CIRCUITS SYSTEMS SIGNAL PROCESSING
VOL. 25, NO. 2, 2006, PP. 225–251

DOI: 10.1007/s00034-005-2505-5

SIMPLIFIED DESIGN OF CONSTANT COEFFICIENT MULTIPLIERS*

Oscar Gustafsson,[1] *Andrew G. Dempster,*[2] *Kenny Johansson,*[1] *Malcolm D. Macleod,*[3] *and Lars Wanhammar*[1]

Abstract. In many digital signal processing algorithms, e.g., linear transforms and digital filters, the multiplier coefficients are constant. Hence, it is possible to implement the multiplier using shifts, adders, and subtracters. In this work two approaches to realize constant coefficient multiplication with few adders and subtracters are presented. The first yields optimal results, i.e., a minimum number of adders and subtracters, but requires an exhaustive search. Compared with previous optimal approaches, redundancies in the exhaustive search cause the search time to be drastically decreased. The second is a heuristic approach based on signed-digit representation and subexpression sharing. The results for the heuristic are worse in only approximately 1% of all coefficients up to 19 bits. However, the optimal approach results in several different optimal realizations, from which it is possible to pick the best one based on other criteria. Relations between the number of adders, possible coefficients, and number of cascaded adders are presented, as well as exact equations for the number of required full and half adder cells. The results show that the number of adders and subtracters decreases on average 25% for 19-bit coefficients compared with the canonic signed-digit representation.
Key words: Constant multiplication, multiplierless, minimum adder graph.

1. Introduction

Multiplication with a constant fixed-point number is a common operation in many digital signal processing (DSP) algorithms, e.g., digital filters. The constant coefficient multiplier can be realized as a network of shifts, adders, and subtracters.

* Received February 14, 2005; revised October 4, 2005.
[1] Department of Electrical Engineering, Linköping University, Sweden. E-mail for Gustafsson: oscarg@isy.liu.se; E-mail for Johansson: kennyj@isy.liu.se; E-mail for Wanhammar: larsw@isy.liu.se
[2] Department of Electronic Systems, University of Westminster, London, United Kingdom. Now with: School of Surveying and Spatial Information Systems, University of New South Wales, Sydney, Australia. E-mail: a.dempster@unsw.edu.au
[3] Qinetiq Ltd., Malvern, United Kingdom. E-mail: m.macleod@signal.qinetiq.com

For bit-parallel arithmetic the shifts can be realized without any additional transistors using hard wiring. To obtain an efficient implementation, it is therefore of interest to minimize the number of adders and subtracters. As the hardware cost is similar for adders and subtracters, we will refer to both as adders, and to the number of adders as adder cost. When the multiplier is realized using shifts and adders, the realization is commonly referred to as "multiplierless." The subject of this work is the properties and topologies for interconnecting adders to minimize the complexity of multipliers capable of multiplying with one and only one constant coefficient. We are not interested in the multiple constant multiplication (MCM) case, commonly used in finite impulse response (FIR) filter implementations to utilize redundancy between coefficients, nor in restricted coefficient set multipliers, using multiplexers and adders to obtain a small set of possible coefficients, nor in general multipliers for any coefficient.

A B-digit signed-digit (SD) representation of a coefficient k can be written as

$$k = \sum_{i=0}^{B-1} b_i 2^{-i}, \tag{1}$$

where $b_i \in \{-1, 0, 1\}$. In the context of digital filters and linear transforms, this is sometimes referred to as a signed powers of two (SPT) representation. A signed-digit representation with a minimum number of nonzero digits (SPT terms) is called a minimum signed-digit (MSD) representation. One MSD representation is the canonic signed-digit (CSD) representation. The CSD representation has the extra condition that no two adjacent digits can both be nonzero, i.e., $b_i b_{i-1} = 0, i = 0, 1, \ldots, B-2$. This condition leads to a minimum nonredundant representation [24]. Assuming that the CSD representation has c nonzero digits, $c - 1$ adders are required for a realization. It is sometimes argued that multipliers based on a CSD representation result in the minimum number of adders.

CSD multipliers are based on sums of signed powers of two of the input word. However, other network topologies, not restricted to CSD representation, are available that require an even smaller number of adders [1], [6]–[9], [17]. An optimal method was described in [6] and extended in [12]. Using at most four adders, coefficients with wordlengths of up to 12 bits are covered, and using five adders, all coefficients up to 19 bits can be realized.

For digital filter design, this means that the search space for realizations using, e.g., three nonzero digits (SPT terms) can be extended to all coefficients that require two adders. Hence, a higher flexibility is provided during the filter design process. In [22] and [23] a design procedure for lattice wave digital filters is proposed that first limits the coefficient range and then performs an exhaustive search with all possible combinations to find the best solution. For this type of algorithm, it is straightforward to include coefficients that can be realized with n adders, but do not necessarily have a CSD representation with $n + 1$ nonzero digits.

The concept of constant coefficient multiplication using a minimum number

of adders has also been investigated for serial arithmetic [15] as well as carry-save arithmetic [3], [13]. However, this work focuses on bit-parallel nonredundant arithmetic using a two's-complement representation of the data.

In Section 2 an optimal approach to constant coefficient multipliers using a minimum number of adders is proposed. This section also includes a discussion on how the maximum number of nonzero digits of the coefficient, the maximum number of cascaded adders, and the number of adders are related. Parts of this were previously published in [12]. In Section 3 a heuristic method based on the work in [9] is presented. A bit-level cost measure of the number of full and half adder cells required to implement a constant coefficient multiplier is derived in Section 4. In Section 5 the results are presented, and in Section 6 an example constant coefficient multiplier is investigated. In Section 7 some conclusions are drawn.

2. Optimal approach

In this section an optimal approach to realizing constant coefficient multiplication with a minimum number of adders is described. Compared to the previously published optimal results in [6], this approach reduces the search space drastically by showing that several of the different adder topologies introduced result in identical sets of possible coefficients. This means that only one of them needs to be included in the search. Furthermore, by classifying the graphs, it is possible to reduce the search space further by covering even more cases in each search run.

2.1. Graph representation of multiplication

A network of shifts and adders can be represented using directed acyclic graphs (DAGs) with the following characteristics:

- For adder-cost C there are $C + 1$ vertices.
- Each vertex has an in-degree of two except for one vertex which has in-degree zero. That vertex is referred to as the input vertex and corresponds to the input of the multiplier.
- Each vertex has out-degree larger than or equal to one, except for one which has out-degree zero. That vertex is referred to as the output vertex and corresponds to the output of the multiplier.
- A vertex with in-degree two corresponds to an adder (subtracter).

Each edge can be assigned a value which corresponds to a shift and a possible negation (realized in the adder/subtracter). When the input vertex is assigned a value of one, it is possible to assign values to the remaining vertices of the graph as

$$f_i = e_j f_j + e_k f_k, \tag{2}$$

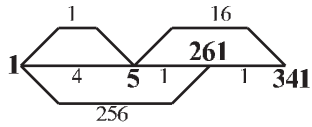


Figure 1. Graph representation of multiplication with 341. Fundamentals are bold.

where e_j and e_k are the edge values of two edges that both have the vertex corresponding to f_i as the terminal vertex. f_j and f_k are the values corresponding to the initial vertices of the two edges. The values f_i, f_j, and f_k are called fundamentals [6]. An example graph with fundamentals and edge values indicated is shown in Figure 1. Note that even though the graph is directed, we will not indicate the direction of the edges; instead, all graphs throughout the paper are directed from left to right. This multiplier only requires three adders, whereas a CSD multiplier for the same coefficient requires four adders.

By introducing an edge from the output vertex, a new output fundamental is obtained as

$$f_i' = e_i f_i. \tag{3}$$

Thus, it is only necessary to store values for positive odd integers at the output vertex. Even values can be obtained at the same cost as the odd fundamental by successively multiplying by two. Negative values can be obtained by negating the corresponding positive value.

2.1.1. Coefficient sets

Each graph can generate a limited number of values at its output vertex. We call the set of values the graph coefficient set of that graph. The coefficient set for a graph i with edge values of at most 2^B is denoted

$$G_{i,B} \in Z^+, \tag{4}$$

where Z^+ denotes all positive integer numbers.

We also define the odd graph coefficient set for a graph, which is all the odd values that a graph may have at its output vertex. The odd coefficient set is denoted and defined as

$$G_{i,B}^O \equiv G_{i,B} \cap Z_O^+, \tag{5}$$

where Z_O^+ denotes all odd positive integers.

All values that can be realized with a certain adder cost are called the cost coefficient set. The coefficient set with adder cost C and edge values of at most 2^B is denoted and defined as

$$H_{C,B} \equiv \bigcup_{\forall i \in D_C} G_{i,B}, \tag{6}$$

where D_C is the set of all graphs with adder cost C. An odd cost coefficient set can be defined in a similar way as shown in (5).

To find the coefficient sets and adder costs for different coefficient values, an exhaustive search is performed and the minimum adder cost and corresponding graph are stored in a lookup table. To reduce the effort required by this search, it is of importance to identify graphs that have identical graph coefficient sets.

2.2. Equivalent graphs

To reduce the number of unique graphs, some different equalities are discussed. This leads to a new form of representing the graph.

To see that two graphs generate the same set of coefficient, some transformations can be applied to obtain isomorphic graphs. If we can show that we can transform one graph to another without changing the value of the output vertex, these two graphs are equivalent in terms of coefficient sets, and only one of them must be considered.

When implementing the corresponding circuit, a fully specified graph can be derived by expanding the vertex reduced graph. These expansions will have the same output fundamental and the same number of adders. However, as will be discussed later, they may differ in adder depth (number of cascaded adders), power consumption, and the number of full adder cells required to implement the adders.

2.2.1. Addition reordering

Observation 1. *If the out-degree of a vertex is one, it is possible to change the place of the initial and the terminal vertex of the outgoing edge without changing the overall value of the multiplier.*

Motivation. Consider the subgraphs in Figure 2a, where f_1, f_2, f_3, f_4, and f_5 are the fundamentals at the internal vertices. The left-hand graph has a value at its output vertex of

$$f_5 = e_3 f_3 + e_4(e_1 f_1 + e_2 f_2). \tag{7}$$

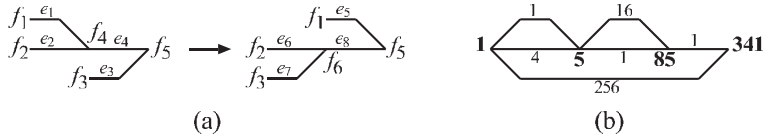
The right-hand graph has an output vertex value of

$$f_5 = e_5 f_1 + e_8(e_6 f_2 + e_7 f_3). \tag{8}$$

This gives

$$e_5 = e_1 e_4, \qquad e_6 e_8 = e_2 e_4, \qquad e_7 e_8 = e_3. \tag{9}$$

Thus, it is possible to select the edge values for the right-hand graph so that the overall graph has the same output value as the left-hand graph. The values of the internal vertices f_4 and f_6 are not the same, but as the output vertex values f_5 are identical, both graphs will have identical graph coefficient sets.



(a) (b)

Figure 2. (a) Reordering of vertices in graphs. (b) Multiplier graph from Figure 1 with reordered vertices.

Applying addition reordering to the graph in Figure 1 yields the graph shown in Figure 2b.

2.2.2. Transposition

Observation 2. *A transposed graph has a coefficient set identical to that of the original graph.*

Motivation. As the multiplier graph represents a single input, single output system, it follows from the transposition theorem [19] that the transfer function from the input vertex to the output vertex is identical to that from the output vertex to the input vertex of the transposed graph.

When the out-degree of any vertex is larger than two, more than one transposed graphs are obtained. This is due to the fact that there are more than one additions corresponding to that vertex in the transposed graph and there are several ways to order these vertices. An example is shown in Figure 3, where two different graphs are obtained after transposition. To obtain these graphs, the directions of all edges are reverted (note that the graphs are directed from left to right). The two vertices corresponding to the fundamentals 261 and 341 in the original graph, respectively, now become the input vertices of the transposed graphs. The previous vertex corresponding to 5 is transposed to a vertex corresponding to 17. Finally, as the original input vertex has an out-degree of three, this results in two different vertices in the transposed graph. As the ordering of these two vertices is arbitrary, there are two different alternatives, as shown in Figure 3. Notice that one of the equivalent graphs is identical to the original graph, whereas the other is the same as was obtained using addition reordering in Figure 2b.

2.3. Vertex reduction

As the internal fundamentals of the graphs are not considered here, we can reduce the number of vertices by allowing a vertex to have an in-degree higher than two. This means that the adder cost of each vertex is now not one adder; instead, the adder cost is the in-degree minus one adder.

Observation 3. *When the out-degree of a vertex is one, the initial and terminal*

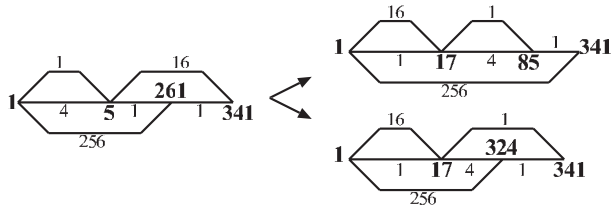


Figure 3. Transposition of the multiplier graph in Figure 1.

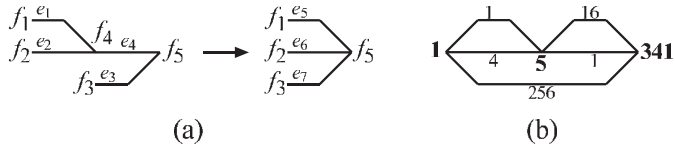


Figure 4. (a) Reduction of vertices. (b) Vertex reduction applied to the multiplier graph in Figure 1.

vertices of that edge can be merged to one vertex without changing the fundamental of the output vertex.

Motivation. Consider the subgraphs in Figure 4a, where again f_1, f_2, f_3, f_4, and f_5 represent the values of the internal vertices. The output value of the left-hand graph can be written as in (7). The output value of the right-hand graph is

$$f_5 = e_5 f_1 + e_6 f_2 + e_7 f_3. \tag{10}$$

This gives

$$e_5 = e_1 e_4, \qquad e_6 = e_2 e_4, \qquad e_7 = e_3, \tag{11}$$

and the edge values of the right-hand graph can be selected to obtain a vertex reduced graph that is identical to the original graph.

A vertex reduced representation of the graph in Figure 1 is shown in Figure 4b.

The reduction of vertices reduces the number of possible graphs, as will be shown in Section 5. The possible vertex reduced graphs for up to cost five are shown in Figure 5.

The vertex reduced graphs can be described as follows:

- The graph is a DAG.
- The adder cost of a vertex is the in-degree minus one.
- Each vertex has an in-degree larger than or equal to two, except for one vertex which has in-degree zero. That vertex is referred to as the input vertex and corresponds to the input of the multiplier.

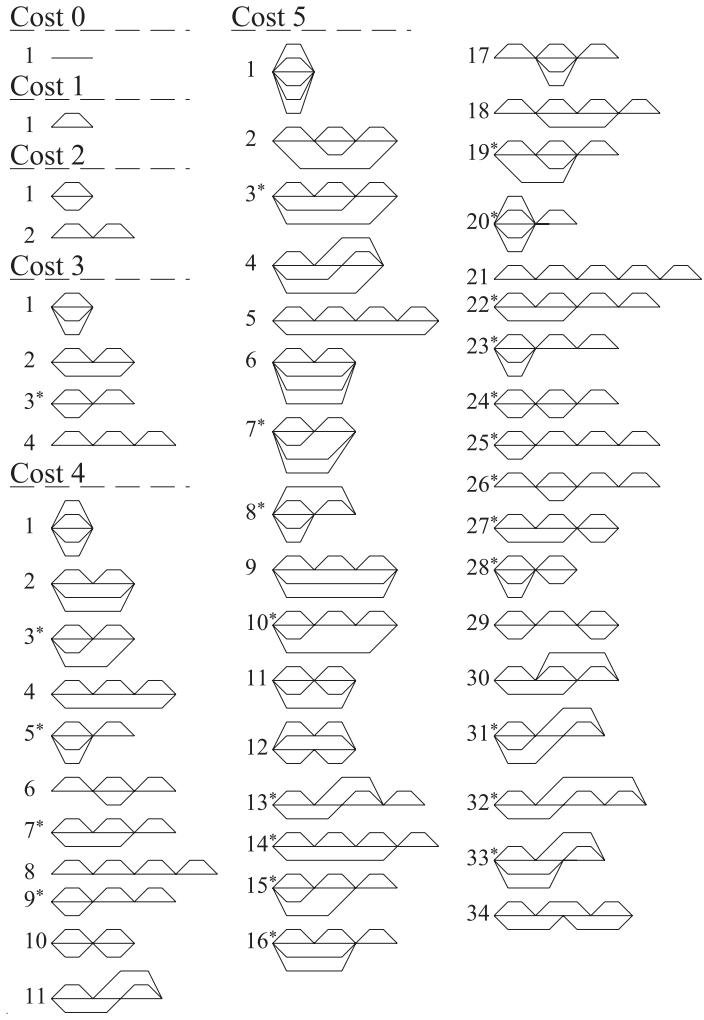


Figure 5. Possible unique vertex reduced graphs with adder cost up to five when transposition symmetry is considered. * denotes that the transposed graph is not isomorphic to the original graph. The graphs are directed from left to right.

- Each vertex has out-degree larger than or equal to two, except for one which has out-degree zero. That vertex is referred to as the output vertex and corresponds to the output of the multiplier.

The introduction of vertex reduced graphs makes Observation 1 obsolete, as no vertex reduced graph can have a vertex with out-degree one.

2.4. Classification

To simplify the search further, we will divide the graphs into classes depending on the structure of the graphs, and, thus, on how the new coefficients are derived from lower cost coefficients.

2.4.1. Additive graphs

The first method to obtain new coefficient sets is simply to add the fundamental of the output vertices for two graphs. The resulting adder cost is the sum of the adder costs of the two subgraphs plus one for the summation adder. The structure of an additive graph is shown in Figure 6a.

To identify additive graphs, we will differentiate between two cases. The first case is when a cost-$C-1$ number is added to a cost-0 number to form an odd cost-C number of the form

$$a_C = \begin{cases} \left|a_{C-1} \pm 2^b\right| \\ 2^b a_{C-1} \pm 1, \end{cases} \tag{12}$$

where a_{C-1} is an odd fundamental of the cost-$C-1$ subgraph. This corresponds to an edge from the input vertex to the output vertex as for, e.g., cost-4 graph 3 in Figure 5. If instead the graph formulation in Section 2.1 is considered, the following observation can be made.

Observation 4. *If there is an edge from the input vertex such that its terminal vertex only has one path to the output vertex, the resulting coefficients can be expressed as in (12).*

Motivation. By using addition reordering, it is possible to move the edge to the output edge (as each edge along the path will have out-degree one).

The second case is when both subgraphs have a cost higher than zero. However, if one subgraph is a cost-1 graph, the total graph will belong to the first case. This is also true if one of the subgraphs belongs to the first case of additive graphs. The coefficients generated by the second case of additive graphs can be expressed as

$$a_C = \begin{cases} \left|a_i \pm 2^b a_j\right| \\ \left|2^b a_i \pm a_j\right| \end{cases}, \tag{13}$$

where a_i and a_j are odd fundamentals of a cost-i and cost-j graph, respectively. The only graph of this type up to adder cost five is cost-5 graph 12 in Figure 5, which is formed from two cost-2 graph 2 subgraphs.

2.4.2. Multiplicative graphs

By cascading two graphs, the resulting coefficient set will be a product of the fundamentals at the output vertex of the two subgraphs. The resulting adder cost

is equal to the sum of the adder costs of the two subgraphs. The structure of a multiplicative graph is shown in Figure 6b. All multiplicative graphs can be identified using the following observation. The odd fundamentals will be of the form

$$a_C = a_{C-m} a_m, \tag{14}$$

where a_i is the odd fundamental of a cost-i graph. For example, the cost-2 graph 2 in Figure 5 is a multiplicative graph formed from two cascaded cost-1 graph 1 subgraphs.

Observation 5. *If a graph has an articulation point, the graph is multiplicative.*

Motivation. The articulation point will be the output vertex of the first subgraph and the input vertex of the second subgraph. The fundamental of the output vertex will thus be the product of the two fundamentals at the output of the two subgraphs.

2.4.3. Leapfrog graphs

The previous classes of graphs have only utilized simple combinations of the fundamentals of the output vertices. This works for up to three adders, but with four or more adders there exist graphs that utilize internal fundamentals. This can also be seen as a connection of more than two subgraphs.

One class that covers all remaining graphs up to five adders is what we will refer to as leapfrog graphs. As the name implies, there are subgraphs "leaping" over other subgraphs. Figures 6c and d show the structures of a leapfrog-2 and leapfrog-3 graph, respectively. The number n in leapfrog-n denotes the number of subgraph "leaps".

It is necessary for the left-hand and right-hand graphs in Figures 6c and d to have adder cost one or higher. If not, the graph will be reduced to another graph class (possibly a lower-order leapfrog graph). For example, cost-4 graph 11 in Figure 5 is a leapfrog-2 graph, where the right-most and left-most subgraphs in Figure 6c are cost-1 graph 1, and the other three subgraphs are cost-0 graph 1. The only leapfrog-3 graph in Figure 5 is cost-5 graph 34, formed by left-most and right-most cost-1 graph 1 subgraphs in Figure 6d, while the other five subgraphs are cost-0 graph 1.

2.5. Adder depth and coefficient range

It can be shown that several conclusions can be drawn from the number of nonzero digits of the coefficient's CSD representation, despite the fact that the CSD representation only corresponds to the first of the graphs for each cost in Figure 5. Of most importance is that the minimum number of cascaded adders, the adder depth, for a coefficient is determined by the number of nonzero digits in the CSD

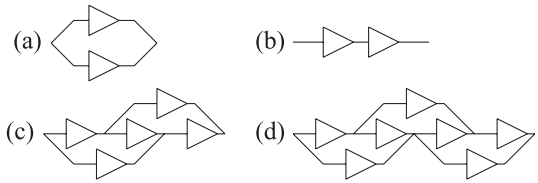


Figure 6. Graph classes covering all graphs up to cost 5. (a) additive, (b) multiplicative, (c) leapfrog-2, and (d) leapfrog-3.

representation of a coefficient. This is a trivial result if a CSD-type multiplier is used, but will be shown to be valid even for the general graph multiplier case. The adder depth is a good estimate of the power consumption, so shallower graphs are preferred [4], [5], [18].

Observation 6. *The sum of two coefficients with c_1 and c_2 nonzero digits, respectively, has at most $c_1 + c_2$ nonzero digits.*

Motivation. By dividing the addition into subsequent additions of one nonzero digit at a time, it is straightforward to see that the number of nonzero digits increases with at most one in each iteration. For more details see [11].

Observation 7. *A multiplier graph with n_1 adders has an adder depth of at least $\lceil \log_2(n_1) \rceil$ and at most n_1.*

Motivation. The upper bound follows directly from arranging the adders in a cascaded structure, e.g., cost-4 graph 8 in Figure 5, and the lower bound follows from arranging the adders in a binary tree type structure.

Observation 8. *A multiplier graph with n_1 adders can generate coefficients with at most 2^{n_1} nonzero digits.*

Motivation. The input of the multiplier graph corresponds to a coefficient with one nonzero digit. After one adder there are at most two nonzero digits according to Observation 6. Cascading an adder to that output yields at most four nonzero digits, and so on. The graphs with the maximum number of nonzero digits are cost-1 graph 1, cost-2 graph 2, cost-3 graph 4, cost-4 graph 8, and cost-5 graph 21 shown in Figure 5.

Observation 9. *A multiplier graph with adder depth d_1 can generate coefficients with at most 2^{d_1} nonzero digits.*

Motivation. The number of nonzero digits at adder depth d_1 is at most twice the number of nonzero digits for a fundamental at adder depth $d_1 - 1$. The number of nonzero digits at adder depth zero (at the input) is one. Hence, the maximum number of nonzero digits at adder depth d_1 is 2^{d_1}.

Observation 10. *A number with a CSD representation containing c_1 nonzero digits has a minimum adder depth of $\lceil \log_2(c_1) \rceil$.*

Motivation. From Observation 9 we have that $c_1 \leq 2^{d_1}$, hence $d_1 \geq \log_2(c_1)$, and as d_1 is an integer, $d_1 \geq \lceil \log_2(c_1) \rceil$.

3. Heuristic approach

Several heuristic approaches to the constant coefficient problem have previously been proposed. Aiming primarily at software compilers, Bernstein proposed an algorithm in 1986 [1]. Bull and Horrocks proposed an algorithm for the implementation of FIR filters (several coefficients), which can also be used for the single coefficient case 2. This was later improved [6], and the improved version is here denoted BHM (Bull-Horrocks modified). Lefèvre proposed an algorithm based on subexpression sharing, i.e., finding common patterns in the representation of the coefficient [16]. In this section an algorithm based on subexpression sharing using a signed-digit representation of the coefficients is proposed. The signed-digit representation possibly uses extra nonzero digits compared with the minimum to reduce the number of adders. A discussion on the maximum number of extra digits to consider is also provided.

3.1. Proposed heuristic approach

Subexpression sharing was introduced in [14] as a method to utilize redundancy between FIR filter coefficients to reduce the number of required adders. There exist many heuristics based on subexpression sharing; see for example, [14], [18], [21]. These algorithms usually represent each coefficient using CSD representation, and subexpressions are sought among the coefficients. However, the results are representation dependent, and in [20] it was shown that if other MSD representations are used, better results may be found.

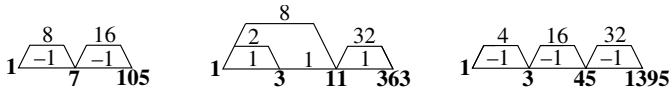
Most work considering subexpression sharing has focused on the design of FIR filters. However, subexpression sharing can also be applied to the design of single coefficient multipliers. In [10] an algorithm that generates all SD representations of an integer using a specified number k of nonzero digits above that used by CSD was introduced. So, if $k = 0$, then all MSD representations are produced. However, it can be shown that when $k > 0$, multipliers can be designed using fewer adders.

In the proposed approach, all SD representations using at most k nonzero extra digits are generated using the algorithm in [10]. Then a subexpression sharing algorithm, in this case the algorithm in [14], is applied to all SD representations. The multiplier design using the fewest adders is then selected. The proposed approach using k extra nonzero digits is denoted $H(k)$.

The CSD and the best SD representations for three different coefficients are shown in Table 1, where the resulting number of required adders is also included. The smallest integer for which another MSD representation gives a better result

Table 1. Results using the proposed heuristic compared with applying subexpression sharing to the CSD representation. k is the number of extra nonzero digits

	CSD		Best SD		
Coefficient	Representation	Adders	k	Representation	Adders
105	$1010\bar{1}001$	3	0	10011001	2
363	$10\bar{1}00\bar{1}0\bar{1}0\bar{1}$	4	1	101101011	2
1395	$10\bar{1}0\bar{1}00\bar{1}010\bar{1}$	4	2	$10\bar{1}0\bar{1}\bar{1}11010\bar{1}$	3



(a) (b) (c)

Figure 7. Graph representation of single coefficient multipliers using subexpression sharing for the coefficients (a) 105, (b) 363, and (c) 1395.

than CSD is 105. The coefficient 363 is the smallest integer where a representation with one extra nonzero digit, i.e., $k = 1$, gives a better result than any MSD representation. The smallest integer where a representation with two extra nonzero digits, i.e., $k = 2$, gives a better result than any representation with up to one extra nonzero digit is 1395. These three coefficients can be realized as shown in Figure 7.

3.2. How far to search

It would appear from the examples in the previous section that simply allowing more digits in the SD representation increases the likelihood of finding lower-cost designs. This is only true to a limited degree. First, there are some cases that are not helped by adding extra digits. For instance, a multiplication by 805 can be realized with 3 adders using the optimal approach previously discussed in Section 2, but $H(k)$ can only find 4-adder solutions. The optimal approach finds a 3-adder graph which is the cascade of a 1-adder graph, with value 5, and a 2-adder graph, with value 161. This is the only optimal solution. The reason that subexpression elimination does not work in this case is illustrated in Figure 8. Both 161 and 5 each have only one MSD representation, which are also the binary representations. Multiplying 161 by 5, however, produces a clash (ringed) that means that the 161 SD pattern does not appear in the 805 SD representation. Therefore, the algorithm cannot be expected to find it. The 5 pattern does appear, but only once, and hence does not indicate any redundancy. As more digits are allowed, by increasing k, the new representations always produce similar clashes, and, hence, in this case, $H(k)$ will never find the optimum result. Also note that

			(1)	0	1	0	0	0	0	1	$161 \times 1 +$
	1	0	(1)	0	0	0	0	1	0	0	$161 \times 4 =$
	1	1	0	0	1	0	0	1	0	1	805 or
1	0	$\overline{1}$	0	0	1	0	0	1	0	1	805_{CSD}

Figure 8. Generating 805 as 161 (in its CSD and binary form) $\times$ 5 (also both CSD and binary) to produce 805 in binary. The only other MSD representation is also its CSD representation, which is also shown. The clash that breaks the patterns is ringed.

if a 3-adder solution is sought, the maximum number of nonzero digits is eight, according to Observation 8. Hence, all representations with more than three extra nonzero digits (as there are five in the MSD representation of 805) require at least four adders.

Furthermore, there is an infinite number of SD representations for a given integer. This is easy to realize by noting that any 1 can be replaced by $1\overline{1}$ ad infinitum. Therefore, it is sensible to seek a limit to the number of digits we add before recognizing no further saving is likely to be made.

Observation 11. *For a given integer n there is a limit to the number of new representations that are produced by each increment of k, and this saturation limit is*

$$S(n) = \begin{cases} 1 & n = 1 \\ S(n/2) & n \text{ even} \\ S(n+1) + S(n-1) & n \text{ odd} \end{cases} \tag{15}$$

Motivation. The general principle can be demonstrated by referring to the algorithm used in [10] to generate all SD representations. It uses a tree such as that in Figure 9. An odd integer can only have 1 or -1 in the least significant position (a 0 would mean the number is even). The algorithm assigns these two possibilities to the two sides of the tree, subtracting 1 or -1 from the input, and associating the resulting remainders with the corresponding branch. The branch values, i.e., the remainders, are both even and one is divisible by a power of two greater than 2^1. For each branch, two subbranches are then formed by adding and subtracting the highest power of two factor of the branch value, leaving remainders with even higher powers of two as factors. This process proceeds until the remainder is itself a power of two. From such a remainder, 2^a, the process produces remainders of 0 (a solution, i.e., a new SD representation) and 2^{a+1}, to which an identical process occurs. In other words, once all remainders are powers of two, which will be true for values $k \geq k_{sat}$, each produces one representation and one power of two remainder. This means that the number of new solutions for each increment of k remains constant, and that constant (the saturation limit, $S(n)$) is the number of paths down the tree that terminate with a power of two remainder (indicated with * in Figure 9), but have no power of two remainders in the path.

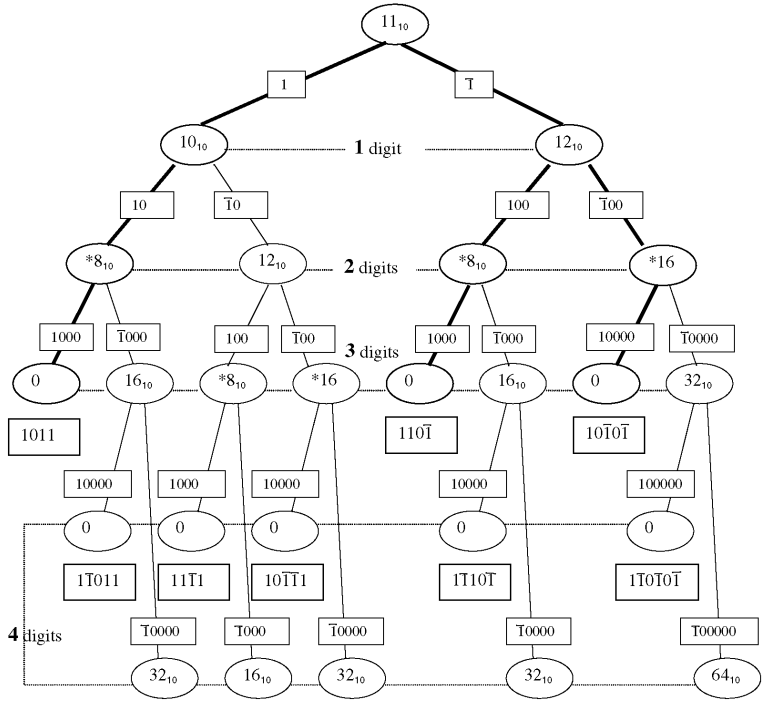


Figure 9. The design tree for integer 11_{10}. The representations with 3 ($k = 0$) and 4 ($k = 1$) nonzero digits are shown. First occurrences of power of two remainders are marked with an asterisk.

Evaluating the saturation limit is straightforward. For $n = 1$, we observe that the possible representations are 1, $1\bar{1}$, $1\bar{1}\bar{1}$, etc., i.e., there is only one representation per value of k (not surprising, because we have started with a power of two remainder). So $S(n) = 1$. To transform a tree for an odd n into a tree for $2n$, we simply shift all the branch labels (see Figure 9) left one bit, giving an identical representation shifted one bit, and identical numbers and locations of power of two remainders, i.e., $S(n) = S(n/2)$ for even n. The tree for odd n can be seen (again see Figure 9) to be a combination of the trees for $n + 1$ and $n - 1$, because these remainders have trees of their own that are generated in the same way as for n. Hence, the number of power of two paths for n is the sum of the number of paths for $n + 1$ and $n - 1$, or $S(n) = S(n + 1) + S(n - 1)$.

Observation 12. *All of the new representations in the saturated state (i.e., for $k \geq k_{sat}$) for $k = m$ are simply the representations for $k = m - 1$ with the leading* 01 *replaced by* $1\bar{1}$ *(for positive n).*

Motivation. Once a power of two remainder is reached, it has a solution representation starting with 01. The power of two remainder passes the 01 to the next remainder, which adds 10 for another solution.

Observation 13. *The total number of representations of positive n not having a leading* $1\overline{1}$ *is* $S(n)$.

Motivation. Following Observation 12, to get to a point in the tree where $1\overline{1}$ are leading digits in the representation, at least two power of two remainders must be encountered; the first adds 01 and the second 10. Therefore, each path that leads to a power of two remainder without passing through another power of two remainder must not have leading 11. The path to that remainder then continues and leads to a sequence of 11-type solutions.

The preceding observations mean that it is likely that there is little use in searching for better solutions for $k \geq k_{sat}$, because new patterns are not being generated.

Consider the generation of SD representations in Figure 9. From (15) we have that $S(11) = 5$. Studying the different representations derived in Figure 9, we see that there are exactly five representations, as predicted by Observation 13, that do not start with $1\overline{1}$, namely 1011, $110\overline{1}$, $10\overline{1}0\overline{1}$, $11\overline{1}1$, and $10\overline{1}\overline{1}1$. All remaining representations start with $1\overline{1}$, and according to Observation 12, all subsequent representations also will, as is easily verified.

In [6], theorems 4 through 6 showed that there were limits to the vertex and edge values that need to be searched in order to guarantee that all possible graphs had been covered when using the graphical method of describing multipliers. Theorems of the same type cannot be used here. However, one simple insight that may help reduce runtime is that when one extra digit is used to represent a number, that new representation must create two new eliminatable subexpressions in order for there to be a reduction in the number of adders in the final design (i.e., eliminating the new digit plus one other). For k digits, $k + 1$ subexpressions must be created.

4. Adder-bit cost

To differentiate designs of equal adder cost, a more detailed cost, measure can be used. In [6] a cost corresponding to the required number of full adders was introduced. This was referred to as the adder-bit cost. Here we refine this method. It is assumed that ripple-carry adders are used. A discussion about other adder types is provided at the end of this section.

As can be seen in Figure 10a, each fundamental, f_i, of a graph has the value

$$f_i = e_j f_j + e_k f_k. \tag{16}$$

As discussed earlier, even coefficients and fraction numbers can be obtained by using a shift operation at the output of the graph multiplier. Hence, only the case where all fundamentals, f_i, are odd integers is considered.

There are two possibilities associated with the edge values, e_j and e_k. In the

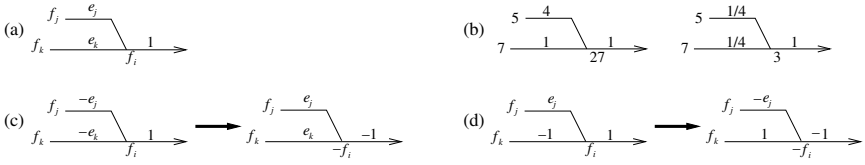


Figure 10. (a) General graph multiplier segment. (b) Example of the normal and special case, respectively. (c) Transformation to eliminate the case when both edge values are negative. (d) Transformation to eliminate the use of half adders.

normal case, the value at one of the initial vertices is left shifted at least once while the significance of the other value is unchanged. The shift operation is, for simplicity, always associated with the initial vertex f_j. There is also a special case when the magnitude of both edge values is less than one. In order to obtain an odd fundamental, the edge values must then be of equal significance. An example of each of the two cases is shown in Figure 10b. Hence, we have

$$|e_j| > 1,\ |e_k| = 1 \text{ (normal case)} \quad or \quad |e_j| = |e_k| < 1 \text{ (special case)}. \tag{17}$$

For simplicity, we eliminate the case when both edge values are negative. A transformation that can be used to avoid this is shown in Figure 10c. The requirement for the signs of the edge values is stated as

$$(sign\{e_j\}, sign\{e_k\}) \in \{(1, 1), (-1, 1), (1, -1)\}. \tag{18}$$

The number of output bits, W_{out}, from the add operation associated with the fundamental f_i is

$$W_{out} = W_0 + \lceil \log_2(|f_i|) \rceil, \tag{19}$$

where W_0 is the wordlength of the graph multiplier input. To be an actual add operation, the number of left shifts must be less than the wordlength of the edge without any shift operation. Hence, for the normal case, we have

$$\log_2(|e_j|) < W_0 + \lceil \log_2(|f_i|) \rceil. \tag{20}$$

The required number of full adders, $n_{i,FA}$, to perform the add operation is

$$n_{i,FA} = W_{out} - \log_2(|e_j|). \tag{21}$$

This means that the number of full adders, in the normal case, is W_{out} minus the number of shifts associated with the edge value e_j. In the special case, on the other hand, the magnitude of the edge values is less than one, which gives more full adders than W_{out}. This occurs because, although the sum bits corresponding to the fractional bits are known to be zero, the carry bit of significance one is computed using full adders.

If the edge without any shift operation, in the normal case, is negative, i.e., $e_k = -1$, half adders are required to compute the least significant output bits.

Hence, we have

$$n_{i,HA} = \begin{cases} \log_2(e_j) & e_j > 1,\ e_k = -1 \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

The use of half adders can be eliminated by the transformation shown in Figure 10d. A condition for this transformation usually is that it must be possible to compensate for the sign of the multiplier output in the subsequent operations of the algorithm, which is the case in most applications.

Note that there are several cases where a complete full adder is not required. For example, the full adder corresponding to the most significant output bit does not need to produce a carry bit. As mentioned previously, the sum bits corresponding to the fractional bits are known to be zero in the special case, hence, only the carry bit is required. Another example is additions where the full adder of least significance always has carry input equal to zero, which means that a half adder would be sufficient. These kind of design improvements have not been considered here, hence, all full adders are assumed to be complete.

If an accelerated adder structure is used, clearly the number of full and half adder cells will not be a relevant measure. However, the results in (21) and (22) can still be used as a start to form suitable complexity measures. Also, for most adder structures, the proposed transformation should be advantageous as the part of the adder corresponding to half adder cells will be removed. For adder structures where the area complexity increases faster than linearly in the wordlength, which is the case for most accelerated adder structures, the relative savings are expected to be larger than for ripple-carry adders. This of course depends on how the part corresponding to half adder cell is realized.

5. Results

Results will be presented for the number of possible graphs, classification of graphs, and average adder cost. Furthermore, results on the adder depth, coefficient range, and power comparison will be presented for the multiplier graphs in Figure 5.

5.1. Possible graphs

The number of possible graphs has been examined up to adder cost 7. As expected [6], the number of possible graphs increases rapidly. Further, the number of possible reduced graphs has also been examined. The results are shown in Table 2.

Table 2. Number of possible graphs for different costs without transposition symmetry considerations

Adder cost	Fully specified graphs	Vertex reduced graphs	Type		
			Additive	Multiplicative	Leapfrog
0	1	1	1	-	-
1	1	1	1	-	-
2	2	2	1	1	-
3	7	5	2	3	-
4	32	15	5	9	1
5	193	54	16	30	8
6	1444	227			
7	13228	1162			

5.2. Classification

Using the classifications described in Section 2.4, all adders up to adder cost five have been classified. The results are shown in Table 2. To search all possible graphs up to cost-5 the cases in Table 3 must be investigated. These 17 cases cover all 236 possible fully specified graphs derivable from Figure 5. For leapfrog graphs the coefficient sets in Table 3 correspond to the multiplier graphs in Figures 6c and d from left to right.

5.3. Average adder cost

The average adder cost for various algorithms with coefficients up to 19 bits is shown in Figure 11. The algorithms are CSD, BBB (best of Bernstein and BHM), the heuristic approach of Section 3.1 with zero, one, or two extra nonzero digits, and graph multipliers with and without leapfrog graphs. The search process for leapfrog graphs is costly compared with the additive and multiplicative graphs. We have therefore included results when the leapfrog graphs are excluded. The results without leapfrog graphs coincide with those obtained using the method in [17]. Coefficients up to 19 bits can be expressed using at most five adders for graph multipliers, whereas up to nine adders may be required for CSD multipliers.

The average number of extra adders required by BBB, the proposed heuristic method, and graph multipliers without leapfrog graphs compared to the optimal approach is shown in Figure 12. For 19-bit coefficients the average increase in adders is just above 1% for the proposed heuristic method using up to two extra nonzero digits.

For filter design it is often desirable to keep the number of adders in a single coefficient to a given number. The ratio of coefficients realizable with a given

Table 3. Graph types that must be examined to obtain all possible coefficients with adder cost up to five

Adder cost	Graph type	Coefficient set 1	Coefficient set 2	Covered graphs
1	Additive	$H_{0,B}$	$H_{0,B}$	1
2	Additive	$H_{1,B}$	$H_{0,B}$	1
2	Multiplicative	$H_{1,B}$	$H_{1,B}$	2
3	Additive	$H_{2,B}$	$H_{0,B}$	1, 2
3	Multiplicative	$H_{2,B}$	$H_{1,B}$	3, 4
4	Additive	$H_{3,B}$	$H_{0,B}$	1–4
4	Multiplicative	$H_{3,B}$	$H_{1,B}$	5–9
4	Multiplicative	$H_{2,B}$	$H_{2,B}$	8–10
4	Leapfrog-2	$H_{1,B}$, $H_{0,B}$, $H_{0,B}$, $H_{0,B}$, $H_{1,B}$		11
5	Additive	$H_{4,B}$	$H_{0,B}$	1–11
5	Additive	$H_{2,B}$	$H_{2,B}$	12
5	Multiplicative	$H_{4,B}$	$H_{1,B}$	13–26
5	Multiplicative	$H_{3,B}$	$H_{2,B}$	21–29
5	Leapfrog-2	$H_{1,B}$, $H_{0,B}$, $H_{1,B}$, $H_{0,B}$, $H_{1,B}$		30
5	Leapfrog-2	$H_{2,B}$, $H_{0,B}$, $H_{0,B}$, $H_{0,B}$, $H_{1,B}$		31, 32
5	Leapfrog-2	$H_{1,B}$, $H_{1,B}$, $H_{0,B}$, $H_{0,B}$, $H_{1,B}$		33
5	Leapfrog-3	$H_{1,B}$, $H_{0,B}$, $H_{0,B}$, $H_{0,B}$, $H_{0,B}$, $H_{0,B}$, $H_{1,B}$		34

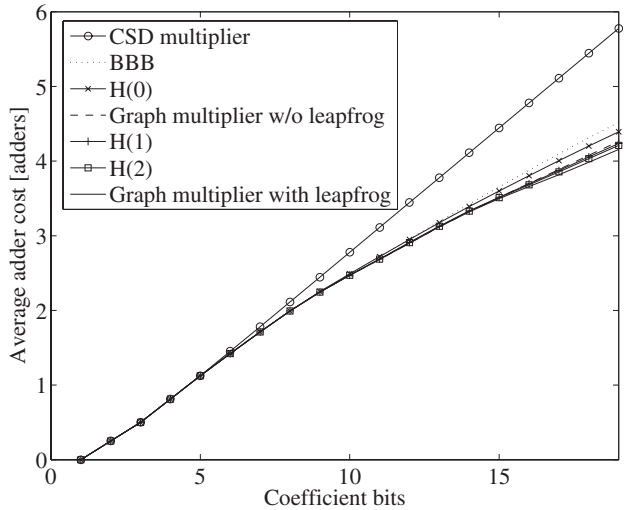


Figure 11. Average adder cost for CSD, BBB, the proposed heuristic method with up to zero, one, and two extra digits, and graph multipliers with and without leapfrog graphs. For BBB the numbers are averages over 5000 values.

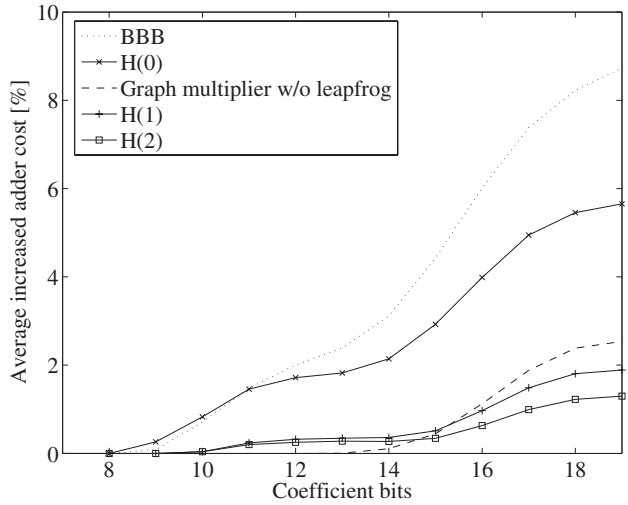


Figure 12. Average increase in adder cost using BBB, the proposed heuristic method with up to zero, one, and two extra nonzero digits, and graph multipliers without leapfrog graphs compared with the optimal approach. For BBB the numbers are averages over 5000 values.

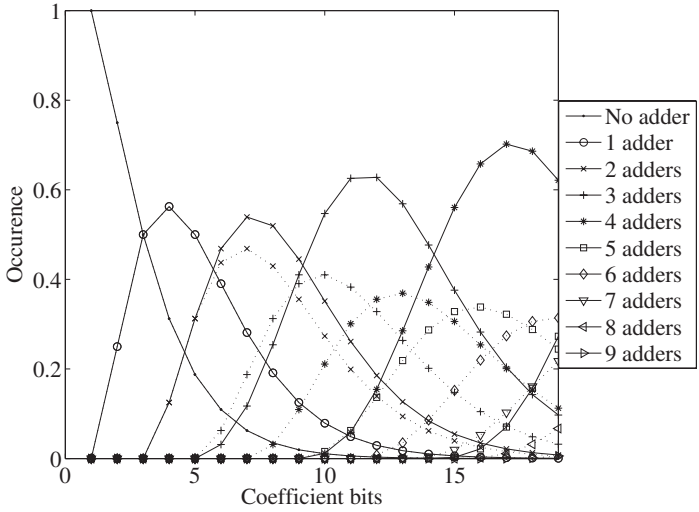


Figure 13. Frequency of occurrence for adder costs as a function of the coefficient wordlength using CSD representation (dotted) and the optimal approach (solid).

number of adders is shown in Figure 13 for the optimal approach and CSD representation.

5.4. Adder depth and coefficient range

Using the observations presented in Section 2.5, it is possible to obtain bounds on the maximum number of nonzero digits and minimum adder depth for any coefficients generated using a given graph. These results are presented in Table 4. The adder depths within parentheses denote the minimum adder depth for the transposed graph.

5.5. Power estimation

Traditionally, the number of adders has been used as an estimate, not only for the complexity, but also for the power consumption. However, the power consumption depends on the switching activity, so different multiplier graphs would have different switching activities. In [4] a measure called GP count was introduced for estimating the glitching/switching activity. This was extended in [5] to a more detailed estimate called GP score. However, GP count works on the topology of the network, whereas GP score takes into account the edge weights. Hence, for comparing different topologies, GP count is a more suitable choice. Furthermore, in GP score there are certain parameters that are technology dependent.

Another factor that affects the power consumption is the capacitive load of a node. For high capacitive loads, the driving current, and therefore the power consumption, must be increased. Hence, if a node in a multiplier graph has a high out-degree, the corresponding adder cells must have a higher driving capability. A similar situation exists for the input node. A high input load leads to the driving circuits having a higher power consumption, even though this power is not consumed within the actual multiplier.

In Table 4, the minimum and maximum GP counts are shown for all multiplier graphs in Figure 5, as well as the input load and maximum internal fan-out (out-degree). These results are based on all possible realizations of a given vertex reduced graph. Loads and fan-outs in parentheses correspond to the transposed multiplier graph.

From Table 4 it can be seen that different realizations of the same vertex reduced adder graph may differ significantly in terms of GP count. Hence, it is clear that just counting the number of adders is not a satisfactory way to estimate power consumption. This issue is further elaborated in the example below.

6. Example

The ideas discussed in previous sections will now be illustrated in an example. Consider the coefficient 1717, which can be computed as

$$1717 = 11 \cdot 63 + 1024 = (1 + 2 + 8) \cdot (64 - 1) + 1024. \tag{23}$$

Table 4. Characteristics of the multiplier graphs in Figure 5

Adder cost	Graph number	Possible realizations	Max. nonzero digits	Min. adder depth	Min. GP count	Max. GP count	Input load	Max. internal fan-out
1	1	1	2	1	1	1	2	-
2	1	1	3	2	3	3	3	1
	2	1	4	2	4	4	2	2
3	1	2	4	2	5	6	4	1
	2	2	5	3	7	8	3	2
	3	2	6	3	8	9	3 (2)	2 (3)
	4	1	8	3	11	11	2	2
4	1	3	5	3	8	10	5	1
	2	6	6	3	10	13	4	2
	3	6	7	4 (3)	12	15	4 (3)	2 (3)
	4	2	9	4	16	19	3	2
	5	4	8	3	12	16	4 (2)	2 (4)
	6	1	12	4	20	20	2	3
	7	4	10	4	16	20	3 (2)	2 (3)
	8	1	16	4	26	26	2	2
	9	2	12	4	19	22	3 (2)	2 (3)
	10	1	9	4	16	16	3	3
	11	2	8	4	14	15	3	2
5	1	6	6	3	11	15	6	1
	2	2	13	5	27	32	3	3
	3	13	11	5 (4)	22	20	4 (3)	2 (3)
	4	9	9	4	18	23	4	2
	5	2	17	5	35	42	3	2
	6	15	7	3	13	19	5	2
	7	21	8	4	15	22	5 (4)	2 (3)
	8	13	9	4	17	24	5 (3)	2 (4)
	9	6	10	4	19	28	4	2
	10	6	13	5 (4)	26	34	4 (3)	2 (3)
	11	4	10	4	20	25	4	3
	12	4	8	3	15	17	4	2
	13	4	16	5	29	35	3 (2)	2 (3)
	14	4	18	5	33	43	3 (2)	2 (3)
	15	6	14	5 (4)	25	35	4 (2)	2 (3)
	16	12	12	4	21	31	4 (2)	2 (4)
	17	2	16	4	29	31	2	4
	18	2	20	5	37	39	2	3
	19	6	12	4 (5)	25	33	3 (2)	3 (4)
	20	6	10	4	17	25	5 (2)	2 (5)
	21	1	32	5	57	57	2	2
	22	4	20	5	35	45	3 (2)	2 (3)
	23	4	16	4	27	37	4 (2)	2 (4)
	24	2	18	5	33	37	3 (2)	3
	25	2	24	5	42	49	3 (2)	2 (3)
	26	2	24	5	43	45	2	3
	27	4	15	5	30	33	3	3
	28	4	12	4	23	27	4 (3)	3 (4)
	29	1	18	5	36	36	3	3
	30	4	12	5	24	28	3	3
	31	6	11	5 (4)	22	26	4 (3)	2 (3)
	32	4	14	5	28	32	3	2
	33	10	10	4	19	24	4 (3)	2 (3)
	34	2	13	5	26	28	3	2

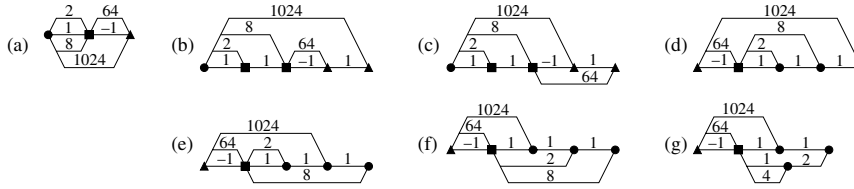


Figure 14. (a) Vertex reduced graph for the coefficient 1717. (b)–(g) Fully specified graphs for the coefficient 1717. The node symbols (●■▲) indicate the relation between nodes in the vertex reduced graph and nodes in the fully specified graphs.

Table 5. Adder depth, GP count, input load, and maximum internal fan-out for the multiplier graphs in Figures 14b–g

Realization	Adder depth	GP count	Input load	Maximum internal fan-out
Figure 14b	4	14	4	2
Figure 14c	4	12	4	2
Figure 14d	4	15	3	3
Figure 14e	4	14	3	3
Figure 14f	4	13	3	3
Figure 14g	3	12	3	3

The corresponding vertex reduced graph is shown in Figure 14a. Note that there exist other vertex reduced graphs that can be used to realize the coefficient 1717, but none with lower adder cost than four. There also exist other possibilities using the same vertex reduced graph, for example

$$1717 = 13 \cdot 33 \cdot 4 + 1 = (1 + 4 + 8) \cdot (32 + 1) * 4 + 1. \tag{24}$$

However, in this example we will use the design first suggested and compare the six associated fully specified graphs shown in Figures 14b–g. Note that the realizations in Figures 14d–g correspond to the transposed vertex reduced graph. For the graphs in Figures 14b–d and f it is possible to reorder the additions corresponding to the 2 and 8 weights. However, this will always lead to an intermediate result with longer wordlength, so it is not considered here.

The realization in Figure 14g has the lowest adder depth. The input load is higher for the nontransposed realizations, whereas the maximum internal fan-out is higher for the transposed graphs. The adder depth, GP count, input load, and maximum internal fan-out for the different realizations are shown in Table 5. The difference in GP count is too small to give a reliable estimate about the relative power consumption. However, it indicates which realizations are more likely to be better. From the results in Table 5 it is not obvious which of the realizations should be selected.

Table 6. Extra fundamental values and the associated number of full adders required in addition to the wordlength of the input, W_0, for the multiplier graphs in Figures 14b–g. The number of half adders are in parentheses

	Fund. 1		Fund. 2		Fund. 3		Output	Total
Realization	Value	N_{FA}	Value	N_{FA}	Value	N_{FA}	N_{FA}	N_{FA}
Figure 14b	3	1	11	1	693	4 (6)	1	7 (6)
Figure 14c	3	1	11	1	1013	0 (10)	5	7 (10)
Figure 14d	63	0 (6)	189	7	693	7	1	15 (6)
Figure 14e	63	0 (6)	189	7	1213	1	8	16 (6)
Figure 14f	63	0 (6)	1087	1	1213	10	8	19 (6)
Figure 14g	63	0 (6)	1087	1	315	7	10	18 (6)

The final cost measure to be studied is the adder-bit cost. The results are shown in Table 6, where the associated extra fundamental values also are given. If the transformation in Figure 10d is applied, the use of half adders can be eliminated for all realizations, resulting in the coefficient -1717. Hence, the cost in half adders will be ignored. To obtain the total number of full adders, the input wordlength, W_0, should be included for each adder. For example, the total number of full adders, $n_{tot,FA}$, for the realization in Figure 14b is

$$n_{tot,FA} = (W_0 + 1) + (W_0 + 1) + (W_0 + 4) + (W_0 + 1) = 4W_0 + 7. \quad (25)$$

Assume that the input wordlength, W_0, is 16 bits. If the realizations in Figure 14b or c are used instead of that in Figure 14g, which has the lowest adder depth, the adder-bit cost is decreased by more than 13%. When the results from both Table 5 and Table 6 are considered, the realization in Figure 14c seems to be the best one to select as it has minimum GP count as well as a minimum number of full adder cells.

7. Conclusions

In this work, the realization of constant coefficient multiplication was discussed. Both an optimal and a heuristic approach were presented to find realizations with few additions. Results showed that, by using the proposed approaches, on average 25% of the required additions can be saved compared with the commonly used canonic signed-digit representation. Furthermore, some theoretical limits on the number of additions and number of cascaded additions were introduced. The previously proposed optimal approach was simplified, and, hence, its execution time can be greatly reduced.

From an implementation point of view, a power estimation measure was applied to differentiate between several similar realizations. Exact equations for the

number of required full and half adder cells were also given, and it was shown that if it is allowable to negate the coefficient value, all half adder cells can be avoided with no increase in full adder cells. For most DSP algorithms, this is possible as it can be compensated for by replacing a subsequent addition by a subtraction or vice versa.

It was shown that the proposed heuristic method in most cases obtains as few adders as the optimal approach. However, taking the power estimation measures into account, the optimal approach provides numerous different solutions that are optimal in the number of adders. From these the best solution based on other criteria can be selected.

Acknowledgments

The authors are grateful to the anonymous reviewers for their helpful comments and suggestions.

References

[1] R. Bernstein, Multiplication by integer constants, *Softw.–Pract. Exp.*, vol. 16, pp. 641–652, July 1986.

[2] D. R. Bull and D. H. Horrocks, Primitive operator digital filters, *IEE Proc. G*, vol. 138, pp. 401–412, June 1991.

[3] D. Chen, T. Aoki, N. Homma, T. Terasaki, and T. Higuchi, Graph-based evolutionary design of arithmetic circuits, *IEEE Trans. Evolutionary Computation*, vol. 6, no. 1, pp. 86–100, Feb. 2002.

[4] S. S. Demirsoy, A. G. Dempster, and I. Kale, Transition analysis on FPGA for multiplier-block based FIR filter structures, in *Proc. IEEE Int. Conf. Elec. Circuits Syst.*, Lebanon, Dec. 2000, pp. 862–865.

[5] S. S. Demirsoy, A. G. Dempster, and I. Kale, Power analysis of multiplier blocks, in *Proc. IEEE Int. Symp. Circuits Syst.*, Phoenix, AZ, May 26–29, 2002, vol. 1, pp. 297–300.

[6] A. G. Dempster and M. D. Macleod, Constant integer multiplication using minimum adders, *IEE Proc. Circuits Devices Syst.*, vol. 141, no. 6, pp. 407–413, Oct. 1994.

[7] A. G. Dempster and M. D. Macleod, General algorithms for reduced-adder integer multiplier design, *Electron. Lett.*, vol. 31, no. 21, pp. 1800–1802, Oct. 1995.

[8] A. G. Dempster and M. D. Macleod, Comments on "Minimum number of adders for implementing a multiplier and its application to the design of multiplierless digital filters," *IEEE Trans. Circuits Syst. II*, vol. 45, no. 2, pp. 242–243, July 1998.

[9] A. G. Dempster and M. D. Macleod, Using all signed-digit representations to design single integer multipliers using subexpression elimination, in *Proc. IEEE Int. Symp. Circuits Syst.*, Vancouver, Canada, May 23–26, 2004, vol. 3, pp. 165–168.

[10] A. G. Dempster and M. D. Macleod, Generation of signed-digit representations for integer multiplication, *IEEE Signal Process. Lett.*, vol. 11, no. 8, pp. 663–665, Aug. 2004.

[11] O. Gustafsson, *Contributions to Low-Complexity Digital Filters*, Linköping Studies in Science and Technology, Dissertations, No. 837, Linköping University, Sweden, Sept. 2003.

[12] O. Gustafsson, A. G. Dempster, and L. Wanhammar, Extended results for minimum-adder

constant integer multipliers, in *Proc. IEEE Int. Symp. Circuits Syst.*, Phoenix, AZ, May 26–29, 2002, vol. 1, pp. 73–76.

[13] O. Gustafsson, H. Ohlsson, and L. Wanhammar, Minimum-adder integer multipliers using carry-save adders, in *Proc. IEEE Int. Symp. Circuits Syst.*, Sydney, Australia, May 6–9, 2001, vol. 2, pp. 709–712.

[14] R. I. Hartley, Subexpression sharing in filters using canonic signed digit multipliers, *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 677–688, Oct. 1996.

[15] K. Johansson, O. Gustafsson, and L. Wanhammar, Low-complexity bit-serial constant-coefficient multipliers, in *Proc. IEEE Int. Symp. Circuits Syst.*, Vancouver, Canada, May 23–26, 2004, vol. 3, pp. 649–652.

[16] V. Lefèvre, Multiplication by an integer constant, Rapport de recherche RR-4192, INRIA, May 2001. (Available: `http://www.vinc17.org/research/papers/rr_mulcst2.pdf`)

[17] D. Li, Minimum number of adders for implementing a multiplier and its application to the design of multiplierless digital filters, *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 453–460, July 1995.

[18] M. Martínez-Peiró, E. Boemo, and L. Wanhammar, Design of high speed multiplierless filters using a nonrecursive signed common subexpression algorithm, *IEEE Trans. Circuits Syst. II*, vol. 49, no. 3, pp. 196–203, Mar. 2002.

[19] A. P. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall, 1989.

[20] I.-C. Park and H.-J. Kang, Digital filter synthesis based on an algorithm to generate all minimal signed digit representations, *IEEE Trans. Computer-Aided Design*, vol. 21, no. 12, pp. 1525–1529, Dec. 2002.

[21] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, A new algorithm for elimination of common subexpressions, *IEEE Trans. Computer-Aided Design Integrated Circuits,* vol. 18, no. 1, pp. 58–68, Jan. 1999.

[22] J. Yli-Kaakinen and T. Saramäki, Design of very low-sensitivity and low-noise recursive filters using a cascade of low order lattice wave digital filters, *IEEE Trans. Circuits Syst. II*, vol. 46, no. 7, pp. 906–914, July 1999.

[23] J. Yli-Kaakinen and T. Saramäki, A systematic algorithm for the design of multiplierless lattice wave digital filters, in *Proc. IEEE Int. Symp. Control, Communications, Signal Processing*, Hammamet, Tunisia, March 21–24, 2004, pp. 393–396.

[24] L. Wanhammar, *DSP Integrated Circuits*, Academic Press, 1999.