Minimum Number of Operations under a General Number Representation for Digital Filter Synthesis

Levent Aksoy
Istanbul Technical University
Istanbul, Turkey

Eduardo Costa
Universidade Catolica de Pelotas
Pelotas, Brazil

Paulo Flores
INESC-ID/IST
Lisbon, Portugal

José Monteiro
INESC-ID/IST
Lisbon, Portugal

***Abstract*— In this work, we introduce an algorithm for the optimization of the number of operations in the multiplier block of a digital filter based on a general number representation for the coefficients. In common subexpression elimination algorithms, constants are generally represented with the minimum number of non-zero digits based on their CSD, or MSD representations. We observe that these representations may yield a solution far from the minimum. The general number representation used in our algorithm considers a much larger set of alternative implementations of a constant, which includes the CSD and MSD representations. To cope with the increased search space, we propose model simplification and problem reduction techniques. In this paper, we show that the proposed exact algorithm using general number representation achieves a significant reduction in the number of operations, which can be up to 15% with respect to the solutions obtained under MSD representation.**

I. 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. The proposed methods have focused on the design of filters with minimum area by replacing the multiplication operations with constant coefficients by addition, subtraction, and shifting operations. 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. In fact, this is known, more generally, as the Multiple Constant Multiplications (MCM) problem.

The proposed algorithms for the optimization of the number of operations in digital filter design can be categorized in two classes: common subexpression elimination (CSE) and graph-based techniques. In CSE algorithms, constants are represented in a number representation, commonly, canonical signed digit (CSD) and minimal signed digit (MSD). Both CSD and MSD representations use a signed digit system with the digit set $\{\overline{1},0,1\}$, where $\overline{1}$ denotes -1, and have the property that the number of non-zero digits is minimum. The CSD representation provides a unique representation for every constant, since two non-zero digits are not adjacent. A constant can have several MSD representations, because non-zero digits can be consecutive in MSD. CSE algorithms generally find the most common non-zero digit combinations while optimizing the number of operations. In [1], a two-term subexpression elimination technique is presented under CSD representation. In [2], it is shown that using MSD representation yields better solutions than CSD in MCM problems, since it has the same minimum number of non-zero digits as CSD, but provides multiple alternative representations for a constant. However, all these algorithms are heuristics, i.e., provide no indication how far from the minimum their solutions are. An exact CSE algorithm that considers the maximum sharing of partial terms is introduced in [3]. In this algorithm, all the possible implementations of constants are found, represented as a Boolean network and then converted to a 0-1 integer linear programming (ILP) problem. The cost function to be minimized is the linear function of optimization variables that represent partial terms. Finally, an exact solution is found by a generic satisfiability (SAT)-based 0-1 ILP solver. In graph-based methods, constants are implemented without a restriction to any particular number representation. Thus, a constant has more possible implementations than CSD and MSD. Two heuristic algorithms are introduced for multiple constants in [4] and [5]. While there is an exact algorithm [6] for a single constant in maximum 19 bits, no exact graph-based algorithm has been proposed for multiple constants.

In this work, we make the observation that being limited to CSD or MSD will prevent from finding a minimum solution in terms of the number of operations. While it is true that there is a higher probability of a representation with the minimal number of non-zero digits being selected for the optimized solution, it is also true that there are situations where a non-minimal representation may fit better with existing partial terms and lead to a better solution. In this paper, we introduce an exact algorithm that can handle multiple constants using general number representation. Since the implementations of a constant are not limited to any digit representation, we increase the search space, allowing our algorithm to be significantly more effective in the optimization of the number of operations. To help the search in this larger solution space, we introduce problem reduction and model simplification techniques. We present results on FIR filters and randomly generated instances which demonstrate that we can achieve large gains over the exact solutions obtained with CSD and MSD.

The rest of the paper is organized as follows. In Section II, the proposed exact algorithm is introduced. Results are given in Section III. Finally, the paper concludes in Section IV.

II. Triple-A: The Exact Algorithm

In this section, we present the problem definition and the exact algorithm called *Triple-A* that can handle multiple constants in CSD, MSD, and general number representations.

A. Definitions

In the optimization of the number of operations in digital filter synthesis, filter coefficients and partial terms are considered as odd numbers, since shifts can be implemented with only wires in hardware. So, an *operation* represents an addition or a subtraction with two odd inputs, I_1 and I_2, input shifts, S_1 and S_2, and an output, O, given as $O = I_1 \ll S_1 \pm I_2 \ll S_2$, where $S_1 = 0, S_2 > 0$ or $S_1 > 0, S_2 = 0$, i.e., without loss of generality one of the shifts at the input is zero and the other is greater than zero. A *partial term* is an odd constant that is neither a coefficient nor a filter input and is determined as an input of an operation that implements a coefficient. Thus, *the problem of the optimization of the number of operations* can be defined as finding the minimum number of partial terms to be added to a set that contains filter coefficients and filter inputs such that each coefficient and partial term in the set can be implemented with the set elements using only one operation. So, the minimum number of operations is the sum of the number of odd coefficients and the minimum number of required partial terms.

B. Partial Term Generation

In *Triple-A*, the optimization of the number of operations is defined as a binate covering problem, a special case of a 0-1 ILP problem where every constraint is interpreted as a propositional clause. In the preprocessing phase of the algorithm, after the filter coefficients are made positive and odd, they are stored without repetition in a set called *Cset*. They are labeled as filter coefficients and unimplemented. If the filter input, i.e., 1, does not exist in *Cset*, it is also inserted into *Cset* and labeled as implemented. The part of the algorithm where the partial terms are found for each element in *Cset* is as follows:

1) Take an unimplemented element from *Cset*, $Cset_i$. Form an empty set of arrays called $Pset_i$ associated with $Cset_i$. $Pset_i$ will contain all partial terms that are required to implement $Cset_i$.
2) Find an operation that implements $Cset_i$;
   a) Find the non-repeated inputs of the operation that are neither a filter coefficient nor a filter input and store them in an empty array called *Iarray*. Note that *Iarray* may contain a single partial term or a pair of partial terms.
   b) If *Iarray* is empty, then make $Pset_i$ empty and go to Step 5. In this case, $Cset_i$ can be implemented with an operation whose inputs are filter coefficients or filter inputs and this is the minimum implementation.
   c) If *Iarray* is not empty, then check for each array of $Pset_i$, $Pset_i(k)$, if $Pset_i(k) \subseteq Iarray$. If *Iarray* is included in $Pset_i$, then go to Step 3.
   d) If *Iarray* is not empty, then check for each array of $Pset_i$, $Pset_i(k)$, if $Iarray \subset Pset_i(k)$. If *Iarray* dominates $Pset_i(k)$, then delete $Pset_i(k)$.
   e) Add *Iarray* to $Pset_i$.
3) Repeat Step 2 until all the implementations of $Cset_i$ are considered.
4) Add all the partial terms in $Pset_i$ to *Cset*, if they are not in *Cset* and label them as unimplemented.
5) Label $Cset_i$ as implemented and repeat Step 1 until all elements in *Cset* are labeled as implemented.

Observe that in the first iteration of the algorithm, *Cset* contains the filter coefficients and in later iterations, it contains also the partial terms.

In CSD and MSD, the operations for a constant (Step 2 of the algorithm) are determined by addition/subtraction of non-zero digit combinations to the constant as described in [3]. As an example, consider 51 ($10\overline{1}010\overline{1}$ in CSD) as a filter coefficient. The operations that implement 51 are given in Fig. 1. Observe that the implementation of 64-13 is the same as the implementation of 52-1, since these operations have the same odd inputs. Therefore, 52-1 is not listed in Fig. 1. Similarly, the duplications of implementations are not presented in this figure, e.g., 63-12 is equal to -12+63. As can be seen from Fig. 1, since 1 represents the filter input, the implementation of 51 requires 3,13,17,47,67 as single partial terms and 3 and 63 as a pair of partial terms. Since the single partial term 3 dominates the pair of 3 and 63 (Step 2d of the algorithm), $Pset_{51}$ includes only single partial terms. After these partial terms are obtained, they are added to *Cset* and operations that implement them are also found in this way.

With 1 non-zero digit combinations	With 2 non-zero digits combinations
51=1000000+00$\overline{1}$010$\overline{1}$=1$\ll$6 - 13$\ll$0	51=10$\overline{1}$0000+000010$\overline{1}$=3$\ll$4 + 3$\ll$0
51=00$\overline{1}$0000+100010$\overline{1}$=-1$\ll$4 + 67$\ll$0	51=1000100+00$\overline{1}$000$\overline{1}$=17$\ll$2 - 17$\ll$0
51=0000100+10$\overline{1}$000$\overline{1}$=1$\ll$2 + 47$\ll$0	51=100000$\overline{1}$+00$\overline{1}$0100=63$\ll$0 - 3$\ll$2

Fig. 1. Implementations of 51 in CSD.

In general number representation, finding the operations that implement a constant has some limitations, since it must be ensured that the obtained solution has no feedback. To illustrate this problem, consider the coefficients of a filter as 7, 11, and 19. The possible implementations of these coefficients under general number representation are given in Fig. 2.

Implementations of 7	Implementations of 11	Implementations of 19
7=1+6=1$\ll$0 + 3$\ll$1	11=1+10=1$\ll$0 + 5$\ll$1	19=1+18=1$\ll$0 + 9$\ll$1
7=2+5=1$\ll$1 + 5$\ll$0	11=2+9=1$\ll$1 + 9$\ll$0	19=2+17=1$\ll$1 + 17$\ll$0
...	...	...
7=11-4=11$\ll$0 - 1$\ll$2	11=7+4=7$\ll$0 + 1$\ll$2	19=7+12=7$\ll$0 + 3$\ll$2
...	...	...
7=19-12=19$\ll$0 - 3$\ll$2	11=19-8=19$\ll$0 - 1$\ll$3	19=11+8=11$\ll$0 + 1$\ll$3
...	...	...

Fig. 2. Implementations of 7, 11, and 19 in general number representation.

If all operations listed in Fig. 2 are accepted for these coefficients, a minimum solution that includes a feedback loop can be obtained, e.g., $7 = 11-4$, $11 = 19-8$, and $19 = 8+11$. To avoid these feedback loops, only addition operations can be considered or additional constraints that break the loops should be added to the 0-1 ILP problem. Since more promising results are obtained with both addition and subtraction operations and the number of additional constraints grows exponentially with the number of partial terms, neither approaches are used.

Instead, for each constant n, odd numbers between 1 and $2^{\lceil log_2(n)\rceil+1}-1$ are sorted in ascending order of non-zero digits in their CSD representations in a set called *Nset*. In fact, *Nset* is a set where its elements are ordered according to the number of operations required to implement each single element in CSD. After *Nset* is formed, the operations for a constant are found by traversing from the first element to the element before the constant in *Nset* and assigning each element to the first input of an adder with positive and negative sign. The operation that implements the constant is accepted, if its second input is placed in a lower position than the position of the constant in *Nset*. As an example for the filter coefficient 51, suppose 23 ($10\bar{1}00\bar{1}$ in CSD) taken from *Nset* is assigned to the first input of addition operations with positive and negative sign. The operations, 51=23$\ll$0 + 7$\ll$2 and 51=-23$\ll$0 + 37$\ll$1, are accepted for the implementation of 51, since the second inputs, i.e., 7 and 37, are located before 51 in *Nset*. The partial terms that are only considered using *Nset* with respect to CSD and MSD are the odd numbers that cannot be represented with the non-zero digit combinations of the constant. For the filter coefficient 51 ($10\bar{1}010\bar{1}$ in CSD), we note that 23 cannot be considered in both CSD and MSD. Also, the partial terms can include equal number of non-zero digits as opposed to the partial terms obtained by decomposing the CSD and MSD representations of constants. Again, for the filter coefficient 51, the partial term 43 ($10\bar{1}0\bar{1}0\bar{1}$ in CSD) cannot be considered using CSD or MSD. With the use of *Nset* we avoid feedback loops and increase the possible sharing of partial terms by providing more possible implementations of a constant than operations obtained under CSD and MSD.

C. Conversion to 0-1 ILP Problem

After all partial terms required to implement each coefficient and partial term are found, the optimization problem is converted into a combinational network. The network only includes AND and OR gates. An OR gate, representing a coefficient or a partial term, combines all partial terms that can be used for the synthesis of the associated coefficient or partial term. An AND gate, representing a pair of partial terms, combines two partial terms. The primary inputs of the network are the coefficients and partial terms that can be implemented with an operation whose inputs are filter coefficients or filter inputs. As an example, the network constructed for the coefficient 51 in CSD is given in Fig. 3 by inserting the redundant 3 and 63 partial term pair into the network to make a clarification in the definition of AND gate. We note that this partial term pair is not required for the implementation of 51, since it is dominated by the single partial term 3.

The network is converted into a 0-1 ILP problem, after additional hardware (a 2-input AND gate for each partial term) with the optimization variables is added to the network. The optimization variables that represent the filter coefficients are assigned to 1 and the conjunctive normal form (CNF) formulas of each gate in the network are found. Each clause in CNF formulas is expressed as a linear inequality. A cost function, i.e, the linear function of the optimization variables that represent the partial terms, is constructed. Finally, the problem is given to the SAT-based 0-1 ILP solver, MiniSat+ [7], to obtain an exact solution. In the construction of the network and the translation of the network into CNF, the issues described in [8] that speed-up the 0-1 ILP solver were also considered.

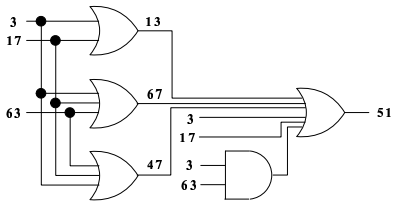

Fig. 3. The network for the coefficient 51 in CSD.

In the post-processing phase of the algorithm, after the minimum solution including filter coefficients and required partial terms is obtained, the filter coefficients and partial terms are synthesized from inputs to outputs. In the selection of an operation for each coefficient and partial term among possible implementations whose inputs are in the solution or filter inputs, the minimization of the number of operations in series, i.e., the delay, is considered. Thus, the minimum delay synthesis of the found minimum area solution is realized.

III. Experimental Results

In this section, experimental results of *Triple-A* on FIR filter and randomly generated instances under CSD, MSD, and general number representations are presented and the exact solutions obtained with these representations are compared.

As the first experiment set, FIR filters where the coefficients were computed using the Remez algorithm in MATLAB were used. The filter specifications are given in Table I where *pass* and *stop* are normalized frequencies that define the passband and stopband respectively; *#tap* is the number of coefficients; and *width* is the bit-width of the coefficients.

TABLE I
FILTER SPECIFICATIONS.

Filter	*pass*	*stop*	*#tap*	*width*
1	0.15	0.25	40	12
2	0.20	0.25	80	12
3	0.24	0.25	120	12
4	0.15	0.25	60	14
5	0.15	0.20	60	14
6	0.10	0.15	60	14
7	0.10	0.15	100	16
8	0.15	0.25	120	16
9	0.10	0.15	160	16

The 0-1 ILP problem sizes of filter instances and solutions obtained by *Triple-A* are given in Table II. In this table, *vars*, *cons*, and *optv* stand for the number of variables, constraints, and optimization variables respectively. Also, *adder* denotes the number of operations and *step* denotes the maximum number of operations in series needed to synthesize the filter coefficients. CPU is the CPU time in seconds that is used by the 0-1 ILP solver [7] to compute the exact solutions on a PC with dual Pentium Xeon at 2.4GHz, with 4GB of main memory, running Linux. Since the CPU times required to construct the network in the preprocessing phase and to find

TABLE II

0-1 ILP PROBLEM SIZES AND RESULTS OF TRIPLE-A ON FIR FILTER INSTANCES.

Filter	0-1 ILP Problem Sizes									Minimum Number of Operation Solutions								
	CSD			MSD			General Number			CSD			MSD			General Number		
	vars	cons	optv	vars	cons	optv	vars	cons	optv	adder	step	CPU	adder	step	CPU	adder	step	CPU
1	77	119	50	151	302	80	21231	96222	475	16	3	0	16	3	0	15	4	1.2
2	61	83	47	92	137	64	28	28	28	29	3	0	29	4	0	28	3	0
3	34	34	34	34	34	34	34	34	34	34	3	0	34	3	0	34	3	0
4	168	345	95	107	146	74	20	20	20	23	3	0	22	3	0	20	4	0
5	241	562	107	203	466	93	29	29	29	35	4	0	34	3	0	29	4	0
6	331	799	137	541	1446	200	17647	77268	556	35	4	0	33	3	0.1	29	5	0.1
7	938	3131	259	4009	16037	779	45	45	45	51	4	0.2	49	4	12.8	45	5	0
8	511	1271	218	673	1918	239	1722	5489	449	54	4	0.3	53	4	0.2	48	4	0.4
9	1866	6671	467	9510	40050	1384	70	70	70	77	5	4.7	77	4	63.9	70	4	0
Total	4227	13015	1414	15320	60534	2947	40826	179205	1706	354	33	5.2	347	31	77.0	318	36	1.7

a solution with minimum delay in the post-processing phase are also negligible, CPU only indicates the CPU time of the 0-1 ILP solver required to find the exact solution.

In this experiment, we observe on some instances, such as Filter 7 and 9, that each coefficient of a filter can be implemented with other coefficients and filter input using general number representation. This occurs because the general number representation considers more possible implementations that can cover the coefficients. In this case, there is no need to represent the problem as an optimization problem. Thus, the 0-1 ILP problem size can be much smaller in general number representation than CSD and MSD. On the other hand, the problem size can be too large, when partial terms are required to implement some coefficients in general number representation. For example, on Filter 1, only two coefficients need to be synthesized with the partial terms. Despite a larger search space, the optimization problem to be solved is not a hard problem for the 0-1 ILP solver, as can be seen from the required time to find an exact solution. We note that the reduction in the number of operations when using general number representation is 10% and 8% on average, and up to a maximum of 17% and 15%, with respect to CSD and MSD respectively. Although the minimum number of operation solutions are obtained in general number representation, the delay of the solutions are increased with respect to the solutions obtained in CSD and MSD.

As the second experiment set, randomly generated instances between the number of 10 and 80 constants in 12 bit-widths were used. We generated 30 instances for each number of constants. We compare the results of *Triple-A* under CSD, MSD, and general number representations in Fig. 4.

In this experiment, we observe that while the difference of average number of operations between CSD and MSD tends to increase up to 2.1, the difference of average number of operations between MSD and general number reaches up to 6.5, as the number of constants increases. This clearly shows the advantage of using general number representation over CSD and MSD, when searching for the maximal sharing of partial terms in the optimization of the number of operations.

IV. CONCLUSIONS

In this work, an exact algorithm that can handle general number representation of multiple coefficients for the minimization of the number of operations in digital filter synthesis was introduced. Model simplifications and problem reduction techniques were also presented to cope with the possible increased search space. It is shown that significant savings can be obtained using general number representation over exact solutions obtained under commonly used CSD and MSD representations with very low computational effort.

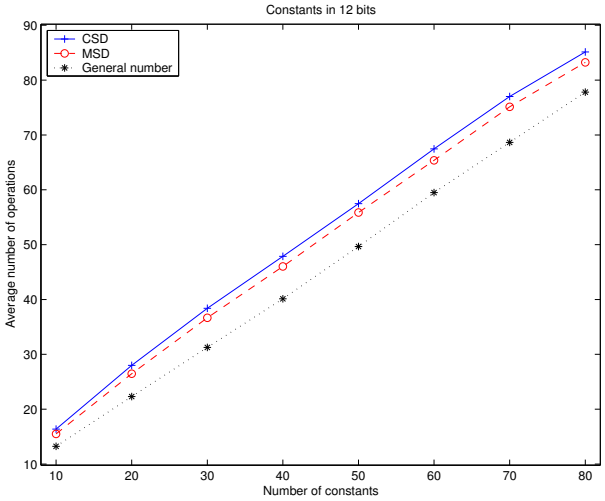

Fig. 4. Results on randomly generated instances.

REFERENCES

[1] R. Hartley, "Subexpression Sharing in Filters using Canonic Signed Digit Multipliers," *IEEE Transactions on Circuits and Systems II*, vol. 43, no. 10, pp. 677–688, 1996.

[2] I.-C. Park and H.-J. Kang, "Digital Filter Synthesis Based on Minimal Signed Digit Representation," in *Proceedings of Design Automation Conference (DAC)*, 2001, pp. 468–473.

[3] P. Flores, J. Monteiro, and E. Costa, "An Exact Algorithm for the Maximal Sharing of Partial Terms in Multiple Constant Multiplications," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, 2005, pp. 13–16.

[4] A. Dempster and M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *IEEE Transactions on Circuits and Systems II*, vol. 42, no. 9, pp. 569–577, 1995.

[5] Y. Voronenko and M. Puschel, "Multiplierless Multiple Constant Multiplication," *To appear in ACM Transactions on Algorithms*.

[6] O. Gustafsson, A. Dempster, and L. Wanhammar, "Extended Results for Minimum-adder Constant Integer Multipliers," in *Proceedings of the International Symposium on Circuits and Systems*, 2002, pp. 73–76.

[7] N. Een and N. Sorensson, "Translating Pseudo-Boolean Constraints into SAT," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006.

[8] M. Velev, "Efficient Translation of Boolean Formulas to CNF in Formal Verification of Microprocessors," in *Proceedings of the IEEE Asia and South Pacific Design Automation*, 2004, pp. 310–315.