Maximal Sharing of Partial Terms in MCM under Minimal Signed Digit Representation

Eduardo da Costa* Paulo Flores† José Monteiro†

Abstract — **We propose a new algorithm that maximizes the sharing of partial terms in Multiple Constant Multiplication (MCM) operations. MCM operations are required by many algorithms in digital signal processing and have been the subject of extensive research. Recently, the Minimal Signed Digit (MSD) number representation has been proposed as an extension to the Canonical Signed Digit (CSD) representation. By properly exploiting the redundancy of the MSD representation, the hardware implementation can be significantly optimized. The initial algorithm described in this paper is able to perform a better search for the optimal sharing of the redundant coefficient representations under MSD than previous methods. However, during its search the depth of adder-steps is not considered. We present a modified version of this algorithm that is able to reduce the maximum depth of partial terms at the expense of some extra hardware. The results show that for more complex problems our algorithm performs significantly better than previous approaches, in some cases obtaining solutions that require 25% less hardware.**

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.

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 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

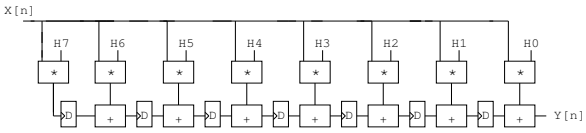

Figure 1: Transposed form of a hardwired FIR filter implementation.

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. Two key strategies have a large impact in the optimization of MCMs. One is to consider of not only adders, but also subtracters to combine partial terms, thus increasing the opportunity for the sharing of common subexpressions. The second is the usage of the Canonical Sign Digit (CSD) representation for the coefficients. This representation minimizes the number of non-zero digits, hence the maximal subexpression sharing search starts from a minimal level of complexity.

In a recent paper, Park et al. [2] propose the usage of a Minimal Signed Digit (MSD) representation for the coefficients. The MSD representation is obtained from the CSD representation by relaxing the requirement that there cannot be two consecutive non-zero digits. 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 [2] exploits the redundancy of the MSD representation by choosing the MSD instance that leads to a maximal sharing in the implementation efficient FIR filters.

The algorithm we describe in this paper also explores the redundancy of the MSD representation. By augmenting the search conditions, we have developed a significantly more effective area optimization algorithm. The more complex the example, the larger the gain we obtain, which can represent a reduction of up to 25% less 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 control this increase,

*Universidade Católica de Pelotas, Pelotas, RS, Brazil, ecosta@ucpel.tche.br

†INESC-ID / Technical University of Lisbon, Lisbon, Portugal, pff@inesc-id.pt, jcm@inesc-id.pt

0-7803-9066-0/05/$20.00 ©2005 IEEE

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 increases the least 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 present the MSD representation. 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].

Synthesis algorithms have been proposed that 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 [2], 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 maximum number of 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 [2], *Cset* represents the coefficient set to be synthesized and contains all MSD representations for all coefficients. The first representation the matches a combination of subexpressions is used. The results are shown to be an improvement to [6] and [7]. The major limitation of [9] is the usage of a lookup table with size 4096, which in pratice limits the coefficient bit-width to 12.

3 PROPOSED ALGORITHMS

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 Algorithm for Maximal Sharing

Similarly to the algorithm of [2], we have two sets: *Cset* maintains all MSD representations of all coefficients not yet covered; *Patset* is the set with the partial terms found so far. Before being inserted into *Cset*, all MSD representations are shifted right such that the least significant bit is 1, 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 MSD representation as a shifted value of an element in *Patset*.
2. remove all coefficients in *Cset* whose MSD representation 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 MSD representation 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 an MSD 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 MSD representation. Go to Step 1.

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

In this algorithm, all pairwise combinations are valid. In the case of [2], 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. Figure 2(a) shows the behavior of the algorithm of [2]. When the first combination is performed, there are no conflicts between the elements. Thus, the addition and subtraction of the elements are obtained. When the first element is shifted left, there is a conflict between elements, where the second least significant bits of each element are equal to 1 simultaneously. Thus, the results from the addition and subtraction are not considered. Figure 2(b) shows the behavior under the algorithm proposed above. The conflict between elements is not a deterrent and we are able to test two new subexpressions with these combinations.

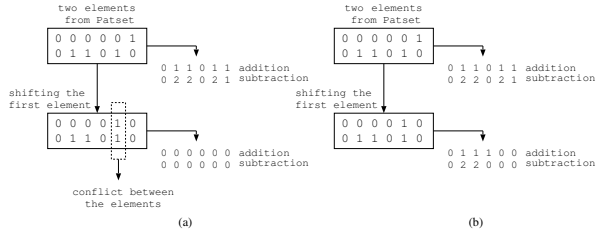

Figure 2: Example of combination and shift of elements.

3.2 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.

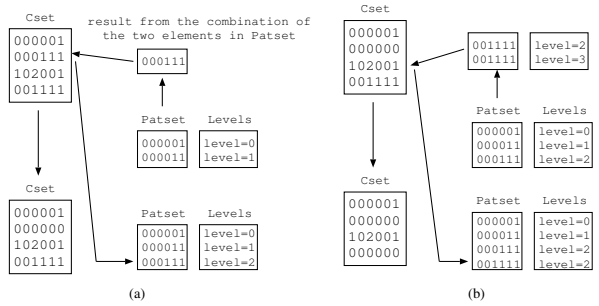

Figure 3: Limiting the depth of adder-steps.

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 `001111` 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* this second combination with a lower level. Note that this implies all combinations must be generated before we match them with *Cset*.

4 RESULTS

In this section, we compare the results obtained with the two algorithms described in the previous section against the algorithm proposed in [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 same FIR filters of [2] and added four new instances. The filters' coefficients were computed with the MATLAB using the Remez algorithm. The first five 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.

The next three columns of Table 1 give the results obtained by applying the algorithm of [2] to this set of benchmarks: *adders* gives the minimum number of adders found by this method required to implement the filter, *steps* gives the maximum depth in terms of adder-steps for all coefficients and *CPU* is the CPU time used to compute this solution. Note that "adder" may refer to both an adder or a subtracter.

The three columns under *Maximal Sharing* in Table 1 present the results obtained with the first implementation of our algorithm. In terms of total amount of hardware, we can observe that we always obtain a solution that is at least as good as [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 25% less hardware. 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 fact that we allow the combination of subexpressions that have non-zero digits in the same place increases the dependency between the partial terms generated, which may cause the logic depth to increase. Since our search reaches a solution more easily, our algorithm is much more efficient also in run time.

We present the results obtained using the second version of our algorithm in the last two columns of Table 1, under *Depth Control*. 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 the most of the filters. In fact, we observe that the greatest impact of this approach can occur in the 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 combina-

Table 1: Summary of the results obtained.

Filter	Filter Specification				Park [2]			Maximal Sharing			Depth Control	
	pass	stop	#tap	width	adders	steps	CPU(s)	adders	steps	CPU(s)	adders	steps
1	0.20	0.25	120	8	10	3	0.03	10	3	0.02	10	3
2	0.10	0.25	100	10	18	4	0.65	17	3	0.07	17	3
3	0.15	0.25	40	12	18	4	1.64	17	4	1.03	19	3
4	0.20	0.25	80	12	29	4	1.44	29	4	1.29	28	5
5	0.24	0.25	120	12	34	3	0.71	34	4	0.48	34	3
6	0.15	0.25	60	14	22	4	1.35	22	5	1.09	23	5
7	0.15	0.20	60	14	35	3	92.5	32	6	11.92	32	5
8	0.15	0.20	100	16	52	5	629.92	45	6	67.96	46	6
9	0.10	0.15	60	14	37	4	21.17	31	6	1.81	37	6
10	0.10	0.15	100	16	50	5	76.23	48	5	227.54	48	5
11	0.10	0.12	100	16	73	5	1891.36	58	14	473.54	65	13
12	0.10	0.12	120	18	107	6	20550.43	81	9	2999.24	88	7

tions. In our second approach, step 3 is only finished after generating the combinations for all elements, so that we can find different 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 9 to 7 adder-steps. For this filter, the algorithm synthesizes 30 coefficients in step 3. This means that for the total of 88 adders, 60 adders were synthesized in step 3 of the algorithm, where the potential for combinations with different levels to choose from is higher.

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 [2], with about the same number of adder-steps.

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. The MSD representation is used for the coefficients and its redundancy is exploited by selecting the representation 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 are currently working on two different avenues of research. One is to explore more general representations for the coefficients. The other is 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.

References

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

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

[3] M. Mehendale, S. Sherlekar, and G. Venkatesh. Techniques for Low Power Realization of FIR Filters. In *DAC*, 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 Trans. 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 *ISCAS*, 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 *DAC*, pages 189–194, 1994.

[7] R. Hartley. Subexpression Sharing in Filters using Canonic Signed Digit Multipliers. *IEEE Trans. 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. *TCAD*, 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.