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, 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].

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:

\[
29x = \left(111010\right)_{bin}x = x \ll 4 + x \ll 3 + x \ll 2 + x \\
43x = \left(101011\right)_{bin}x = x \ll 5 + x \ll 3 + x \ll 1 + x
\]

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” sub-expression, 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 \(3x = \left(11\right)_{bin}x\) and \(5x = \left(101\right)_{bin}x\) when constants are defined under binary. (Figure 2(b)). The exact GB algorithm [2] finds the minimum number of operations solution with 3 opera-
Figure 2: Shift-adds implementations of $29x$ and $43x$: (a) without partial product sharing [7]; (b) the algorithm of [1]; (c) the algorithm of [2].

Implementations by sharing the common partial product $7x$ in both multiplications, (Figure 2(c)). Observe that the partial product $7x = (111)_{29}$ 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 additions under the shift-adds architecture.

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 additions under the shift-adds architecture.

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.

$$a_i \Rightarrow c(i + ls) \mod d$$
$$\#FF_{ai} = \begin{cases} 
\lfloor ls/d \rfloor & \text{if } i < d - (ls \mod d) \\
\lfloor ls/d \rfloor + 1 & \text{otherwise} 
\end{cases}$$ (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. Thus, necessary bits must be appended to the input data $x$, i.e., $0$s, 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),

1In 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 $\lfloor (\log_2 c) + N \rfloor/d$, 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 one 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:

$$w = A(u, v) = |2^{l_1}u + (-1)^{l_2}v|2^{-l}$$

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 substractor 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, $T = \{t_1, \ldots , t_m\} \subset \mathbb{N}$, 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 $T = \{t_1, \ldots , t_m\} \subset \mathbb{N}$, 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: \((R, T) = \text{Synthesize}(R, T)\)
3: while \( T \neq \emptyset \) do
4: for \( j = 1 \) to \( 2^{w+i} - 1 \) step 2 do
5: if \( j \notin R \) and \( j \notin T \) then
6: \( \text{impcost} = \text{ComputeCost}\{\{j\}, R\} \)
7: if \( \text{impcost} \neq 0 \) then
8: \( A \leftarrow R \cup \{j\} \)
9: \( \text{impcost} = \text{ComputeTCost}(T, A) \)
10: \( \text{iccost}_{j} = \text{impcost}_{j} + \text{impcost}_{T} \)
11: Find the intermediate constant, \( ic_{j} \), with the minimum \( \text{iccost}_{j} \) cost among all possible constants, \( j \)
12: \( R = R \cup \{j\} \)
13: \((R, T) = \text{Synthesize}(R, T)\)
14: \( D = \text{SynthesizeMinArea}(R) \)
15: return \( D \)

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

ComputeCost(\(\{\}, C\))
1: \( \text{cost}_{0} = 0 \)
2: for all operations \( c = [2^{l}u + (-1)^{y}2^{v}v][2^{-r}] \), 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 \( \text{cost}_{c} \)
4: return \( \text{cost}_{c} \)

ComputeTCost(\(B, C\))
1: \( \text{cost}_{B} = 0 \)
2: repeat
3: \( \text{isadded} = 0 \)
4: for each \( b_{k} \in B \) do
5: \( \text{cost}_{b_{k}} = \text{ComputeCost}(\{b_{k}\}, C) \)
6: if \( \text{cost}_{b_{k}} \neq 0 \) then
7: \( \text{isadded} = 1 \)
8: \( C \leftarrow C \cup \{b_{k}\} \)
9: \( B \leftarrow B \backslash \{b_{k}\} \)
10: \( \text{cost}_{b} = \text{cost}_{b} + \text{cost}_{b_{k}} \)
11: until \( \text{isadded} = 0 \)
12: for each \( b_{k} \in B \) do
13: \( \text{cost}_{b_{k}} = \text{cost}_{b} + \text{maxcost}(b_{k}) \)
14: return \( \text{cost}_{b} \)

SynthesizeMinArea(R)
1: Find all possible implementations of target and intermediate constants using the \( \text{GenerateImp}(R) \) 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: \( A \leftarrow \{1\} \), \( R \leftarrow R \backslash \{1\} \)
2: repeat
3: for each \( r_{l} \in R \) do
4: \((B, C) = \text{Synthesize}(A, \{r_{l}\})\)
5: if \( C = \emptyset \) then
6: Find all operations, \( r_{l} = [2^{l}u + (-1)^{y}2^{v}v][2^{-r}] \), where \( u, v \in A \) and determine their implementation costs under the digit-serial architecture
7: \( A \leftarrow A \cup \{r_{l}\} \)
8: \( R \leftarrow R \backslash \{r_{l}\} \)
9: until \( R = \emptyset \)

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
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 UCMLogic 0.18μm 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 μm²) 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.

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

<table>
<thead>
<tr>
<th>number of constants ($n$)</th>
<th>oper</th>
<th>shift</th>
<th>Icost</th>
<th>oper</th>
<th>shift</th>
<th>Icost</th>
<th>oper</th>
<th>shift</th>
<th>Icost</th>
<th>oper</th>
<th>shift</th>
<th>Icost</th>
<th>oper</th>
<th>shift</th>
<th>Icost</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>13.5</td>
<td>28.6</td>
<td>3735.6</td>
<td>15.1</td>
<td>32.5</td>
<td>3832.1</td>
<td>12.8</td>
<td>23.9</td>
<td>3181.2</td>
<td>7.1</td>
<td>13.7</td>
<td>2380.7</td>
<td>13.1</td>
<td>18.9</td>
<td>2875.8</td>
</tr>
<tr>
<td>20</td>
<td>26.5</td>
<td>42.5</td>
<td>6044.5</td>
<td>22.2</td>
<td>36.5</td>
<td>5111.4</td>
<td>21.4</td>
<td>33.2</td>
<td>4834.7</td>
<td>29.2</td>
<td>19.7</td>
<td>5239.6</td>
<td>21.5</td>
<td>26.5</td>
<td>4493.4</td>
</tr>
<tr>
<td>30</td>
<td>36.7</td>
<td>53.3</td>
<td>8091.3</td>
<td>30.1</td>
<td>39.8</td>
<td>6432.9</td>
<td>30.1</td>
<td>39.3</td>
<td>6406.5</td>
<td>39.9</td>
<td>23.1</td>
<td>6975.4</td>
<td>30.1</td>
<td>28.8</td>
<td>5847.3</td>
</tr>
<tr>
<td>40</td>
<td>46.0</td>
<td>60.6</td>
<td>9827.4</td>
<td>39.4</td>
<td>47.9</td>
<td>8195.1</td>
<td>39.4</td>
<td>47.9</td>
<td>8195.1</td>
<td>50.2</td>
<td>24.9</td>
<td>8548.1</td>
<td>39.4</td>
<td>28.5</td>
<td>7140.9</td>
</tr>
<tr>
<td>50</td>
<td>55.9</td>
<td>68.7</td>
<td>11672.5</td>
<td>49.0</td>
<td>54.4</td>
<td>9922.0</td>
<td>49.0</td>
<td>53.2</td>
<td>9860.5</td>
<td>59.8</td>
<td>27.6</td>
<td>10072.3</td>
<td>49.0</td>
<td>26.5</td>
<td>8411.8</td>
</tr>
<tr>
<td>60</td>
<td>65.4</td>
<td>77.0</td>
<td>13482.9</td>
<td>59.0</td>
<td>59.7</td>
<td>11652.5</td>
<td>59.0</td>
<td>58.7</td>
<td>11597.7</td>
<td>69.8</td>
<td>29.3</td>
<td>12155.3</td>
<td>59.0</td>
<td>23.9</td>
<td>9693.1</td>
</tr>
<tr>
<td>70</td>
<td>75.1</td>
<td>83.3</td>
<td>15238.3</td>
<td>68.2</td>
<td>62.6</td>
<td>13143.5</td>
<td>68.2</td>
<td>61.8</td>
<td>13102.9</td>
<td>79.5</td>
<td>31.0</td>
<td>13806.2</td>
<td>68.2</td>
<td>23.1</td>
<td>10980.6</td>
</tr>
<tr>
<td>80</td>
<td>83.2</td>
<td>89.5</td>
<td>16716.1</td>
<td>77.7</td>
<td>74.1</td>
<td>15111.4</td>
<td>77.7</td>
<td>74.5</td>
<td>15125.5</td>
<td>87.6</td>
<td>31.3</td>
<td>14275.5</td>
<td>77.7</td>
<td>20.7</td>
<td>12202.1</td>
</tr>
<tr>
<td>90</td>
<td>92.7</td>
<td>99.1</td>
<td>18646.6</td>
<td>86.8</td>
<td>74.7</td>
<td>16466.7</td>
<td>86.8</td>
<td>73.8</td>
<td>16414.8</td>
<td>97.2</td>
<td>32.0</td>
<td>15690.6</td>
<td>86.8</td>
<td>20.9</td>
<td>13504.9</td>
</tr>
<tr>
<td>100</td>
<td>101.7</td>
<td>104.7</td>
<td>20204.0</td>
<td>96.5</td>
<td>84.0</td>
<td>18343.9</td>
<td>96.5</td>
<td>86.0</td>
<td>18442.3</td>
<td>106.4</td>
<td>32.4</td>
<td>21704.1</td>
<td>96.5</td>
<td>19.8</td>
<td>14827.8</td>
</tr>
</tbody>
</table>

Table 2: Filter specifications.

<table>
<thead>
<tr>
<th>Filter</th>
<th>pass</th>
<th>stop</th>
<th>#tap</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>0.15</td>
<td>200</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>0.15</td>
<td>240</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>0.10</td>
<td>0.25</td>
<td>180</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>0.10</td>
<td>0.25</td>
<td>200</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>0.10</td>
<td>0.20</td>
<td>240</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>0.10</td>
<td>0.20</td>
<td>300</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>0.15</td>
<td>0.25</td>
<td>200</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>0.15</td>
<td>0.25</td>
<td>240</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>0.20</td>
<td>0.25</td>
<td>240</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>0.20</td>
<td>0.25</td>
<td>300</td>
<td>16</td>
</tr>
</tbody>
</table>

denote respectively the area in μm², 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.

2The FIR filter instances are available at http://algos.inesc-id.pt/multicon.
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, as opposed to accounting 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 using the algorithms designed for the MCM problem and the optimization of area problem in a digit-serial MCM operation at gate-level.

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 Multiplications) PTDC/EIA-EIA/103532/2008 and under INESC-ID multiannual funding through the PIDDAC Program funds.

7. REFERENCES


