Efficient Shift-Adds Design of Digit-Serial Multiple Constant Multiplications

Levent Aksoy, Cristiano Lazzari
INESC-ID
Lisboa, Portugal
{levent, lazzari}@algos.inesc-id.pt

Eduardo Costa
Univ. Católica de Pelotas
Pelotas, Brazil
ecosta@ucpel.tche.br

Paulo Flores, José Monteiro
INESC-ID/IST TU Lisbon
Lisboa, Portugal
{pff, jcm}@inesc-id.pt

ABSTRACT

Bit-parallel realization of the multiplication of a variable by a set of constants using only addition, subtraction, and shift operations has been explored extensively over the years as large number of constant multiplications dominate the complexity of many digital signal processing systems. On the other hand, digit-serial architectures offer alternative low-complexity designs since digit-serial operators occupy less area and are independent of the data wordlength. This paper introduces an approximate algorithm that targets the optimization of gate-level area in digit-serial constant multiplications under the shift-adds architecture. Experimental results indicate that our approximate algorithm gives better solutions than the previously proposed algorithms in terms of area at gate-level and yields alternative low-complexity designs relatively to the bit-parallel design. It is also observed on digit-serial filter designs that the use of shift-adds architecture yields area reduction up to 43.6% with respect to designs that use generic digit-serial constant multipliers.

Categories and Subject Descriptors

B.2.0 [**Arithmetic and Logic Structures**]: General; B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids

General Terms

Algorithms, Design

Keywords

Multiple constant multiplications (MCM), digit-serial arithmetic, gate-level area optimization, finite impulse response filters

1. INTRODUCTION

Multiplication of a variable by a set of constants, generally known as Multiple Constant Multiplications (MCM), is essential in many Digital Signal Processing (DSP) applications such as, digital Finite Impulse Response (FIR) filters (illustrated in Figure 1), Fast Fourier Transforms (FFT), and Discrete Cosine Transforms (DCT). However, the implementation of a multiplication operation in hardware is considered to be expensive as it occupies significant area and has large delay. Since the constants in multiplications are determined beforehand by the DSP algorithms, the full-flexibility of a multiplier is not necessary and the constant multiplications can be replaced by addition/subtraction and shift operations [12].

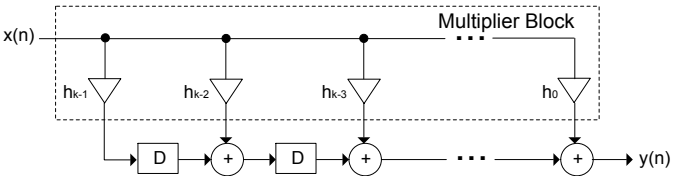

Figure 1: Transposed form of a digital FIR filter design.

For the implementation of constant multiplications using addition/subtraction and shift operations, a straightforward method, generally known as the digit-based recoding [7], initially defines the constants in multiplications in binary representation. Then, for each 1 in the binary representation of the constant, according to its bit position, it shifts the variable and adds up the shifted variables to obtain the result. As a simple example, consider the constant multiplications $29x$ and $43x$ using the digit-based recoding method [7]. The decompositions of $29x$ and $43x$ in binary are listed as follows:

$$29x = (11101)_{bin}x = x \ll 4 + x \ll 3 + x \ll 2 + x$$
$$43x = (101011)_{bin}x = x \ll 5 + x \ll 3 + x \ll 1 + x$$

and require 6 addition operations as illustrated in Figure 2(a).

However, the implementation of constant multiplications in a shift-adds architecture enables the sharing of common partial products among the constant multiplications, that significantly reduces the area and power dissipation of the MCM design. Hence, the MCM problem is defined as finding the minimum number of addition/subtraction operations that implement the constant multiplications, since shifts can be realized using only wires in hardware. Note that the MCM problem is an NP-complete problem [4].

The algorithms designed for the MCM problem can be categorized in two classes as Common Subexpression Elimination (CSE) methods and graph-based (GB) techniques. While the maximization of the partial product sharing is common in these algorithms, they differ in the search space that they explore. The CSE algorithms [1, 13] initially define the constants under a particular number representation namely, binary, Canonical Signed Digit (CSD), or Minimal Signed Digit (MSD), and then, find the “best" subexpression, generally the most common, among the constant multiplications. The GB algorithms [2, 6, 14] are not restricted to any particular number representation and consider a large number of alternative implementations of a constant multiplication, yielding better solutions than the CSE algorithms as shown in [2, 14].

Returning to our example in Figure 2, the exact CSE algorithm [1] gives a solution with 4 operations by finding the most common partial products $3x = (11)_{bin}x$ and $5x = (101)_{bin}x$ when constants are defined under binary, (Figure 2(b)). The exact GB algorithm [2] finds the minimum number of operations solution with 3 opera-

GLSVLSI'11, May 2–4, 2011, Lausanne, Switzerland.

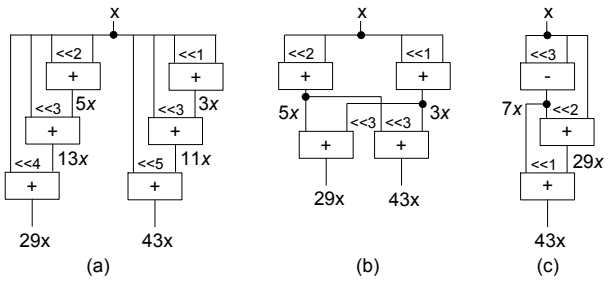

Figure 2: Shift-adds implementations of $29x$ and $43x$: (a) without partial product sharing [7]; with partial product sharing: (b) the algorithm of [1]; (c) the algorithm of [2].

tions by sharing the common partial product $7x$ in both multiplications, (Figure 2(c)). Observe that the partial product $7x = (111)_{bin}x$ cannot be extracted from the binary representations of both multiplications $29x$ and $43x$ in the exact CSE algorithm [1].

However, all these algorithms assume that the input data x is processed in parallel and hence, shifts do not represent any cost in parallel processing since they are implemented with wires. On the other hand, in digit-serial arithmetic, the data words are divided into digit sets, consisting of d bits, that are processed one at a time [8]. In this case, although the addition/subtraction operations are realized using low-complexity digit-serial operations, the implementation of shifts requires D flip-flops. Hence, the optimization algorithms should consider the sharing of shifts as well as the sharing of addition/subtraction operations while implementing digit-serial constant multiplications under the shift-adds architecture.

Hence, in this paper, we introduce a GB algorithm that focuses on the optimization of gate-level area in the digit-serial MCM design. The experimental results on randomly generated constants and FIR filter instances indicate that our GB algorithm yields digit-serial MCM designs using less area when compared to those obtained by the previously proposed algorithms. It is also shown that the digit-serial realization of an FIR filter yields alternative low-complexity filter designs in addition to its bit-parallel realization and the design of the digit-serial FIR filter under the shift-adds architecture yields significant savings in area when compared to those implemented using generic digit-serial constant multipliers [9].

The rest of the paper proceeds as follows. Section 2 gives the background concepts. The approximate GB algorithm is described in Section 3. Experimental results are presented in Section 4 and finally, Section 5 concludes the paper.

2. BACKGROUND

In this section, we present the main concepts on digit-serial arithmetic, introduce the problem definitions, and give an overview on previously proposed algorithms.

2.1 Digit-Serial Arithmetic

In digit-serial arithmetic, data words are divided into digits, with a digit size of d bits, which are processed in one clock cycle. The special cases of the digit-serial computation, called bit-serial and bit-parallel processing, occur when the digit size d is equal to 1 and input data wordlength, respectively. The digit-serial computation plays an important role when the bit-serial implementations cannot meet delay requirements and the bit-parallel designs require excessive hardware. Thus, an optimal tradeoff between area and delay can be obtained by changing the digit size parameter (d).

The fundamental digit-serial operations were introduced in [8]. The digit-serial addition, subtraction, and left shift operations are depicted in Figure 3 when d is equal to 3. Notice from Figure 3(a) that in a digit-serial addition operation, in general, the number of

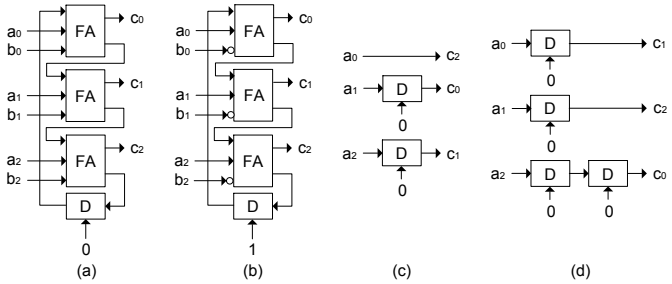

Figure 3: The digit-serial operations when d is 3: (a) addition operation; (b) subtraction operation; (c) left shift by 2 times; (d) left shift by 4 times.

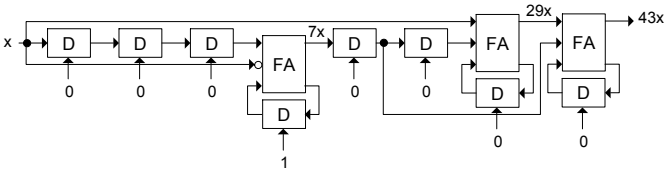

Figure 4: Bit-serial realization of shift-adds implementation of $29x$ and $43x$ given in Figure 2(c).

required full adders (FAs) is equal to d and the number of necessary D flip-flops is always 1. The subtraction operation (Figure 3(b)) is implemented using 2's complement, requiring the initialization of the D flip-flop with 1 and additional d inverter gates with respect to the digit-serial addition operation. In a left shift operation (Figure 3(c)-(d)), the number of required D flip-flops is equal to the amount of shift. The input-output correspondence and the number of flip-flops cascaded serially for each input in a digit-serial left shift operation are given in Eqn. (1) and (2) respectively, where i ranges from 0 to $d-1$ and ls denotes the amount of left shift.

$$a_i \Rightarrow c_{(i\ +\ ls)\ mod\ d} \tag{1}$$

$$\#FF_{a_i} = \begin{cases} \lfloor ls/d \rfloor & \text{if } i < d - (ls\ mod\ d) \\ \lceil ls/d \rceil & \text{otherwise} \end{cases} \tag{2}$$

As an example on digit-serial realization of constant multiplications under the shift-adds architecture, Figure 4 illustrates the bit-serial implementation of $29x$ and $43x$ obtained by the exact GB algorithm [2] given in Figure 2(c). The network includes 2 bit-serial additions, 1 bit-serial subtraction, and 5 D flip-flops for all the left shift operations. Observe from Figure 4 that at each clock cycle, one bit of the input data x is applied to the network input and one bit of the constant multiplication output is computed. Note that the digit-serial design of the MCM operation occupies significantly less area when compared to its bit-parallel design and the area of the design is not dependent on the bit-width of the input data. However, the latency of the MCM computation is increased due to the serial processing. Suppose that x is a 16-bit input value. To obtain the actual output of $29x$ and $43x$ in the bit-serial network of Figure 4, 21 and 22 clock cycles are required respectively[1]. Thus, necessary bits must be appended to the input data x, *i.e.*, 0s, if x is an unsigned input or sign bits, otherwise. Moreover, in the case of the conversion of the outputs obtained in digit-serial to the bit-parallel format, storage elements and control logic are required.

Note that while the sharing of addition/subtraction operations reduces the complexity of the digit-serial MCM design (since each addition and subtraction operation requires a digit-serial operation),

[1] In general, in the design of a digit-serial constant multiplication cx under shift-adds architecture, the number of clock cycles required to obtain the computation is $\lceil (\lceil log_2 c \rceil + N)/d \rceil$, where N is the bit-width of the input data and d is less than N.

the sharing of shift operations for a constant multiplication reduces the number of D flip-flops, and consequently, the design area. Observe from Figure 4 that two D flip-flops cascaded serially to generate the left shift of $7x$ by two can also generate the left shift of $7x$ by one without adding any hardware cost.

2.2 Problem Definitions

For the multiplierless realization of the constant multiplications, the fundamental operation, called *A-operation* in [14], 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:

$$w = A(u,v) = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r} \quad (3)$$

where $s \in \{0,1\}$ is the sign, which determines if an addition or a subtraction operation is to be performed, $l_1, l_2 \geq 0$ are integers denoting left shifts of the operands, and $r \geq 0$ is an integer indicating a right shift of the result.

In the MCM problem, it is supposed that the input data is processed in parallel and hence, the shifting operation has no cost in hardware. It is also assumed that the sign of the constant can be adjusted at some part of the design and the complexity of an adder and a subtracter is equal in hardware. Thus, only positive and odd constants are considered in the MCM problem. Observe from Eqn. (3) that in the implementation of an odd constant considering any two odd constants at the inputs, one of the left shifts, l_1 or l_2, is zero and r is zero, or both l_1 and l_2 are zero and r is greater than zero. Also, it is necessary to constrain the left shifts, l_1 and l_2, otherwise there exist infinite ways of implementing a constant. In the algorithm of [2], the number of shifts is allowed to be at most $bw+1$, where bw is the maximum bit-width of the constants to be implemented. Thus, the MCM problem [14] can be defined as follows:

DEFINITION 1. THE MCM PROBLEM. *Given the target set composed of positive and odd unrepeated target constants to be implemented, $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$, find the smallest ready set, $R = \{r_0, r_1, \ldots, r_m\}$, 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 A-operation $r_k = A(r_i, r_j)$.*

As described in Section 2.1, the digit-serial MCM operation includes digit-serial addition and subtraction operations and D flip-flops for the left shift operations, each having different implementation cost at gate-level. Hence, the optimization of area problem in the digit-serial MCM operation can be defined as follows:

DEFINITION 2. THE OPTIMIZATION OF AREA PROBLEM IN DIGIT-SERIAL MCM OPERATION. *Given the digit size d and the target set $T = \{t_1, \ldots, t_n\} \subset \mathbb{N}$, find the ready set $R = \{r_0, r_1, \ldots, r_m\}$ such that under the same conditions on the ready set given in Definition 1, the set of A-operations yields a digit-serial MCM design using optimal area at gate-level.*

In an *A-operation* that realizes a constant multiplication under the digit-serial architecture, its right shift is always assumed to be 0 in [10, 11], since the complexity of the control logic is significantly increased to realize the MCM operation in this case.

2.3 Related Work

For the MCM problem, the exact CSE algorithm of [1] initially defines the target constants under a number representation and finds all possible implementations of constant multiplications that can be extracted from the representations of the constants. Then, the MCM problem is formalized as a 0-1 Integer Linear Programming (ILP) problem with constraints to be satisfied and a cost function to be minimized. Finally, the minimum number of operations solution is obtained using a generic 0-1 ILP solver. The exact GB algorithms that search the minimum number of operations solution in breadth-first and depth-first manners were introduced in [2]. An efficient GB heuristic algorithm, called RAG-n, that includes two parts, optimal and heuristic, was introduced in [6]. In the optimal part, each target constant that can be implemented with a single operation are synthesized. If there exist unimplemented elements left in the target set, the algorithm switches to the heuristic part. In this iterative part of the algorithm, RAG-n initially chooses a single unimplemented target constant with the smallest single coefficient cost evaluated by the algorithm of [5] and then, synthesizes it with a single operation including one(two) intermediate constant(s) that has(have) the smallest value among the possible constants. However, since the intermediate constants are selected for the implementation of a single target constant in each iteration, the intermediate constants chosen in previous iterations may not be shared for the implementation of not-yet synthesized target constants in later iterations, thus yielding a local minimum solution. The GB heuristic of [14], called Hcub, includes the same optimal part of RAG-n, but uses a better heuristic that considers the impact of each possible intermediate constant on the not-yet synthesized target constants.

For the optimization of area problem in digit-serial MCM operation, the exact CSE algorithm of [3] formalizes this problem as a 0-1 ILP problem taking into account the gate-level implementation cost of digit-serial operations and D flip-flops for the shifts. Also, two GB algorithms, called RSAG-n and RASG-n, that target the reduction on the number of addition/subtraction operations and the amount of shifts were introduced in [10, 11] respectively. Both algorithms are based on the RAG-n algorithm designed for the MCM problem. However, in each iteration, while the RSAG-n algorithm chooses the intermediate constant(s) that require the minimum number of shifts, the RASG-n algorithm selects the intermediate constant(s) with the minimum cost value as done in RAG-n but, if there are more than one possible intermediate constant, it favors the one that requires the minimum number of shifts.

3. THE APPROXIMATE ALGORITHM

As done in algorithms designed for the MCM problem given in Definition 1, in our approximate algorithm, called MINAS-DS, we find the fewest number of intermediate constants such that all the target and intermediate constants are synthesized using a single operation. However, while selecting an intermediate constant for the implementation of the not-yet synthesized target constants in each iteration, we favor the one that can be synthesized using the least hardware and enables to implement the not-yet synthesized target constants in a smaller area with the available constants. After the set of target and intermediate constants that realizes the MCM operation is found, each constant is synthesized using an *A-operation* that yields the minimum area in the digit-serial MCM design. In MINAS-DS, the area of the digit-serial MCM operation is determined as the total implementation cost of each digit-serial addition, subtraction, and shift operation, as described in Section 2.1.

In the preprocessing phase of the MINAS-DS algorithm, the target constants to be implemented are made positive and odd, are added to the target set, T, without repetition, and the maximum bit-width of the target constants, bw, is determined. The main part of the MINAS-DS algorithm is given in Algorithm 1.

In MINAS-DS, the ready set, $R = \{1\}$, is formed initially and then, the target constants that can be implemented with the elements of the ready set using a single operation are found and moved to the ready set iteratively using the *Synthesize* function. If there exist unimplemented constants in the target set then, in each iteration of its heuristic part (line 3), an intermediate constant is added to the

Algorithm 1 The MINAS-DS algorithm.

MINAS-DS(T)

1: $R \leftarrow \{1\}$
2: (R, T) = Synthesize(R, T)
3: **while** $T \neq \emptyset$ **do**
4: **for** $j = 1$ to $2^{bw+1} - 1$ step 2 **do**
5: **if** $j \notin R$ and $j \notin T$ **then**
6: $impcost_j$ = ComputeCost($\{j\}$, R)
7: **if** $impcost_j \neq 0$ **then**
8: $A \leftarrow R \cup \{j\}$
9: $impcost_T$ = ComputeTCost(T, A)
10: $iccost_j = impcost_j + impcost_T$
11: Find the intermediate constant, ic, with the minimum $iccost_j$ cost among all possible constants, j
12: $R \leftarrow R \cup \{ic\}$
13: (R, T) = Synthesize(R, T)
14: D = SynthesizeMinArea(R)
15: **return** D

Synthesize(R, T)

1: **repeat**
2: $isadded = 0$
3: **for** each $t_k \in T$ **do**
4: **if** t_k can be implemented using a single *A-operation* whose inputs are the elements of R **then**
5: $isadded = 1$
6: $R \leftarrow R \cup \{t_k\}$
7: $T \leftarrow T \setminus \{t_k\}$
8: **until** $isadded = 0$
9: **return** (R, T)

ComputeCost({c}, C)

1: $cost_c = 0$
2: **for** all operations $c = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r}$, where $u, v \in C$ **do**
3: Determine the cost of each operation under the digit-serial architecture, compute the minimum implementation cost of constant c, and assign it to $cost_c$
4: **return** $cost_c$

ComputeTCost(B, C)

1: $cost_B = 0$
2: **repeat**
3: $isadded = 0$
4: **for** each $b_k \in B$ **do**
5: $cost_{b_k}$ = ComputeCost($\{b_k\}$, C)
6: **if** $cost_{b_k} \neq 0$ **then**
7: $isadded = 1$
8: $C \leftarrow C \cup \{b_k\}$
9: $B \leftarrow B \setminus \{b_k\}$
10: $cost_B = cost_B + cost_{b_k}$
11: **until** $isadded = 0$
12: **for** each $b_k \in B$ **do**
13: $cost_B = cost_B + maxcost(b_k)$
14: **return** $cost_B$

SynthesizeMinArea(R)

1: Find all possible implementations of target and intermediate constants using the *GenerateImp(R)* function
2: Formalize the problem as a 0-1 ILP problem
3: Find D as a set of *A-operations* that yields minimum area under the digit-serial architecture
4: **return** D

GenerateImp(R)

1: $A \leftarrow \{1\}$, $R \leftarrow R \setminus \{1\}$
2: **repeat**
3: **for** each $r_k \in R$ **do**
4: (B, C) = Synthesize(A, $\{r_k\}$)
5: **if** $C = \emptyset$ **then**
6: Find all operations, $r_k = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r}$, where $u, v \in A$ and determine their implementation costs under the digit-serial architecture
7: $A \leftarrow A \cup \{r_k\}$
8: $R \leftarrow R \setminus \{r_k\}$
9: **until** $R = \emptyset$

ready set until there is no element left in the target set. The MINAS-DS algorithm considers the positive and odd constants that are not included in the current ready and target sets (lines 4-5) and that can be implemented with the elements of the current ready set using a single operation (lines 6-7) as possible intermediate constants. On line 6, the *ComputeCost* function searches all *A-operations* that compute the constant with the elements of the current ready set. If the implementations of the constant are found then, it determines the cost of each operation under the digit-serial architecture as described in Section 2.1 and returns its minimum implementation cost among possible operations. Otherwise, it returns 0 value indicating that the constant cannot be synthesized using an operation with the elements of the current ready set. After the possible intermediate constant is found, it is included into the working ready set, A, and its implications on the current target set are found by the *ComputeTCost* function. In this function, similar to the *ComputeCost*, the minimum implementation costs of the target constants under the digit-serial architecture that can be synthesized with the elements of the working ready set A are determined. For each target constant, t_k, that cannot be implemented with the elements of A, its cost value is determined as its maximum implementation cost, $maxcost(t_k)$, computed as if it requires a digit-serial addition operation with digit size d and $\lceil log_2 t_k \rceil$ D flip-flops for the left shifts. Then, the cost of the intermediate constant is determined as its minimum implementation cost plus the implementation costs of the not-yet synthesized target constants. After the cost value of each possible intermediate constant is found, the one with the minimum cost is added to the current ready set and its implications on the current target set are found using the *Synthesize* function.

When there are no elements left in the target set, the *SynthesizeMinArea* function is applied on the final ready set to find the set of *A-operations* that yields a solution with the smallest area. Because, in each iteration of MINAS-DS, the cost of an intermediate constant is determined by an operation whose inputs are available in the current ready set. However, the recently added intermediate constants may yield better realizations of previously added constants. Hence, we formalize this problem as a 0-1 ILP problem, similar to the formalization described in [3], where the sharing of addition, subtraction, and shift operations is maximized considering their implementation costs. Note that the possible implementations of the constants are found by the *GenerateImp* function.

4. EXPERIMENTAL RESULTS

This section presents the high-level results of the MINAS-DS algorithm on FIR filter and randomly generated instances and the comparison of its results with those obtained by the previously proposed algorithms designed for the MCM problem [1, 2, 6] and the optimization of area problem in digit-serial MCM operation [3]. Also, the gate-level results on the multiplier blocks of digital FIR filters designed using the solutions of MINAS-DS are given and compared with those of designs obtained by the algorithms of [1, 2, 3, 6]. Finally, we introduce the gate-level results of digit-serial FIR filters whose multiplier blocks are designed using digit-serial constant multipliers [9] and using digit-serial addition, subtraction, and shift operations determined by the solution of MINAS-DS. Note that the gate-level results on the digit-serial multiplier block of an FIR filter or on the whole digit-serial FIR filter itself also include the storage elements and control logic that are necessary to convert the digit-serial computation results to parallel.

To design a digit-serial MCM operation at gate-level, we implemented a design tool called SAFIR that takes as inputs, the bit-width of the input data N, the digit size parameter d, and the set of addition/subtraction operations found by a high-level algorithm and

Table 1: Summary of results of the algorithms on randomly generated 12-bit constants when d is 1.

number of constants (n)	Optimization of the number of operations									Optimization of area in bit-serial MCM design					
	Exact CSE - MSD [1]			RAG-n [6]			Exact GB [2]			Exact CSE - MSD [3]			MINAS-DS		
	oper	shift	Icost	oper	shift	Icost	oper	shift	Icost	oper	shift	Icost	oper	shift	Icost
10	15.5	28.6	3735.6	15.1	32.5	3872.5	12.8	25.0	3161.3	17.1	15.7	3280.7	13.1	18.9	2875.8
20	26.5	42.5	6044.5	22.2	36.5	5111.4	21.4	33.2	4834.7	29.2	19.7	5239.6	21.5	26.5	4493.4
30	36.7	53.3	8091.3	30.1	39.8	6432.9	30.1	39.3	6406.5	39.9	23.1	6975.4	30.1	28.8	5847.3
40	46.0	60.6	9827.4	39.4	47.9	8195.1	39.4	47.9	8191.5	50.2	24.9	8548.1	39.4	28.5	7140.9
50	55.9	68.7	11672.5	49.0	54.4	9922.0	49.0	53.2	9860.5	59.8	27.6	10072.3	49.0	26.5	8411.8
60	65.4	77.0	13482.9	59.0	59.7	11652.5	59.0	58.7	11597.5	69.8	29.3	11595.3	59.0	23.9	9693.1
70	75.1	83.3	15238.3	68.2	62.6	13143.5	68.2	61.8	13102.9	79.5	31.0	13086.2	68.2	23.1	10980.6
80	83.2	89.5	16716.1	77.7	74.1	15111.4	77.7	74.5	15125.5	87.6	31.3	14275.5	77.7	20.7	12202.1
90	92.7	99.9	18646.9	86.8	74.7	16466.7	86.8	73.8	16414.0	97.2	32.0	15690.6	86.8	20.9	13504.9
100	101.7	104.7	20204.0	96.5	84.0	18343.9	96.5	86.0	18442.3	106.4	32.4	17034.1	96.5	19.8	14827.8

generates the VHDL code of the digit-serial MCM operation automatically. The SAFIR tool has capabilities to describe a digit-serial MCM operation using generic digit-serial constant multipliers [9] in VHDL and to design digit-serial FIR filters. In SAFIR, we use the Synopsys Design Compiler with the UMCLogic $0.18\mu m$ Generic II library to synthesize digit-serial MCM and FIR filter circuits.

As the first experiment set, we used sets of instances that include the number of constants (n) ranging from 10 and 100 where each set includes 30 instances and the constants are 12-bit randomly generated integers. Table 1 presents the high-level results of the algorithms [1, 2, 3, 6] and MINAS-DS. In the exact CSE algorithms [1, 3], the constants were defined under MSD and in the exact CSE algorithm [3] and MINAS-DS, d was taken as 1. In this table, *oper* and *shift* stand for the average number of operations and shifts respectively and *Icost* denotes the average implementation cost of the MCM operation obtained by the algorithms under the bit-serial architecture. The implementation cost of an FA, a D flip-flop, and an inverter was taken as 90, 52, and 6 respectively, as the area (in μm^2) of these components in the design library and N was 16.

Observe from Table 1 that although the exact CSE and GB algorithms [1, 2] find an MCM design with fewer number of operations than the exact CSE algorithm [3] and MINAS-DS respectively, their solutions yield bit-serial MCM designs that occupy larger area than those obtained by these algorithms. Because the algorithms designed for the MCM problem [1, 2, 6] do not consider the sharing of shifts that require D flip-flops in the digit-serial arithmetic. Note that while the average number of operations on solutions found by the exact GB algorithm [2] and MINAS-DS is the same on instances where n is larger than 20, the ratio of average number of shift operations on solutions obtained by the exact GB algorithm [2] and MINAS-DS reaches up to 4.34 when n is 100.

As the second experiment set, we used the FIR filter instances[2] given in Table 2 where filter coefficients were computed with the *remez* algorithm in MATLAB. In this table, *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 filter coefficients.

The high-level results of algorithms on FIR filter instances are given in Table 3. Again, it is assumed that the multiplier blocks of the FIR filters are to be designed under the bit-serial architecture, *i.e.*, d is 1. Observe from Table 3 that while MINAS-DS finds MCM designs with the same number of operations as the exact GB algorithm [2], it gives solutions including less number of shifts that consequently lead to bit-serial MCM designs with less hardware cost when compared to the solutions of [1, 2, 6]. Also, MINAS-DS obtains better MCM designs than the exact CSE algorithm [3], since it considers more possible implementations of a constant yielding better solutions in terms of the number of operations.

The gate-level results of the bit-serial MCM designs obtained by the algorithms are given in Table 4, where *area*, *delay*, and *power*

[2]The FIR filter instances are available at http://algos.inesc-id.pt/multicon.

Table 2: Filter specifications.

Filter	*pass*	*stop*	*#tap*	*width*
1	0.10	0.15	200	16
2	0.10	0.15	240	16
3	0.10	0.25	180	16
4	0.10	0.25	200	16
5	0.10	0.20	240	16
6	0.10	0.20	300	16
7	0.15	0.25	200	16
8	0.15	0.25	240	16
9	0.20	0.25	240	16
10	0.20	0.25	300	16

denote respectively the area in μm^2, the delay of the longest path in *ns*, and the total dynamic power dissipation in nW. During the technology mapping phase of the synthesis tool, the bit-serial MCM operations were synthesized under the minimum area design strategy without a constraint on the clock frequency.

Observe from Table 4 that the MINAS-DS algorithm, whose objective is to optimize the gate-level area of a digit-serial MCM operation, leads to significant improvements on area when the bit-serial MCM designs are implemented at gate-level.

Table 5 presents the gate-level results of digit-serial designs of Filter 4. This FIR filter was chosen among others to be designed since its multiplier block requires the largest number of addition and subtraction operations as shown in Table 3. In SAFIR, Filter 4 was designed under two architectures denoted as *shift-adds* and *cons. mult.* in Table 5. When d is 1, 2, 4, and 8, the multiplier block of the FIR filter (illustrated in Figure 1) is designed using digit-serial addition, subtraction, and shift operations determined by the solution of MINAS-DS under the *shift-adds* architecture and it is implemented using digit-serial constant multipliers [9] under the *cons. mult.* architecture. When d is 16, *i.e.*, for bit-parallel processing, the multiplier block is designed using addition and subtraction operations obtained by the solution of the exact GB algorithm [2] under the *shift-adds* architecture and it is described in VHDL as constant multiplications under the *cons. mult.* architecture. Note that the hardware except the multiplier block, that is required to compute the filter output (additions and registers as illustrated in Figure 1), is the same for both architectures under the same d. During the technology mapping, the FIR filters were synthesized under two design strategies, *i.e.*, the minimum area (*MA*) and the minimum area under the maximum clock frequency (*MCF*) constraint. In the former, there was no constraint on the clock frequency and in the latter, we found the maximum clock frequency that can be applied to the FIR filter iteratively in a binary search manner.

Observe from Table 5 that as the digit size is decreased, the area of the FIR filter is also decreased under both design architectures. However, the maximum clock frequency that can be applied to the FIR filter decreases, as the digit size increases. Also, observe that the area reduction obtained under the *shift-adds* architecture with respect to the *cons. mult.* architecture reaches up to 34.1% and 43.6% with the *MA* and *MCF* design strategies respectively when d is equal to 8.

Table 3: Summary of results of high-level algorithms on the multiplier blocks of the FIR filters when d is 1.

Filter	Optimization of the number operations									Optimization of area in bit-serial MCM design					
	Exact CSE - CSD [1]			RAG-n [6]			Exact GB [2]			Exact CSE - CSD [3]			MINAS-DS		
	oper	shift	Icost	oper	shift	Icost	oper	shift	Icost	oper	shift	Icost	oper	shift	Icost
1	83	156	20126	80	134	18556	79	130	18164	96	41	16028	79	31	12908
2	88	144	20254	83	117	18098	83	126	18554	97	49	16604	83	32	13540
3	56	118	14280	49	97	12146	47	93	11642	62	36	10814	47	34	8508
4	94	127	20252	87	124	19048	87	111	18348	101	54	17408	87	43	14668
5	66	117	15690	63	110	14870	63	115	15130	78	37	13198	63	26	10394
6	74	124	17172	71	119	16462	68	104	15220	82	44	14154	68	29	11272
7	65	105	14882	59	86	12994	59	92	13270	74	32	12364	59	24	9740
8	73	116	16626	69	92	14786	69	99	15126	76	49	13538	69	23	11048
9	80	125	18076	78	92	16088	78	92	16052	89	33	14606	78	32	12818
10	84	114	18102	81	97	16810	81	94	16654	89	30	14444	81	21	12660
Total	763	1246	175460	720	1068	159858	714	1056	158160	844	405	143158	714	295	117556

Table 4: Summary of gate-level results of the high-level algorithms on the multiplier blocks of the FIR filters when d is 1.

Filter	Optimization of the number operations									Optimization of area in bit-serial MCM design					
	Exact CSE - CSD [1]			RAG-n [6]			Exact GB [2]			Exact CSE - CSD [3]			MINAS-DS		
	area	delay	power	area	delay	power	area	delay	power	area	delay	power	area	delay	power
1	74869	4.13	152	74377	4.13	153	74344	4.13	153	73626	4.13	146	72367	5.03	148
2	77877	4.13	159	77333	4.13	161	77280	4.13	161	76782	4.13	153	75677	5.47	156
3	42720	4.12	91	41914	4.12	89	41740	4.12	91	41497	4.12	84	40642	4.12	87
4	84584	4.14	170	84186	4.14	172	83921	4.14	172	83590	4.14	164	82757	4.47	165
5	59629	4.13	121	59363	4.13	122	59432	4.13	123	58747	4.13	117	57783	4.98	118
6	63529	3.99	131	63310	3.99	131	62723	3.99	132	62486	3.99	126	61492	5.13	127
7	55521	4.13	113	54883	4.13	113	54943	4.13	114	54658	4.13	110	53756	4.41	111
8	63001	4.13	128	62549	4.13	129	62488	4.13	130	62059	4.13	125	61159	4.94	126
9	73670	4.13	148	73175	4.13	146	73145	4.13	147	72670	4.13	143	71986	5.69	145
10	75619	4.13	151	75235	4.13	151	75168	4.13	154	74369	4.13	145	73726	4.09	151
Total	671019	41.16	1364	666325	41.16	1367	665184	41.16	1377	660484	41.16	1313	651345	48.33	1334

Table 5: Gate-level results of digit-serial Filter 4 of Table 2.

Arch.	DS	d	1	2	4	8	16
shift-adds	MA	area (mm^2)	201.7	214.8	228.9	281.1	322.9
		delay (ns)	5.45	6.16	6.94	7.70	9.90
		power (nW)	503	593	694	923	1060
	MCF	area (mm^2)	220.6	222.8	233.2	291.1	490.8
		delay (ns)	1.78	2.21	3.04	3.89	3.88
		power (mW)	271	258	224	241	464
cons. mult. [9]	MA	area (mm^2)	252.0	264.8	269.7	377.9	439.0
		delay (ns)	3.97	5.79	6.92	12.00	9.00
		power (nW)	619	706	779	1023	1220
	MCF	area (mm^2)	299.6	316.4	310.4	418.8	612.7
		delay (ns)	1.48	1.81	2.39	3.98	5.20
		power (mW)	440	431	376	281	490

5. CONCLUSIONS

This paper introduced an approximate GB algorithm that aims to optimize the gate-level area of a digit-serial MCM design. It considers more possible implementations of a constant multiplication than the recently proposed exact CSE algorithm and takes into account the gate-level implementation cost of digit-serial addition, subtraction, and shift operations while selecting the intermediate constants required for the constant multiplications, as opposed to the previously proposed GB algorithms. It was observed that the proposed algorithm obtains better solutions in terms of area than the algorithms designed for the MCM problem and the optimization of area problem in a digit-serial MCM operation at gate-level. It was also shown that the realization of digit-serial FIR filters under the shift-adds architectures yields significant area and power reductions when compared to those whose multiplier blocks are implemented using digit-serial constant multipliers.

6. ACKNOWLEDGMENT

This work was supported by the *Portuguese Foundation for Science and Technology* (FCT) research project (*Multicon - Architectural Optimization of DSP Systems with Multiple Constants Multiplications*) PTDC/EIA-EIA/103532/2008 and under INESC-ID multiannual funding through the PIDDAC Program funds.

7. 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 TCAD*, 27(6):1013-1026, 2008.

[2] L. Aksoy, E. Gunes, and P. Flores. Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate. *Elsevier Journal on Microprocessors and Microsystems*, 34:151-162, 2010.

[3] L. Aksoy, C. Lazzari, E. Costa, P. Flores, and J. Monteiro. Optimization of Area in Digit-Serial Multiple Constant Multiplications at Gate-Level. In *ISCAS*, to appear, 2011.

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

[5] A. Dempster and M. Macleod. Constant Integer Multiplication using Minimum Adders. *IEE Proceedings - Circuits, Devices, and Systems*, 141(5):407-413, 1994.

[6] A. Dempster and M. Macleod. Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters. *IEEE TCAS II*, 42(9):569-577, 1995.

[7] M. Ercegovac and T. Lang. Digital Arithmetic. *Morgan Kaufmann*, 2003.

[8] R. Hartley and P. Corbett. Digit-Serial Processing Techniques. *IEEE TCAS II*, 37(6):707-719, 1990.

[9] B. Parhami. Computer Arithmetic: Algorithms and Hardware Designs. *Oxford University Press*, 2000.

[10] K. Johansson, O. Gustafsson, A. Dempster, and L. Wanhammar. Algorithm to Reduce the Number of Shifts and Additions in Multiplier Blocks Using Serial Arithmetic. In *MELECON*, pages 197-200, 2004.

[11] K. Johansson, O. Gustafsson, and L. Wanhammar. Multiple Constant Multiplication for Digit-Serial Implementation of Low Power FIR Filters. *WSEAS Tran. on Circuits and Systems*, 5(7):1001-1008, 2006.

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

[13] M. Potkonjak, M. Srivastava, and A. Chandrakasan. Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination. *IEEE TCAD*, 15(2):151-165, 1996.

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