Optimization of Area under a Delay Constraint in Digital Filter Synthesis Using SAT-Based Integer Linear Programming

Levent Aksoy
Istanbul Technical University
Department of Electronics Engineering
Maslak, 34469, Istanbul, Turkey
+90-212 2853637
levent@ehb.itu.edu.tr

Eduardo Costa
Universidade Catolica de Pelotas
Rua Félix da Cunha, 412, Centro
Pelotas-RS, Brazil
+55-533 2848216
ecosta@atlas.ucpel.tche.br

Paulo Flores, Jose Monteiro
INESC-ID/IST
Rua Alves Redol, 9, 1000-029
Lisbon, Portugal
+351-21 3100300
{pff, jcm}@inesc-id.pt

ABSTRACT

In this paper, we propose an exact algorithm for the problem of area optimization under a delay constraint in the synthesis of multiplierless FIR filters. To the best of our knowledge, the method presented in this paper is the only exact algorithm designed for this problem. We present the results of the algorithm on real-sized filter instances and compare with an improved version of a recently proposed exact algorithm designed for the minimization of area. We show that in many cases delay can be minimized without any area penalty. Additionally, we describe two approximate algorithms that can be applied to instances which cannot be solved, or take too long, with the exact algorithm. We show that these algorithms find similar solutions to the exact algorithm in less CPU time.

Categories and Subject Descriptors

B.2.0 [**Arithmetic and Logic Structures**]: General.

General Terms: Algorithms, design.

Keywords: Multiple constant multiplication, multiplierless digital filter design, area optimization, delay optimization.

1. INTRODUCTION

Finite impulse response (FIR) digital filters are widely used in digital signal processing by virtue of stability and easy implementation. The problem of designing FIR filters has received a significant amount of attention during the last decade, as the filters require a large number of multiplications, leading to excessive area, delay, and power consumption even if implemented in a full custom integrated circuit. Early works have focused on the design of filters with minimum area by replacing the multiplication operations with constant coefficients by addition, subtraction, and shifting operations as shown in Figure 1. Since shifts are free in terms of hardware, the design problem can be defined as the minimization of the number of addition/subtraction operations to implement the coefficient multiplications. Also, to reduce the complexity of the design, coefficients are expressed in canonical signed digit (CSD) [7] or represented in minimum signed digit (MSD) [12].

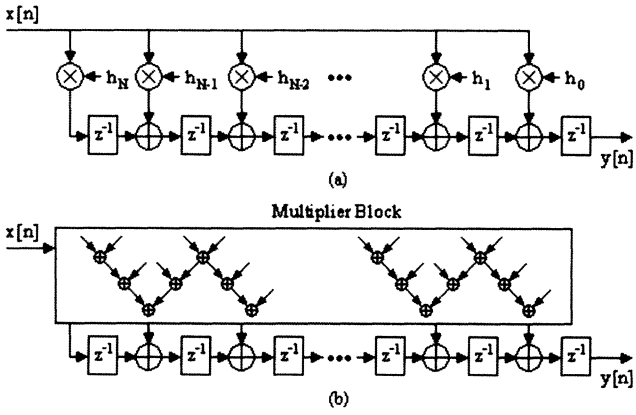

Figure 1. FIR filter structure: (a) General transposed form; (b) Replacing coefficient multiplications by a multiplier block.

Many algorithms [3,4,6,7,12,13] have been proposed to minimize the number of addition/subtraction operations in the multiplier block of the filter. Most of these algorithms are heuristic, providing no indication as to how far from the optimum their solution is. An exact algorithm for the maximal sharing of the partial terms for more than one coefficient is given in [4]. In this algorithm, the filter design problem is defined as a binate covering problem, a special case of 0-1 linear integer programming (ILP) problem where every constraint is interpreted as a propositional clause. The problem is modeled as a combinational network that covers all possible partial terms that may be used to generate the set of coefficients. In this way, a network that consists of only AND and OR gates is computed. Then, all the conjunctive normal form (CNF) formulas for each gate output are obtained. Each clause in the CNF formula is defined as a constraint by expressing each clause as a linear inequality. Finally, an objective function to be minimized is constructed. A generic SAT-based 0-1 ILP solver is used to obtain the exact solution.

However, all these algorithms do not consider the delay of the filter, the most important design parameter for high performance systems. Despite the large number of algorithms proposed for minimum area, there are a few methods [2,8,9] that deal with both area and delay. In [9], initially, the number of addition/subtraction operations is reduced and then, a set of transformations in an iterative loop is

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

used to reduce the delay. In [2,8], while minimizing area, delay is also considered in the selection criterion.

Although these algorithms take delay into account when they minimize the area, they are heuristic. In this work, we propose an exact algorithm for the problem of optimizing area under a delay constraint. We start by improving the exact algorithm of [4] by changing the structure of the network given to the 0-1 ILP solver, lowering significantly the computation time and permitting its applicability to larger filter instances. Then, we implement an exact algorithm for the problem of area optimization under a delay constraint. In this algorithm, we add extra constraints to the 0-1 ILP problem to ensure that the solution to be obtained by the improved exact algorithm designed for minimum area has the minimum delay. Also, we present two approximate algorithms that are able to compute good solutions for larger filter instances in reasonable CPU time. The first approximate algorithm is based on the exact algorithm, but deletes some operations that can increase the minimum delay of the network, thus reducing the search space to be explored. The second algorithm reduces the delay in the found minimum area solution by replacing the operations that present a higher delay with operations that meet the desired delay, in a post-processing phase. This is similar to [9] except that we start from an exact minimum area solution.

The rest of the paper is organized as follows. In Section 2, definitions used throughout the paper are given. In Section 3, the implementation of the exact algorithm for the minimization of area and the structure of the network constructed by the algorithm are presented. An exact algorithm for the problem of optimization of area under a delay constraint is described in Section 4 and the two approximate algorithms for this problem are presented in Section 5. Experimental results are given in Section 6. Finally, the paper concludes in Section 7.

2. DEFINITIONS

An instance C of a *covering problem* is defined as follows:

$$\text{Minimize} \sum_{j=1}^{n} c_j.x_j \qquad \text{(I)}$$

$$\text{subject to } \mathbf{A.x} \geq b, \ \mathbf{x} \in \{0,1\}^n \qquad \text{(II)}$$

where c_j is a nonnegative integer cost value associated with variable x_j, $1 \leq j \leq n$, in the objective function (I) and $\mathbf{A.x} \geq b$, denotes the set of m linear constraints (II). If every entry in the $m \times n$ matrix A is in the set $\{0,1\}$ and $b_i = 1$, $1 \leq i \leq m$, then C is an instance of the unate covering problem. Moreover, if the entries a_{ij} of A belong to $\{-1, 0, 1\}$ and $b_i = 1 - |\{a_{ij}: a_{ij} = -1, 1 \leq j \leq n\}|$, then C is an instance of the binate covering problem. Observe that, if C is an instance of the binate covering problem, then each constraint can be interpreted as a propositional clause.

A *propositional formula* φ in CNF denotes a Boolean function $f:\{0,1\}^n \rightarrow \{0,1\}$. The CNF formula φ consists of a conjunction of propositional clauses where each clause ω is a disjunction of literals and a literal l_j is either a variable x_j or its complement $\neg x_j$. If a literal assumes value 1, then the clause is satisfied. If all literals of a clause assume value 0, then the clause is unsatisfied. The CNF formula of a combinational network is the conjunction of the CNF formulas for each gate output, where the CNF formula of each gate denotes the valid input-output assignments to the gate. The derivation of CNF formulas of the basic gates can be found in [10]. A clause ω to be satisfied in the formula, $l_1+...+l_k$, $k \leq n$, can be interpreted as a linear inequality, $l_1+...+l_k \geq 1$, and the complement of variable x_j can be represented by $1-x_j$.

The *Canonical Signed Digit* (CSD) representation is a signed digit system with the digit set {1,0,2} where 2 denotes -1. The CSD representation is unique and has two main properties: (i) the number of non-zero digits is minimal, (ii) two non-zero digits are not adjacent. Hardware requirements are reduced because the numerical values are represented with a maximal number of zero digits. This representation is widely used in multiplierless implementations, because it reduces the number of non-zero digits by 33%, on average compared with the binary representation [5].

The *Minimum Signed Digit* (MSD) representation is obtained by dropping the second property of the CSD representation. Thus, a constant can have several MSD representations, but all with a minimum number of non-zero digits. For example, the value 6 is represented using 4 digits in CSD as 1020, but both 1020 and 0110 are valid representations in MSD.

The *problem of optimizing area under a delay constraint* [8] can be defined as the minimization of the number of adders/subtracters such that a user-specified delay is not exceeded. As the delay is dependent on several implementation issues, such as circuit technology, placement, and routing, we consider the delay as the number of adder-steps, which denotes the maximal number of adders/subtracters in series to produce any multiplication as given in [9]. Since the definition of adder-steps is identical to the definition of level in combinational circuits, we use both definitions interchangeably in this paper.

3. AN EXACT ALGORITHM FOR MINIMUM AREA

In this section, we present the implementation of the exact algorithm designed for the minimization of area and describe the improvements we have made. The basic model description of the algorithm can be found in [4]. The implemented algorithm can be used for any type of coefficient representation: binary, CSD or MSD. However, using the MSD representation results in a more general algorithm, because several representations may exist for the same value. We will describe first, the MSD implementation of the algorithm and then, we summarize the changes for binary or CSD representations.

3.1 The Implementation

In the preprocessing phase of the algorithm, all coefficients are converted to positive and then, made odd by successive divisions by 2, i.e., we shift all coefficients to the right so that zero bits on the right are eliminated. Each new resulting coefficient is added to a set called *Iset*. This set represents the minimum number of coefficients necessary to be synthesized. For each element i in the *Iset*, all MSD representations are determined using $\lceil \log_2(i) \rceil + 1$ bits and inserted in the *Cset*. Therefore, the *Cset* begins with all the MSD representations of the coefficients as in [12]. However, during the execution of our algorithm the *Cset* will be augmented with MSD representations of partial terms. Then, we enter in the main algorithm loop where an element c, removed from *Cset* and representing a number i, is processed to determine its covers: 1) compute all non-symmetric partial term pairs that covers the element c; 2) convert to positive and make odd each element of the cover pair; 3) add each cover pair to the corresponding set of covers

of the element being processed, $Aset_i$; 4) add the MSD representations of each cover to the *Cset*, if the representation has not been processed yet and it is not already in the set. Covers with only one non-zero digit are skipped. This loop is repeated until there are no more elements in the *Cset*. The pair of elements in each $Aset_i$ represents all possible alternatives of partial terms for a value i based on its MSD representations.

The final 0-1 ILP optimization model is generated in three steps: 1) for each pair element in $Aset_i$, generate the corresponding AND gate, with an additional input that represents an optimization variable for the AND gate. Generate an OR gate for the value i with the outputs of all the ANDs resulting from $Aset_i$; 2) identify all the OR outputs that represent a coefficient (values belonging to *Iset*) and force their outputs to be 1; 3) generate the objective function to be minimized. The objective function is a linear combination of the optimization variables at inputs of the AND gates.

Note that this algorithm can be easily adapted to obtain the 0-1 ILP optimization model with different coefficient representations. When the MSD representation of a coefficient or partial term is determined, one needs only to compute a binary or CSD representation instead. Moreover, mixed representations, i.e., binary and CSD or binary and MSD, can also be computed and added to the *Cset*.

The algorithm designed for minimization of area, which we refer to as Minimum Area Algorithm (MAA), is implemented by taking the cost value of each optimization variable, representing an adder/subtracter, in the objective function as 1. To obtain an exact solution, we use an efficient SAT-based solver [11] that incorporates several advanced optimization techniques and has been applied to several classes of problems.

3.2 The Network

The combinational network constructed by the algorithm only includes AND and OR gates. In this network, an AND gate represents an addition/subtraction operation and an OR gate combines the possible ways of implementation of a partial term. The primary inputs of the network represent the filter input or its shifted versions. The primary outputs of the network are the OR gate outputs that generate the coefficients of the filter, which are forced to evaluate to 1. The number of inputs for each AND gate is three: two are either primary inputs or OR gate outputs (partial terms); the third is an optimization variable. The inputs of an OR gate are the outputs of AND gates associated with the partial term. As an example, assume that the value 15 (in binary, 1111) is the coefficient of the filter to be synthesized. In Figure 2, the network generated by the algorithm is given with the elimination of 1-input OR gates for 3, 5, and 9 partial terms.

3.3 Improvements to the Algorithm

In [4], an optimization variable in the objective function is used to represent a partial term with an additional AND gate and by doing so, a reduction in the number of variables and clauses is aimed. In the MAA, an optimization variable represents an operation to handle the delay constraints. Also, this is a more adequate representation of a 0-1 ILP problem that is generated for minimum area. Besides, in [4], the primary inputs are represented with different variables even if they represent the same shifted version of the filter input. In the MAA, all primary inputs are represented with the number 1 and a 0-1 ILP problem with less number of variables is obtained. This reduction in the number of primary inputs also accelerates the preprocessing phase of the 0-1 ILP solver [11] where detection of necessary assignments and simplification techniques are used. In the design of the algorithms for optimization of area under a delay constraint, the improved exact algorithm for minimum area, MAA, is used.

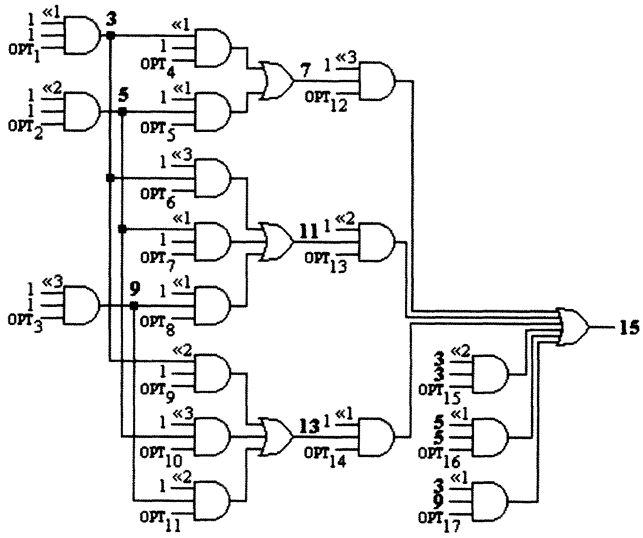

Figure 2. The network generated for the coefficient 15 in binary.

4. AN EXACT ALGORITHM FOR OPTIMIZATION OF AREA UNDER A DELAY CONSTRAINT

In this section, we describe the exact algorithm designed for the optimization of area under a delay constraint. The exact algorithm can find a solution with either the minimum delay of the network or a maximum user-specified delay constraint. In this paper, by using as constraint the minimum delay of the network, the proposed algorithm is also an exact algorithm for minimum delay.

Note that a partial term in the network can be implemented with operations that have different adder-steps. For a partial term with N non-zero digits, the minimum latency implementation of a partial term has $\lceil \log_2 N \rceil$ adder-steps and the maximum latency implementation of a partial term has N-1 adder-steps. In the network, an OR gate associated with the partial term gathers all of these operations. So, a partial term can be generated with the number of adder-steps ranging from its minimum to maximum latency implementations. As can be seen from Figure 2, the coefficient 15 can be implemented using an operation with OPT_{12} optimization variable yielding a maximum of 3 adders-steps or using an operation with OPT_{15} optimization variable yielding a minimum of 2 adder-steps. Hence, the coefficient 15 has minimum 2 and maximum 3 adder-steps implementations. In the preprocessing phase of the algorithm, we find the minimum and maximum level of each operation and partial term in the network by traversing from primary inputs to primary outputs. Then, we find the minimum delay of the network, *min_delay*, by computing the maximum value of the minimum levels of the primary outputs.

An exact algorithm can be designed using this information by finding the paths in the network that exceed *min_delay*. We add these paths as constraints to the 0-1 ILP problem preventing them

from being selected for the final solution. In this algorithm, which we call Minimum Area with Minimum Delay Algorithm (MAMDA), initially, partial terms whose maximum levels are higher than *min_delay* are determined and stored in a set called *Pset*. Then, for each element in the *Pset*, *Pset(i)*, if each operation that implements *Pset(i)* has minimum level higher than *min_delay* and *Pset(i)* is not used to implement other partial terms, then this operation is deleted from the network. Otherwise, if each operation has maximum level higher than *min_delay*, then this operation is added to a new set called $path_j$ as an initial node. Also, this operation is added to a set called *Oset* with a target level, *min_delay*-1, and the associated path identifier, *j*. The paths and *Oset* are formed, when all elements in the *Pset* are considered. In the MAMDA, the paths are constructed in a breadth-first manner as follows. In an iterative loop, an operation with its target level, *level(i)*, and the associated path identifier is removed from the *Oset*. For each input of the operation (a partial term), if each operation that implements the partial term has minimum level higher than *level(i)*, then the associated path is constructed, i.e., the terminal node of the associated path is found. Otherwise, if each operation has maximum level higher than *level(i)*, then this operation is added to the associated path and inserted into the *Oset* with its associated target level, *level(i)*-1, and path identifier. This loop iterates until there is no element left in the *Oset*.

Suppose that a situation given in Figure 3 is encountered when finding the paths that exceed *min-delay*. In this figure, operations and partial terms are labeled with letters inside the gates. The minimum and maximum levels of operations are given with a-b pair above the gates respectively. $path_n$ includes the operations that exceed *min-delay* determined so far to the operation G. Also, operations are shown without optimization variables for the sake of clarity in this figure.

Assume that the operation G with its target level, *level(G)*=4, and associated path identifier, *n*, is removed from the *Oset* considering the partial term H as the input of G. The algorithm finds a terminal node, K, for the $path_n$ and constructs the path. Also, it adds a new path as $path_{n+1}$ by inserting the operation I to the path. Since the maximum level of the operation I is higher than *level(G)*, there is a path(s) that exceeds *min_delay*. So, the operation I with its target level, *level(I)=level(G)*-1, and associated path identifier, *n+1*, is added to the *Oset*. In Figure 3, the paths that can increase the minimum delay are highlighted.

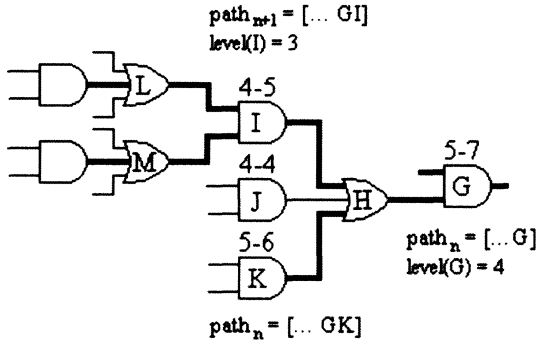

Figure 3. An illustrative example when finding the paths that increase the minimum delay of the network.

After all paths are found, for each path a delay constraint, $-OPT_1 - OPT_2 - \ldots - OPT_m \geq 1 - m$, where OPT_j, $1 \leq j \leq m$, denotes the optimization variable of an operation in the path and *m* is the number of operations in the path, is added to the 0-1 ILP problem. This constraint expresses that the operations in the path must not be included in the solution altogether, by guaranteeing that the solution to be found by the 0-1 ILP solver has the minimum delay and allowing the possible sharing of partial terms that are inputs of the operations in the path with other partial terms. Then, the network with the delay constraints is given to the 0-1 ILP solver to find a solution with minimum area.

5. APPROXIMATE ALGORITHMS

In this section, we present two approximate algorithms for optimization of area under a delay constraint that are more efficient in terms of CPU time than the proposed exact algorithm.

5.1 Minimum Area with Minimum Delay Heuristic

This algorithm, which we refer to as Minimum Area with Minimum Delay Heuristic (MAMDH), deletes the operations that can cause the network to be implemented with a delay higher than the minimum delay of the network, thus reducing the search space to be explored. This algorithm removes the operations from the network that are the terminal nodes of the paths constructed in the MAMDA. As an example, the operation K given in Figure 3 is removed from the network in the MAMDH. Then, this network is given to the 0-1 ILP solver to find a solution with minimum area. The MAMDH is not exact because when it deletes an operation from the network, it also deletes the possible sharing of partial terms even if the possible sharing does not increase the minimum delay of the network. However, it may greatly simplify the 0-1 ILP problem to be solved as shown in Section 6.

5.2 Post Process for Minimum Delay

Optimization of area under a minimum delay can be realized in a post-processing phase as used in logic synthesis systems [1] after a solution with minimum area is found. This method called Post Process for Minimum Delay (PPMD) is applied after a solution with minimum area is obtained by the MAA. In the preprocessing phase, we find the minimum level of each operation and partial term in the network given to the 0-1 ILP solver. Then, we find the minimum delay of the network. In the post-processing phase, each partial term in the exact solution that increase the minimum delay of the network is implemented by an operation selected among the set of possible implementations of the partial term that meets the desired delay. These possible implementations are the operations that ensure the minimum delay of the network. PPMD uses a greedy method in the selection of operations. In this method, the operations whose inputs are the partial terms already existing in the solution are favored.

6. EXPERIMENTAL RESULTS

In this section, we present the results of the algorithms described in this paper. Also, we give the results of a heuristic method [12] designed for minimum area in order to compare with the exact algorithm, MAA. Since [2,9] use different models to obtain the partial terms and [8] represents the coefficients of filters with 24 digits precision that is hard for the exact algorithm, we could not compare the MAMDA with these heuristic methods.

Table 1. Characteristics of the FIR filters and 0-1 ILP problem sizes of the algorithms

Filter	Filter Specification				MAA			MAMDA			MAMDH		
	pass	stop	#tap	width	vars	clauses	optv	vars	clauses	optv	vars	clauses	optv
1	0.20	0.25	120	8	417	996	190	331	781	147	331	781	147
2	0.10	0.25	100	10	920	2208	424	920	2332	424	800	1908	364
3	0.15	0.25	40	12	1931	4671	909	1931	5386	909	1453	3476	670
4	0.20	0.25	80	12	2940	7161	1398	2940	8346	1398	2184	5271	1020
5	0.24	0.25	120	12	1717	4150	800	1717	4470	800	1397	3350	640
6	0.15	0.25	60	14	6653	16317	3215	6499	18469	3138	5165	12597	2471
7	0.15	0.20	60	14	2810	6789	1317	2810	7460	1317	2188	5234	1006
8	0.15	0.20	100	16	37370	92277	18289	36038	97895	17623	31280	77052	15244
9	0.10	0.15	60	14	5106	12426	2431	5106	14798	2431	3710	8936	1733
10	0.10	0.15	100	16	41177	101557	20112	39825	109799	19436	33629	82687	16338
Avg.	-	-	-	-	100%	100%	100%	97.1%	108.5%	97%	81.3%	81%	80.7%

Table 2. Results of the algorithms on the given filters in MSD representation

Filter	MAA			MAMDA			MAMDH			PPMD		[12]		
	Adders	Steps	CPU	Adders	Steps	CPU	Adders	Steps	CPU	Adders	Steps	Adders	Steps	CPU
1	10	3	0.2	10	2	0.1	10	2	0.4	10	2	10	3	0.0
2	18	3	1.3	18	3	1.0	18	3	0.6	18	3	18	4	0.6
3	16	3	13.1	16	3	7.3	16	3	9.0	16	3	18	4	1.6
4	29	4	14.8	29	3	20.9	29	3	9.5	29	3	29	4	1.4
5	34	3	4.9	34	3	5.9	34	3	2.6	34	3	34	3	0.7
6	22	4	79.8	22	3	130.2	22	3	75.8	23	3	22	4	1.3
7	34	3	20.7	34	3	27.5	34	3	7.3	34	3	35	3	92.5
8	47	4	3604.7	47	3	3829.6	47	3	2209.5	47	3	52	5	629.9
9	33	4	320.2	33	3	1293.8	33	3	56.6	33	3	37	4	21.1
10	*49*	*4*	*7200.1*	*50*	*3*	*7200.1*	*50*	*3*	*7200.1*	*50*	*3*	50	5	76.2
Avg.	100%	120.7%	90%	100.3%	100%	100%	100.3%	100%	76.5%	100.7%	100%	104.5%	134.5%	6.6%

The results of the algorithms described in this paper are based on filter instances where the coefficients were computed with MATLAB using the *remez* algorithm. The specifications of filters and the problem sizes of the algorithms are given 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. The next three sets of three columns indicate the size of the 0-1 ILP problems for the MAA, MAMDA, and MAMDH in terms of the number of variables, clauses and optimization variables. Note that the problems generated by the MAMDA include more constraints than the MAA, since delay constraints are added to find a solution with minimum area under a delay constraint. Also, the MAMDA can represent a problem using fewer variables as a consequence of operation eliminations. On the other hand, the sizes of problems generated by the MAMDH are much smaller than the MAMDA, because some operations are deleted to guarantee the minimum delay.

The results of the algorithms described in this paper and the heuristic method [12] on the filter instances in MSD representation are given in Table 2. In this table, *Adders* denotes the number of adders/subtracters required to implement the filter, *Steps* denotes the maximum depth in terms of adder-steps for all coefficients, and *CPU* is the CPU time in seconds that is used by the 0-1 ILP solver to compute the exact solutions on a PC with dual Pentium Xeon at 2.4GHz, with 4GB of main memory, running Linux. The allowed CPU time was 7200 seconds. In Table 2, the italic results indicate the non-optimal solutions found in the allowed CPU time. Note that most of the filter instances are solved in a very small period of CPU time.

In this experiment, we observe that the MAMDA finds the minimum delay for each filter and minimizes the number of operations under this constraint with no area overhead with respect to the MAA, except Filter 10. Since additional delay constraints are added by the MAMDA, the required time to find a solution is higher than the MAA. Since the MAMDH reduces the problem size on average 19% with respect to the MAA problem size, this results in a significant reduction of CPU time. Also, the MAMDH finds identical solutions with respect to the MAMDA. Besides, although the PPMD is implemented as a post-processing phase after a solution with minimum area is obtained by the MAA and is a greedy method, the results of the PPMD are promising. Note that the CPU time for this algorithm is not given because it is practically the same as the MAA, since the post-processing phase is negligible with respect to the 0-1 ILP solver time. Also, all algorithms proposed for the problem of optimizing area under a delay constraint give better results on the average number of adders than the heuristic algorithm designed for the minimization of area. This is because all these algorithms use the exact algorithm designed for minimum area. One interesting result of this work is that we can actually obtain exact results for real-sized filter instances. However, there are still those for which only a heuristic algorithm is able to run.

In a different experiment, coefficients with 10-digits precision were generated randomly. The number of coefficients ranges between 10 and 70 and each of them includes 30 instances. The results of the algorithms on the randomly generated instances are given in Figure 4. Average numbers of adder-steps in the solutions found by the proposed algorithms are presented in Figure 4(a). Observe that a great improvement in delay with respect to the MAA is obtained. To compare the area overhead between the algorithms, the average differences of the number of operations in the solutions between the MAA and other algorithms are given in Figure 4(b). Observe that, the cost of obtaining the minimum delay solution is very small, less than 0.5 operations on average, when compared to the MAA. Also, the MAMDA obtains better solutions than the approximate algorithms.

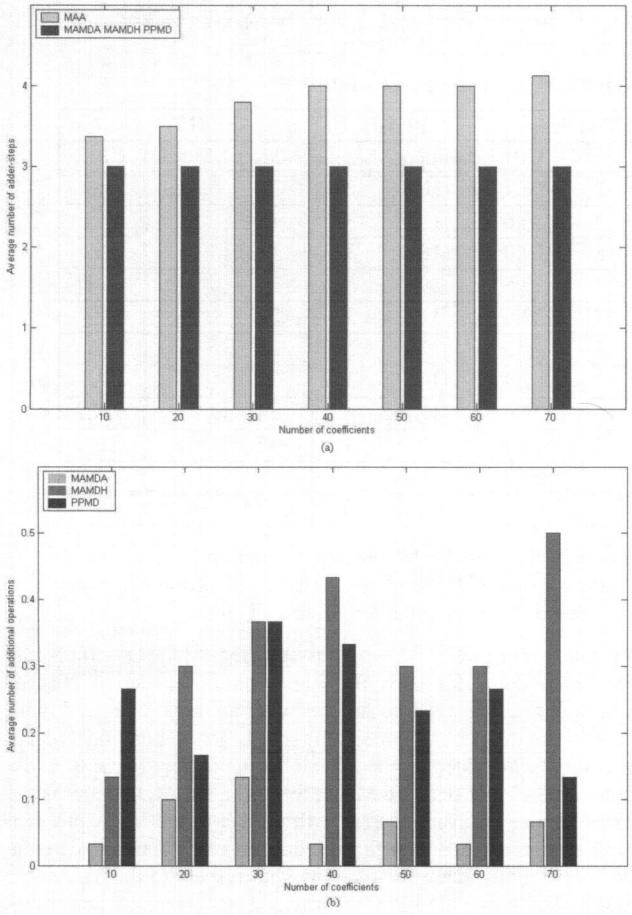

Figure 4. Results on randomly generated coefficients: (a) average number of adder-steps; (b) area overhead.

7. CONCLUSIONS

In this paper, we examined the problem of optimizing area under a delay constraint and proposed an exact algorithm for this problem. Initially, we improved the exact algorithm designed for minimum area. To introduce the delay constraint, new clauses are added to the model. Since the number of constraints is increased in the exact algorithm and so, the required time to find a solution, we also proposed two approximate algorithms for this problem. It is shown by the experimental results that delay can be minimized practically with no area overhead in a reasonable time for real-sized instances. We are currently working on the application of the exact algorithms presented in this paper to different types of filters and the exploration of more general representations for the coefficients.

8. ACKNOWLEDGMENTS

This research was supported in part by the portuguese FCT under program POCTI.

9. REFERENCES

[1] Brayton, R. K., Rudell, R., Sangiovanni-Vincentelli, A., and Wang, A. R. MIS: a multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design*, Vol. 6, No. 6, (1987), 1062-1081.

[2] Costa, E., Flores, P., and Monteiro, J. Maximal sharing of partial terms in MCM under minimal signed digit representation. In *Proc. of ECCTD*, 2005, 221-224.

[3] Dempster, A. G. and MacLeod, M. Digital filter design using subexpression elimination and all signed-digit representations. In *Proc. of ISCAS*, 2004, 24-26.

[4] Flores, P., Monteiro, J., and Costa, E. An exact algorithm for the maximal sharing of partial terms in multiple constant multiplications. In *Proc. of ICCAD*, 2005, 13-16.

[5] Garner, H. L. Number systems and arithmetic. *Advances in Computers*, Vol. 6, (1965), 131-194.

[6] Gustafsson, O., Dempster, A. G., and Wanhammer, L. Extended results for minimum-adder constant integer multipliers. In *Proc. of ISCAS*, 2002, 73-76.

[7] Hartley, R. I. Subexpression sharing in filters using canonic signed digit multipliers. *IEEE Transactions on Circuits and Systems II*, Vol. 43, No. 10, (1996), 677-688.

[8] Hosangadi, A., Fallah, F., and Kastner, R. Simultaneous optimization of delay and number of operations in multiplierless implementation of linear systems. In *Proc. of IWLS*, 2005.

[9] Kang, H. –J. and Park, I. –C. FIR filter synthesis algorithms for minimizing the delay and the number of adders. *IEEE Transactions on Circuits and Systems II*, Vol. 48, No. 8, (2001), 770-777.

[10] Larrabee, T. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 1, (1992), 4-15.

[11] Manquinho, V. and Marques-Silva, J. P. Effective lower bounding techniques for pseudo-boolean optimization. In *Proc. of DATE*, 2005, 660-665.

[12] Park, I. –C. and Kang, H. –J. Digital filter synthesis based on minimal signed digit representation. In *Proc. of DAC*, 2001, 468-473.

[13] Pasko, R., Schaumont, P., Derudder, V., Vernalde, S., and Durackova, D. A new algorithm for elimination of common subexpressions. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 18, No. 1, (1999), 58-68.