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

Hyeong-Ju Kang, *Student Member, IEEE*, and In-Cheol Park, *Member, IEEE*

***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.**

***Index Terms*—Digital filter, filter optimization, FIR filter, multiplier block.**

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. Early works have focused on replacing multiplications by decomposing them into simple operations such as addition, subtraction and shifting. As the coefficients of an application specific filter are constant, the decomposition is more efficient than employing multipliers. The complexity of FIR filters in this case is dominated by the number of additions/subtractions used to implement the coefficient multiplications. To reduce the complexity, the coefficients can be restricted to powers-of-two or expressed in canonical signed-digit (CSD) or graph representation to minimize the number of additions/subtractions required in each coefficient multiplication. On the average, the CSD representation can reduce 33% of nonzero digits compared with the binary representation. The above approaches deal with each coefficient individually, whereas there is another approach in which all coefficients are considered as a whole. As shown in Fig. 1, the hardware block called a multiplier block is 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.

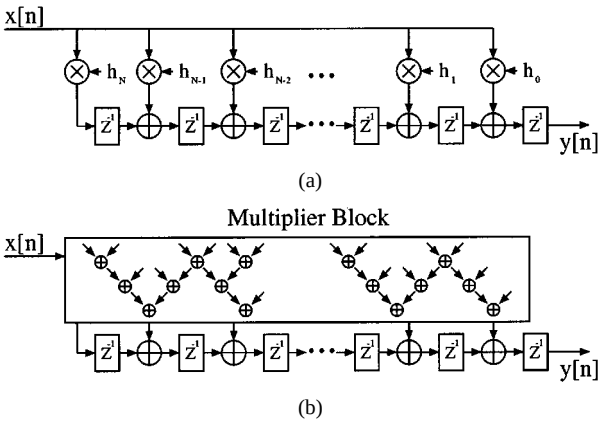

Fig. 1. FIR filter structure. (a) General transposed form. (b) Multiplications are replaced by a multiplier block.

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]–[6]. The main purpose of these algorithms is to minimize the number of additions/subt ractions, as the number is proportional to the number of two-input adders required in the implementation of a multiplier block and the shifting 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

Manuscript received July 31, 2001. This paper was recommended by Associate Editor S. Sriram.

The authors are with the Division of Electrical Engineering, Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Taejeon, Korea 305-701 (e-mail: dk@ics.kaist.ac.kr; icpark@ics.kaist.ac.kr).

Publisher Item Identifier S 1057-7130(01)09606-9.

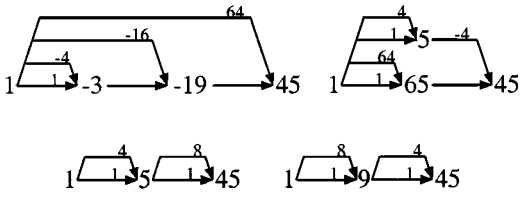

Fig. 2. Examples of graph representation.

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 terms to be used throughout this paper:

1) CSD Representation: The number $a_{N-1}a_{N-2}\ldots a_0$ is said to be in CSD representation if no two nonzero digits are consecutive and the number of nonzero digits is minimal, where each b_i is in the set $\{0, +1, -1\}$ and the -1 is often denoted by $\bar{1}$;

2) Graph Representation: This is a graphical method introduced in [2], [7] to represent multiplication by a constant integer, where each vertex except the initial and terminal vertices means an adder, and each edge is associated with a value to be multiplied with the left vertex of the edge. The value is a positive or negative constant of a power of two. The initial vertex is assigned to 1 and the result of the multiplication is obtained from the terminal vertex. The example of 45 quoted from [7] is redrawn in Fig. 2 for easy understanding;

3) 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. For example, the CSD value of 45 can have 2 adder-steps or 3 adder-steps if it is implemented in serial or parallel, respectively, as shown in Fig. 2.

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.

To investigate the results of previous algorithms, we will use the following lemmas.

Lemma 1: For a coefficient multiplication in which the coefficient, c_i, is represented by a CSD number with k_i nonzero digits, the adder-steps required in the multiplication, n_i, is given by

$$n_i \geq \lceil \log_2 k_i \rceil \tag{1}$$

where $\lceil x \rceil$ represents an integer no less than x. The equality holds when the multiplication is constructed by using a complete binary tree of adders.

This lemma straightforwardly leads to another lemma described below:

Lemma 2: For a set of coefficients, $\{c_0, c_1, c_2, \ldots, c_m\}$, the low bound of adder-steps, N, required in implementing the multiplier block is given by

$$N = \max\{\lceil \log_2 k_i \rceil\} \tag{2}$$

where k_i is the number of nonzero digits in the CSD format of c_i.

One simple method of achieving 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. For a set of coefficients, the minimal delay of a multiplier block can be calculated by Lemma 2. 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 which is inversely proportional to performance. The less delay, the more performance. Since the previous approaches have tried to minimize the number of adders in implementing a multiplier block, they have not taken into account the delay of the multiplier block. Therefore, their results are not suitable for the implementation of fast filters. This does not mean that there is no technical method to achieve a faster filter from the multiplier block generated from the previous approaches. One simple solution to increase performance is to insert pipeline registers in the middle of the multiplier block, as shown in Fig. 3. The insertion of pipeline registers has two drawbacks. First, the number of pipeline registers is numerous in practical filters, thus leads to excessive area and power. Second, the latency of the multiplier block can be increased much as each pipeline stage has the same cycle time. For example, let us assume that the original latency of the multiplier is 110 ns and the cycle time is 100 ns. In this case the multiplier block has to be partitioned into two pipeline stages, resulting in the latency of 200 ns. Hence, it is desired to develop a new algorithm that can generate a multiplier block under a given delay constraint.

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, BH 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.

The first step of the BH algorithm [2] is to convert all the given coefficients into the positive odd numbers by negating the

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

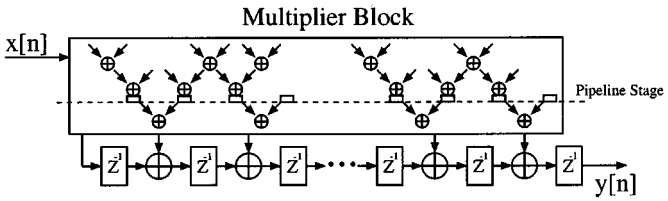

Fig. 3. Pipelined multiplier block.

negative ones and dividing them by the greatest power-of-two. Sorting them in ascending order, the algorithm synthesizes the changed coefficients one by one by using add, subtract, and shift operations. To synthesize a coefficient, a pair of partial sums whose sum or difference of scaled versions is closest to the coefficient being synthesized 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 as the coefficient being synthesized. This process is continued until all coefficients are synthesized.

As identified in [1], the BH algorithm has three limitations. The first is that partial sums are generated with values only up to, but not exceeding, the coefficient. The second is that even valued partial sums can be entered in the partial sum set, and the third is that the coefficients are processed in numerical order. In order to overcome these limitations a modified version of the BH algorithm, called BHM, is proposed and 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 first part is optimal and the second part is heuristic. In the optimal part, the coefficients that can be synthesized with one adder are synthesized. If all the coefficients are synthesized in the optimal part, it is guaranteed that the number of adders is minimal. This is the reason why this part is called optimal. When some coefficients cannot be synthesized in the optimal part, the heuristic part is progressed. The first step in the heuristic part is to synthesize the coefficients which can be synthesized with two adders. Only two cases are considered in [1], but there are three cases to be considered. It will be discussed in the next section. The second step in the heuristic part is to synthesize a coefficient that requires the least number of adders. In [1], the minimum adder graph (MAG) algorithm [7] is used for the comparison of the number of adders in this step, but its use is limited up to 12 bandwidth. In this paper, therefore, the number of nonzero digits is used for the comparison. Although the multiplier block

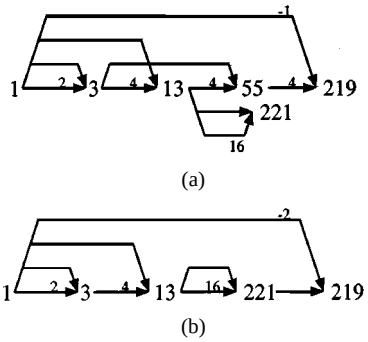

Fig. 4. Graph representations of synthesizing 3, 13, 219, and 221. (a) By the BHM algorithm. (b) By the RAGn algorithm.

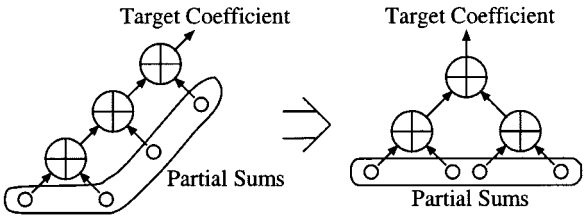

Fig. 5. Tree reduction.

resulted from the RAGn algorithm is superior to that from the BHM algorithm, the RAGn algorithm suffers from large amount of memory and computation time.

In Fig. 4, a set of coefficients $\{3, 13, 219, 221\}$ is synthesized to compare the BHM algorithm and the RAGn algorithm. The coefficients are synthesized with 5 adders in the BHM algorithm and 4 adders in the RAGn algorithm while with 9 adders in the form of canonical signed digit (CSD) representation. Comparing the delay, only 2 adder-steps are required in the form of CSD while 4 adder-steps are taken in the BHM algorithm and the RAGn algorithm. Therefore, the BHM algorithm and the RAGn algorithm are good for reducing area but not optimized in the view of speed. As the partial sums are added in a serial manner and the number of adder-steps increased by including a partial sum is not considered in selecting the partial sum, those algorithms have a limitation in reducing the number of adder-steps. To overcome the limitation, the following three methods are proposed.

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. 5. 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. If such a case occurs, it increases the

number of adder-steps by 3 or more. To reduce the number of steps for the cases, we can employ the tree reduction technique illustrated in Fig. 5. The tree reduction technique is used to convert the serial adding structure to a parallel one. Since each partial sum requires a different number of adder-steps, the partial sums requiring a less number of adder-steps should be added earlier.

It is difficult to consider this in the BHM algorithm in which the sum or difference of a selected pair of partial sums becomes a new partial sum. In the proposed tree reduction technique, 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, and the partial sums with smaller numbers of adder-steps are added earlier. The algorithm to make a tree structure of adders is as follows, where *NumTemp* is the number of partial sums in the initial temporary set.

Step 1) Sort the partial sums in the temporary set in ascending order of their numbers of adder-steps.
Step 2) Select the first two partial sums in the order, that is, two partial sums that have the smallest number of adder-steps.
Step 3) Add (or subtract) them and insert the sum (or difference) into the partial sum set. They are removed in the temporary set and the sum (or difference) is inserted into the temporary set as a new partial sum.
Step 4) Decrement *NumTemp*.
Step 5) If *NumTemp* is not one, go to Step 1.

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 investigations 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. 6, where the following terms are used:

InitRange: upper limit of the number of adder-steps that the partial sums in the first selected pair can have;
SearchRange: upper limit of the number of adder-steps that the partial sums selected at that moment can have;
CandidateSet: subset of partial sums that have the number of adder-steps are equal to or less than *SearchRange*.

In Fig. 6, 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. 6, it is guaranteed that the number of adder-steps for the coefficient is less than or equal to $(InitRange + 3)$. As started from a limited set of partial sums, this method is referred to as *Limited Selection Method*.

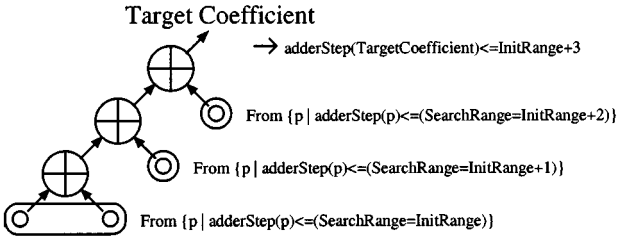

Fig. 6. Limited selection method.

The complete description of the limited selection method is as follows. In order to least restrict the *CandidateSet* for the first pair, we begin with the maximally allowable *InitRange* that is one less than the specified number of adder-steps. And then *SearchRange* is set to *InitRange* and is increased after each selection of a partial sum. Therefore, the partial sum selected later will be selected from the less restricted *CandidateSet*. After the first pair is selected, the error between the coefficient being synthesized and the sum or difference of the selected pair is calculated and a new partial sum is generated. The selection procedure is iterated until the sum or difference coincides with the coefficient. In the iteration, partial sums are selected one by one. As mentioned above, *SearchRange* is increased after each selection, and a less restricted *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* and reducing the *CandidateSet* for the first pair. If the iteration reaches to a situation in which *InitRange* is less than 1 or 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.

The detailed procedure of this method is as follows.

Step 1) *InitRange* is set to the limit of the number of adder-steps specified by the user subtracted by 1.
Step 2) $SearchRange = InitRange$.
Step 3) *CandidateSet* $= \{p | p$ is a partial sum that has the number of adder-steps equal to or less than *SearchRange*$\}$.
Step 4) Select a pair of partial sums in *CandidateSet* whose sum or difference of scaled versions is closest to the coefficient being synthesized. The sum or difference becomes a new partial sum. If the error value representing the difference between the coefficient and the sum or difference is zero, go to Step 7).

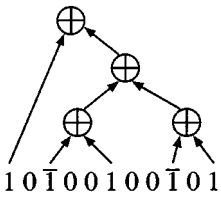

Fig. 7. Adding structure for achieving the minimum number of adder-steps.

Step 5) If *SearchRange* is less than the limit of the number of adder-steps specified by the user subtracted by 1, *SearchRange* is incremented by 1, and the *CandidateSet* is constructed again with the new *SearchRange*.

Step 6) Select a partial sum in *CandidateSet* whose scaled version is closest to the error value. A new partial sum is generated with the scaled partial sum, and the error value is reset to the difference between the current error value and the partial sum. If the error value is not zero, go to Step 5).

Step 7) If the number of adder-steps for the coefficient being synthesized is equal to or less than the specification of the number of adder-steps, end.

Step 8) *InitRange* is decremented by 1. If *InitRange* is less than zero, give up. Otherwise, go to Step 2).

C. Minimum Adder-Step Method

The 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 nonzero digits using the tree structure illustrated in Fig. 7.

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 nonzero digits in the CSD form of c_i is selected. Though any pair can be randomly selected, we select two nonzero 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 the coefficient c_i is synthesized with satisfying the specification in the new iteration, another coefficient not synthesized with a satisfactory number of adder-steps is selected and a new partial sum is generated by selecting a new pair of nonzero digits in the CSD form of the coefficient. If the coefficient c_i does not satisfy the specification in the new iteration, another pair of nonzero 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. This procedure can be iterated until the coefficient c_i has no pair of nonzero digits. 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. An example of this method is illustrated in Fig. 8, where we assume that a coefficient of 405, or $10\bar{1}0010101$, is not synthesized in the above methods.

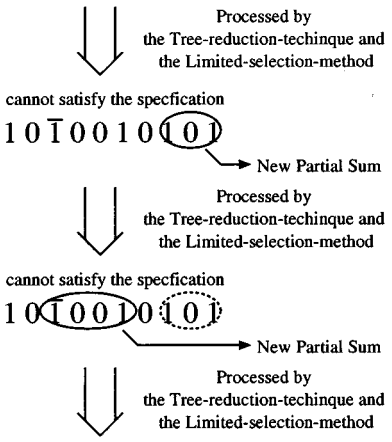

Fig. 8. An example of the minimum adder-step method.

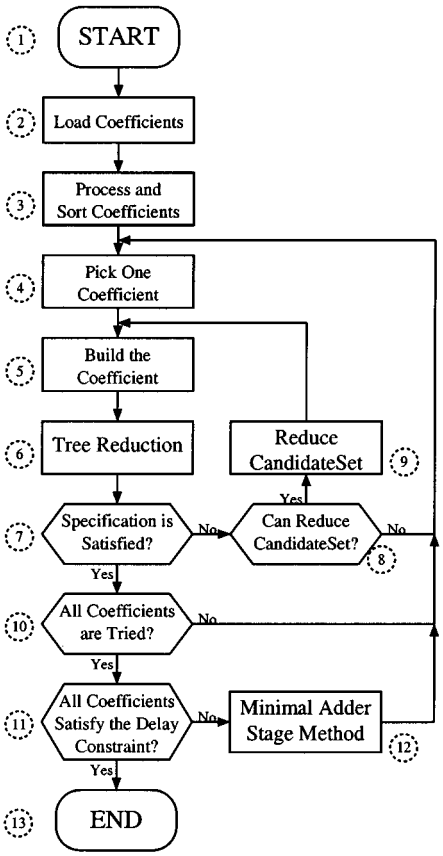

Fig. 9. Overall procedure of SLBHM.

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, the BHM algorithm and the RAGn algorithm.

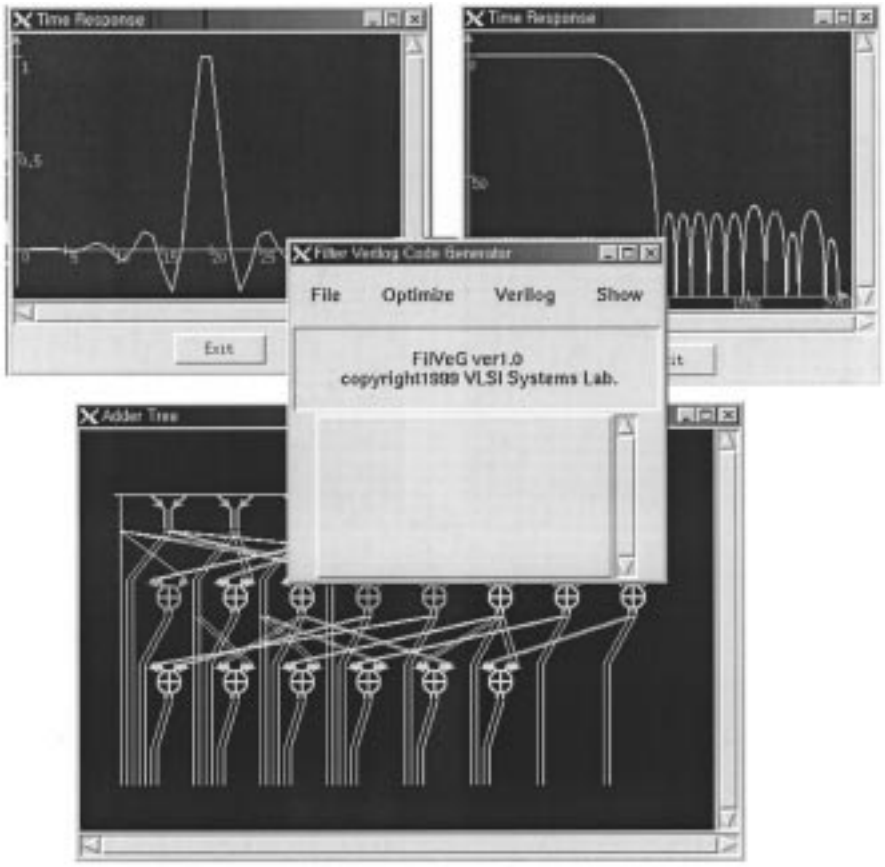

Fig. 10. Appearance of the filter syntheisis tool.

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. The flow chart of the new algorithm named as "Step-Limiting BHM algorithm(SLBHM)" is shown in Fig. 9. Details are described below only for the steps that need to be explained.

- Step 3) After negating the negative coefficients, even coefficients are divided by the largest power-of-two to make them odd numbers.
- Step 4) Pick one coefficient that is not synthesized yet.
- Step 5) Build the coefficient selected in Step 4). A pair of partial sums is selected repetitively, and new partial sums are generated as in the BHM algorithm.
- Step 6) The partial sums which are selected in Step 5) are rearranged by the tree reduction technique.
- Step 8) When the specification is not satisfied, the limited selection method is progressed in Steps 8) and 9). Examine whether the candidate set can be reduced. If the candidate set cannot be reduced, give up the coefficient and pick another coefficient Step 4).
- Step 12) When some coefficients are given up, the minimum adder-step method is applied. One new partial sum is generated from a coefficient that is given up in Step 8).

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

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

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 algorithm(SLRAGn)."

C. Filter Synthesis Tool

The proposed algorithms are implemented in C language and incorporated into a filter synthesis tool, as shown in Fig. 10. Given the set of coefficients for a target filter and the number of adder-steps, the tool can synthesize a multiplier block of the filter and generate a hardware description in Verilog hardware description language. For the implementation of the multiplier block, carry-save adders are used to minimize area and delay, while for the serial addition stage under the multiplier block shown at the bottom of Fig. 10, the tool supports two adder types, carry-propagation adders and carry-save adders.

Besides the proposed algorithms, several previous algorithms such as the BHM algorithm and the RAGn algorithm are supported in the tool.

V. EXPERIMENTAL RESULTS

The proposed algorithms are applied to several FIR filters and compared with previous algorithms. The specifications of those filters are summarized in Table II, where f_p and f_s are normalized passband frequency and stopband frequency, respectively, *#Tap* is the number of taps, and *Width* is the word size in fixed point integer representation. The coefficients of test filters are generated with specifying f_p, f_s, and *#Tap* using the Remez algorithm in MATLAB and are converted to integer numbers with rounding. The passband and stopband frequency of the first filter in Table II are quoted from the example in [8].

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 that 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.

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

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.

Similar results can be seen in Tables IV–VI which represent the results for filter 2, 3, and 4, respectively. For filter 2, three multiplier blocks are synthesized using the SLBHM algorithm and the SLRAGn algorithm. In addition, one result of the SLRAGn is superior to that of the RAGn in that both the number of adders and the number of adder-steps are smaller than those of the RAGn. For filter 3, 3 and 6 multiplier blocks are generated by the SLBHM algorithms and the SLRAGn algorithm, respectively. For filter 4, 4 and 8 multiplier blocks are generated. The overhead of reducing the number of adder-steps in the proposed algorithms is not significant. For filter 2, 10 and 4 additional adders are required to reduce 2 and 4 adder-steps. For filter 3, 4 and 8 adders are needed to reduce 2 and 5 adder-steps and for filter 4, 21 and 12 adders are needed to reduce 5 and 7 adder-steps, respectively.

This implies the proposed algorithms enable the trade-off between the number of adders and the number of adder-steps, e.g.,

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.

VI. Conclusion

Delay is as important as area. But in the previous works, 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 algorithm and the RAGn algorithm 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, a trade-off between the number of adders and the number of adder-steps, that is, 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 Trans. Circuits Syst. II*, vol. 42, pp. 569–577, Sept. 1995.

[2] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *Proc. Inst. Elect. Eng.*, pt. G, vol. 138, pp. 401–412, Mar. 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 *Proc. 31st ACM/IEEE Design Automation Conf.*, 1994, pp. 189–194.

[4] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 677–688, Oct. 1996.

[5] R. Paško, P. Schaumont, V. Derudder, S. Vernalde, and D. Ďuračková, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 58–68, Jan. 1999.

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

[7] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," *Proc. Inst. Elect. Eng.—Circuits Devices Systems*, vol. 141, pp. 407–413, May 1994.

[8] H. Samueli, "An improved search algorithm for the design of multiplier-less FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. CAS-36, pp. 1044–1047, July 1989.

[9] R. Jain, P. T. Yang, and T. Yoshino, "FIRGEN: A computer-aided design system for high performance FIR filter integrated circuits," *IEEE Trans. Signal Processing*, vol. 39, pp. 1655–1668, July 1991.

[10] T. Yoshino, R. Jain, P. T. Yang, H. Davis, W. Gass, and A. H. Shah, "A 100-MHz 64-Tap FIR digital filter in 0.8-um BiCMOS gate array," *IEEE J. Solid-State Circuits*, vol. 25, pp. 1494–1501, June 1990.

[11] Y. C. Lim, J. B. Evans, and B. Liu, "Decomposition of binary integers into signed power-of-two terms," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 667–672, June 1991.

[12] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, "Synthesis of multiplier-less FIR filters with minimum number of additions," in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design*, 1995, pp. 668–671.

Hyeong-Ju Kang (S'00) received the M.S. and B.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Taejeon, Korea, in 1998 and 2000, respectively. He is currently pursuing the Ph.D. degree in the Department of Electrical Engineering and Computer Science at KAIST.

His research interests include VLSI design for signal processing and multimedia.

In-Cheol Park (S'88–M'92) received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Taejeon, Korea, in 1986, 1988, and 1992, respectively.

From 1995 to 1996, he was with the postdoctoral technical staff at IBM T.J. Watson Research Center, Yorktown, NY. In 1996, he became a professor in the Department of Electrical Engineering and Computer Science at KAIST. His research interest includes CAD algorithms for high-level synthesis and VLSI architectures for general-purpose microprocessors.