{(1)} \neq \emptyset$, then additional common terms can be extracted by forming new subsets from the filters in $\Lambda^{(1)}$, i.e. the whole procedure explained for $\Lambda^{(0)}$ can be repeated for $\Lambda^{(1)}$ and the resulting filters will form $\Lambda^{(2)}$. This recursion continues until $\Lambda^{(i)} = \emptyset$. Let the final recursion that leads a nonzero set be represented with I. For every filter $H_l^{(i)}$ in $\Lambda^{(i)}$, there exist a number of reconstruction equations that use the filters of $\Lambda^{(i+1)}$. Let $H_{l,z}^{(i)}$ be the z'th reconstruction equation for $H_l^{(i)}$ and let $Q_{l,z}^{(i)}$ be the set of filters that are used in the reconstruction equation $H_{l,z}^{(i)}$. Then,

$$H_{l,z}^{(0)} = \sum_{n=1}^{\left|Q_{l,z}^{(1)}\right|} s_{l,z,n} H_n^{(1)} \ , \forall H_l \in \Lambda^{(0)} \tag{14}$$

where $s_{l,z,n}$ is the sign of H_n in $H_{l,z}^{(i)}$. Based on these definitions a 0-1 integer programming model can be formed to solve the problem stated in Theorem 2



$Q_{3,1}=\{H_1,H_3,H_4\}$
H_1
H_3
H_4
(a)
H_1
1
0
$H^1 = H_1 \cap^1 H_3$
$H^0 = H_1 \cap^0 H_3$
0
$H^{10} = H^1 \cap^0 H_4$
$H_1 = (H^{10})_{3,1} + H_{1,(3,1)} = (H^{10}+H^0)_{3,1}$
$H_3 = (H^{10})_{3,1} + H_{3,(3,1)} = (H^{10}-H^0)_{3,1}$
$H_4 = (-H^{10})_{(3,1)} + H_{4,(3,1)}$
$H_{4,(3,1)}$
(b)

Fig. 3. Generation of common terms in $Q_{3,1} = \{H_1, H_3, H_4\}$: (a)Filters in $Q_{3,1}$, (b)the generated tree and the reconstruction equations.

with minimum number of adders. The binary variables of the problem are

$$q_l^{(i)} = \begin{cases} 1\ , \text{ if filter } H_l^{(i)} \text{ is selected} \\ 0\ , \text{otherwise} \end{cases} \qquad (15)$$

$$q_{l,z}^{(i)} = \begin{cases} 1\ , \text{ if equation } H_{l,z}^{(i)} \text{ is selected} \\ 0\ , \text{otherwise} \end{cases}$$

$$p_l^{(i)} = \begin{cases} 1\ , \text{ if either } H_l^{(i)} \text{ or } H_{l,z}^{(i)} \text{ is selected} \\ 0\ , \text{otherwise} \end{cases}$$

The constraints of the problem are as follows:

Constraint 4 *Exactly one equation or the sub-filter itself is required to implement each filter $H_l^{(0)}$ in $\Lambda^{(0)}$:*

$$q_l^{(0)} + \sum_y q_{l,z}^{(0)} = 1 \ , 1 \leq l \leq \lambda \tag{16}$$

Constraint 5 *At most one equation or the sub-filter itself is required to implement each filter $H_l^{(i)}$ in $\Lambda^{(i)}$ for all $i \geq 1$:*

$$p_l^{(i)} - \left(q_l^{(i)} + \sum_y q_{l,z}^{(i)} \right) = 0 \ , 1 \leq l \leq \left| \Lambda^{(i)} \right| , 1 \leq i < I \tag{17}$$

Constraint 6 *If an equation is selected for the implementation, then all of the filters in the equation must be selected simultaneously:*

$$\left| Q_{l,z}^{(i+1)} \right| q_{l,z}^{(i)} - \sum_{n=1}^{\left| Q_{l,z}^{(i+1)} \right|} p_n^{(i+1)} = 0 \quad \begin{array}{l} , 1 \leq l \leq \left| \Lambda^{(i)} \right| \\ , \forall y, 0 \leq i < I \end{array} \tag{18}$$

Since the aim is to minimize the number of adders in the implementation, the objective function should be as

$$A_c = \sum_{i=0}^{I} \sum_{l=1}^{\lambda} \sum_{y} \left(\left| Q_{l,z}^{(i+1)} \right| - 1 \right) q_{l,z}^{(i)} + \sum_{i=0}^{I} \sum_{l=1}^{\lambda} c_l^{(i)} q_l^{(i)} \tag{19}$$

Therefore, the 0-1 integer programming model can be summarized as "$A_c^* = \min A_c$ such that equations 16, 17, 18 hold simulataneously."

2.3 Common Partial Sum Extraction

The solution of the common partial sum extraction problem is a number of selected filters and equations. Starting from the filters of $\Lambda^{(I)}$, Eq 14 can be iteratively used to obtain the filters and equations at $i-1$. At the end of this procedure, each sub-filter in $\Lambda^{(0)}$ will be written in terms of sub-filters that might belong to different recursion levels:

$$H_l = \sum_{i=0}^{I} \sum_{n=1}^{\left| \Lambda^{(i)} \right|} s_{l,n}^{(i)} q_n^{(i)} H_n^{(i)} \ , \forall H_l \in \Lambda^{(0)} \tag{20}$$

Here, the inner sum is actually Eq 14 that is written in a way to include the binary filter selection variables. The outer sum is required to scan sets of all recursion levels. Common partial sum extraction will be handled on these equations for all $H_l \in \Lambda^{(0)}$.

Example 1 Assume that there are two filter reconstruction equations, $H_1 = A+B+C$ and $H_2 = A+B-C$. If H_1 and H_2 are implemented as they are, then four adders will be required. However studying these equations simultaneously will easily show that $A+B$ is a partial sum that is common in both filters. Then an implementation like $K = A+B$, $H_1 = K+C$ and $H_2 = K-C$ will require three adders.

Let us form a bit string of length w for each sub-filter $H_l \in \Lambda^{(0)}$:

$$w = \sum_{i=0}^{I} \left|\Lambda^{(i)}\right| \tag{21}$$

Note that $\Lambda^{(i)}$ is an ordered set and each sub-filter of $\Lambda^{(i)}$ is represented with a binary variable $q_n^{(i)}$, $1 \leq n \leq \left|\Lambda^{(i)}\right|$. Let σ_l denote the corresponding bit string for H_l. The j'th entry in b_l is given by the following equation:

$$\sigma_l\left[j\right] = \sigma_l\left[n + \textstyle\sum_{m=0}^{i}\left|\Lambda^{(m)}\right|\right] = s_{l,n}^{(i)} q_n^{(i)} \quad \begin{array}{l} ,1 \leq n \leq \left|\Lambda^{(i)}\right| \\ ,0 \leq i < I \\ ,1 \leq j \leq w \end{array} \tag{22}$$

Note that the sum term in the calculation of the index is tricky: The summation is not carried out when $m = i$. In this way, the implementation style of each sub-filter $H_l \in \Lambda^{(0)}$ will be encoded in the related bit-string b_l.

Let p_j be a bit-pattern of length w. Note that the entries of p_j are from the set $\{-1, 0, 1\}$. Let P be the set of all possible bit-patterns.

Theorem 3 *Implementing a bit-string as a sum of bit-patterns is an NP-complete problem*

Proof: An instance of the problem can be stated as follows: "Given a bit string σ_l and a set of bit patterns P and a binary variable q_j, is $\left\{q_j \in B : \sum_j q_j p_j = \sigma_l\right\}$"? Note that this is quite similar to the problem defined in Theorem 2. Since the proof is also similar, it is skipped.

3 The Algorithm

In this section, the developed heuristic algorithm is explained. It consists of three subroutines as in previous section.

Let $\overline{h}_{ij}$ be the odd integer equivalent of h_{ij}, which is the (i,j)'th entry of H:

$$\overline{h}_{ij} = \text{round}\left(h_{ij}2^{L}\right)2^{-m_{ij}} \text{ such that } \overline{h}_{ij} \in \mathbf{Z}^{odd} \tag{23}$$

Note that the quantized entries of the filter, h^{q}_{ij}, can be obtained from $\overline{h}^{q}_{i,j}$. Let H^{q} be the quantized filter where each entry is represented by h^{q}_{ij}. An example is shown in 4(a).

The heuristic algorithm operates on the quantized 2-D filter It recursively generates the scalars and filters of Eq. 3. However, these filters are *super-binary* filters. A super-binary filter is a binary filter where each entry of a super-binary filter consists of a signed bit and the number of shifting operations. Note that a super-binary filter can be easily decomposed into binary filters by applying Eq. 2 on the super-binary filter. The algorithm tries to maximize the number of adders in the scalars so that the number of sub-filters is decreased. In this way, it is aimed to minimize the number of adders in the overall system. The initial call of the algorithm requires l to be initialized with 1 so that Eq. 3 will hold. Below, the functions of the algorithm are explained with the aid of the example in 4. Here, the quantization wordlength of the coefficients is eight-bit for the fractional part:

Step 1 *Using H_q, generate $\mathbf{C}$, the array of unique odd integers given by Eq. 23 (4(b)).*

Step 2 *Generate an adder dag (*direct *a*cyclic *g*raph*), τ, from $\mathbf{C}$ such that where the source is an imaginary input signal, d and each sink is the scaled signal with a coefficient in $\mathbf{C}$. This can be done with any algorithm that minimizes the number of adders in a 1-D system. In this study, the algorithm in [3] is used due to its fast computation time and low memory requirement (refsamplerun2Ddecompose(c)).*

Step 3 *Extract τ^{*}, the most expensive common sub-dag of τ in terms of adder count. The union of all paths from a sink to the source defines the sub-dag due to the related coefficient. Let Υ be the hypergraph that contains all sub-dags. Generate all possible sub-hypergraphs of Υ so that each sub-hypergraph will contain at least two sub-dags. The intersection of a hypergraph can be defined as the sub-dag of weighted-edges and nodes that are common in all the dags of the hypergraph. As the aim is the minimization of the number of adders in the whole system, firstly select the sub-hypergraph with maximum cardinality. If there are more than one candidate, then select the sub-hypergraph with the maximum number of adders in its intersection. If these conditions do not suffice, then introduce additional criteria like adder-width, distinction*

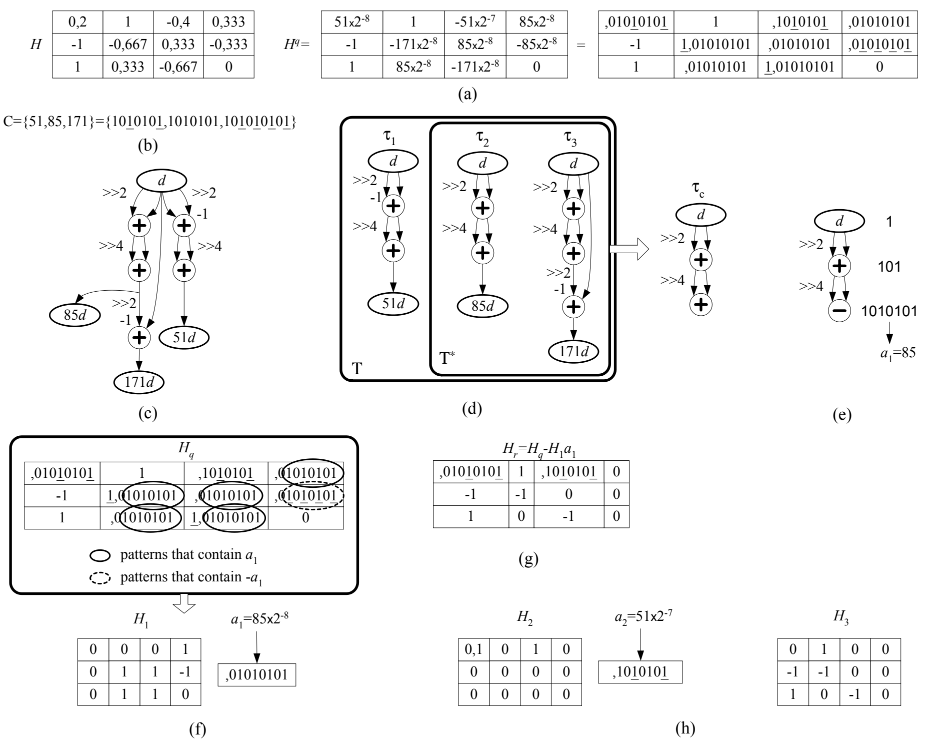


Fig. 4. Sample execution of the 2Ddecomposition algorithm: (a)A 2-D filter and its eight-bit quantized version, (b)the **C** array, (c)the minimum-adder dag, (d)The hypergraph of sub-dags and extraction of the common sub-dag, (e)calculation of the first scalar, (f)generation of the first super-binary filter, (g)generation of the remainder, (h)other sub-filters and scalars after the execution is over.

between addition and subtraction until one sub-hypergraph, Υ^*, remains. The intersection of Υ^* is the sub-dag τ^*. In 4(d), the first condition is sufficient.

Step 4 *Calculate the scalar a_l from the sub-dag τ^* as in refsamplerun2Ddecompose(d).*

Step 5 *Extract the 2-D super-binary filter from the original filter. The algorithm starts with a null filter, i.e. all entries of H_l are 0. Recall that the sink of a sub-dag in Υ^* correspond to a coefficient in H^q. Firstly, the positions of these coefficients in H_q are determined. Then within each coefficient, a pattern that corresponds to a_l or $-a_l$ is identified and the corresponding entries in H_l are set to 1 or -1, respectively. The starting bit-position of the pattern in the coefficient of H^q determines the amount of shift in the relevant entry of H_l. In 4(f), extraction of the first sub-filter is shown.*

Step 6 *Form the remainder: $H_r = H_q - H_l a_l$ (4(g)).*

Step 7 *Copy H_r to H_q and repeat the process until $a_l = 1$, i.e. the original filter is completely decomposed into a set of scalars and super-binary filters as shown in 4(h).*

In some cases, the algorithm might produce similar sub-filters and scalars. Hence, if there are more than one a_l (or H_l) with the same value, then the related H_l's (or a_l's) must be added up and the iterative process is repeated until uniqueness of H_l and a_l is satisfied.

3.2 Common Term Extraction in Sub-filters

Each super-binary filter is decomposed into binary filters by using Eq. 2:

$$H = \sum_{l=1}^{\lambda} H_l a_l = \sum_{l=1}^{\lambda} \left(\sum_{b=0}^{L-1} (H_b)_l \, 2^{-b} \right) a_l \tag{24}$$

Using the binary filters $(H_b)_l$, the ordered set $\Lambda^{(0)}$ is obtained. Let H_j represent one of these filters in $\Lambda^{(0)}$. The heuristic algorithm for extracting the common terms in this set is simply a greedy table-search method:

Step 8 *Call the function* find_common() *(2) repeatedly to generate all possible common filters of* $\Lambda^{(0)}$*. Note that if no common terms exist in a subset, say* $Q_{\widehat{x},\widehat{y}}$ *where* $\widehat{x} \geq 2$*, then there is no need for calling the function for the subsets that contain* $Q_{\widehat{x},\widehat{y}}$ *. In this way, the generation time of all common terms can be reduced drastically. Let set* Λ *contain these common filters and the sub-filters in* $\Lambda^{(0)}$*.*

Step 9 *Form the following tables and arrays:*

(1) *Table* Θ *such that the columns stand for the place of the nonzero terms in filters in* $\Lambda^{(0)}$ *and rows stand for the filters in* Λ*. The super-columns in the table are the groups of columns such that* j*'th super-column represents* $H_j \in \Lambda^{(0)}$*. Fill the table in such a way that the* (n, γ_j)*'th entry in* Θ *is the value of the* γ*'th entry in* $H_j \in \Lambda^{(0)}$ *if the* n*'th filter in* Λ *contains it, otherwise it is* 0*.*

(2) *Table* Φ *such that columns stand for the filters in* $\Lambda^{(0)}$ *and rows stand for the filters in* Λ*. Fill the table in such a way that the* (n, j)*'th entry in* Φ *is the number of nonzero terms in the* n*'th row and* j*'th super-column of* Θ*.*

(3) *Table* R *so as to store the selected rows of* Θ*.*

(4) *Array* Ω_1 *that holds the number of nonzero entries in each row of* Φ*.*

(5) *Array* Ω_2 *holding the sum of numbers in each row of* Φ*.*

(6) *Array* Ω_3 *that holds the maximum number of the nonzero entries in each row of* Φ*.*

Step 10 *Process the tables and arrays:*

(1) If a row n_e of Θ has all the entries that another row n_f has, then n_e is covered by n_f. Then delete row n_e in all tables and arrays if $\Omega_1[n_e] \leq \Omega_1[n_f]$.
(2) Among the rows where $\Omega_3[n] = \Omega_2[n]/\Omega_1[n]$, select row n^* in Θ such that $\Omega_1[n^*]$ is the maximum in Ω_1. If there exists more than one candidate, then select the one such that $\Omega_2[n^*]$ is also maximum.
(3) Include row n^* in R.
(4) In Θ, delete columns that correspond to the nonzero entries of n^*. Recalculate values in Φ, Ω_1, Ω_2 and Ω_3.
(5) In all tables and arrays, delete rows that do not contain nonzero entries.
(6) Repeat the process until no rows exist in Θ.

In 5, the execution of the algorithm on an example is shown. In this example, the starting set $\Lambda^{(0)}$ consists of four filters (5(a)). All non-empty common filters of this set are generated (5(b)) and the final set Λ is simply the union of both sets. After the tables and arrays formed for Λ (5(c)), the iteration starts. Firstly, the covered rows are erased. Here rows E and I are covered by row J. Since $\Omega_1[E] = \Omega_1[I] \leq \Omega_1[J]$, then E and I are deleted. Similarly row B is erased since it is covered by row H and $\Omega_1[B] \leq \Omega_1[H]$. The reduced tables and arrays are shown in 5(d). According to Step 10.2, the last row is the first selected filter to implement the system (5(e)). After the tables and arrays are processed to reflect the effects of the selection, the second iteration starts. After deleting covered rows, the tables and arrays are as in 5(f). In this iteration, again the first row is selected according to Step 10.2. The iterations continue until Θ is empty and the final form of table R and the corresponding filters are shown in 5(g). Using these filters, the filters in $\Lambda^{(0)}$ are written as $A = J^* + F^*$, $B = G^*$, $C = J^* - F^*$, and $D = -J^* + H^*$. Note that the implementation of initial set requires nine adders in 5(a) while six additions are required in 5(g).

3.3 Common Partial Sum Extraction

Common partial sum extraction problem has been two-fold due to the previously defined heuristic algorithms. Firstly, using the equations derived from table R, the bit strings can be formed as described in Eq. 22. Secondly, recall that the decomposition of the initial filter generates super-binary filters which are decomposed into binary filters as specified in Eq. 24. Hence, the initial problem definition of Section 2.3 has to be enhanced to handle this new structure. An example is shown in 6(a). It has been shown that the original problem in Theorem 3 is NP-complete. Obviously the expanded problem will be much harder than the initial one. The following heuristic algorithm is the slightly modified version of the algorithm presented in [3]:

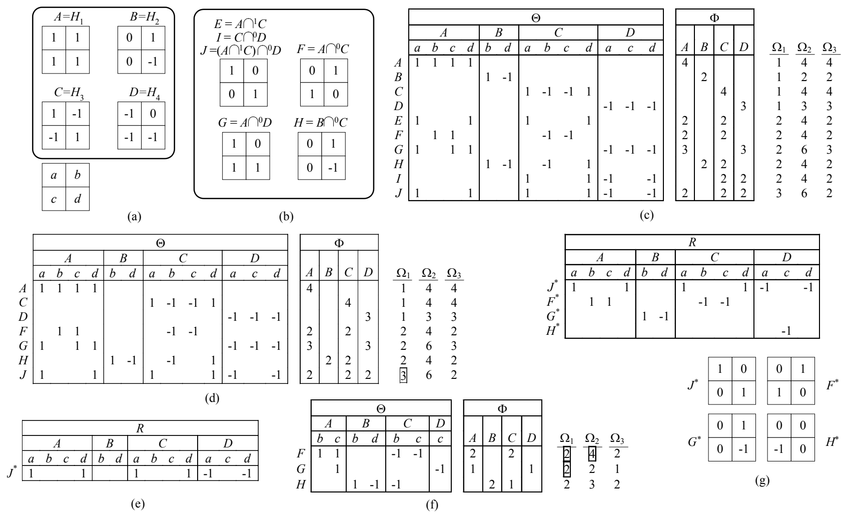


Fig. 5. Heuristic extraction of common filters and remainders : (a)The starting filter set $\Lambda^{(0)}$ and the position template, (b)the first level common-filter set, (c)initial tables and arrays, (d)the first iteration: Tables and arrays after covered rows are deleted and selection of the first common term, i.e. J, (e)the solution table R after the first iteration (f)the second iteration: Tables and arrays after covered rows are deleted and selection of the second common term, i.e. F, (g)the final solution table R and the filters.

Step 11 *For each super-binary filter H_l, form an array α_l of length $\left|\Lambda^{(0)}\right|$. If a sub-filter in $\Lambda^{(0)}$, say H_j, is used in the representation of H_l, then find the amount of shifts, b, that is required at its output by using Eq 24. If $H_j = \pm\left(H_b\right)_l$, then set the j'th entry of the array to $\pm 2^{-b}$.*

Step 12 *Using all arrays simultaneously, do the following procedure:*

(1) *Compute all two-terms in all strings. Here, the two-term ρ is the modified version of the two-term that was initially introduced in [9]. A modified two-term can be computed as follows: Find two nonzero entries in a string. Let j_1 and j_2 be the places and s_1 and s_2 be the signs of these two entries. Also let $v_1 = s_1 2^{-b_1}$ and $v_2 = s_2 2^{-b_2}$ be the values of the first entry and the second entry respectively. Here, $\rho = (j_1, j_2)$ pair is the two-term. The value of the two-term is*

$$v_\rho = \left|v_1 2^{j_1+b_1} + v_2 2^{j_2+b_1}\right| \qquad (25)$$

Let the process iteration variable i be the level of the entry. Initially, all entries are at zeroth level. The level of the two-term is given by the $\max(i_1, i_2)$. *Note that a two-term appears only once in a string.*

(2) *If a common partial sum exists, then obviously there will be multiple copies of the same two-term. The cost of a two-term is the number of copies. Pick the most expensive two-term. If there exists more than one candidate, then pick the one with lowest level. If still there is not a single candidate, then pick the one with maximum value. Let this two-term be defined as*

$H_1 = A + 2^{-1}B + 2^{-2}C + D$	(3 adders, 2 shifters)
$H_2 = 2^{-1}A + 2^{-2}B + 2^{-3}C - D$	(3 adders, 3 shifters)
TOTAL	(6 adders, 5 shifters)

(a)

sub-filter name	A	B	C	D
position	3	2	1	0
α_1	1	2^{-1}	2^{-2}	1
α_2	2^{-1}	2^{-2}	2^{-3}	-1

ρ	(j_1, j_2)	*cost (array #)*	*level*	$j_1 - j_2$	v_ρ
ρ_1	(3,2)	$\boxed{2\ (1,2)}$	$\boxed{0}$	$\boxed{1}$	10
ρ_2	(3,1)	$\boxed{2\ (1,2)}$	$\boxed{0}$	2	8,5
ρ_3	(3,0)	1 (1)	0	3	9
ρ_4	(3,0)	1 (2)	0	3	6
ρ_5	(2,1)	$\boxed{2\ (1,2)}$	$\boxed{0}$	1	5
ρ_6	(2,0)	1 (1)	0	2	6
ρ_8	(2,0)	1 (2)	0	2	2
ρ_7	(1,0)	1 (1)	0	1	5
ρ_9	(1,0)	1 (2)	0	1	-6

(b)

position	3	2	1	0
α_1	ρ_1^*	0	2^{-2}	1
α_2	$\rho_1^*\, 2^{-1}$	0	2^{-3}	-1

ρ	(j_1, j_2)	*cost (array #)*	*level*	$j_1 - j_2$	v_ρ
ρ_2	(3,1)	$\boxed{2\ (1,2)}$	1	2	80,5
ρ_3	(3,0)	1 (1)	0	3	81
ρ_4	(3,0)	1 (2)	0	3	78
ρ_7	(1,0)	1 (1)	0	1	5
ρ_9	(1,0)	1 (2)	0	1	-6

(c)

position	3	2	1	0
α_1	ρ_2^*	0	0	1
α_2	$\rho_2^*\, 2^{-1}$	0	0	-1

(d)

$\rho_1^* = A + 2^{-1}B$	(1 adder, 1 shifter)
$\rho_2^* = \rho_1^* + 2^{-2}C$	(1 adder, 1 shifter)
$H_1 = \rho_2^* + D$	(1 adder)
$H_2 = 2^{-1}\rho_2^* - D$	(1 adder, 1 shifter)
TOTAL:	(4 adders, 3 shifters)

(e)

Fig. 6. Extraction of common partial sums: (a)The starting filter set with partial sums, (b)the starting arrays and the two-terms in the first iteration, (c)the arrays and the two-terms in the second iteration, (d)the arrays after at the end of second iteration (e)the resulting implementation utilizing common partial sums

ρ^*.

(3) In all strings that contain ρ^*, do the following: Delete the entry at j_2. Replace the entry at j_1 with $\rho^* s_1 2^{-b_1}$. Note that the value of this entry is updated as $v_1 = v_{\rho^*} s_1 2^{-b_1}$

(4) Repeat the procedure until there exists exactly one nonzero entry in all strings.

The first two iterations of the algorithm is shown in 6. The selection of the two-term is shown by boxed entries in every table of two-terms.

4 Experiments

The algorithm has operated on a number of infinite precision 2-D filters. The first example is from [8], a nonseparable perfect reconstruction 2-D filter used for block transform. In this table, *1-D* stands for the implementation in 1(a) and *2-D* stands for the implementation proposed in this paper, i.e. 1(b). According to the results given in 1 the gain in the number of adders is more than 50% when wordlength required for the quantization wordlength is greater than

Table 1
Number of adders for [8]

Wordlength	org	1-D	2-D
8	95	83	69
12	167	129	90
16	229	171	113
24	303	225	130

Table 2
Percent gain in the number of adders for different filter sizes

L	8		12		16		24	
Size	1-D	2-D	1-D	2-D	1-D	2-D	1-D	2-D
5×5	16	37	19	41	19	41	20	38
7×7	18	34	24	43	26	44	28	46
9×9	16	31	22	38	25	43	28	45

twelve bits. The algorithm is also executed in fifty different infinite-precision filters with different sizes. 2 shows that the average percent-gain in the number of adders increases as the wordlength increases. This is due to the fact that the probability of common patterns increases in quantized coefficients with the wordlength. As a more detailed illustration, the number of adders in different 7×7 filters for different quantization wordlengths is presented in 7. Also, 8 shows the adder requirements in different implementations. In this figure, org represents the direct implementation where no subexpression sharing is done.

The algorithm has also been executed on a number of finite-precision filters in [7]. The results are presented in 9. In this figure, $a1_a$ stands for the A filter of Table A1 in [7]. As it can be observed, the adder gain in finite precision filters is not as high as infinite precision filters, but still there is an improvement after the new algorithm is applied.

The experimental results show that with the newly-developed 2-D CSE method, the number of adders can be reduced more than original and the traditional 1-D implementations of 2-D filters. The interconnects in the 2-D filters generated with this method are local, hence it is estimated that the hardware implementation will not consume too much area and power. Exact 0-1 integer programming models have not been implemented for these filters because they are memory and run-time inefficient.

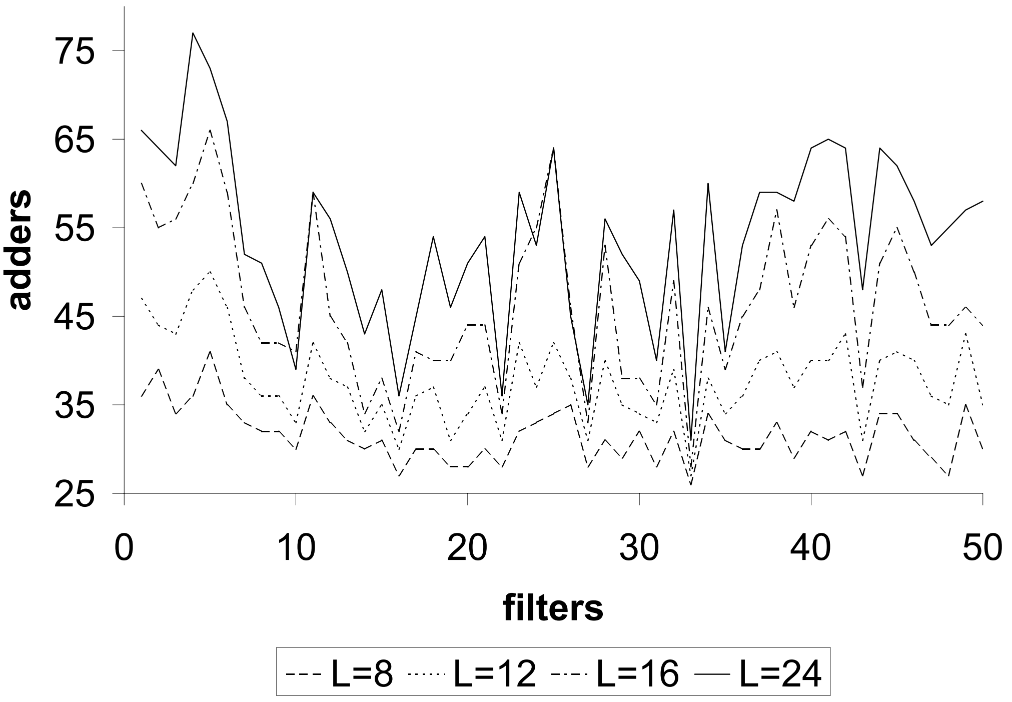


Fig. 7. The number of adders for different wordlengths. Size of the filters is 7×7

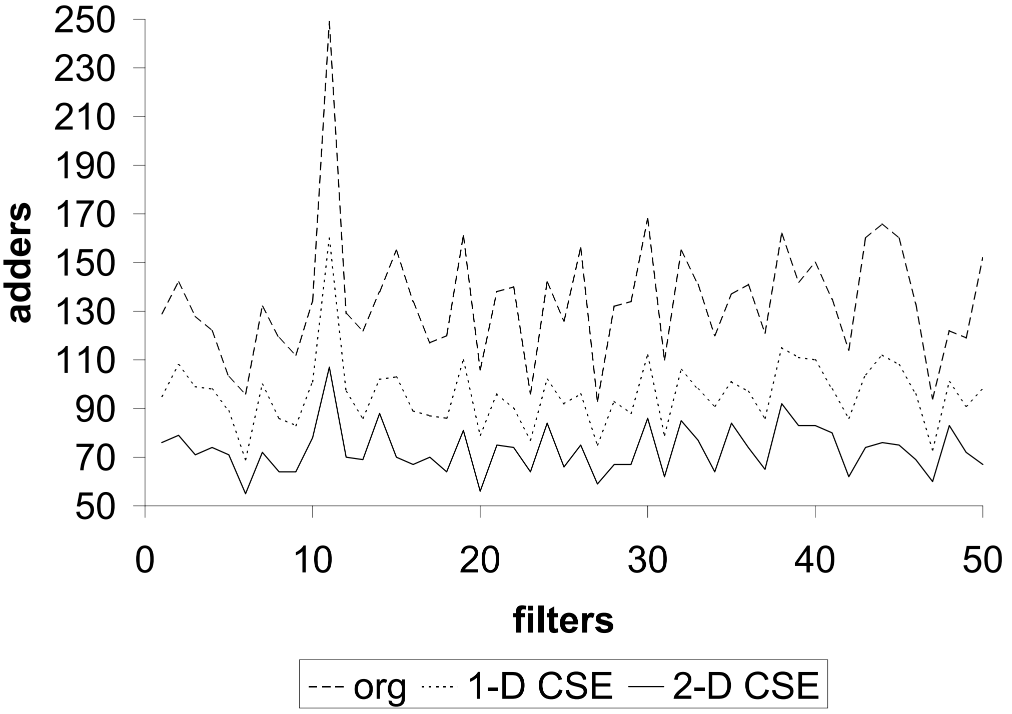


Fig. 8. The number of adders in different implementations. Size of the filters is 9×9 and $L = 16$.

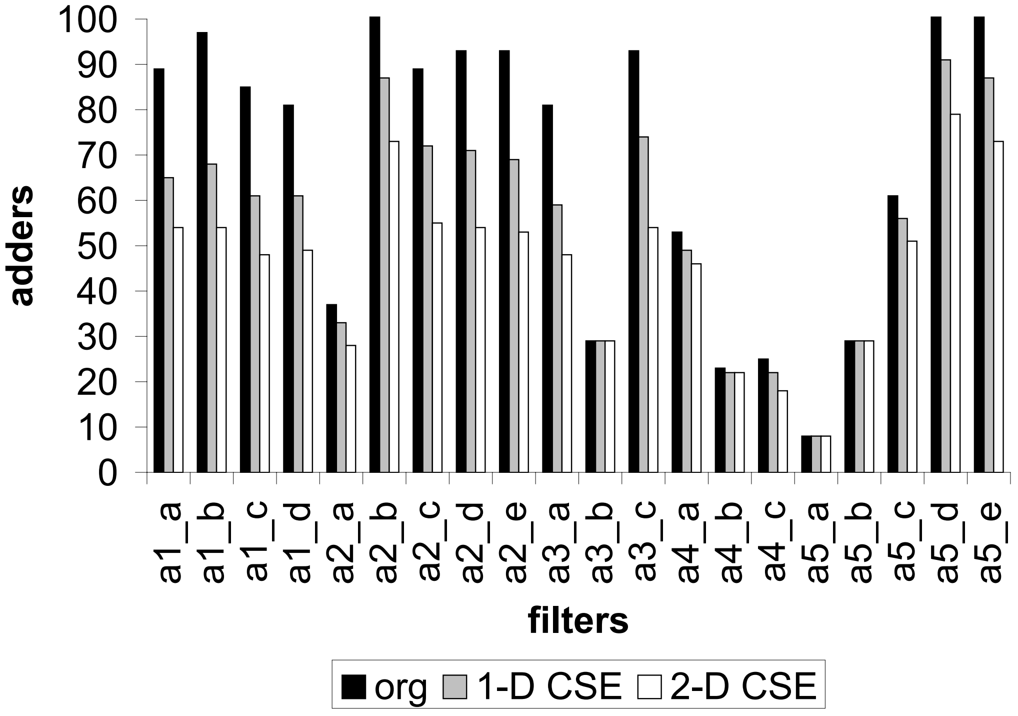


Fig. 9. Adders in different finite-precision filters.

References

[1] K. K. Parhi, *VLSI Digital Signal Processing Systems.* John Wiley & Sons, 1999.

[2] M. Martnez-Peir, E. I. Boemo, and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," *IEEE Trans. Circuits Syst.-II*, vol. 49, pp. 196-203, March 2002

[3] A. Yurdakul and G. Dündar, "Fast and Efficient Algorithm for the Multiplierless Realization of Linear DSP Transforms," *IEE-Proceedings-Circuits, Devices, and Syst*, vol.149, pp.205-211, August 2002.

[4] S. Pei and S. Jaw, "Efficient design of 2-D multiplierless filters by transformation," *IEEE Trans. Circuits Syst.*, vol. CAS-34, pp.436-438, Apr. 1987.

[5] P. Siohan and A. Benlismane, "Finite precision design of optimal linear phase 2-D FIR digital filters," *IEEE Trans. Circuits Syst.*, vol. CAS-36, pp.11-21, Jan. 1989.

[6] H. K. Kwan and C. L. Chan, "Circularly symmetric two-dimensional multiplierless FIR digital filter design using an enhanced McClellan transformation," *IEE-Proceedings-Circuits, Devices, and Syst*, vol.136, pp.129-134, June 1989.

[7] L. Banzato, N. Benvenuto and G. M. Cortelazzo, "A design technique for

two-dimensional multiplierless FIR filters for video applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. CSVT-2, pp.273-284, Sep. 1992.

[8] I. Celasun, *Design of multidimensional perfect-reconstruction filter banks with compression applications*. PhD Thesis, Bogazici University, Turkey, 1996.

[9] A. Yurdakul and G. Dündar, "Multiplierless realization of linear DSP tranforms by using common two-term expressions," *Journal of VLSI Signal Processing Systems*, vol.22, pp.163-172, September/October 1999.