Proc. of the WSEAS International Conference on Circuits (ICC), pg. 81-82, July 2009

Optimization of Area under a Delay Constraint in Multiple Constant Multiplications

LEVENT AKSOY
Istanbul Technical University
Istanbul, TURKEY
aksoyl@itu.edu.tr

ECE OLCAY GUNES
Istanbul Technical University
Istanbul, TURKEY
ece.gunes@itu.edu.tr

PAULO FLORES
INESC-ID/IST
Lisbon, PORTUGAL
pff@inesc-id.pt

Abstract: The Multiple Constant Multiplications (MCM), i.e., the multiplication of a variable by a set of constants, has been a central operation and performance bottleneck in many digital signal processing applications such as, image and video processing, digital television, and wireless communications. Since the design of multiplications is expensive in terms of area, delay, and power consumption in hardware, the area-delay optimization of the MCM operation has often been accomplished by using the shift-adds architecture. However, most of the previously proposed algorithms have focused on the optimization of area ignoring the crucial tradeoff between area and delay of the computation. In this paper, we introduce an approximate algorithm that can find near optimal area solutions under the user specified delay constraint. It is shown by the experimental results that the proposed algorithm finds better area-delay solutions than the previously proposed efficient algorithms.

Key–Words: Multiple constant multiplications, area and delay optimization, high-level synthesis, digital FIR filters.

1 Introduction

In several computationally intensive operations, such as Finite Impulse Response (FIR) filters and fast Fourier transforms, the same input is multiplied by a set of coefficients, an operation known as Multiple Constant Multiplications (MCM). These operations are typical in Digital Signal Processing (DSP) applications and hardwired dedicated architectures are the best option for maximum performance and minimum power consumption. However, the design complexity of these applications is dominated by a large number of constant multiplications leading to excessive area, delay, and power consumption even if implemented in a full custom integrated circuit. Since the values of the constants are known beforehand, the constant multiplications can be designed using addition/subtraction and shifting operations in the shift-adds architecture [12]. When the same input is to be multiplied by a set of constant coefficients, significant reductions in hardware can also be obtained by sharing the partial products of the input among the set of multiplications. Since shifts are free in terms of hardware, the MCM problem is defined as finding the minimum number of addition/subtraction operations to implement the constant multiplications. The MCM problem has been proven to be NP-complete in [5].

As a small example, suppose the multiplication of multiple constants 11 and 13 by the variable x. Observe from Figure 1(a) that the multiplierless implementation without partial product sharing requires four operations. However, the sharing of partial product $9x$ in both multiplications reduces the number of required operations to 3 as given in Figure 1(b).

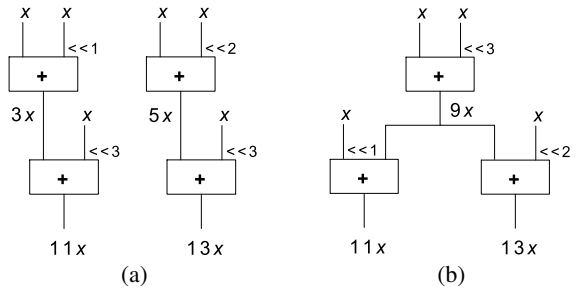


Figure 1: The shift-add implementations of constant multiplications $11x$ and $13x$: (a) without partial product sharing; (b) with partial product sharing.

In the last two decades, many efficient algorithms have been proposed for the optimization of the number of operations in MCM. These methods can be categorized in two classes: the Common Subexpression Elimination (CSE) and the graph-based algorithms. The CSE algorithms basically find common non-zero digit patterns on the representations of the constants. The exact CSE algorithms that formalize the MCM problem as a 0-1 Integer Linear Programming (ILP) problem and find the minimum number of operations solution of the MCM problem by maximizing the partial product sharing have been proposed in [1, 10]. On the other hand, the graph-based algorithms are not restricted to a particular number representation and syn-

thesize the constants iteratively by building a graph. The exact graph-based algorithm that searches all possible partial products in a breadth-first manner has been proposed in [3]. It is shown in [3] that the graph-based algorithm finds superior solutions than the exact CSE algorithm [1], since it considers more possible implementations of a constant. The prominent graph-based heuristics have been introduced in [2, 8, 13].

However, in many designs, particularly in DSP systems, performance is an important and crucial parameter. Hence, circuit area is generally expandable in order to achieve a given performance target. Although the delay is dependent on several implementation issues, such as circuit technology, placement, and routing, the delay in MCM is generally considered as the number of adder-steps, which denotes the maximal number of adders/subtracters in series to produce any constant multiplication [11]. Also, as shown in [7], the number of adder-steps in MCM has significant impacts on the power consumption. Despite the large number of techniques proposed for the optimization of area, there are only a few methods [1, 6, 7, 11] that also consider the delay of the design. However, these algorithms are not equipped with the recently proposed efficient graph-based heuristics [2, 13].

In this work, we introduce an approximate graph-based algorithm designed for the optimization of the number of operations under a delay constraint that is able to compensate the decrease in delay with the increase in area efficiently. To do this, we resort to the graph-based heuristic of [2] designed for the MCM problem that finds better solutions than the prominent graph-based heuristics of [8, 13]. To deal with the delay constraint, in the selection of operations that implement the constant multiplications, we take into account the delay introduced by each operation. The approximate algorithm is applied on randomly generated and FIR filter instances, and it is observed that our algorithm finds better solutions in terms of both area and delay than the previously proposed algorithms.

The rest of the paper is organized as follows: Section 2 gives the background concepts and the problem definitions. The proposed approximate graph-based algorithm is introduced in Section 3 and experimental results are given in Section 4. Finally, Section 5 concludes the paper.

2 Background

In this section, initially, we introduce the basic notations in the graph-based algorithms and the MCM problem. Then, we present the MCM problem under a delay constraint. Finally, we give an overview of the previously proposed graph-based algorithms.

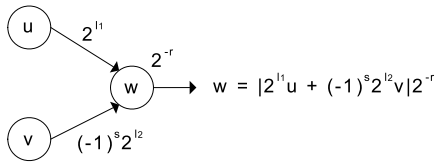


Figure 2: Representation of A-operation in a graph.

2.1 The MCM Problem

In the graph-based algorithms, the main operation, called *A-operation* in [13], is an operation with two integer inputs and one integer output that performs a single addition or a subtraction, and an arbitrary number of shifts. It is defined as follows:

$$w = A(u,v) = |(u \ll l_1) + (-1)^s (v \ll l_2)| \gg r \quad (1)$$
$$= |2^{l_1}u + (-1)^s 2^{l_2} v| 2^{-r}$$

where $l_1, l_2 \geq 0$ are integers denoting left shifts, $r \geq 0$ is an integer indicating the right shift, and $s \in \{0,1\}$ is the sign that denotes the addition/subtraction operation to be performed. The operation that implements a constant can be represented in a graph where vertices are labeled with the constants and edges are labeled with the sign and shifts as given in Figure 2.

In the MCM problem, the complexity of an adder and a subtracter is assumed to be equal in hardware. It is also assumed that the sign of the constant can be adjusted at some part of the design and the shifting operation has no cost, since shifts can be implemented with only wires in hardware. Thus, in the MCM problem, only positive and odd constants are considered. Observe from Eqn. (1) that in the implementation of an odd constant considering odd constants at the inputs, one of the left shifts, l_1 or l_2, is zero and r is zero, or l_1 and l_2 are zero and r is greater than zero. In finding an operation to implement a constant, it is necessary to constrain the left shifts l_1 and l_2 otherwise, a constant can be implemented in infinite ways. As shown in [8], it is sufficient to limit the shifts by the maximum bit-width of the constants to be implemented, i.e., bw, and allowing larger shifts than bw does not improve the solutions obtained with the former limits. In the algorithms of [2, 3, 13], the number of shifts is allowed to be at most $bw+1$.

Thus, the MCM problem can be defined as;

Definition 1 THE MCM PROBLEM. *Given the target set including the positive and odd unrepeated target constants, $T = \{t_1, \ldots, t_m\} \subset \mathbb{N}$, find the smallest ready set $R = \{r_0, r_1, \ldots, r_n\}$ with $T \subset R$ such that $r_0 = 1$ and for all r_k with $1 \leq k \leq m$, there exist r_i, r_j with $0 \leq i, j < k$ and an operation $r_k = A(r_i, r_j)$.*

Hence, the number of operations required to implement the MCM is $|R| - 1$ [13].

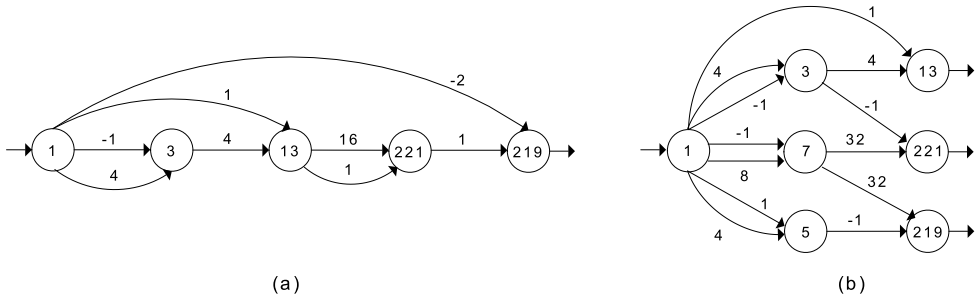


Figure 3: Implementations of the target set $\{3, 13, 219, 221\}$: (a) with the minimum number of operations; (b) with the minimum number of adder-steps.

2.2 The MCM Problem under a Delay Constraint

A single constant represented with n non-zero digits can be implemented in a tree of operations with the minimum latency, i.e., $\lceil \log_2 n \rceil$ adder-steps, or in a chain of operations with the maximum latency, i.e., $n-1$ adder-steps. Obviously, the maximum of the minimum number of adder-steps of each constant in an MCM problem defines the minimum delay of the computation. Hence, for a target set, $T = \{t_1, \ldots, t_m\}$, the minimum delay in MCM [11] is determined as;

$$min\_delay = \max_{t_i} \{\lceil log_2 S(t_i) \rceil\} \quad (2)$$

where $S(t_i)$ is the number of non-zero digits in the Canonical Signed Digit (CSD) representation of the target constant t_i. Note that the CSD representation of a constant includes minimum number of non-zero digits [4].

The minimization of the number of operations under a delay constraint problem can be defined as;

Definition 2 THE MCM PROBLEM UNDER A DELAY CONSTRAINT. *Given a set of target constants and a maximum number of adder-steps, find the minimum number of addition/subtraction operations required to implement the MCM such that the user-specified maximum number of adder-steps is not exceeded.*

As an example, consider a set of target constants $T = \{3, 13, 219, 221\}$. The minimum delay of the MCM implementation is 2 adder-steps as computed by Eqn. (2). Figure 3(a) presents the minimum number of operations solution [3] including 4 operations with 4 adder-steps. However, as can be observed from Figure 3(b), the solution obtained by our approximate algorithm under the minimum delay constraint includes 6 operations with 2 adder-steps. We note that the solution given in Figure 3(b) is the minimum number of operations solution under the minimum delay constraint guaranteed by the graph-based heuristic [2].

As can be observed from Figure 3, the solution under the minimum delay constraint includes number of operations equal to, or generally, greater than that of the minimum number of operations solution.

2.3 Related Work

In the graph-based heuristic of [11], three methods that reduce the number of adder-steps are applied in BHM [8] and RAG-n [8] algorithms designed for the MCM problem, leading to two algorithms called SLBHM and SLRAGn respectively. The graph-based heuristic of [7], called C1, initially, finds a solution using BHM or RAG-n including generally more number of operations but, with a small number of adder-steps and then, reduces the number of operations without increasing the delay in an iterative loop. The heuristic of [6] controls the increase in delay by choosing an operation that yields the least total number of adder-steps during the synthesis of a constant in each iteration. However, note that the algorithms of [6, 7] cannot find a solution under a specific delay constraint.

3 The Approximate Algorithm

In the approximate graph-based algorithm, called ADA, the target constants are implemented including fewest number of intermediate constants into the ready set such that at the end of the algorithm, for each target and intermediate constant in the ready set, there is an *A-operation* as given in Eqn. (1) where the inputs are the elements of the ready set. In each iteration of the algorithm, an intermediate constant that can synthesize the greatest number of target constants without violating the delay constraint is selected.

In the preprocessing phase of the approximate graph-based algorithm, the target constants are made positive and odd, added to the target set, T, without repetition, and the maximum bit-width of the target constants, bw, is determined. The pseudo-code of the ADA algorithm is given in Algorithm 1.

Algorithm 1 The ADA algorithm. The algorithm takes the target set, T, including target constants to be implemented and the delay constraint, dc, not to be exceeded, and returns the ready set, R, including 1, target, and intermediate constants.

ApproximateSearchUdc(T, dc)

```
1: R ← {1}
2: (R, T) = Synthesize(dc, R, T)
3: if T = ∅ then
4:    return R
5: else
6:    while 1 do
7:       for j = 1 to 2^(bw+1) − 1 step 2 do
8:          if ⌈log2 S(j)⌉ < dc then
9:             if j ∉ R and j ∉ T then
10:               (A, B) = Synthesize(dc, R, {j})
11:               if B = ∅ then
12:                  (A, B) = Synthesize(dc, A, T)
13:                  if B = ∅ then
14:                     A = RemoveRedundant(A, dc)
15:                     return A
16:                  else
17:                     cost_j = EvaluateCost(B)
18:      Find the intermediate constant, ic, with the minimum cost among all possible intermediate constants, j.
19:      R ← R ∪ {ic}
20:      (R, T) = Synthesize(dc, R, T)
```

Synthesize(dc, R, T)

```
1: repeat
2:    isadded = 0
3:    for k = 1 to |T| do
4:       if t_k can be synthesized with the elements of R without violating dc then
5:          isadded = 1
6:          R ← R ∪ {t_k}
7:          T ← T \ {t_k}
8: until isadded = 0
9: return (R, T)
```

EvaluateCost(T)

```
1: cost = 0
2: for k = 1 to |T| do
3:    cost = cost + SingleConstantCost(t_k)
4: return cost
```

RemoveRedundant(R, dc)

```
1: for k = 1 to |R| do
2:    if r_k is an intermediate constant then
3:       R ← R \ {r_k}
4:       (R, T) = Synthesize(dc, {1}, R)
5:       if T ≠ ∅ then
6:          R ← R ∪ {r_k}
7: return R
```

In the main function of ADA, *ApproximateSearchUdc*, initially, the ready set including only 1 is formed and then, using the *Synthesize* function, all the target constants that can be implemented with the elements of the ready set are found iteratively without exceeding the delay constraint, i.e., dc. If all the target constants are synthesized at this phase of the algorithm, then the found solution is the minimum number of operations solution under the delay constraint. Otherwise, in each iteration of the infinite loop, i.e., the line 6 of the algorithm, an intermediate constant is added to the ready set until there is no element left in the target set. The ADA algorithm, as given on lines 7-11 of its main function, considers the positive and odd constants, whose minimum delay implementations are less than dc, that are not included in the current ready and target sets and can be implemented with the elements of the current ready set, as possible intermediate constants. Note that the ready and target sets denoted by A and B represent the working ready and target sets respectively. Then, with the inclusion of the possible intermediate constant into the working ready set its implications on the current target set are found by the *Synthesize* function. If there exist unimplemented target constants in the working target set, the cost of unimplemented target constants is found in terms of their single minimum implementation costs [9] stored in a look-up table and is assigned to the cost value of the intermediate constant using the *EvaluateCost* function as given on line 17. After the cost value of each intermediate constant is found, the one with the minimum cost is chosen to be added to the current ready set and the target constants that can be implemented with the elements of the ready set are found. The infinite loop is interrupted whenever there is no element left in the working target set thus, the solution is obtained with the working ready set. However, note that by adding an intermediate constant to the ready set in each iteration, the previously added intermediate constants can be redundant due to the recently added constant. Hence, the *RemoveRedundant* function is applied on the final ready set to remove the redundant intermediate constants. After the ready set is obtained, each element in the ready set, except 1, is synthesized with a single operation whose inputs are the elements of the ready set.

As a small example, again consider the target set, $T = \{3, 13, 219, 221\}$, when the delay constraint, dc, is 2. After the ready set including only 1 is formed, initially, the target constant 3 and then, 13 are removed from the target set to the ready set, since each of them can be implemented using a single operation with the elements of the current ready set without violating dc. Thus, the current target set includes two unimplemented constants, i.e., 219 and 221, and in-

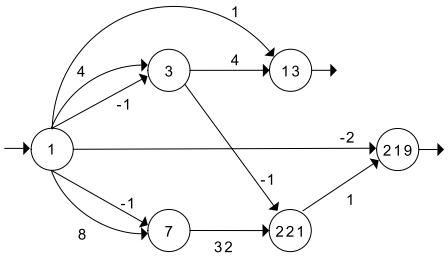


Figure 4: Implementation of the target set $\{3, 13, 219, 221\}$ with 3 adder-steps.

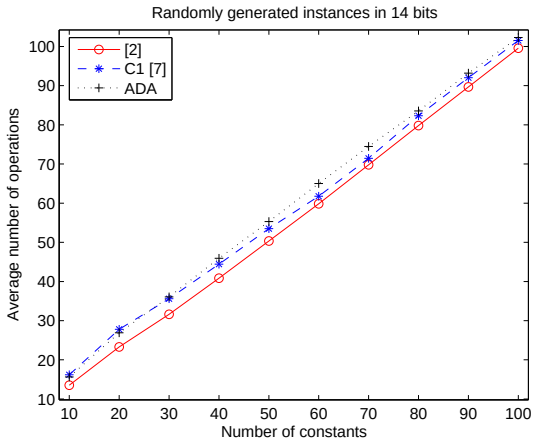
termediate constants are required. In the first iteration of the algorithm, the intermediate constant 7 that can be synthesized as $7 = 1 \ll 3 - 1$ is chosen and added to the ready set. Then, the target constant 221 is synthesized with the elements of the current ready set and added to the ready set. In the second iteration of the algorithm, the intermediate constant 5 is selected and the last target constant 219 is synthesized. The solution of the ADA algorithm including 6 operations with 2 adder-steps is given in Figure 3(b). Figure 4 presents the solution of the ADA algorithm when dc is 3. The solution of the algorithm when dc is 4, is given in Figure 3(a) where the target constants are synthesized in the optimal part of the algorithm. Observe that the amount of decrease in delay is equal to the amount of increase in area on this example.

4 Experimental Results

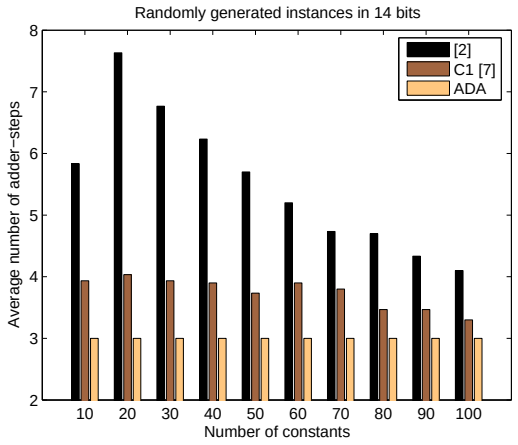
In this section, we present the results of the ADA algorithm and compare with those of the graph-based heuristics [7, 11] on randomly generated and FIR filter instances. The C1 algorithm [7] was provided by O. Gustafsson.

As the first experiment set, we used randomly generated instances where constants are defined under 14 bit-width. The number of constants ranges between 10 and 100, and for each of them, we generated 30 instances, totally 300 instances. Figure 5 presents the results of the graph-based heuristic [2] designed for the MCM problem, and the C1 [7] and ADA algorithms designed for the MCM problem under a delay constraint. In the ADA algorithm, the delay constraint is set to the minimum delay of the MCM problem computed by Eqn. (2).

As can be easily observed from Figure 5, a restriction on the delay of the design yields solutions including more number of operations with respect to those obtained without a delay constraint. However, we note that on the instances including 20 constants, while the difference of average number of operations between the graph-based heuristic [2] and the ADA algorithm is 3.6, the difference of average number of



(a)



(b)

Figure 5: Results of the graph-based algorithms on randomly generated instances: (a) Area; (b) Delay.

adder-steps between these algorithms is 4.6, meaning that the amount of decrease in delay is less than the amount of increase in area. Recall that the C1 algorithm [7] does not aim to find a solution under the minimum number of adder-steps. However, as can be observed from Figure 5(a)-(b), on instances including 10 and 20 constants, the ADA algorithm finds better area-delay solutions than the C1 algorithm. We also ran the ADA algorithm with the delay constraint that is equal to the adder-step of the solution obtained by the C1 algorithm on each instance. In this case, it is observed that the difference of average number of operations between the C1 and ADA algorithms is 1.18 on overall 300 instances. Note that the number of instances that the C1 algorithm finds a better solution than the ADA algorithm is only 1 on these 300 instances.

As the second experiment set, we used three FIR filter instances given in [11]. The specifications of filters are presented in Table 1 where *pass* and *stop* are normalized frequencies that define the passband and stopband respectively, *#tap* is the number of coefficients, and *width* is the bit-width of the coefficients.

Table 1: Characteristics of the FIR filters.

Filter	pass	stop	#tap	width
1	0.10	0.15	60	14
2	0.15	0.20	60	16
3	0.10	0.12	100	18

The results of the graph-based algorithms under different number of adder-steps are given in Table 2. In this table, *adder* and *step* denote the number of operations and the number of adder-steps respectively. Since the SLRAGn algorithm of [11] generally obtains better results than its SLBHM algorithm, the results of the SLRAGn algorithm are given in this table.

As can be easily observed from Table 2, the ADA algorithm finds similar or better area-delay solutions than the graph-based heuristics of [7, 11]. Note that the reason for the large increases on number of operations when the delay constraint is decreased from 4 to 3 is simply because in the latter case, there are much less possible intermediate constants due to the delay constraint yielding poor sharing.

Table 2: Summary of the results of the algorithms on FIR filters.

Filter	step	[2]	C1 [7]	SLRAGn [11]	ADA
		adder	adder	adder	adder
1	3			36	31
	4		29	31	29
	5			29	29
	6	28		28	28
2	3			37	34
	4		35	33	31
	5			32	30
	6	30			30
3	3			73	69
	4			69	62
	5		64	68	59
	6			64	58
	7			62	57
	8			62	56
	9	56		61	56

5 Conclusions

In this work, we introduced an approximate graph-based algorithm designed for the MCM problem under a delay constraint. The proposed algorithm is based on an efficient graph-based heuristic designed for the MCM problem. To deal with the delay constraint, in each iteration of the algorithm, the intermediate constants that do not violate the delay constraint are considered. It was observed from the experimental results that the approximate algorithm balances the area and delay of the design efficiently and obtains better area-delay solutions than the previously proposed prominent graph-based heuristics.

References:

[1] L. Aksoy, E. Costa, P. Flores and J. Monteiro, Exact and Approximate Algorithms for the Optimization of Area and Delay in Multiple Constant Multiplications, *IEEE Transactions on Computer-Aided Design of Integrated Circuits* 27(6), 2008, pp. 1013–1026.

[2] L. Aksoy and E.O. Gunes, An Approximate Algorithm for the Multiple Constant Multiplications Problem, *SBCCI*, 2008, pp. 58–63.

[3] L. Aksoy, E.O. Gunes and P. Flores, An Exact Breadth-First Search Algorithm for the Multiple Constant Multiplications Problem, *NORCHIP*, 2008, pp. 41–46.

[4] A. Avizienis, Signed-digit Number Representation for Fast Parallel Arithmetic, *IRE Transactions on Electronic Computers* EC-10, 1961, pp. 389–400.

[5] P. Cappello and K. Steiglitz, Some Complexity Issues in Digital Signal Processing, *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32(5), 1984, pp. 1037–1041.

[6] E. Costa, P. Flores and J. Monteiro, Maximal Sharing of Partial Terms in MCM under Minimal Signed Digit Representation, *ECCTD*, 2005, pp. 221–224.

[7] A. Dempster, S. Demirsoy and I. Kale, Designing Multiplier Blocks with Low Logic Depth, *ISCAS*, 2002, pp. 773–776.

[8] A. Dempster and M. Macleod, Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters, *IEEE Transactions on Circuits and Systems II* 42(9), 1995, pp. 569–577.

[9] O. Gustafsson, A. Dempster and L. Wanhammar, Extended Results for Minimum-adder Constant Integer Multipliers, *ISCAS*, 2002, pp. 73–76.

[10] O. Gustafsson and L. Wanhammar, ILP Modelling of the Common Subexpression Sharing Problem, *ICECS*, 2002, pp. 1171–1174.

[11] H-J. Kang and I-C. Park, FIR Filter Synthesis Algorithms for Minimizing the Delay and the Number of Adders, *IEEE Transactions on Circuits and Systems II* 48(8), 2001, pp. 770–777.

[12] H. Nguyen and A. Chatterjee, Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis, *IEEE Transactions on VLSI* 8(4), 2000, pp. 419–424.

[13] Y. Voronenko and M. Püschel, Multiplierless Multiple Constant Multiplication, *ACM Transactions on Algorithms* 3(2), 2007.