Exploiting General Coefficient Representation for the Optimal Sharing of Partial Products in MCMs

Eduardo Costa
UCPel
Pelotas, Brazil
ecosta@inf.ufrgs.br

Paulo Flores
IST/INESC-ID
Lisboa, Portugal
pff@inesc-id.pt

José Monteiro
IST/INESC-ID
Lisboa, Portugal
jcm@inesc-id.pt

ABSTRACT

We propose a new algorithm that maximizes the sharing of partial terms in Multiple Constant Multiplication (MCM) operations under a general number representation for the coefficients. MCM operations are required by many algorithms in digital signal processing and have been the subject of extensive research. By making no assumptions as to the number representation, the algorithm described in this paper is able to perform a better search for the optimal sharing of partial terms than previous methods based on MSD or CSD representations. We have applied our algorithm for the hardware minimization of FIR filters. The results show that we can obtain solutions that require between 20% to 50% less hardware when compared against the solutions using the MSD representation.

Categories and Subject Descriptors

B.2.2 [**Hardware**]: Arithmetic and Logic Structures—*Performance Analysis and Design Aids*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Signal processing systems*

General Terms

Algorithms, Design, Performance.

Keywords

Multiple Constant Multiplication (MCM), Minimal Signed Digit (MSD), Common Subexpression Elimination (CSE), Digital Filter Design.

1. INTRODUCTION

Several computationally intensive operations, such as, Finite Impulse Response (FIR) filters and Fast Fourier Transforms (FFT), involve a sequence of Multiply-Accumulate (MAC) operations with constant coefficients. These operations are typical in Digital Signal Processing (DSP) applications. Hardwired dedicated architectures are the best option for maximum performance and minimum power consumption.

SBCCI'06, August 28–September 1, 2006, Minas Gerais, Brazil.

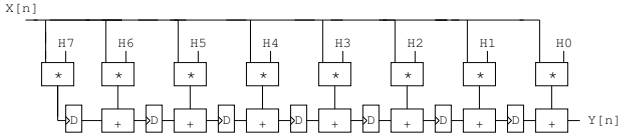

Figure 1: Transposed form of a hardwired FIR filter implementation.

Constant coefficients allow for a great simplification of the multipliers, which can be reduced to shift-adders [1]. In these multipliers, a bit set to 1 in position m of the coefficient implies the sum of the input shifted left by m positions. Shifts are free in terms of hardware, hence the hardware required for a multiplication with a constant with n bits set to 1 is simply $n-1$ adders.

In many MAC operations, the same input is to be multiplied by a set of constant coefficients, a problem known as Multiple Constant Multiplications (MCM). An example of this is the transposed form architecture of a FIR filter, exemplified in Figure 1. In this situation, significant reductions in hardware, and consequently power, can be obtained by sharing the partial products of the input. In this paper, we address the problem of maximizing the amount of sharing of the partial products in a MCM operation over the same input. This problem has been the subject of extensive research in the last years, with different strategies for finding the partial products that lead to a minimal hardware implementation [2, 3, 4, 5, 6, 7, 8, 9].

In many of these works, the Canonical Sign Digit (CSD) representation is used for the coefficients. The reason for this is that the CSD representation minimizes the number of non-zero digits, hence it allows the maximal subexpression sharing search to start from a minimal level of complexity. Recently, Park et al. [10] proposed the usage of a Minimal Signed Digit (MSD) representation for the coefficients. Under the MSD representation, a given numerical value can have multiple representations. However, in all of them, the number of non-zero digits is the same as the CSD representation. The algorithm proposed in [10] exploits the redundancy of the MSD representation by choosing the MSD instance that leads to a maximal sharing in the implementation efficient FIR filters.

In this paper, we make the observation that the minimum area solution, in general, is not obtained using all coefficient representations with minimum number of non-zero digits. In fact, we claim that the representation used for the coefficients is irrelevant and that during the optimization process numerical values of the coefficients should be used! While it is true that there is a higher probability of a representation with a 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.

By using numerical values for the coefficients and partial terms we increase significantly the search space, allowing our algorithm to be significantly more effective in area optimization. We present results that demonstrate that we can achieve large gains over previously proposed methods, in some cases a reduction of up to 50% of adders/subtracters. We should emphasize that this comparison is made against highly optimized solutions.

The downside of this algorithm is that it is not able to take into account the depth of the subexpression sharing, meaning that the hardware reduction is obtained at the cost of an increase in the number of adder-steps. In order to introduce some control this increase, we have added information about the depth of each subexpression to the initial algorithm. During the search process, when there are several possible combinations of subexpressions, we choose the one that leads to a minimum increase of the total number of adder-steps.

This paper is organized as follows. In Section 2 we give an overview of relevant work related to our work and presents the CSD and MSD representations. Section 3 describes the algorithms we propose. We present results obtained for FIR filters in Section 4. Finally, in Section 5 we conclude this paper, discussing the main contributions and future work.

2. RELATED WORK

A large amount of work has addressed the use of efficient implementations of multiplier-less MCMs. The techniques include the use of different number representation schemes, the use of different architectures and implementation styles and the coefficient optimization techniques, *e.g.*, [3, 4, 5].

In this work, we will concentrate on the coefficient optimization techniques applied to a fully-parallel transposed form FIR filter architecture. This type of filter is used because it can accept the subexpression sharing which will be explored in this work.

In FIR filter algorithms, 2's complement is the most used encoding for signed operands. However, since number systems have a great influence on the hardware complexity of the filters, the use of signed digit representation has been frequently more efficient [10].

Synthesis algorithms that have been proposed are based on the Canonical Signed Digit (CSD) representation [6, 7, 8, 9]. CSD is a signed digit system with the digit set {1,0,2}, where 2 denotes -1. The CSD representation is unique and presents two main properties: (1) the number of non-zero digits is minimal, (2) two non-zero digits are not adjacent. Hardware requirements are reduced because the numerical values are represented with a maximal number of zero digits. In these methods, a common subexpression is searched among multiple constants and implemented into one hardware block in order to share the result of the subexpressions in evaluating all the constants.

In [10], the MSD representation is proposed for the coefficients. The MSD representation is obtained by removing the second property of the CSD representation. Thus, a constant can have several MSD representations, but all with a minimum number of non-zero bits. For example, the value 6 is represented using 4 bits as 1020 in CSD, but both 1020 and 0110 are valid representations in MSD. In the algorithm described in [10], *Cset* represents the coefficient set to be synthesized and contains all MSD representations for all coefficients. The first representation that matches a combination of subexpressions is used. The results are shown to be an improvement to [6] and [7]. On the other hand, the major limitation of [9] is the usage of a lookup table with size 4096, which in practice limits the coefficient bit-width to 12.

The fundamental difference between our approach and previously proposed methods is that we do not combine bit-patterns and match them against coefficients in some given representation, be it binary, CSD or MSD. Instead, we store numerical values and combine a pair of these values, either by addition or subtraction, using all possible different shift amounts (multiplication by a power of 2) of one of them, and compare the numerical result with the stored values. In this search process, we use an algorithm similar to the one described in [10].

3. PROPOSED METHOD

In this section, we describe the algorithm we propose for the maximal sharing of subexpressions and its modification to allow for the control of the number of adder-steps.

3.1 Number Representation

As referred, by considering both positive and negative digits, the values in CSD are represented with a minimum number of non-zero digits. Hence, when multiplying against a single constant coefficient, this is the representation that leads to a minimum number of operators in the shift-add implementation. For example, the value 29 is represented as 011101 in binary, while it is represented as 100201 in CSD (again, 2 stands for -1), thus one less operator would be required.

When the sharing of partial terms is to be considered among the multiplication of several constant coefficients, the MSD representation has advantages over the CSD representation. Values using the MSD representation have the same number of non-zero bits as their CSD representation, however because in MSD they are allowed to be consecutive, several representations with a minimum number of non-zero digits are valid (of which, the CSD representation is a particular case). Valid MSD representations for 29 are 100201 and 100022, and current methods can exploit this fact by testing either match.

However, we note that 29 can be represented, using the digit set {1,0,2}, as 011101, 011112, 100022, 100201, 100212, 102101, 102112, 121101, 121112. Suppose that at some stage of the search process we have the partial terms 100011 and 102 (both valid MSD representations, for values 35 and 3, respectively). Existing methods will not be able to use these terms as no combination of them yields a valid MSD pattern for 29. Yet, we can use a subtracter to obtain 100011-1020=102101, which represents the value 29. Hence,

we can implement 29 with a single operator, whereas that was not possible with just MSD.

We go one step further to note that we really don't care about the bit patterns, we are only concerned with the numerical value. Hence, we don't need to store all the representations of a given value, only its numerical value! During the search process, instead of combining and matching bit patterns, what we do is to add or subtract shifted versions of already found partial terms, and compare the numerical value with the value of the coefficients we need to implement.

3.2 Algorithm for Maximal Sharing

Similarly to the algorithm of [10], we have two sets: *Cset* maintains numerical values of all coefficients not yet covered; *Patset* is the set with the partial terms found so far. Before being inserted into *Cset*, all values are made odd by successive shifted rights and any duplicates that may appear in this process are eliminated. *Patset* is initialized with a single element, the value 1. We then enter a loop where all shifted versions of elements in *Patset* are pair-wised added and subtracted:

1. remove all coefficients in *Cset* that have the same value as any shifted value of an element in *Patset*.
2. remove all coefficients in *Cset* whose value can be obtained by adding or subtracting shifted versions of two elements in *Patset*. Insert the elements removed into *Patset*.
3. remove all coefficients in *Cset* whose value can be obtained by adding or subtracting shifted versions of three elements in *Patset*. Insert the elements removed into *Patset*. If no element was removed from *Cset* in the previous steps, go to Step 4. Otherwise, go to Step 1.
4. check which of the partial terms obtained by adding or subtracting shifted versions of two elements in *Patset* maximally matches a subset of bits of a CSD representation. Register the combination as a new partial term in *Patset* and insert into *Cset* a new element obtained by removing the new partial term from the selected decimal representation. Go to Step 1.

This loop is repeat until there are no more coefficients in *Cset*.

In this algorithm, as in [2], all pairwise combinations are valid. In the case of [10], the algorithm does not consider a combination of shifted elements of *Patset* with non-zero bits in the same position.

Figure 2 presents an example of shifting and combination of two elements of the *Patset*. This combination in particular is performed in step 2 of the algorithm. We use the example in the previous section where we have the values `100011` and `102` in *Patset* and 29 is the coefficient to implement. Figure 2(a) shows the behavior of the algorithms of [10] and [2]. Because there are non-zero digits in the same position of both terms, the results from the addition and subtraction for this shift amount are not considered in [10]. Algorithm [2] is not deterred by this conflict and computes the addition and subtraction, but the result is not a valid MSD representation of the value 29, hence no match will be flagged.

The execution of algorithm we propose in this paper is described in Figure 2(b). In *Patset* we simply have the numerical values of the partial terms and their shifted versions

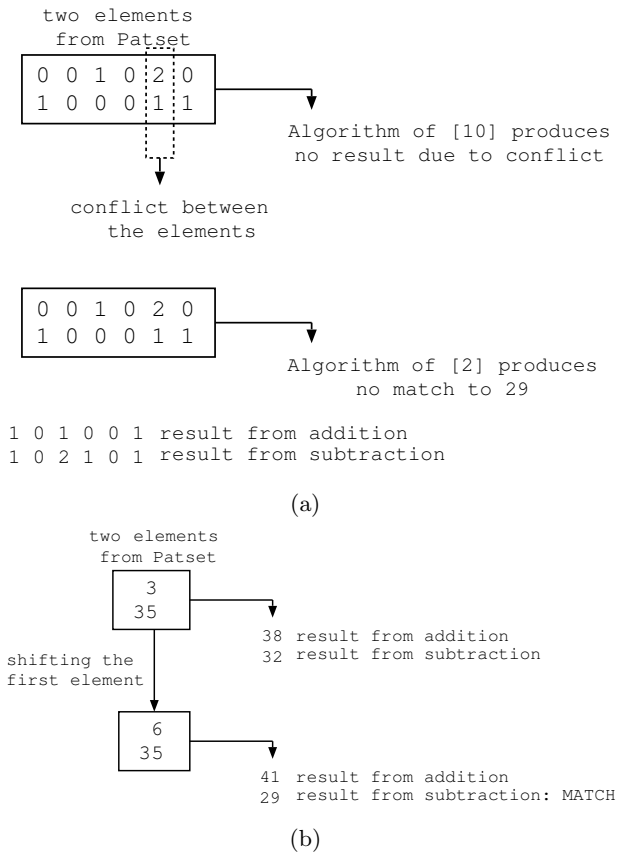

Figure 2: Example of combination and shift of elements.

are added and subtracted. The numerical result of these operations is compared against 29 and a match is immediately found.

3.3 Accounting for the Adder-Step Depth

The algorithm described above does not have any mechanism to account for, and therefore minimize, the depth in terms of adder-steps. In order to control the number of adder-steps, we associate the depth of each partial term in *Patset*. While we maintain the main goal of maximizing the sharing of partial terms, when we have an option, we select the combination that minimizes the depth of the new combination (minimum number of levels). This procedure is illustrated in Figure 3.

Figure 3(a) shows how step 2 of the algorithm is modified to track the level of each partial term in *Patset*. The level of the new combination is one more than the maximum level of the partial terms used. The other steps are modified in the same manner.

Figure 3(b) presents the situation where the same combination is obtained through different partial terms in *Patset*. In this example, the element `15` can be obtained from the combination of the first and third elements of *Patset*, meaning that the combination will be at level 3, or from the combination of two shifted versions of the second element in *Patset*, which will be at level 2. Hence, we insert into *Patset*

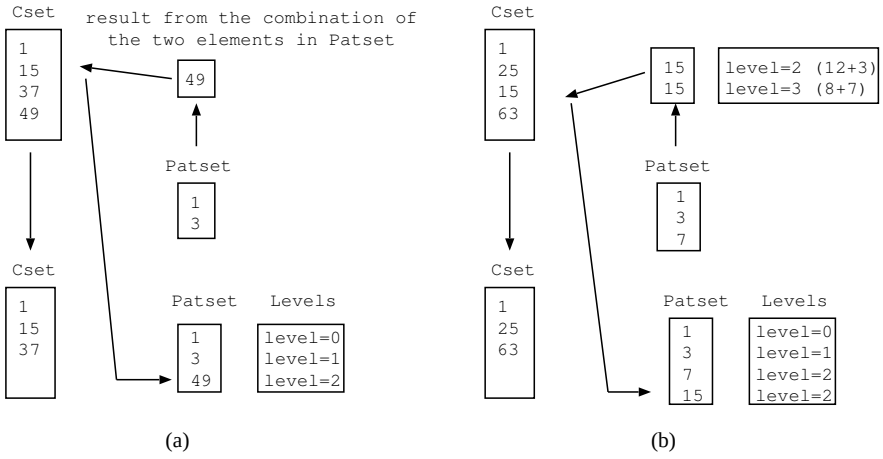

Figure 3: Limiting the depth of adder-steps.

this second combination with a lower level. Note that we do not need to repeatedly compute all possible combinations. Instead, we keep an updated list of these combinations ordered by their level and select the first match in this list.

4. RESULTS

In this section, we compare the results obtained with the two algorithms described in the previous section against the algorithm proposed in [10] and [2], which in turn has been shown to be an improvement over [6] and [7]. We have applied these algorithms to the optimization of FIR filters. We used the FIR filters from [2] which are the same as [10] added with four new instances. The filters' coefficients were computed with the MATLAB using the Remez algorithm. The columns of Table 1 present the filters' specification: *filter* is just an index for each example, *passband* and *stopband* are normalized frequencies, *#tap* is the number of coefficients of the filters and *width* is the bit-width of each coefficient.

Table 2 shows the results obtained by applying the algorithms of [10], [2] and our proposed algorithms on this set of benchmarks. For each algorithm the column *adders* gives the minimum number of adders required to implement the filter found by each method, the column *steps* gives the maximum depth, in terms of adder-steps, for all coefficients and the column *CPU* is the CPU time used to compute the solution. Note that "adder" may refer to both an adder or a subtracter.

Table 1: Benchmark FIR filter specifications.

Filter	passband	stopband	#tap	width
1	0.20	0.25	120	8
2	0.10	0.25	100	10
3	0.15	0.25	40	12
4	0.20	0.25	80	12
5	0.24	0.25	120	12
6	0.15	0.25	60	14
7	0.15	0.20	60	14
8	0.15	0.20	100	16
9	0.10	0.15	60	14
10	0.10	0.15	100	16
11	0.10	0.12	100	16
12	0.10	0.12	120	18

The results obtained by the proposed algorithms are presented by the name of *Numeric Algorithms*. The columns under *Maximal Sharing* in Table 2 present the results obtained by the different methods when maximizing the sharing of partial terms, thus reducing the circuit area. In terms of the hardware required to implement the MCM of the filter, we can observe that we always obtain a solution that is at least as good as [10] and [2]. For the more complex examples, our more comprehensive search is able to produce significantly better results. For the larger example, filter 12, we are able to reach a solution with 40% or 27% less hardware, when compared with [10] or [2]. Note that this comparison is made against a highly optimized result. This reduction is obtained at the cost of a higher number of adder-steps (more significant in filters 7, 11 and 12).

The proposed algorithm is also much more efficient in run time. The reduction in the computation time results from three factors. First, our method reaches a solution more easily, by considering partial terms that are discarded by previous methods. Second, since our method does not have to deal with different representations of the coefficients, we can use the unique computer native integer representation of the coefficients, thus increasing the efficiency of most operations of the algorithm. Third, since each coefficient and partial term is represented by a single numeric value we have less combinations of partial terms and less elements when looking for a match.

We present the results obtained using the second version of our algorithm in the last two columns of Table 2. The run times are very similar to the first implementation, hence have been omitted from the table. As can be observed, the second approach does not present a significant variation for most of the filters. In fact, we observe that the greatest impact of this approach can occur in step 3 of the algorithm. In the first version, when a coefficient is found in *Cset* in step 3, as a result of a shifted combination of three elements in *Patset*, the flow of the algorithm returns to the step 1, thus before testing the remaining combinations. In our second approach, step 3 is only finished after generating the combinations for all elements, so that we can find different

Table 2: Summary of the results obtained.

Filter	MSD [10] Maximal Sharing			MSD [2] Maximal Sharing			Numeric Algorithms				
							Maximal Sharing			Depth Control	
	adders	steps	CPU(s)	adders	steps	CPU(s)	adders	steps	CPU(s)	adders	steps
1	10	3	0.03	10	3	0.02	10	3	0.01	10	3
2	18	4	0.65	17	3	0.07	17	3	0.02	17	3
3	18	4	1.64	17	4	1.03	15	4	0.69	17	4
4	29	4	1.44	29	4	1.29	28	5	0.03	28	5
5	34	3	0.71	34	4	0.48	34	3	0.12	34	3
6	22	4	1.35	22	5	1.09	20	4	0.03	20	4
7	35	3	92.50	32	6	11.92	29	6	0.05	29	6
8	52	5	629.92	45	6	67.96	44	6	560.52	44	6
9	37	4	21.17	31	6	1.81	28	6	0.05	28	6
10	50	5	76.23	48	5	227.54	46	6	286.22	47	6
11	73	5	1891.36	58	14	473.54	51	13	684.16	51	13
12	107	6	20550.43	81	9	2999.24	64	12	517.86	67	9

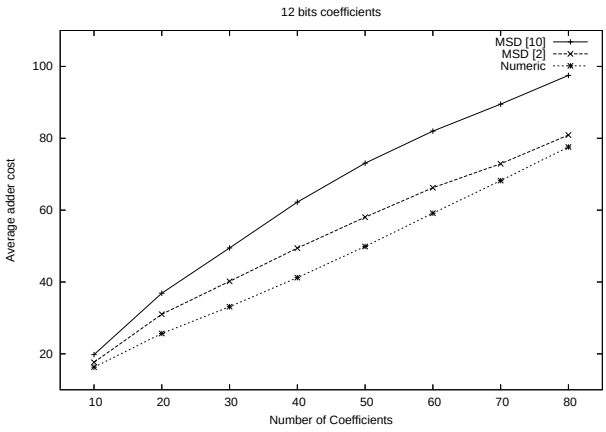

Figure 4: Comparison of heuristic algorithms for random instances.

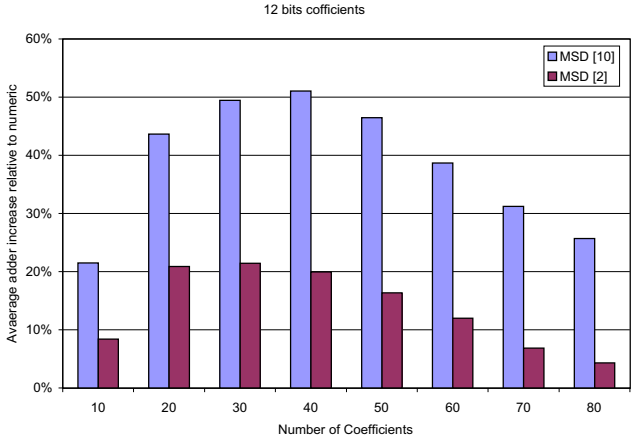

Figure 5: Relative increase on adder cost with respect to the numeric algorithm.

combinations for the same element with a smaller number of adder-steps. However, in our first approach most of the coefficients for the smaller examples are synthesized in the step 2 of the algorithm. For this reason, the second approach does not present a great impact on the results. To substantiate this reasoning, we can observe that there is a significant reduction in the number of adder-steps for filter 12, where we have a reduction from 12 to 9 adder-steps for the numeric algorithms. For this filter, the algorithm synthesizes 5 coefficients in step 3. This means that for the total of 64 adders, 10 adders were synthesized in step 3 of the algorithm, where the potential for combinations with different levels to choose from is higher. The other filters are not practically synthesized in step 3 of the algorithm and thus, there is no impact in reduction of the adder-steps. An important point to be emphasized is that, although filter 12 has an increased number of adders when compared with the first version, this value is still significantly less than that obtained with [10] and [2], with about the same number of adder-steps.

In order to further characterize the proposed algorithm we compare its efficiency on random MCM instances. Figure 4 gives a plot of the average number of adders/subtracters obtained with the maximal sharing heuristics of [10], [2] and our numeric algorithm, versus the number of coefficients. We used coefficients with 12 bits and, for each number of coefficients, we run 45 instances with randomly generated coefficients. We observe that the average solution obtained with the numeric method requires always less hardware than other heuristic solutions. On average we compute 17 adders less than the solution of [10] and the difference between solutions does not reduce with the number of coefficients. However when comparing to the solutions of [2] we need 6 adders less, on average. Moreover, the solutions difference is reduced when the number of coefficients increases, because the probability of finding a generated partial term that covers a coefficient becomes higher.

Figure 5 shows the relative average adders increase of [10] and [2] methods regarding to the proposed numeric method. Those methods compute solutions with an hardware overhead that can go up to 50% and 20%, respectively. However, for instance with large number of coefficients the relative hardware overhead is reduce due to the natural increase on the average number of adders need to implement the MCM.

5. CONCLUSIONS

We have described a new algorithm that computes the minimum number of adder/subtracter modules by maximizing the sharing of common subexpressions in the implementation of MCM structures. Numerical values are used for the coefficients and their redundancy in terms of number representation is exploited by selecting the instance that minimizes the total hardware. We presented results for digital filter synthesis where we demonstrate that our algorithm is able to perform significantly better that previously proposed approaches. We have implemented a modified version of our algorithm in order to allow for the reduction of the depth of adder-steps. As future developments of this work, we plan to develop non-heuristic algorithms/models for the optimal sharing of partial terms.

Acknowledgments

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

6. REFERENCES

[1] H. Nguyen and A. Chatterjee. Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*, 8(4):419–424, August 2000.

[2] E. da Costa, P. Flores, and J. Monteiro. Maximal Sharing of Partial Terms in MCM under Minimal Signed Digit Representation. In *European Conference on Circuits Theory and Design*, pages 468–473, 2005.

[3] M. Mehendale, S. Sherlekar, and G. Venkatesh. Techniques for Low Power Realization of FIR Filters. In *Design Automation Conference*, pages 404–416, 1995.

[4] H. Samueli. An Improved Search Algorithm for the Design of Multiplierless FIR Filters with Power-of-Two Coefficients. In *IEEE Transactions on Circuits and Systems*, pages 1044–1047, 1989.

[5] A. Nannarelli, M. Re, and G. Cardarilli. Tradeoffs between residue number system and traditional FIR Filters. In *IEEE International Symposium on Circuits and Systems*, May 2001.

[6] M. Potkonjak, M. Srivastava, and A. Chandrakasan. Efficient substitution of multiple constant multiplication by shifts and additions using iterative pairwise matching. In *Proceedings of the 31st ACM/IEEE Design Automation Conference*, pages 189–194, 1994.

[7] R. Hartley. Subexpression sharing in filters using canonic signed digit multipliers. *IEEE Transactions on Circuits and Systems II.*, 43(10):677–688, 1996.

[8] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova. A New Algorithm for Elimination of Common Subexpressions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, 18:58–68, January 1999.

[9] A. Dempster and M. Macleod. Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters. *IEEE Trans. on CAS-II*, 42(9):596–577, September 1995.

[10] I-C. Park and H-J. Kang. Digital Filter Synthesis Based on Minimal Signed Digit Representation. In *Design Automation Conference*, pages 468–473, 2001.