FIR Filter Synthesis Algorithms for Minimizing the Delay and the Number of Adders

Hyeong-Ju Kang , Hansoo Kim , and In-Cheol Park
VLSI Systems Lab.
Dept. of Electrical Eng. and Comp.Science
Korea Advanced Institute of Science and Technology, Korea

Abstract

As the complexity of digital filters is dominated by the number of multiplications, many works have focused on minimizing the complexity of multiplier blocks that compute the constant coefficient multiplications required in filters. Although the complexity of multiplier blocks is significantly reduced by using efficient techniques such as decomposing multiplications into simple operations and sharing common subexpressions, previous works have not considered the delay of multiplier blocks which is a critical factor in the design of complex filters. In this paper, we present new algorithms to minimize the complexity of multiplier blocks under the given delay constraints. By analyzing multiplier blocks in view of delay, three delay reduction methods are proposed and combined into previous algorithms. Since the proposed algorithms can generate multiplier blocks that meet the specified delay, a trade-off between delay and hardware complexity is enabled by changing the delay constraints. Experimental results show that the proposed algorithms can reduce the delay of multiplier blocks at the cost of a little increase of complexity.

I. Introduction

Finite impulse response (FIR) digital filters are frequently used in digital signal processing by virtue of stability and easy implementation. Although programmable filters based on digital signal processing cores can take an advantage of flexibility, they are not suitable for recent consumer applications demanding high throughput and low power consumption. In such an application, therefore application specific FIR filters are frequently adopted to meet the constraints of performance and power consumption.

The problem of designing FIR filters has received a great attention during the last decade, as the filters are suffering from a large number of multiplications, leading to excessive area and power consumption even if implemented in full custom integrated circuits. Many works have focused on replacing multiplications by decomposing them into simple operations such as addition, subtraction and shift and reducing the number of simple operations. In this approach, the hardware block called a multiplier block is often used to implement all coefficient multiplications[1]. The concept of the multiplier block is significant in both terms of area and power because some adders and shifters can be shared among different multiplications.

Many algorithms have been proposed to make the multiplier block as simple as possible: Bull-Horrocks(BH) algorithm [2], n-dimensional reduced adder graph(RAGn) algorithm [1], recursive bipartite matching algorithm [3], and common subexpression sharing algorithm [4], [5], [6]. The main purpose of these algorithms is to minimize the number of additions/subtractions, as the number is proportional to the number of two-input adders required in the implementation of a multiplier block and the shift can be implemented by wire connections. However, the algorithms do not take into account a factor critical in high performance filters, the delay of the multiplier block, leading to slow filters that may not be suitable for high performance systems.

In this paper, we propose new multiplier block synthesis algorithms that consider both the delay and the number of adders. Since the proposed algorithm can generate a multiplier block satisfying a given delay constraint, it enables a trade-off between delay and area. The rest of this paper is organized as follows. In Section II, the problem to be solved is formally defined, and in Section III, some basic operations proposed to make the multiplier block meet the specified delay constraint are described. We explain the proposed algorithm and its implementation in Section IV. Then, we show experimental results in detail in Section V, and finally conclusions are made in Section VI.

II. Problem Definition

In this section, the problem to be solved will be defined formally. We will start from introducing the following term to be used throughout this paper.

Adder-Step: One adder-step represents an adder/subtractor in a maximal path of decomposed multiplications. A multiplication can have different adder-steps, depending on the structure of multiplication.

The problem to be solved is described as follows:

Problem 1: Given a delay constraint and a set of filter coefficients, generate a multiplier block satisfying the delay constraint such that the number of adders/subtractors is minimal.

As the delay is dependent on several implementation issues such as circuit technology, placement and routing, we regard in this paper the delay is specified by the number of adder-steps that denotes the maximal number of adders/subtractors allowed to pass though to produce any multiplication. In this case, the above definition is restated as follows.

Problem 2: Given a maximal number of adder-steps and a set of filter coefficients, generate a multiplier block that needs a minimal number of adders/subtractors and does not violate the number of adder-steps.

One simple method of achieving the minimum number of adder-steps N is to construct coefficients individually by using a separate binary tree of adders for each c_i, meaning that adders associated with c_i are not shared with those of other c_j.

TABLE I
FILTER SYNTHESIS RESULTS OF THE PREVIOUS ALGORITHMS

Test filter	Simple method		BHM[2], [1]		RAGn[1]	
	#adder-steps	#adders	#adder-steps	#adders	#adder-steps	#adders
1	3	49	4	14	5	14
2	3	57	5	33	7	33
3	3	54	5	29	8	28
4	3	112	8	68	10	61

In Table I, the number of adder steps resulted from previous algorithms is compared with N for several FIR filters. In the simple method, each coefficient is represented by a CSD value and constructed with a separate binary tree of adders. From this comparison, we can conclude that the previous algorithms are effective in reducing the number of adders but not optimized for delay. The previous algorithms have not taken into account the delay of the multiplier block. Therefore their results are not suitable for the implementation of fast filters.

III. Methods for Reducing the Number of Adder-Steps

In this section, we explain three basic methods that are essential in reducing the number of adder-steps. Before starting the explanation, we briefly introduce two previous filter synthesis algorithms, BHM and RAGn, that are based on the graph representation. The two algorithms are selected here as they produce the minimal number of adders among many published algorithms

In the BHM algorithm [1], [2], to synthesize a coefficient, a pair of partial sums whose sum or difference is closest to it is selected. And the sum or difference becomes a new partial sum. If the sum or difference is not the same as the coefficient, selecting a pair of partial sums is repeated until the sum or difference has the same value.

A new algorithm, called RAGn, is also proposed in [1]. The major difference between the BHM algorithm and the RAGn algorithm is that the coefficient which requires the least number of adders is first synthesized in the RAGn algorithm while in the BHM algorithm the coefficients are synthesized in the previously defined order. The RAGn algorithm is divided into two parts: the optimal part and the heuristic part.

A. Tree Reduction

In constructing coefficients by using the BHM algorithm or the RAGn algorithm, several partial sums are selected and added in a serial manner as shown in the left of Fig. 1. It is obvious that the serial structure increases the number of adder-steps. Though the cases do not occur frequently, their effect on the number of adder-steps is significant. To reduce the number of steps for the cases, we can employ a tree reduction technique illustrated in Fig. 1. The tree reduction technique is used to convert a serial adding structure to a parallel one.

In applying the proposed tree reduction technique to the BHM algorithm, the sum or difference is put into a temporary set instead of directly putting into the partial sum set. When the synthesis of a coefficient is completed, the partial sums stored in the temporary set are sorted in ascending order of their number of adder-steps,

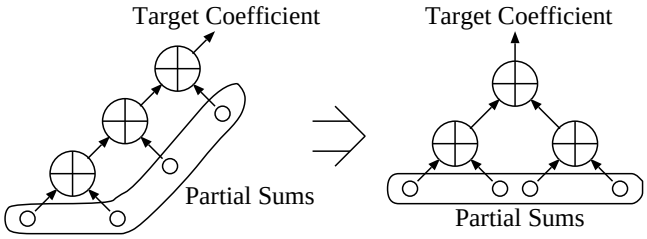

Fig. 1. Tree reduction.

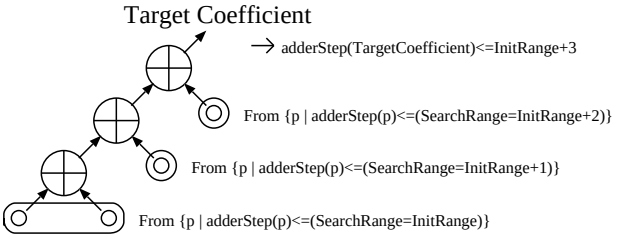

Fig. 2. Limited selection method.

and the partial sums with smaller numbers of adder-steps are added earlier.

B. Limited Selection Method

In this and the next subsection, we propose methods to design a multiplier block satisfying a given delay specified by the number of adder-steps. In our investigation on the previous algorithms, we found that a coefficient is synthesized by a series of partial sums and the number of adder-steps for the coefficient is determined mostly by the first pair of partial sums in that series, that is, the adder-steps required to synthesize the first pair of partial sums has a great effect on the final number of adder-steps. If we can start from a pair requiring small numbers of adder-steps in implementing its partial sums, the coefficient can be synthesized with a less number of adder-steps. The basic idea is to select the first pair from a limited set of partial sums whose adder-steps are less than or equal to a given number. An example is illustrated in Fig. 2, where the following terms are used.

InitRange: the upper limit of the number of adder-steps that the partial sums in the first selected pair can have.

SearchRange: the upper limit of the number of adder-steps that the partial sums selected at that moment can have.

CandidateSet: a subset of partial sums that have the number of adder-steps are equal to or less than *SearchRange*.

In Fig. 2, the first pair of partial sums shown at the bottom is selected by setting *SearchRange* to *InitRange*. Then the *CandidateSet* is limited to $\{p \mid adderStep(p) \leq (SearchRange = InitRange)\}$, where *adderStep(p)* is the number of adder-steps needed for the partial sum p. To select a new partial sum, *SearchRange* is increased by one and the *CandidateSet* is less limited to $\{p \mid adderStep(p) \leq (SearchRange = InitRange+1)\}$. At the next time, *SearchRange* is increased by one again. If a coefficient is to be synthesized with four partial sums as shown in Fig. 2, it is guaranteed that the number of adder-steps for the coefficient is less than or equal to (*InitRange*+3).

The complete description of the limited selection method is as follows. We begin with the maximally allowable *InitRange*

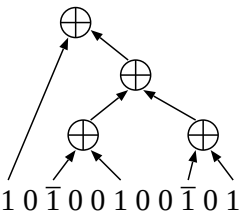

Fig. 3. Adding structure for achieving the minimum number of adder-steps.

that is one less than the specified number of adder-steps, and *SearchRange* is set to *InitRange*. After the first pair is selected, the error between the coefficient being synthesized and the sum or difference of the selected pair is calculated. Then a new partial sum closest to the error is selected with *SearchRange* increased and the error is re-calculated. The selection procedure is iterated until the sum or difference coincides with the coefficient. As mentioned above, *SearchRange* is increased after each selection, and a less limited *CandidateSet* is considered in the later selection. After synthesizing the coefficient, it is examined whether the synthesis of the coefficient meets the specification or not. If not, another iteration is repeated after decrementing *InitRange* i.e. reducing the *CandidateSet* for the first pair. If the iteration reaches to a situation in which *InitRange* is less than 1, i.e., the *CandidateSet* cannot be reduced, we conclude that this method cannot synthesize the coefficient under the given delay constraint. The coefficient given up will be synthesized by the method to be explained in the next subsection.

C. Minimum Adder-Step Method

This method is invoked when some coefficients are not synthesized using the above two methods. It is induced from the structure of the minimum number of adder-steps. If we want to synthesize a coefficient with the minimum number of adder-steps, we represent it in the CSD form and add the non-zero digits using the tree structure illustrated in Fig. 3.

In the minimum adder-step method, the procedure for the minimum number of adder-steps is progressed step by step. One of the remained coefficients that do not satisfy the specification is selected, and for convenience let us call the coefficient c_i. A pair of non-zero digits in the CSD form of c_i is selected. Though any pair can be randomly selected, we select two non-zero digits at the lower bit location in our implementation. The value of the pair is calculated and becomes a new partial sum. Next, the methods described in the above subsections are progressed again for the remained coefficients. If coefficient c_i is synthesized with satisfying the specification in the new iteration, another coefficient that is not synthesized with a satisfactory number of adder-steps is selected and a new partial sum is generated by selecting a new pair of non-zero digits in the CSD form of the coefficient. If coefficient c_i does not satisfy the specification in the new iteration, another pair of non-zero digits is selected from its CSD, excluding the previously selected pair. The selected pair becomes a new partial sum and the methods described in the above subsections are processed again. As the procedure is basically the same as synthesizing a coefficient with the minimum number of adder-steps, any coefficient can be synthesized with satisfying the specification unless the specification is less than the minimum number of adder-steps.

IV. Proposed Algorithms

In this section, we describe two proposed algorithms that can generate multiplier blocks satisfying the given delay constraint. The proposed algorithms are based on three methods of reducing the number of adder-steps and two previous algorithms, BHM and RAGn.

A. Step-Limiting BHM Algorithm (SLBHM)

Three methods explained in the previous subsections, tree reduction, limited selection method, and minimum adder-step method, can be combined with the BHM algorithm. To synthesize a coefficient, the partial sums selected for the coefficient are put into the temporary set and rearranged by the tree reduction technique. After each synthesis, it is examined whether the synthesis satisfies the specification. If it is, a new synthesis starts for another coefficient. Otherwise, the candidate set where the partial sums are selected is changed by the limited selection method. This is iterated until all coefficients are tried. As stated before, however, the limited selection method does not guarantee the synthesis of all coefficients. If all coefficients are not synthesized, a new partial sum is generated by the minimum adder-step method and the procedure is repeated. We name this algorithm as step-limiting BHM(SLBHM) algorithm.

B. Step-Limiting RAGn Algorithm (SLRAGn)

The limitation method can be easily applied to the RAGn algorithm. In the optimal part of the RAGn algorithm, the partial sums whose number of adders-steps are less than the specification are searched. If one coefficient is synthesized at the optimal part only using such partial sums, it satisfies the specification. The heuristic part can be divided into two parts: the cost-2 part that requires two adders and the cost-more part that needs more than two adders. Two cases of the cost-2 part are described in [1]. One is that a scaled version of 1 and two partial sums are added and the other is that two scaled versions of 1 and one partial sum are added. However, there is one more case that three scaled partial sums are added. If we regard 1 as a partial sum, these three cases can be assembled to only one case that three scaled partial sums are added. As the case that more than two selected partial sums require (*specification*-1) adder-steps is unacceptable, such partial sums that make the case are excluded in the selection of partial sums. In order to reduce adder-steps, the cost-more part is replaced by the minimum adder-step method, because the cost-more part is very heuristic and can be replaced by any reasonable procedure. We name this algorithm as step-limiting RAGn(SLRAGn) algorithm.

V. Experimental Results

The proposed algorithms are applied to several FIR filters and compared with previous algorithms. The specification of those filters are summarized in Table II, where f_p and f_s are normalized passband frequency and stopband frequency respectively, *#tap* is

TABLE II
TEST FILTER SPECIFICATION

Filter	f_p	f_s	#tap	Width
1	0.15	0.25	60	14
2	0.15	0.20	60	16
3	0.10	0.15	60	14
4	0.10	0.12	100	18

TABLE III
NUMBERS OF ADDERS FOR FILTER 1

#adder-steps	Simple	BHM	RAGn	SLBHM	SLRAGn
3	49			17	16
4		14		14	14
5			14		14

the number of taps, and *Width* is the word size in fixed point integer representation. All of algorithms are implemented by C language and the heuristic part of the RAGn algorithm is some modified because it can't be applied to the filter design with coefficients of width more than 12 bits.

In Table III, the results of filter 1 obtained by the previous and proposed algorithms are shown. The first column is the number of adder-steps for the multiplier block optimized by the algorithms identified in the first row and the contents of the table is the number of adders needed to implement the multiplier block. So the BHM algorithm produces a multiplier block of 14 adders and 4 adder-steps and the RAGn algorithm produces one of 14 adders and 5 adder-steps. The SLBHM algorithm produces two multiplier blocks: one is with 14 adders and 4 adder-steps and the other with 17 adders and 3 adder-steps. The SLRAGn algorithm provides three multiplier blocks. The first one is with 14 adders and 5 adder-steps, the second one with 14 adders and 4 adder-steps, and the last with 16 adders and 3 adder-steps. Notice that the previous algorithms, BHM and RAGn, give only one result and do not allow to specify the maximum number of adder-steps, while the proposed algorithm, SLBHM and SLRAGn, provide several results under the given delay constraint. It can be seen easily that the number of adder-steps can be reduced by 1 or 2 with additional 2 or 3 adders. In Table IV, V, and VI, we can see the similar results.

This implies the proposed algorithms enable the trade-off between the number of adders and the number of adder-steps, i.e., between the area and the speed. In some cases, only 10% overhead is required to reduce the number of adder-steps by half. The last point to be noted in the tables is that the SLBHM algorithm and the SLRAGn algorithm can produce a multiplier block with the minimum number of adder-steps. This is obvious because the minimum adder-step method guarantees the synthesis of any coefficients when the specification is more than or equal to the minimum number of adder-steps.

TABLE IV
NUMBERS OF ADDERS FOR FILTER 2

#adder-step	Simple	BHM	RAGn	SLBHM	SLRAGn
3	57			43	37
4				35	33
5		33		34	32
6					
7			33		

TABLE V
NUMBERS OF ADDERS FOR FILTER 3

#adder-step	Simple	BHM	RAGn	SLBHM	SLRAGn
3	54			33	36
4				31	31
5		29		29	29
6					28
7					28
8			28		28

TABLE VI
NUMBERS OF ADDERS FOR FILTER 4

#adder-step	Simple	BHM	RAGn	SLBHM	SLRAGn
3	112			89	73
4				77	69
5				69	68
6				67	64
7					62
8		68			62
9					61
10			61		61

VI. Conclusions

Delay is as important as area. In the previous works, however, only area or the number of adders is considered in implementing and optimizing filters. In this paper, we have described FIR filter synthesis algorithms that take into account delay, or the number of adder-steps, as well as area. By combining three proposed methods to the BHM and the RAGn algorithms which are developed in the previous works, we can implement filters satisfying the given specification of the number of adder-steps. Contrast to the previous works that generate only one tuple of the number of adders and the number of adder-steps, many tuples are generated in the proposed algorithms, and therefore the trade-off between area and speed is enabled. Experimental results show that the proposed algorithms can reduce the delay of multiplier blocks at the cost of a little increase of complexity.

References

[1] A. G. Dempster and M. D. Macleod, "Use of minimum adder multiplier blocks in FIR digital filters," *IEEE Transactions on Circuits and Systems-II:Analog and Digital Signal Processing*, vol. 42, no. 9, pp. 569–77, 1995.

[2] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *IEE Proceedings-G*, vol. 138, no. 3, pp. 401–12, 1991.

[3] M. Potkonjak, M. B. Srivastava, and A. Chandrakasan, "Efficient substitution of multiple constant multiplications by shifts and additions using iterative pairwise matching," in *DAC94, Proceedings of 31st ACM/IEEE Design Automation Conference*, 1994, pp. 189–94.

[4] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Transactions on Circuits and Systems-II:Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 677–88, 1996.

[5] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Duraeková, "A new algorithm for elimination of common subexpressions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 1, pp. 58–68, 1999.

[6] J. T. Kim, *Design and implementation of computationally efficient FIR filters, and scalable VLSI architectures for discrete wavelet transform*, Ph.D. thesis, Korea Advanced Institute of Science and Technology, 1998.

[7] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," *IEE Proc.-Circuits Devices Systems*, vol. 141, no. 5, pp. 407–13, 1994.