A Novel Low-Complexity Method for Parallel Multiplierless Implementation of Digital FIR Filters

Yongtao Wang and Kaushik Roy
School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907, USA
Email: {yw, kaushik}@ecn.purdue.edu

***Abstract*— We present a computation reduction method which can be used to obtain low-complexity parallel multiplierless implementation of digital FIR filters, exploring the use of shift inclusive differential (SID) coefficients and common subexpression elimination (CSE). We introduce a new directed multigraph to represent the design space greatly expanded by the use of SID coefficients. A graph-theoretic algorithm is then employed to efficiently explore the greatly expanded design space. Further, we propose a novel CSE method applied to the design space represented by the graph, which recursively eliminates 2-bit subexpressions with a steepest descent approach for subexpression selection. Compared with conventional multiplierless implementation, up to 75% reduction in terms of number of additions has been achieved. In comparison to a recently reported CSE method based on available data, our approach achieves an improvement up to 19%.**

I. Introduction

Future battery-powered wireless communication systems are expected to provide higher data rates with improved energy-efficiency. This has made low-power, high-performance digital signal processing (DSP) an important research area. Low-complexity design, which aims at reducing the number of certain basic operations (e.g., addition) for a given DSP task, is an attractive approach. The resultant complexity reduction can potentially improve the processing speed of a DSP algorithm while achieving high energy-efficiency by removing energy consuming operations. In addition, low-complexity design generally leads to smaller chip area and thus lower cost.

FIR filtering with a set of fixed coefficients is widely used in many DSP and communication applications. In the tranposed direct form of a FIR filter as shown in Fig. 1, the input data $x(n)$ is simultaneously multiplied by the set of M filter coefficeints $\mathbf{c} = [c_0, c_1, ..., x_{M-1}]^T$, where M is the filter length. Complexity reduction on these multiplications in the multiplication network can lead to significant improvements in various design parameters such as speed, area or power dissipation. Since multiplication with a constant can be substituted by shifts and additions/subtractions and in dedicated fully parallel implementation shift operation can be simply done by wiring, considerable amount of research work has been conducted to reduce the number of additions, thereby leading to low-complexity designs.

Common subexpression elimination (CSE) has been extensively studied and various algorithms have been proposed [1]-[4]. One common feature of the CSE method is to identify common bit-patterns in the set of coefficients and to share those identified common subexpressions to reduce the number of additions.

This research was supported in part by SRC (1122.001)

The use of shift inclusive differential (SID) coefficient is proposed in [5]. With the assistance of graph representation, the problem of searching for low-complexity solutions is mapped into a weighted minimum set cover problem and a heuristic algorithm based on a greedy approach is given.

In this paper, we reformulate the idea of SID coefficient by introducing a new graph representation and mapping the optimization problem into an equivalent problem of determining a directed minimum spanning tree (DMST) of a directed multigraph, which is then solved by an optimal graph-theoretic algorithm. To achieve further complexity reduction, we propose a CSE method which recursively eliminates 2-bit subexpressions with a steepest descent approach for subexpression selection.

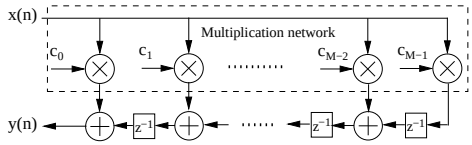

Fig. 1. Transposed direct form of a M-tap FIR filter.

II. A reformulation of the idea of shift inclusive differential coefficient

A. A new graph representation

The key idea behind the use of shift inclusive differential (SID) coefficients is that the result of $c_i x(n)$ could be reused for computing $c_j x(n)$. Note that if $c_i x(n)$ has been computed already, we can obtain $2^L c_i x(n)$ by simple wiring. Thus we can express $c_j x(n) = 2^L c_i x(n) + (c_j - 2^L c_i)x(n)$ or $c_j x(n) = -2^L c_i x(n) + (c_j + 2^L c_i)x(n)$, where $c_j \pm 2^L c_i$ are the SID coefficients [5]. For each value of L, $c_j \pm 2^L c_i$ represents two SID coefficients. For a wordlength of W, L could be in the range from -W to W. Hence for each coefficient, there could be maximally 2(W+1)(M-1) SID coefficients. This greatly expands the design space. In conventional CSE methods [1]-[4], optimizations are performed only over the original set of coefficients $\mathbf{c} = [c_0, c_1, ..., x_{M-1}]^T$. However, by using the SID coefficients, we can explore potentially better solutions in this greatly expanded design space. In the sequel, coefficient $c_j - 2^L c_i$ will be used to represent an SID coefficient, and

the operator '-' may either represent an add or a subtract operation. In addition, for clarity we call each c_i with $0 \leq i \leq (M-1)$ as an *original* coefficient to distinguish it from the SID coefficients.

This greatly expanded design space can be well represented by a directed graph $G = (V, E)$. The vertex set V consists of all the original coefficients and a virtual vertex vn, namely, $V = \{c_0, c_1, ..., c_{M-1}, vn\}$. The virtual vertex vn has no physical meaning and is introduced to help formulate the problem. All the edges of graph G are defined as follows. Each SID coefficient, $c_j - 2^L c_i$, corresponds to an edge directed from c_i to c_j. For different values of L, there would be multiple edges directed from c_i to c_j. There is only one edge directed from the virtual vertex vn to vertex c_i , which represents the original coefficient c_i. Hence G is a directed graph with multiple edges, i.e. a directed multigraph. Each edge of G represents either an original coefficient or an SID coefficient. In [5], cases when $L < 0$ are not considered since it may lead to re-quantization or increase the actual word-length of the intermediate computations. However, that argument is not generally true. For example, for an original coefficient $c_i = 000101000$, there is no problem with considering right shifts with $L = -1, -2$, or -3. Hence $L < 0$ will also be considered in our proposed method when the tailing bits of an original coefficient are zeros. There are two clear distinctions between our graph representation and that of [5]: (1) Introduction of the virtual vertex automatically takes the original coefficients into consideration during our later optimizations; (2) $L < 0$ is considered. This makes our graph representation more comprehensive. A graph representation of a 4-tap filter is illustrated in Fig. 2. Further, in the following subsection, an efficient graph-theoretic algorithm is presented to explore the low-complexity solutions.

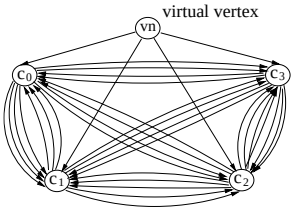

Fig. 2. Directed multigraph for a 4-tap FIR filter.

B. Algorithm SID_DMST

For each edge of graph G, we can assign a weight that represents the associated implementation cost when the corresponding original or SID coefficient is used. In this work, we focus on minimizing the number of additions at a high level of abstraction and the implementation cost is quantified as the number of required additions. Canonic signed digit (CSD) number representation [9] is used, although our proposed approach does not depend on any specific number representation. The weight of each edge of G is defined as follows. For the edge directed from vn to each c_i for $0 \leq i \leq (M-1)$, its weight is the number of additions required for computing $c_i x(n)$, which is one less than the number of nonzero bits of the CSD representation of c_i if c_i is nonzero. The weight is zero if c_i is zero. For each edge associated with an SID coefficient, its weight equals to one plus the number of additions required for computing $(c_j - 2^L c_i)x(n)$ if $c_j - 2^L c_i$ is not zero. If $c_j - 2^L c_i$ is zero, then the weight of the corresponding edge is zero. The optimization problem of constructing a possible implementation with minimum number of additions is to find an acyclic subset of edges in E that connects all of the vertices in V such that the sum of weights of these edges is minimized. In [5], this optimization problem is formulated as a weighted minimum set cover problem and a heuristic algorithm based on a greedy approach is developed.

In this work, we note that the optimization problem is equivalent to determining a directed minimum-weighted spanning tree (DMST) of the directed multigraph G [6]. We present an optimal graph-theoretic algorithm to solve this problem as follows. And we denote the resulting complexity reduction algorithm as *Algorithm SID_DMST*.

We construct a directed graph $G_1 = (V_1, E_1)$, where the vertex set $V_1 = \{0, 1, 2, ..., M\}$ and the edges in E_1 of G_1 are defined as follows. We denote an edge of G_1 directed from vertex i to j by (i, j) and its weight by w_{ij}. For each ordered pair of vertices (c_i, c_j) for $0 \leq i, j \leq (M-1)$, if there is an edge directed from c_i to c_j in G, there will be exactly one edge (i, j) in graph G_1 and its weight is the minimum of the weights of all edges directed from c_i to c_j in G. For each edge directed from the virutal vertex nv to c_i in G, there is one edge (M, i) and its weight is the same as that of the edge directed from nv to c_i.

Assuming we have determined a DMST T_1 of G_1, we can construct a DMST T of G from the DMST of G_1 as follows. For each edge (i, j) of a DMST T_1 of G_1 with $i \neq M$, we select an edge of G directed from c_i to c_j with weight being w_{ij}. For each edge (M, j), we select the edge directed from vn to c_j in G. Let these selected edges form set S. Graph $T = (V, S)$ is a spanning tree of G due to the one-to-one correspondence between the vertices of G and those of G_1. Additionally we will show there is no spanning tree of G that has a total weight smaller than that of T. For descriptive convenience, we denote the total weight of all edges of a graph H as $tw(H)$. Obviously $tw(T_1) = tw(T)$.

Suppose there is a spanning tree T' of G with a smaller total weight than T, i.e., $tw(T) > tw(T')$. We can construct a spanning tree T_1' of G_1 by mapping each edge of T' directed from c_i to c_j into an edge (i, j) of G_1 and each edge directed from vn to c_j into edge (M, j), for $0 \leq i, j \leq (M-1)$. Due to the way we constructed the graph G_1, $tw(T') \geq tw(T_1')$. Thus $tw(T_1) = tw(T) > tw(T') \geq tw(T_1')$. Therefore $tw(T_1) > tw(T_1')$, which contradicts the fact that T_1 is a DMST of G_1. Hence, by contradiction we prove that graph T as constructed above is a DMST of G.

There exists an optimal graph-theoretic algorithm for finding a DMST of G_1[7]. For a directed graph with n vertices and m edges, an implementation of the algorithm which runs in $O(m \log n)$ time was presented in [8]. In finding a DMST of graph G_1, we always select the vertex M as the root of the DMST, which implies that in the DMST of G constructed from the DMST of G_1, the virtual vertex vn will be its root.

As will be evident later, this provides us great convenience in constructing the implementation structure of the multiplication network of a FIR filter.

III. Incorporating Common Subexpression Elimination

The edges of graph G, corresponding to either original or SID coefficients, could have some common subexpressions. It is then natural to consider if we can identify certain common subexpressions and reduce computational complexity through subexpression sharing. This leads us to consider using CSE for further complexity reduction. However, existing CSE methods such as proposed in [1]-[4] are not directly applicable since they are applied to a set of fixed and known coefficients, i.e., the set of *original* coefficients. In this work, we are to apply CSE to a design space represented by a directed multigraph G. Each spanning tree of G corresponds to a possible filter implementation and its edges form a set of coefficients, either original or SID. Hence we need to find a set of subexpressions and a spanning tree of G to minimize the hardware complexity.

One way is to enumerate all the spanning trees of G and on each spanning tree of G, we can apply conventional CSE algorithms since the coefficients in each spanning tree of G are fixed and known. Then the spanning tree with the lowest complexity can be chosen for implementation. However, the number of spanning trees of G is very large and increases exponentially with the number of filter coefficients. Thus it is computationally prohibitive to enumerate all the spanning trees of G.

In this work we propose a CSE method which recursively eliminates 2-bit subexpression with a steepest descent approach for subexpression selection.

A 2-bit subexpression is defined as a bit-pattern with exactly two nonzero bits, where a nonzero bit is an element of set $\{1, -1, 2, -2, 3, -3, ...\}$. In eliminating each occurrence of the subexpression, we replace it by an integer k or -k in place of the second of the two nozero bits making up the subexpression and the first of the two nonzero bits by zero. In the sequel, we will refer to this way of elimination as *elimination by k*. We use $k = 2$ to represent the first eliminated subexpression, $k = 3$ for the second eliminated subexpression, and so on. If the occurrence of the subexpression is exactly same as the subexpression, then it is replaced by k. If the occurrence is the complementary of the subexpression, it is replaced by $-k$. This way of elimination can be best demonstrated by an example. Consider three coefficients, either orignal or SID, in the CSD format, $a_1 = 1010\bar{1}001$, $a_2 = \bar{1}0\bar{1}01000$ and $a_3 = 00000101$, where $\bar{1}$ represents -1. Suppose at the first iteration bit-pattern 101 is selected for elimination. Then after *elimination by 2*, $a_1 = 0020\bar{1}001$, $a_2 = 00\bar{2}01000$ and $a_3 = 00000002$, where $\bar{2}$ represents -2. In the second iteration bit-pattern $20\bar{1}$ is selected and after *elimination by 3*, $a_1 = 00003001$, $a_2 = 0000\bar{3}000$, and $a_3 = 00000002$, where $\bar{3}$ represents -3.

Since in each iteration many different subexpressions will occur, a criterion much be used to select the subexpression for elimination. In conventional CSE algorithms, usually subexpression with the highest frequency is chosen since the amount of complexity reduction achieved by eliminating a subexpression is directly related to its frequency. However, in our case, frequency of the occurrence of a subexpression in all the edges of graph G is usually not a good measure of how much complexity reduction we can achieve if the subexpression is chosen and eliminated. This is mainly because there are so many edges in graph G while just a small portion of the edges will be chosen to form a spanning tree of G. Adopting a steepest descent strategy, in each iteration, among all the subexpressions that occur at least twice, we choose the subexpression whose elimination leads to the greatest complexity reduction. This is elaborated as follows. In each iteration, we first search for all 2-bit subexpressions which occur at least twice, and put them into a set as $SE = \{s_1, s_2, s_3, ...\}$. If we can not find any 2-bit subexpression occurring at least twice, i.e., the set SE is empty, the CSE process is terminated. Otherwise, for each subexpression s_i, let graph $H_i = G$ and eliminate s_i in all edges of graph H_i as described above, update the weights of all edges of H_i by decreasing the edge weight by the number of occurrences of s_i in the edge, determine the DMST of H_i as described in section II, and calculate the complexity reduction in terms of the number of additions as a result of eliminating s_i, which is denoted by Δ_i. Then determine $n = \underset{i}{\operatorname{argmin}} \Delta_i$. If $\Delta_n = 0$, meaning that there is no reduction through subexpression elimination, terminate the CSE process. Otherwiese, put subexpression s_n into a table denoted as CSE_table, update G by letting $G = H_n$ and go into the next iteration with this update graph G. The resulting algorithm is named as *Algorithm SID_CSE*, which is summarized in Fig. 3. *CSE_table* contains the subexpressions that have been eliminated. $|CSE\_table|$ denotes the number of subexpressions in CSE_table.

The final outcome of *Algorithm SID_CSE* is a set of subexpressions contained in *CSE_table* and a DMST of G. Since the DMST of graph G corresponds to a low-complexity implementation of the multiplication network of the filter, we define it as an *implementation tree*. Note that the root of the implementation tree is the virtual vertex vn. Hence, for a vertex c_i of the implementation tree, if its parent is vn, then $c_i x(n)$ is implemented as it is, i.e., using the original coefficient; Otherwise, if the parent of c_i is c_j, $c_i x(n)$ is implemented using the SID coefficient $c_i - 2^L c_j$ corresponding to the specific edge directed from c_j to c_i in the implementation tree. Thus an implementation structure of the filter can be readily derived from the implementation tree.

IV. Numerical Results

We first take 12 example linear-phase filters with filter length ranging from 21 to 161 and wordlength of 16. The filter types include equi-ripple, least-square, low-pass and band-pass. Three techniques are considered, i.e., simple CSD implementation where the filter coefficients are encoded in CSD format, SID_DMST and SID_CSE. The complexity in terms of the number of addition is shown in Fig. 4. In com-

1. Construct G and find a DMST T of G. $cost = tw(T)$. Set CSE_table empty, $noCSE = 0$ and $k = 2$.
2. **while** $noCSE = 0$
   (a) Find the 2-bit subexpressions in the edges of G occurring at least twice. Put them into set $SE = \{s_1, s_2, s_3, ...\}$.
   (b) **if** SE is empty, **then** $noCSE = 1$.
   else
   for each subexpression $s_i \in SE$
   $H_i = G$. Eliminate s_i in edges of H_i by k.
   Update weights of edges of H_i.
   Find a DMST of H_i, denoted as T_i.
   $\Delta_i = cost - tw(T_i) - |CSE\_table| - 1$.
   Determine $n = \underset{i}{\operatorname{argmin}}\, \Delta_i$.
   if $\Delta_n = 0$, **then** $noCSE = 1$.
   else
   Put s_n into CSE_table. $G = H_n$ and $T = T_n$.
   $cost = tw(T_n) + |CSE\_table|$ and $k = k + 1$.
3. Output *implementation tree* as T, the number of additions as $cost$ and CSE_table.

Fig. 3. Algorithm SID_CSE

parison to the simple CSD implementation (CSD), algorithms SID_DMST and SID_CSE achieve 44%-69% reduction and 53%-73%reduciton, respectively.

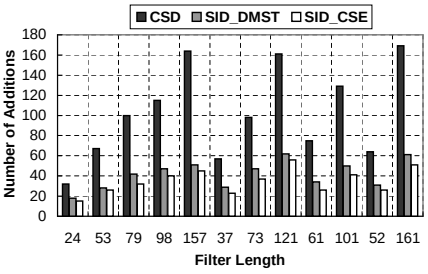

Fig. 4. Complexity reduction on 12 example filters.

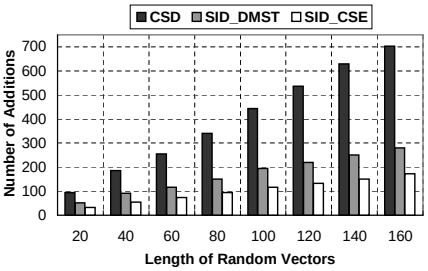

Fig. 5. Complexity reduction on several random vectors.

Our proposed method is also applicable to general multiple constant multiplication (MCM) operations as defined in [1] since the multiplication network of a FIR filter performs exactly a MCM operation. To test the effectiveness of our proposed methods on general MCM operations, we applied our proposed methods to random vectors with length ranging from 20 to 160 and the results are shown in Fig. 5. Compared with simple CSD implementation (CSD), application of SID_DMST algorithm leads to 45%-60% reduction while algorithm SID_CSE leads to 65%-75% reduction. This demonstrates that our proposed methods are effective for general MCM operations.

TABLE I
COMPARISON WITH PAŠKO'S CSE ALGORITHM.

	M	W	CSD	Paško	SID_DMST	SID_CSE
S1	25	9	11	6	6	6
S2	60	14	57	32	29	26
L1	121	17	145	58	61	51
L2	63	13	49	23	24	22
L3	36	11	16	5	5	5

Paško *et al* proposed a CSE algorithm in [4], which achieves comparable or better results than other CSE methods proposed in [1]-[3] . We compare our proposed method with Paško's algorithm based on the data included in [4]. We apply our methods to the filters that are denoted as S1, S2, L1, L2 and L3 in [4]. The results are shown in Table I, where M denotes the filter length and W denotes the wordlength of the filter coefficients. For all the filters, the proposed SID_CSE algorithm yields similar or better results. In particular, for filter L1 and S2, 12% and 19% improvement has been achieved, respectively. The improvement can be attributed to the fact that, in our proposed SID_CSE algorithm, common subexpression elimination is performed over the design space that has been greatly expanded through using SID coefficients. And the expanded design space is represented by a directed multigraph and explored by an efficient graph-theoretic algorithm.

V. CONCLUSIONS

We present a novel low-complexity design method for figital FIR filters. We reformulate the idea of SID coefficients by introducing a new graph representation and employing an efficient graph-theoretic algorithm. Further, we proposed a CSE method which recursively eliminates 2-bit subexpressions with a steepest descent approach. Compared with conventional multiplierless implementation, up to 75% reduction in terms of number of additions has been achieved. In comparison with a recently reported CSE method based on available data, our approach achieves an improvement up to 19%.

REFERENCES

[1] M. Potkonjak, M. B. Srivasta, and P. A. Chandraskasan, "Multiple constant multiplication: Efficient and versatile framework and algorithms for exploring common subexpression elimination", *IEEE Trans. Computer-Aided Design*, vol. 15, no. 2, pp. 151-165, Feb. 1996.

[2] M. Mehendale, S. D. Sherlekar, and G. Vekantesh, "Synthesis of multiplierless FIR filters with minimum number of additions",in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pp. 668-671, Nov. 1995.

[3] R. I. Hartley, "subexpression sharing in filtering using canonic signed digit multiplier", *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 677-688, Oct. 1996.

[4] R. Paško, *et al*, "A new algorithm for elimination of common subexpressions", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 1, pp. 58-68, Jan. 1999.

[5] K. Muhammad and K. Roy, "Minimally redundant parallel implementation of digital filters and vector scaling", *International Conference on Acoustics, Speech and Signal Processing*, vol. 6, pp. 5-9, June 2000.

[6] E. Lawler, Combinatorial optimization: networks and matroid, Saunders College Publishing, 1976.

[7] Y. J. Chu and T. H. Liu, "On the shortest arborescence of a directed graph", Scientia Sinica, vol. 14, pp. 1396-1400, 1965.

[8] R. E. Tarjan, "Finding optimum branchings", *Networks*, vol. 7, pp. 25-35, 1977.

[9] A. Avizienis, "Signed digit number representation for fast parallel arithmetic", *IRE Trans. Electron. Computers*, vol. EC-10, pp. 389-400, 1961.