Short Papers

FIR Filter Synthesis Considering Multiple Adder Graphs for a Coefficient

Jeong-Ho Han and In-Cheol Park

***Abstract*—To reduce the hardware complexity of finite-impulse response (FIR) digital filters, this paper proposes a new filter synthesis algorithm. Considering multiple adder graphs for a coefficient, the proposed algorithm selects an adder graph that can be maximally sharable with the remaining coefficients, whereas previous dependence-graph algorithms consider only one adder graph when implementing a coefficient. In addition, an addition reordering technique is proposed to derive multiple adder graphs from a seed adder graph generated by using previous dependence-graph algorithms. Experimental results show that the proposed algorithm reduces the hardware cost of FIR filters by 22% and 3.4%, on average, compared to the Hartley and n-dimensional reduced adder graph hybrid algorithms, respectively.**

***Index Terms*—Digital filter, filter optimization, finite-impulse response (FIR) filter synthesis, multiplier block.**

NOMENCLATURE

- $\boldsymbol{C}$ A set of coefficients to be synthesized. Each coefficient is uniquely represented by an odd positive number, and $|\boldsymbol{C}|$ represents the number of coefficients in $\boldsymbol{C}$.
- $\boldsymbol{S}$ A set of partial sums that are generated so far. Starting from $\{1\}$, the set grows as coefficients are one by one synthesized. A partial sum that is in the current partial sum set is called a cost-0 partial sum as it can be directly used to synthesize other coefficients.
- $\boldsymbol{G}$ A set of adder graphs generated for a coefficient. This set is generated by applying the proposed addition reordering to a seed adder tree.
- $\boldsymbol{T}$ A table of partial sums. Initially, this table contains the information on the shift amounts and signs of the cost-0 partial sums involved in a seed adder graph. $|\boldsymbol{T}|$ represents the number of entries in $\boldsymbol{T}$, t_i denotes the ith entry, and p_i denotes the partial sum value in t_i.

I. INTRODUCTION

Finite-impulse response (FIR) digital filters are widely employed in digital signal processing due to their stability and linear-phase property. In many applications requiring high-speed processing and low power consumption, FIR filters are implemented as dedicated filter blocks. Because an FIR filter consists of many constant multiplications, earlier papers have decomposed the multiplications into simple operations such as addition, subtraction, and shift and have tried to share as many partial sums as possible to reduce the hardware complexity. As shown in Fig. 1, all coefficients are considered as a whole to design all the constant multiplications into a hardware block called a multiplier block. There are two metrics that are important in the design and comparison of digital FIR filters. The first one is the adder cost, which is the number of adders required to implement a given set of filter coefficients, and the second is the adder step, which is the number of adders passing through the critical path. Many algorithms have been proposed to reduce the adder cost and adder step, which can be categorized into two groups: 1) dependence-graph and 2) common subexpression elimination (CSE) algorithms.

The objective of CSE algorithms is to find bit patterns that are common in more-than-one coefficients to reduce the total adder cost by sharing the common bit patterns among the coefficients. To reduce the number of nonzero bits or to maximize the sharing of common bit patterns, the coefficients are converted to a specific number representation such as the signed digit (SD) number system [3], the canonic SD (CSD) number system [4]–[7], or the minimal SD number system [8] before searching common bit patterns.

The goal of dependence-graph algorithms is to find an adder graph consisting of the minimal number of partial sum nodes. The total adder cost can be reduced by sharing the previous partial sums generated for a subset of constant multiplications in deriving additional constant multiplications. The relationship among partial sums can be graphically represented, as shown in Fig. 2, where a node is associated with the value of the corresponding partial sum. The number on each edge indicates the shift amount to be applied to the input partial sum, and the minus symbol indicates that the input partial sum is negated before adding it to the output node. For the sake of simplicity, the shift amount of zero is not explicitly shown on the edge. The Bull–Horrocks (BH) [1] and modified BH (BHM) [2] algorithms generate an adder graph by inserting new partial sums that are required to minimize the difference between a coefficient to be synthesized and the currently available partial sums. The n-dimensional reduced adder graph (RAGn) algorithm [2] introduces the concept of adder distance. The adder distance of a coefficient is the number of adders that is additionally required to achieve the coefficient value from a given adder graph. In general, the RAGn algorithm shows better performance than other dependence-graph and CSE algorithms. However, a precalculated lookup table (LUT) is used to find the adder distance in the RAGn algorithm, and it induces a limitation on the range of coefficient values. To overcome this limitation, the RAGn hybrid algorithm [2] replaces the heuristic part requiring the precalculated LUT with the BHM algorithm. Recently, a graph-theoretical approach called shift inclusive differential coefficient (SIDC) [10] has been proposed to consider the differences among the coefficients. The differences are first synthesized and then the coefficients are synthesized based on the differences, but partial sums are not shared between the two steps. For the applications in which the logic delay is more crucial than the logic complexity, the adder step [9] and the wire delay [11] are taken into account as additional metrics.

The previous dependence-graph algorithms consider one coefficient at a time and do not take into account the effect on the rest of the coefficients when synthesizing the coefficient. Though the adder graph generated in such a way is the best solution for the coefficient, it may not be the best if the remaining coefficients are considered. In this paper, we propose a new dependence-graph algorithm to minimize the adder cost by considering the effect on the remaining coefficients. Considering multiple adder graphs for a coefficient, the proposed algorithm selects an adder graph that can be maximally shared with the remaining coefficients. To reduce the computation overhead, a simple

Manuscript received July 10, 2007; revised October 10, 2007. This work was supported by the Korean Intellectual Property Office. This paper was recommended by Associate Editor R. Camposano.

The authors are with the Division of Electrical Engineering, School of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Korea (e-mail: jhhan@icslab.kaist.ac.kr; icpark@ee.kaist.ac.kr).

Digital Object Identifier 10.1109/TCAD.2008.917581

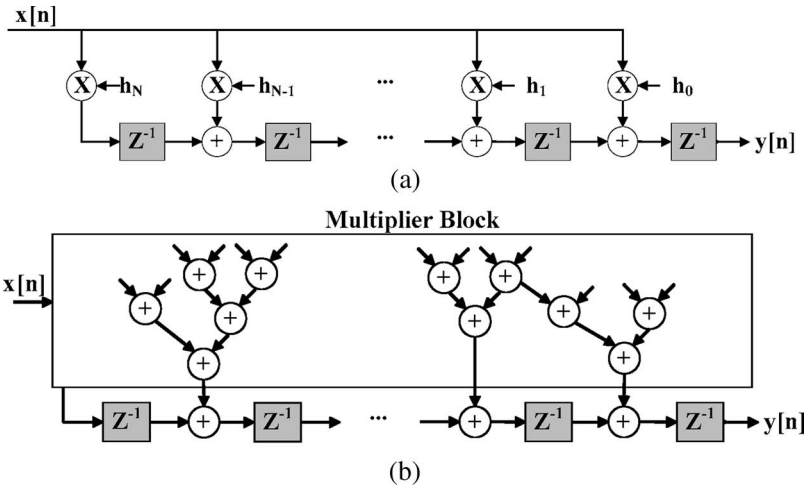

Fig. 1. FIR filter structure. (a) Transposed form. (b) FIR filter using a dedicated multiplier block.

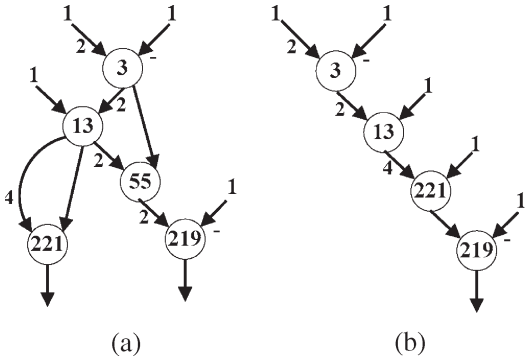

Fig. 2. Filter coefficient set $\{3, 13, 219, 221\}$ generated by (a) the BHM algorithm and (b) the RAGn algorithm.

addition reordering technique is applied to generate multiple candidate graphs from a seed adder graph.

II. Proposed Algorithm

Because synthesizing an adder graph for a set of filter coefficients is nondeterministic-polynomial-time-complete [1], previous dependence-graph algorithms such as the BHM and RAGn algorithms rely on heuristic approaches. Considering one coefficient at a time, the heuristic algorithms minimize the additional adder cost required to incorporate the coefficient into the current partial sum set. The adder graph generated for a coefficient may not be the best if we consider the whole coefficient set because the remaining coefficients are not considered in generating the adder graph. If we consider multiple candidate adder graphs and then select an adder graph that can be maximally shared with the remaining coefficients, the total adder cost can be further reduced.

Fig. 3 illustrates an example of the adder graph generation process for a coefficient set {3, 53, 585}. There are two coefficients 53 and 585 to be synthesized, and the initial condition is presented in Fig. 3(a) where three adder graphs are possible for 53. The previous dependence-graph algorithm generates an adder graph by inserting new partial sums that minimize the difference to a coefficient and does not consider the remaining coefficients when synthesizing the coefficient, as shown in Fig. 3(b). In the example, 53 is synthesized before 583, and only a graph (graph_1) is taken into account for 53. The partial sums generated for 53 cannot be shareable with 585, and the total adder cost becomes 6. Fig. 3(c) shows the proposed adder

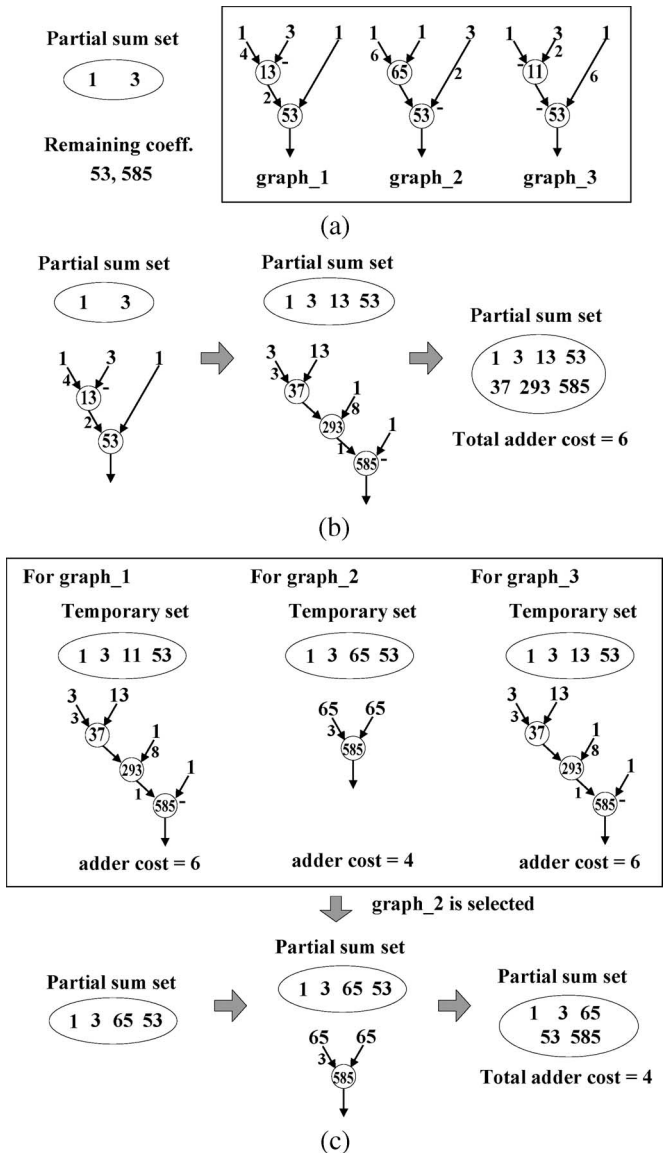

Fig. 3. Synthesis procedure for a coefficient set $\{3, 53, 585\}$. (a) Initial condition and three possible adder graphs for 53. (b) Previous dependence-graph algorithms (BHM and RAGn hybrid). (c) Proposed algorithm that considers three adder graphs for 53.

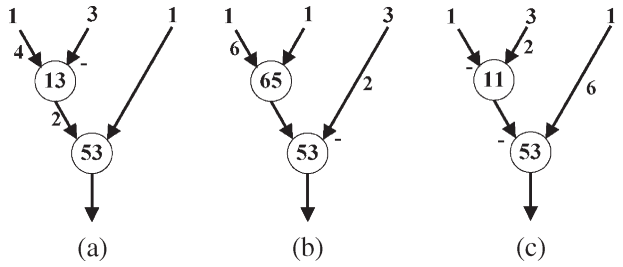

Fig. 4. Two adder graphs (b) and (c) generated by applying the addition reordering to (a).

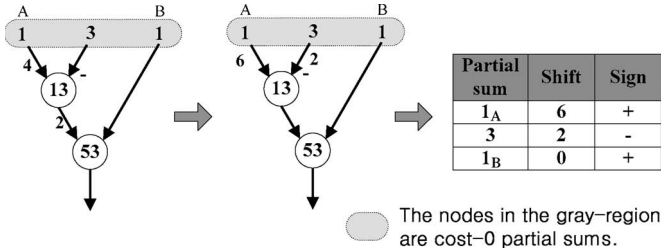

Fig. 5. Expanding a seed adder graph and making a table of cost-0 partial sums. From the root node of 53, the shift amount and the sign of a node are propagated to the upper nodes.

graph generation process that considers the remaining coefficients. In this case, the three graphs are considered for 53. To select a proper graph from the multiple graphs, the proposed algorithm generates a temporary partial sum set for each graph by tentatively inserting the partial sums contained in the graph into the current partial sum set and then synthesizes the remaining coefficients by applying the conventional dependence-graph algorithm such as the BHM algorithm to the temporary set. After examining the total adder cost of each graph, the proposed algorithm selects a proper graph (graph_2, in this case) that minimizes the total adder cost. The partial sums of graph_2 are inserted to the current partial sum set. In this case, 585 can be synthesized by using one additional adder, and the total adder cost becomes 4. As observed in this example, the total adder cost can be reduced if we consider the sharing effect on the remaining coefficients when synthesizing a coefficient.

A. Generating Multiple Adder Graphs

The computational complexity to find all the possible adder graphs is high even for a coefficient. To reduce the computation overhead, an efficient method is proposed in this subsection to generate multiple adder graphs for a coefficient. In the proposed method, a seed adder graph is first generated for a coefficient to be synthesized by applying one of the previous dependence-graph algorithms and then multiple adder graphs are generated by the proposed addition reordering technique that changes the order of additions in the seed adder graph. Fig. 4 illustrates that two additional adder graphs (b) and (c) are generated from a seed adder graph (a) that is generated by the BHM algorithm. The proposed addition reordering is different from the computation reordering [10] that is used to express the filter with the differences of coefficients. In the sense that the differences are computed earlier, the computation is reordered in [10].

Before applying the addition reordering, the seed adder graph is expanded into cost-0 partial sums that are already in the current partial sum set. Fig. 5 illustrates how to expand the seed adder graph. Note that the seed graph can be constructed by using a cost-0 partial sum more than once. Let us assume that the seed graph is a tree. In other words, the number of edges coming out from a node is always one in the seed adder graph. If there is a node violating this assumption, the seed graph can be transformed into a tree by duplicating such a node. In this case, we can move the shift amount and the sign of an outgoing edge to the input edges, as there is only one outgoing edge in a tree. Starting from the root node of 53, the shift amount and the sign of the output edge are propagated to each input edge, i.e., the shift amount and the sign are added to and multiplied by those of each input edge, respectively. This propagation is recursively applied until it reaches to the cost-0 partial sum nodes. Through the adder graph expanding procedure, we make a partial sum table $\boldsymbol{T}$ that stores the shift amount and the sign of each cost-0 partial sum node contained in the seed adder tree.

To generate an adder graph, we select two partial sums t_i and t_j from $\boldsymbol{T}$ and make a new entry t_{new} by adding the shifted values of p_i and p_j. Then, we update $\boldsymbol{T}$ by removing t_i and t_j and inserting t_{new}. To maximize the sharing of partial sums, the value of p_{new} is restricted to a positive odd number by adjusting its shift amount and sign value when inserting the new partial sum into $\boldsymbol{T}$. This procedure is recursively applied until only a partial sum remains in the table. Following the table update sequence, we generate an adder graph that results in the target coefficient. Fig. 6 illustrates an example where the first and third entries are selected first to make a new partial sum of 65 by performing $(1 \ll 6) + (1 \ll 0)$ and then the two entries in the updated table are combined to derive a new partial sum of 53 by performing $(65 \ll 0) - (3 \ll 2)$.

Multiple adder graphs can be obtained by changing the combination of two partial sums to be selected from $\boldsymbol{T}$. The total number of adder graphs N that can be generated from a seed adder graph is expressed as follows:

$$N = \prod_{i=0}^{n-3} {}_{n-i}C_2, (n \geq 3)$$
$$= \frac{n!}{2!(n-2)!} \times \frac{(n-1)!}{2!(n-3)!} \times \cdots \times \frac{3!}{2!1!} = \frac{n! \times (n-1)!}{2^{n-1}} \quad (1)$$

where n is $|\boldsymbol{T}|$. As indicated in (1), the number of adder graphs exponentially grows as n increases. For example, the number of possible adder graphs is 3 for $n = 3$, 18 for $n = 4$, and 180 for $n = 5$. To reduce the computation overhead, multiple adder graphs are considered only when n is less than 5 in this paper. This restriction will be justified later in Section III.

The procedure to generate multiple adder graphs is summarized below, and a pseudocode of the major function called *Build_graphs* is shown in Fig. 7 where *new_psum* (t_i, t_j) generates a new entry in $\boldsymbol{T}$ by adding the shifted values of the partial sums in t_i and t_j, *insert_node* (p_i, g) enlarges the adder tree g by inserting p_i, and *refine_graph*(g) combines a number of nodes that have the same partial sums into a node so as to convert g to an adder graph. The new partial sum is restricted to an odd positive number by adjusting the shift amount and the sign value. In the proposed algorithm, a set of adder graphs $\boldsymbol{G}$ is used to store adder graphs that are found by applying the proposed addition reordering to a seed adder graph.

Step 1) Initialize $\boldsymbol{G}$ to ϕ.
Step 2) Transform the seed adder graph g to an adder tree.
Step 3) Make a partial sum table $\boldsymbol{T}$ from the adder tree by using the expanding procedure.
Step 4) If $|\boldsymbol{T}| > 4$ or $|\boldsymbol{T}| = 2$, then $\boldsymbol{G} = g$ and finish.
Step 5) To generate multiple adder graphs, call function *Build_graphs* $(\boldsymbol{T}, g_{\text{empty}})$ where g_{empty} is an empty tree.

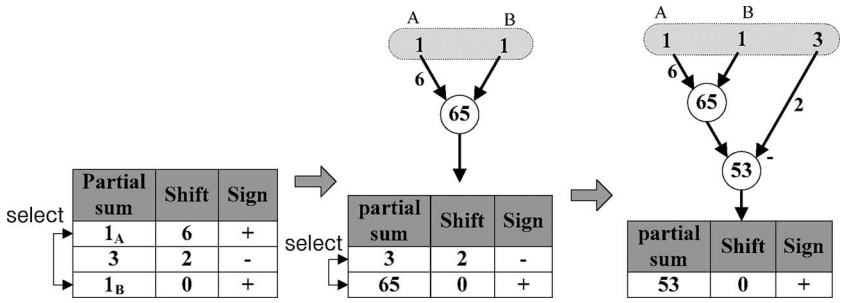

Fig. 6. Generating an adder graph from the partial sum table.

```
Build_graphs (T, g) {
 If (|T| == 2) {
  t_new = new_psum (t1, t2).  (where, t1, t2 ∈ T)
  g_new = insert_node (p_new, g)
  g_cand = refine_graph (g_new)
  G = G ∪ {g_cand} }
 else {
  for (i=1 .. |T|-1) {
   for (j=i+1 .. |T|) {
    t_new = new_psum (t_i, t_j).  (where, t_i, t_j ∈ T)
    g_new = insert_node (p_new, g)
    T_tmp = T − t_i − t_j + t_new
    Build_graphs (T_tmp , g_new) }}}}
```

Fig. 7. Generating multiple adder graphs based on the addition reordering.

B. Description of the Proposed Algorithm

The proposed algorithm is divided into two parts. The first part is to select a coefficient to be synthesized from $\boldsymbol{C}$, which is the set of coefficients that is not yet synthesized. We select a coefficient that is associated with the lowest adder distance. In the second part, multiple adder graphs are generated for the selected coefficient by using the proposed addition reordering, and an adder graph is selected that seems to be maximally shared with the remaining coefficients. Then, the partial sums in the selected adder graph are inserted into the partial sum set. The proposed algorithm is as follows.

Step 1) Prepare for starting.
- 1.1) Make a coefficient set $\boldsymbol{C}$ from the given filter coefficients.
- 1.2) Sort the coefficients in $\boldsymbol{C}$ in the increasing order of the number of nonzero bits in the CSD representation.

Step 2) Select a coefficient to be synthesized.
- 2.1) For each $c \in \boldsymbol{C}$, find an adder graph by using the BHM algorithm.
- 2.2) Select a coefficient $c_{\rm sel}$ associated with the minimum adder distance, and remove $c_{\rm sel}$ from $\boldsymbol{C}$.

Step 3) For $c_{\rm sel}$, make a graph set $\boldsymbol{G}_{\rm cand}$ that contains multiple adder graphs generated by the addition reordering.

Step 4) Select a proper adder graph.
- 4.1) For each $g_i \in \boldsymbol{G}_{\rm cand}$,
  - 4.1.1) Make a temporary partial sum set $\boldsymbol{S}_i$ by merging $\boldsymbol{S}$ and the partial sums contained in g_i.
  - 4.1.2) Synthesize all the remaining coefficients in $\boldsymbol{C}$ by using the BHM algorithm with $\boldsymbol{S}_i$ as the current partial sum set.
- 4.2) Select an adder graph g_j that minimizes the total adder cost.

Step 5) Update $\boldsymbol{S}$ by merging $\boldsymbol{S}_j$ to the current $\boldsymbol{S}$.

TABLE I
TEST FILTER SPECIFICATIONS

Filter	# tab	Wordlength	Type	Fs1	Fp1	Fp2	Fs2
1	60	16	LP	0.14	0.1		
2	100	16	LP	0.6	0.2		
3	101	16	HP	0.3	0.42		
4	101	16	HP	0.3	0.76		
5	60	16	BP	0.2	0.3	0.8	0.9
6	60	16	BP	0.2	0.45	0.65	0.9

TABLE II
ADDER COSTS FOR THE TEST FILTERS

Filter	Hartely [5]	NCSE [12]	BHM [2]	RAGn-hybrid [2]	Proposed
	AC (%)	AC (%)	AC (%)	AC (%)	AC (%)
1	49 (100)	39 (80)	42 (86)	42 (86)	37 (76)
2	77 (100)	62 (81)	57 (74)	57 (74)	54 (70)
3	41 (100)	37 (90)	37 (90)	37 (90)	37 (90)
4	58 (100)	53 (91)	51 (87)	50 (86)	49 (84)
5	40 (100)	32 (80)	31 (78)	29 (73)	29 (73)
6	31 (100)	28 (90)	26 (84)	26 (84)	25 (81)
Ave.	49.3 (100)	41.8 (84.8)	40.7 (82.4)	40.2 (81.4)	38.5 (78)

* AC : adder cost,
* (%) : adder cost ratio in % compared to Hartely

Step 6) Remove the synthesized coefficients from $\boldsymbol{C}$, that is, $\boldsymbol{C} = \boldsymbol{C} - (\boldsymbol{C} \cap \boldsymbol{S})$.

Step 7) If $|\boldsymbol{C}| > 0$, go to Step 2.

III. EXPERIMENTAL RESULTS

The proposed algorithm is applied to six FIR filters to compare with previous CSE algorithms such as the Hartley [4] and novel CSE algorithms [12] and dependence-graph algorithms such as the BHM [2] and RAGn hybrid algorithms [2]. The RAGn hybrid algorithm is used in the comparison rather than the RAGn algorithm having a limitation on the coefficient wordlength. Table I shows the filter specifications. The coefficients are generated by the Remez algorithm in MATLAB, and the 16-bit wordlength is larger than the 12-bit limitation of the RAGn algorithm.

Synthesis results are summarized in Table II where we can see that the proposed algorithm outperforms other algorithms in terms of the adder cost. Compared to the Hartley algorithm, the proposed algorithm reduces the adder cost by 22% on the average, whereas the others reduce by 15% to 18%. As indicated in [10], the SIDC algorithm improves by 11% compared to the Hartley algorithm.

TABLE III
FILTER DELAYS CONSIDERING FAN-OUTS

Filter	BHM[2]		RAGn-hybrid[2]		Proposed	
	AS	delay (ns)	AS	delay (ns)	AS	delay (ns)
1	7	6.67	7	6.67	6	5.84
2	8	7.7	8	7.7	9	8.91
3	6	5.86	4	3.91	6	5.86
4	7	6.72	7	6.81	6	5.97
5	5	4.9	6	5.78	5	4.92
6	6	5.8	5	4.75	5	4.91
Ave.	6.5	6.23	6.17	5.94	6.17	6.07

* AS : Adder step

TABLE IV
DISTRIBUTION OF THE NUMBER OF PARTIAL SUMS

Number of partial sums	2	3	4	5	6	7	8
Occurrence (%)	84.2	13.49	1.69	0.37	0.11	0.1	0.03

TABLE V
SYNTHESIS RESULTS FOR 417 FILTERS COMPARED TO THE BHM ALGORITHM

	RAGn-hybrid[2]	Proposed
Number of improved cases	226	321
Number of worse cases	0	0
Maximum number of reduced adders (%)	12 (13.89%)	14 (15.91%)
Average number of reduced adders (%)	1.43 (1.67%)	3.03 (3.38%)

Maximizing partial sum sharing may increase fan-outs, leading to a delay increment. For the dependence-graph algorithms, Table III shows the adder steps and the delays obtained in a 0.18-μm complementary metal–oxide–semiconductor technology. Therefore, the proposed algorithm reduces more adders than other dependence-graph algorithms while retaining the delays.

In the proposed algorithm, multiple adder graphs are considered only when the number of partial sums is less than 5. To justify the restriction, Table IV shows the distribution of the number of partial sums obtained by expanding seed adder graphs. The percentage that the number of partial sums is greater than 4 is less than 0.62%, meaning that the restriction has a negligible effect on the final synthesis results. We synthesized a number of filters without resorting to the restriction, but there were almost no improvements compared to the results obtained with the restriction. If the restriction is not taken, the computation overhead of the proposed algorithm can exponentially grow as the number of coefficients increases. The restriction plays an important role in dramatically reducing the computation overhead.

For rigorous comparison, additional experiments were performed for 417 FIR filters on a computer system equipped with a 3-GHz Pentium processor and a 1-GB memory. The filter order ranges from 60 to 300, and the wordlength of coefficients is at least 16 bits. Compared to the BHM algorithm, the proposed algorithm improves 321 cases, whereas the RAGn hybrid algorithm improves 226 cases, as shown in Table V. Note that there are no degrading cases. On the average, the proposed algorithm reduces the adder cost by 3.38% including unimproved cases, whereas the RAGn hybrid algorithm reduces by 1.67%. For some filters, the proposed algorithm reduces the adder cost by 15.91%. Table VI shows the average computation times for various filter orders, which reveals that the practical computation time of the proposed algorithm is linearly, not exponentially, proportional to the filter order.

TABLE VI
COMPUTATION TIMES FOR 417 FILTERS

Filter order	60	120	180	240	300
Average computation time (sec)	31.16	63.26	89	117.13	138.23

IV. CONCLUSION

We have presented a new filter synthesis algorithm that minimizes the adder cost. The proposed algorithm considers multiple adder graphs for a coefficient to be synthesized to take into account the effect on the remaining coefficients, whereas previous dependence-graph algorithms consider only one adder graph when implementing a coefficient. An adder graph that can be maximally shared with the remaining coefficients has been selected among multiple adder graphs. Starting from a seed adder graph, we can derive multiple adder graphs by applying the proposed addition reordering technique. To reduce the computational complexity of considering multiple adder graphs, the proposed method has been applied when the number of partial sums of the seed adder graph is less than 5, as the other cases are very rare. Experimental results have shown that the proposed algorithm outperforms the previous dependence-graph algorithms such as the BHM and RAGn hybrid algorithms.

REFERENCES

[1] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *Proc. Inst. Electr. Eng G—Circuits Devices Syst.*, vol. 138, no. 3, pp. 401–412, Jun. 1991.

[2] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 9, pp. 569–577, Sep. 1995.

[3] 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, no. 6, pp. 667–672, Jun. 1991.

[4] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 43, no. 10, pp. 677–688, Oct. 1996.

[5] R. Pasko, P. Schaumont, R. Derudder, S. Vernalde, and D. Durackova, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 1, pp. 58–68, Jan. 1999.

[6] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. CAS-36, no. 7, pp. 1044–1047, Jul. 1989.

[7] A. P. Vinod and E. M.-K. Lai, "On the implementation of efficient channel filters for wideband receivers by optimizing common subexpression elimination methods," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 2, pp. 295–304, Feb. 2005.

[8] I.-C. Park and H.-J. Kang, "Digital filter synthesis based on an algorithm to generate all minimal signed digit representations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 12, pp. 1525–1529, Dec. 2002.

[9] H.-J. Kang and I.-C. Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 8, pp. 770–777, Aug. 2001.

[10] H. Choo, K. Muhammad, and K. Roy, "Complexity reduction of digital filters using shift inclusive differential coefficients," *IEEE Trans. Signal Process.*, vol. 52, no. 6, pp. 1760–1772, Jun. 2004.

[11] D. Kang, H. Choo, K. Muhammad, and K. Roy, "Layout-driven architecture synthesis for high-speed digital filters," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 2, pp. 203–207, Feb. 2006.

[12] C.-Y. Yao *et al.*, "A novel common-subexpression-elimination method for synthesizing fixed-point FIR filters," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 11, pp. 2215–2221, Nov. 2004.