A Graph Theoretic Approach for Design and Synthesis of Multiplierless FIR Filters

Khurram Muhammad and Kaushik Roy
Email: k-muhammad1@ti.com and kaushik@ecn.purdue.edu
Storage Products Group, Texas Instruments, Dallas, TX and School of ECE, Purdue University, West Lafayette, IN, USA.

Abstract—**We present a novel approach which can be used to obtain multiplierless implementations of** *finite impulse response* **(FIR) digital filters. The main idea is to reorder filter coefficients such that an implementation based on differential coefficients requires only a few adders. We represent this problem using a graph in which vertices represent the coefficients and edges represent the resources required when the differential coefficient corresponding to the edge is used in a computation. We also present a graph model for an implementation based on second-order coefficient differences. The optimal solution to the coefficient reordering problem is the well known problem of finding the** *Hamiltonian path* **of smallest weight in this graph. We use two approaches to find the smallest weight** *Hamiltonian cycle*; **a greedy approach, and, the heuristic algorithm proposed by Lin and Kernighan. The power and potential of this approach is demonstrated by presenting results for large filters (***lengths* **up to > 300) which show that, in general, for 16-bit coefficients, the total number of adders required per coefficient is less than 2. Hence, high performance and/or low power filters can be designed and synthesized using the proposed approach.**

I. Introduction

Future mobile radio and portable computing systems are expected to provide increased services, faster data rates and higher processing speeds at reduced power dissipation levels. This provides us with a motivation to explore new approaches in low-complexity design of high-performance *digital signal processing* (DSP) blocks which operate at lower power levels. Complexity reduction in FIR digital filter implementations has been of particular interest to the DSP system design community [1]. Many previous work have been reported [2], [3], [4] which consider simplified parallel implementations of FIR filters for *signed powers-of-two* (SPT) implementations. One approach uses *integer linear programming* (ILP) to search for a filter which conforms to desired frequency response. Another approach is to start from a known optimal filter solution and search for *quantizations* in the vicinity of the optimal solution which gives a lower implementation cost. The prior approach is computationally impractical as the filter size or the number of bits used to represent a coefficient increases. One disadvantage of these methods is that they are not guaranteed to yield a solution conforming to constraints on the frequency response characteristics of the desired filter.

Low power FIR filter realizations have also been extensively studied in recent years [5], [6], [7]. The basic techniques used in power reduction constitute *architectural transformations*, *coding*, *quantizations*, and *computation reordering*. The idea of *computation reordering* was proposed in [6] in the context of power reduction by reducing the dynamic range of computation using the *differential coefficient method* (DCM). The DCM approach computes the filter output using coefficient differences instead of their original values. This approach effectively reduces the word-length of coefficients from a computational point of view, thereby saving power.

In this paper, we explore complexity reduction in FIR filters from the point of view of removing *computational redundancy* rather than following the traditional approach of removing less significant operations. We define *computational redundancy* as the *excess computation over the minimum number of bit operations needed for a given sequence of operations*. Our goal is to propose a methodology which can be used to design and synthesize high-performance FIR filters by removing redundant computation. The main approach in removing such redundancies is to explore appropriate coefficient reordering. Lower complexity in terms of number of operations directly improves power. The resulting parallel filters may be used for high-performance as well as low-power applications. There are two ways to obtain reduction in power dissipation using this approach. First, we get a direct reduction in power dissipation due to removal of redundant computation. This advantage appears in the form of reduced overall *switching activity* [8] because of relatively fewer computational operations. Second, we obtain a high speed multiplierless implementation which can also be used to further reduce power levels by employing *voltage scaling*.

The main idea of our work is to find an ordering of coefficients which minimizes the number of adders required in the filter implementation using a graph theoretic approach. We employ a *differential coefficient* scheme which can be implemented for any coefficient ordering in digital filters. The main contributions of this work are summarized below:

- The frequency response of the given filter is not altered.
- DCMI approach is independent of the number representation scheme used and the choice of the number of bits to represent the coefficients.
- We map the DCMI problem to a well-known problem of finding the smallest weight Hamiltonian path on a graph. Efficient polynomial time algorithms can be employed to obtain "good" solutions.
- Our approach is general. By modifying the values of edge costs further implementation details can also be accounted for. Similarly, solutions which combine various orders of differential coefficients can also be explored.

This work was supported in part by DARPA (F33615-95-C-1625), NSF CAREER award (9501869-MIP), Rockwell, AT&T and Lucent foundation.

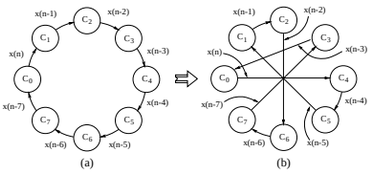

Fig. 1. Graph representation of an example filter with $M = 8$.

This paper is organized in six sections. Section II provides a general background on FIR filtering and the DCM approach in [6]. Section III presents the DCMI approach for removing the computational redundancy from the filter computations. Section IV presents the extension of DCMI approach to second order differential coefficients. Numerical results are presented in section V to quantify the complexity reduction using the proposed methods. Finally, section VI concludes this paper.

II. General Background

Consider a *linear time-invariant* (LTI) FIR filter of length M described by an input-output relationship of the form

$$y(n) = \sum_{i=0}^{M-1} c_i \, x(n-i) = \sum_{i=0}^{M-1} P_i^{(n)} \qquad (1)$$

In this context, c_i represents the *ith* coefficient and $x(n-i)$ denotes the data sample at time instant $n-i$. $P_i^{(n)}$ represents the partial product $c_i x(n-i)$ for $i = 0, 1, \ldots, M-1$ computed at time instant n. Figure 1(a) shows a tour T in a graph representation of the filter at time instant n. In this representation, vertices represent coefficients and the edges, $E_{i,j}$, $i, j = 0, 1, \ldots, M-1$, represent the resources required to multiply the corresponding data sample with the preceding vertex (i.e. coefficient c_i). Note that for both *sign-magnitude* (SM) and SPT representations, the graph is undirected. The coefficients are applied such that c_{j+1} follows c_j, $j = 0, \ldots, M-2$. The appropriate data sample with the corresponding coefficient are shown next to the edges. Since we are considering a parallel filter implementation, $E_{i,j}$ depends only on the number representation scheme and the type of multiplier employed. For example, if an array multiplier is used with SM number representation of coefficients and data, each edge represents the number of adder rows required to compute the product of respective data sample with the coefficient. The total number of adder rows in the multiplier is equal to the number of "1" bits in the corresponding coefficient (rows corresponding to 0's can be removed) and M parallel multipliers are required to obtain the parallel implementation of the M-tap filter.

With the above interpretation of the graph, the total resources required to compute the output given by equation 1 at time instant n is given by the sum of resources required to compute the partial products ($P_i^{(n)}$'s) along each edge in the tour. At the next time instant, $n+1$, each data sample $x(i)$, $i = n, n-1, \ldots n-M+1$ in the graph is replaced by $x(i+1)$. Next, consider the DCM [6] in the context of the graph shown in figure 1(a). The outputs of the filter at time instants $n-1$ and n are given as $y(n-1) = c_0 \, x(n-1) + c_1 \, x(n-2) + \ldots + c_{M-1} \, x(n-M) = P_0^{(n-1)} + P_1^{(n-1)} + \ldots + P_{M-1}^{(n-1)}$ and $y(n) = c_0 \, x(n) + c_1 \, x(n-1) + \ldots + c_{M-1} \, x(n-M+1) = P_0^{(n)} + P_1^{(n)} + \ldots + P_{M-1}^{(n)}$ The first order DCM uses the coefficient difference $c_{i+1} - c_i$, $i = 0, 1, \ldots, M-2$ along the edge $E_{i,i+1}$. Hence, in this case, $E_{i,i+1}$ represents the resources required to compute the product of $c_{i+1} - c_i$ with the corresponding data sample $x(n-i-1)$, at time instant n, for all $i = 0, 1, \ldots, M-2$. Then each vertex can be replaced by the differential coefficient $c_{i+1} - c_i$ except for c_0. The partial product $P_i^{(n)}$ is computed by adding $(c_i - c_{i-1})x(n-i)$ to $P_{i-1}^{(n-1)}$. $P_i^{(n)}$ thus obtained is stored in memory for computing $P_{i+1}^{(n+1)}$ in future and removed subsequently. Hence, multiplication of c_i with $x(n-i)$ is replaced by addition of $P_{i-1}^{(n-1)}$ with the product $(c_i - c_{i-1})x(n-i)$. The authors noticed in [6] that this approach reduced the dynamic range of computation, thereby, saving power due to reduced word-lengths in the multiplication operation. Higher orders of differences may also be considered.

III. The DCMI Approach

Once again, consider the graph representation of the FIR filter of equation 1 in figure 1. We note that the order of computation shown in figure 1(a) is not the only possible order, i.e. c_{i+1} immediately followed by c_i, for $i = 0, 1, \ldots, M-1$ (Note that DCM only considers this particular order.). Consider figure 1(b) where arrows indicate the order in which successive coefficients are applied. It is quite possible that this order yields differential coefficients which are simpler to implement than the order shown in figure 1(a) (e.g. they may be powers-of-two), and hence, the implementation so obtained has lower complexity. Note that in this example, the ordering is given by $c_0, c_4, c_5, c_1, c_2, c_6, c_7, c_3$. The corresponding data sample $x(n-i)$ migrates from the edge $E_{i,j}$ to $E_{i,k}$, such that if T' is the new tour, $E_{i,k} \in T'$, $k \neq j$. This is shown in figure 1 which shows that $x(n-i)$

Let $\mathcal{K} = \{k_0, k_1, \ldots, k_{M-1}\}$ be the set representing the indices of coefficients in the new ordering. Hence, for the example in figure 1(b), $\mathcal{K} = \{0, 4, 5, 1, 2, 6, 7, 3\}$. Then, the new differential coefficients for the order sequence in $\mathcal{K}$ are given by $c_{k_{i+1}} - c_{k_i}$, $i = 0, 1, \ldots, M-1$ and we can calculate the partial products using

$$P_{k_i}^{(n)} = (c_{k_i} - c_{k_{i-1}})x(n-k_i) + P_{k_{i-1}}^{(n-k_i+k_{i-1})} \qquad (2)$$

for $i = 1, \ldots, M-1$ (the first partial product $P_{k_0}^{(n)}$ is computed directly as $c_{k_0}x(n-k_0)$) where $i = 0, 1, \ldots, M-1$. As an example, consider the computation of $P_4^{(n)} = (c_4 - c_0)x(n-4) + P_0^{(n-4)}$. Similarly, we can compute $P_4^{(n-1)} = (c_4 - c_0)x(n-5) + P_0^{(n-5)}$. Notice that in this approach we first need to calculate the partial products $P_{k_i}^{(j)}$, for all time instants $j = 0, 1, \ldots, n$ before calculating $P_{k_{i+1}}^{j}$. Hence, apparently this approach yields a block filter where the computations of output are performed "column-wise", rather than "row-wise" as in DCM. In the worst case, a maximum of n storage elements are required to store the intermediate partial products using a straight-forward implementation. How-

ever, by re-timing the filter, a transpose-direct form can be obtained for any coefficient ordering. Figure 2 shows the

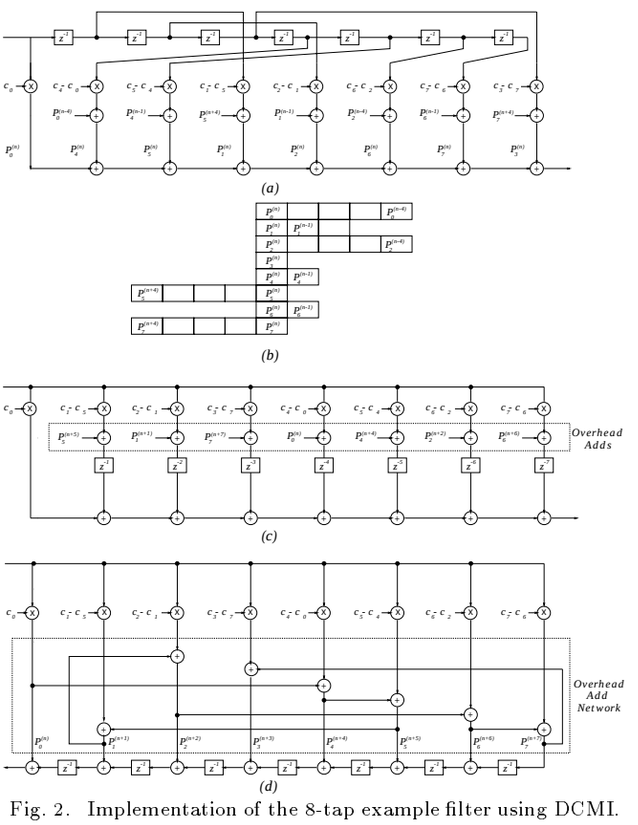

Fig. 2. Implementation of the 8-tap example filter using DCMI.

implementation details of an 8-tap filter using the DCMI approach assuming that it is asymmetric (in symmetric filter case, half of the filter "folds-over"). Figure 2(a) reveals the structure of the DCMI filter. Note that the implementation of this example filter requires reference to future values of partial products ($P_5^{(n+4)}$ and $P_7^{(n+4)}$). The partial products required to compute $P_i^{(n)}$'s for $i = 0, 1, \ldots, M-1$ are shown in figure 2(b). Clearly, at any time instant n, only M such products are required.

We can simplify the implementation shown in figure 2(a) by re-timing the filter. The first step is to move the delay elements to the multiplier inputs. Consequently, the *ith* branch containing a multiplier has i delay elements on it after the first step is completed. Next, we can move the delay elements further down such that the the multiplier precedes the delay elements, and then, move them even further down such that the overhead add operations also precede these delay elements as shown in figure 2(c). Finally, the delay elements are moved out of these branches to get the implementation shown in figure 2(d). By a careful consideration of appropriate partial products at various branches in the figure, we obtain the adder network for the re-timed implementation shown in figure 2(d). Hence, the filter output is available with an extra delay equal to M-adders due to the overhead add network. However, the structure of figure 2(d) can be pipelined to eliminate this delay.

A. Computing Coefficients for DCMI

The DCMI approach computes the set $\mathcal{K} = \{ k_0, k_1, \ldots, k_{M-1}\}$, such that the coefficient sequence $c_{k_0}, c_{k_1}, \ldots, c_{k_{M-1}}$ yields the least number of resources required in the implementation. In order to compute $\mathcal{K}$, we represent this problem using a graph, $G = (V, E)$, in which the set V represents vertices $\{c_0, c_1, \ldots, c_{M-1}\}$ for an M-tap filter and E represents the edges, $E_{i,j}$, for $i, j = 0, 1, \ldots, M-1$. Figure 3(a) shows the graph for a 4-tap ($M = 4$) filter. The edge $E_{k_j,k_{j+1}}$ connects vertex c_{k_j} to $c_{k_{j+1}}$ and represents the number of adders required to represent the difference $c_{k_{j+1}} - c_{k_j}$ in a given number representation scheme. Hence, the values assigned to the edges take into consideration the scheme used for number representation. As an example, if SM number representation is used, $c(k_i) = 17$, and, $c(k_{i+1}) = 33$, then $E_{k_j,k_{j+1}}$ is assigned a value of 1 because $c_{k_{j+1}} - c_{k_j} = 33 - 17 = 16$ requires only one adder in implementation of the multiplier. Note that G is *undirected* and *complete* [9]. There are M elements in V and $M(M-1)/2$ elements in E. Hence, $|V| = M$ and $|E| = M(M-1)/2$ independent of the word-length or the number representation scheme used in the filter implementation.

The implementation which requires least number of resources (total number of adders) can be obtained by computing the *Hamiltonian path* [9] with smallest weight in G. A Hamiltonian path is defined as *a path which visits each vertex exactly once*. In our work, for simplicity, we will compute the Hamiltonian cycle instead of the Hamiltonian path. A Hamiltonian cycle is a *simple cycle* [9] in which each vertex in G is visited. For example, figure 1 shows two Hamiltonian paths in G. We can remove any link in a Hamiltonian cycle to obtain a Hamiltonian path. This offers us added convenience as we can select the first coefficient such that the first column computation ($P_0^{(j)}$, for $j = 0, 1, \ldots, n$) requires *only one* adder, rather than a full multiplier. This is always possible if one of the coefficients is always fixed to a known power-of-two value and the remaining coefficients are calibrated with respect to it. For example, if c_4 in the filter in figure 1 is fixed at 2^{15} in a 16-bit SM representation scheme, and the graph in figure 1(b) represents the minimum weight Hamiltonian cycle for this filter, then the DCMI implementation would use the sequence $\{c_4, c_5, c_1, c_2, c_6, c_7, c_3, c_0\}$, thereby, avoiding the use of a full multiplier for the first partial product column computation. Hence, Hamiltonian cycle computation is more advantageous.

B. Finding the Hamiltonian Cycle

The Hamiltonian cycle can be solved by employing one of the known methods of solving the *traveling salesman problem* (TSP) [9], [10]. In our work, we use two well-known approaches to obtain the Hamiltonian cycle for a given graph. The first approach uses a *greedy strategy* which starts at a given node and extends the cycle in a *depth-first search* (DFS) manner. Initially, all nodes are colored white and the *start* node is initialized to a given node. Next, it looks at the white colored neighboring nodes of the given *start* node and selects the one which can be reached using the smallest

weight edge (minimum resources). The selected node becomes the *start* node in the next step and is colored black. This process is repeated till all the nodes are colored black. Since the graph is complete, this method produces a tour by visiting each node exactly once. The complexity of this algorithm is $\Theta(|V| + |E|) = \Theta(M^2)$ [9]. This algorithm is repeated by initializing the *start* node to each vertex in V. Hence, the complexity of the greedy approach used in this work is $\Theta(M^3)$.

Another popular approach used for solving the TSP is the heuristic algorithm due to Lin and Kernighan [10]. The basic approach in this method is to complete a tour and then perform a local search to improve the tour. When an improvement is found, the algorithm does not necessarily use it immediately, but continues search hoping to find an even greater improvement. An interested reader is referred to [10] for a detailed description of the algorithm. In general, this approach is quite powerful and produces tours which are within 2% of the optimal tour [10] which is acceptable accuracy in the DCMI problem.

IV. Second Order DCMI

The approach presented in section III addresses first-order DCMI. Similar to the DCM [6], we can use higher order differential coefficients to define higher order DCMI. Let $\delta^1_{i-1,i}$ represent the first order coefficient difference, $c_i - c_{i-1}$, and $\delta^2_{i-2,i}$ represent the second order coefficient difference, $(c_i - c_{i-1}) - (c_{i-1} - c_{i-2}) = c_i - 2c_{i-1} + c_{i-2}$. Then, it can be shown [6] that $P_i^{(n)}$ can be calculated as

$$P_i^{(n)} = c_{i-1}x(n-i) + \delta^1_{i-2,i-1}x(n-i) + \delta^2_{i-2,i}x(n-i) \quad (3)$$

where $i = 2, 3, \ldots, M-1$. Hence, using two overhead storage and two addition operations per partial product, we can implement the second order DCM as explained in detail in [6]. It can be verified that the second-order DCMI can be obtained by computing

$$P^{(n)}_{k_i} = c_{k_{i-1}}x(n-k_i) + \delta^1_{k_{i-2},k_{i-1}}x(n-k_i) + \delta^2_{k_{i-2},k_i}x(n-k_i) \quad (4)$$

for $i = 2, 3, \ldots, M-1$, where k_i's give the ordering sequence for the second-order DCMI. Hence, the second order DCMI requires twice as much overhead of add operation similar to the second order DCM. However, similar to the first-order DCMI, by choosing c_{k_0} to be a power-of-two, we can eliminate the full multiplication in the computation of the first column of partial products.

A. Computing Second-Order DCMI Coefficients

The second-order DCMI problem cannot be solved using the graph representation presented in section III. This is because in the second order DCMI, a second-order differential coefficient, $\delta^2_{i-2,i}$, requires reference to three coefficients, c_i, c_{i-1} and c_{i-2}. Hence, if we were to use an edge to express the number of adders required to implement a multiplier with $\delta^2_{i,j}$ $(i \neq j)$ at one input, we would require counting the number of adders required to implement $\delta^2_{i,j}$ in

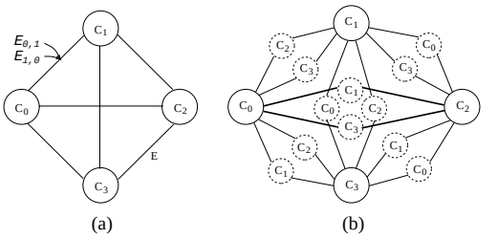

Fig. 3. Graph (G and $\tilde{G}$) Representations of an Example Filter with $M = 4$.

the given number representation scheme. For a given M-tap filter, the second order differential coefficients comprising c_i and c_j as the end points would be $c_j - 2c_k + c_i$, where $k = 0, 1, \ldots, M-1, k \neq i, k \neq j$ and, hence, it would require $M-2$ edges between coefficients c_i and c_j in the graph. Therefore, the graph representation of section III needs to be modified to account for all possible $(M-2)$ intermediate nodes between the given two nodes.

Figure 3 shows the modified graph for a 4-tap filter for second order DCMI problem. The vertices are represented by continuous circles. Each pair of coefficients has $M-2$ edges between them. This is shown using dashed circles which indicates the intermediate vertex corresponding to the edge. Note that the dashed circles do not represent vertices, rather, these illustrate the vertex considered to be the intermediate vertex in the particular edge. Hence, the modified graph, $\tilde{G} = (V, \tilde{E})$, for the second order DCMI can be obtained from G by inserting $M-2$ edges between each pair of edges. In the new graph, $|V| = M$ as in G, and, $|E| = M(M-1)(M-2)/2$.

Next, we need to formulate rules for traversing $\tilde{G}$. Let the edge between vertices c_i and c_j, with intermediate vertex c_k be represented as $E_{i,k,j} \in \tilde{E}$, $i, j, k = 0, 1, \ldots, M-1, i \neq j \neq k$. Now, if $E_{i,k,j}$ is traversed, this implies that we have selected the coefficient order c_i followed by c_k followed by c_j. Hence, c_i and c_k have already been visited and no subsequent edge may be visited which has c_i or c_k as an intermediate or terminal node. The only exception to this rule is when all vertices have already been visited and the tour is completed by one more step. In that case, the first node from which the tour computation was initially started must be visited as the terminal vertex.

Consider the bold path in figure 3, for example. This path shows a valid tour represented by the coefficient sequence c_0, c_1, c_2, c_3 and contains two edges $E_{0,1,2}$ and $E_{2,3,0}$. Then, after arriving at c_2, we cannot visit c_1 because it has already been visited through $E_{0,1,2}$. Further, c_0 can only be visited as the terminal node in order to complete the tour, but it cannot be used as an intermediate node. Hence, $E_{2,3,0}$ is the only edge which can be visited without violating $\tilde{G}$ traversal rules. Hence, for any $k \in 0, 1, \ldots, M-1$, if c_k has been visited, this implies that before the next edge is traversed, all edges in the graph with c_k as the intermediate vertex must be disallowed. Similarly all edges originating from the vertex c_k must also be disallowed. Next, it is possible to devise a greedy algorithm which would start at a given initial node *start* and constructs a tour which visits all the vertices in the graph based on the best selection at the given time. At

each step, the algorithm keeps track of three vertices, *start*, *middle* and *last*. This corresponds to the coefficient order $c_{start}, c_{middle}, c_{last}$. Initially, all nodes are colored white and a user selected node, *initial*, is taken as the *start* node. Next, a decision is taken at c_{start} and the best edge $E_{start,middle,last}$ is selected such that c_{middle} and c_{last} are white nodes. Next, c_{middle} is marked black and it becomes the next *start* node. Similarly, c_{last} becomes the new *middle* node and search for the next best c_{last} is performed such that the *start* and *middle* nodes are already known and c_{last} must be a white node other than *initial*. If no such node can be found, then the tour is completed by selecting $c_{last} = c_{initial}$. In terms of figure 3, if the tour shown in bold were to be the best tour, the edge sequence visited will be $E_{0,1,2}, E_{1,2,3}$ and $E_{2,3,0}$ which corresponds to the coefficient order c_0, c_1, c_2 and c_3. The algorithm is repeated by initializing the *start* node to each vertex in V.

B. The Modified LK-Algorithm

The algorithm presented in section IV-A uses a greedy strategy to find a good coefficient order for second order DCMI. However, we may use better tour computation schemes such as the LK-algorithm. The basic LK-algorithm must be modified such that it does not violate the graph traversal rules outlined in section IV-A. The main idea is to perform a local search around a given tour to find a better tour. This is done by forming a δ-path shown in figure 4. Starting at a node v, the algorithm tries improvement on both its neighboring edges. When a node w is located such that the cost of δ-path shown in figure 4(b) is smaller than the cost of the initial tour T, the δ-path is converted into another tour which updates the best tour found so far. In the algorithm shown below, $c(T)$ represents the cost of a tour T.

Figure 4(c) shows the improved tour obtained using the δ-path. We note that the modification required in this algorithm is to move the intermediate nodes p and r so that no rules are violated while improving the tour. We note that this is not the only possible approach. Better solutions may be obtained by considering the best edge $E_{u,q,w}$, where q may be any node in T other than u and w. The tour may be completed such that none of the graph traversal rules are violated on $\tilde{G}$. The approach presented in this paper is the simplest way to modify the LK-algorithm such that it can execute on $\tilde{G}$. More sophisticated and better approaches for tour update will be a subject of subsequent publication. The modified LK-algorithm is shown below.

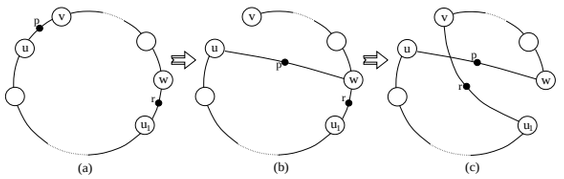

Fig. 4. Tour improvement in modified LK-algorithm.

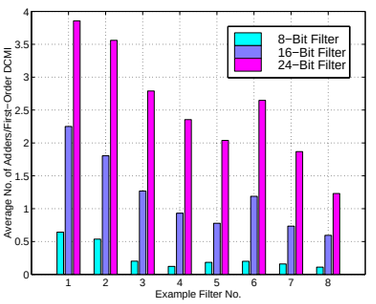

Fig. 5. Average Number of Adders per First-Order Differential Coefficient For Sign-Magnitude Number Representation.

1. Initially, compute the best tour, T, using the greedy approach. Let $B = T$.
2. (Edge Scan) For each vertex v in $\tilde{G}$ and for each vertex uv incident with v in turn (note that there is an intermediate node on these edges), perform steps 3-6
3. Let $u_0 = u$. Remove edge u_0v and find another edge u_0w_0, $w_0 \neq v$, such that it has smaller cost than that of the removed edge. This yields a δ-path. Set i=0. If no such w_0 can be found, break the loop and go to step 2 and try the next node/edge.
4. Construct a tour from the δ-path. Call it $T^{(i)}$. If $c(T^{(i)}) < c(B)$, let $B = T^{(i)}$.
5. (Build-nest δ-path) Let u_{i+1} be the other neighbor of w_i. If $u_{i+1}w_i$ was already added to a δ-path in this iteration, go to step 5. Otherwise, find a w_{i+1} such that $u_{i+1}w_{i+1}$ is not in T and the resulting δ path has a cost smaller than $c(T)$. If no such node exists, go to Step 6, otherwise, set i=i+1 and goto Step 4.
6. If a tour is found with cost less than $c(T)$, replace T with this tour. Return to Step 3 if untested node/edge remain.

V. Numerical Results

We now present some numerical results to demonstrate the power and potential of the proposed approaches. Both SM and SPT number representations for implementing differential coefficients are considered. Table I shows a relative comparison[1] of filter implementations obtained using the proposed DCMI approach when the coefficients are expressed with $N = 16$ bit SM and SPT representations[2]. f_s and f_p represents normalized passband and stopband frequencies, respectively and R_p and R_s represent the passband ripple and stopband attenuation, respectively. None of the solutions presented in this paper required more than a few minutes of CPU time on a Sun Ultra 30 workstation. A close observation of the table reveal some interesting results. The difference in the solutions using the greedy strategy and the LK-algorithm are negligible. Hence, near-optimal solutions are obtainable using the greedy strategy alone, as LK-algorithm is known to yield a tour which is very close to the optimal solution [10]. Further, second-order DCMI does not offer any significant advantage as compared to the first-order DCMI. In many cases, it provides a slightly worse

[1]The entries containing "—" could not be computed due to time constraints and will be provided in the final version of the paper.

[2]Note that by "[Symmetric]" in the specification of filter F in the tables, we mean that it is symmetric about $f = 0.5$.

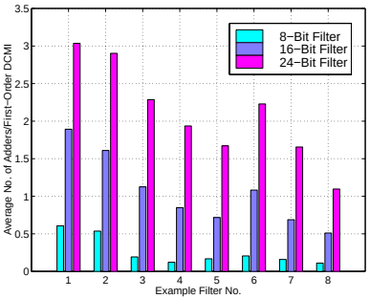

Fig. 6. Average Number of Adders per First-Order Differential Coefficient For SPT Number Representation.

solution. Hence, one may only consider higher-order DCMI if one were to investigate a hybrid solution which combines first and second-order solutions. We also note that the SPT representation offers significant advantage of SM representation in many cases. Finally, we note that the number of adders per coefficient required in DCMI implementation is less than 2, in general, for SPT representation. This compares favorably to the published results of multiplierless filters in literature.

Figures 5 and 6 show a relative comparison of the average number of adders per differential coefficient obtained using the first-order DCMI solutions for SM and SPT number representations, respectively. In the table, BW, EP, PM and LS abbreviate Butter-worth, elliptic, Parks-McClellan and least squares filters, respectively. We compare the number of adders per differential coefficient for 8, 16 and 24 bit coefficients. The example filters considered were 28-tap PM, 41-tap LS, 119-tap PM, 172-tap LS, 131-tap PM, 170-tap LS, 151-tap PM, 217-tap LS, respectively, with specifications shown in table I. These results were obtained using the greedy strategy for first-order DCMI. We note that SPT implementations require less adders than SM implementations for all word-lengths. We also observe a linear relationship between the average number of adders per differential coefficient with the word-length. This relationship is exhibited in all the cases considered. Further, the average number of adders per differential coefficient reduces, in general, as the length of the filter increases. We note that traditional approaches of finding multiplierless implementations for word-lengths > 16 would take enormous computational effort and may not yield good solutions. In contrast, our technique takes polynomial time, independent of the word-length and the number representation scheme, and can be used to obtain good DCMI solutions for large filters within a few minutes of CPU time.

VI. Conclusion

We presented a novel approach which can be used to obtain design and synthesize multiplierless implementations of FIR digital filters. The basic idea presented in this technique is to remove computational redundancy by reordering filter coefficients such that an implementation based on differential coefficients requires only a few adders. This approach does not compromise the frequency response characteristics of the given filter. We presented a graph representation model for the first and second order DCMI approaches in which vertices represent coefficients and edges represent the corresponding resources required in computation. The optimal solution to the coefficient reordering problem is the *Hamiltonian path* in the problem graph. We used two approaches to find the smallest weight *Hamiltonian cycle*; a greedy approach, and, the heuristic algorithm proposed by Lin and Kernighan. The greedy approach was shown to yield solutions that are very close to the ones obtained using the LK-algorithm. The power and potential of this approach was demonstrated by presenting results for filters with *length* up to > 300. In general, for implementations using 16-bit SPT number representation and first order differential coefficients, the total number of adders required per coefficient is less than 2.

References

[1] 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.

[2] D. Li. and Y. C. Lim, "Multiplierless realization of adaptive filters by nonuniform quantization of input signal," *1994 IEEE International Symposium on Circuits and Systems*, Vol. 2, pp. 457-459, 1994.

[3] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits and Systems*, Vol. 36, No. 7, pp. 1044-1047, July 1989.

[4] M. Yagyu, A. Nishihara and N. Fujii, "Fast FIR Digital Filter Structures Using Minimal Number of Adders and its Application to Filter Design," *IEICE Trans. Fundamentals*, Vol. E79-A, No. 8, pp. 1120–1128, Aug. 1996.

[5] D. A. Parker and K. K. Parhi, "Low-area/power parallel FIR digital filter implementations," *Journal of VLSI Signal Processing*, Vol. 17, No. 1, Sept. 1997.

[6] N. Sankarayya, K. Roy, and D. Bhattacharya, "Algorithms for low power and high speed FIR filter realization using differential coefficients," *IEEE Trans. Circuits and Systems*, Vol. 44, No. 6, pp. 488-497, Jun. 1997.

[7] K. Muhammad and K. Roy, "Low Power Digital Filters Based On Constrained Least Squares Solution," *In Proc. 31st Asilomar Conference On Signals, Systems, & Computers*, Nov. 2–5, 1997.

[8] J. M. Rabaey, "Digital Integrated Circuits: A Design Perspective," Prentice Hall, New Jersey, 1996.

[9] T. H. Cormen, C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms," The MIT Press, 1990.

[10] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank and A. Schrijver, "Combinatorial Optimization," John Wiley & Sons, Inc., 1998.

Type	M	Total Adds	DCMI-1 Greedy	DCMI-1 LK	DCMI-2 Greedy	DCMI-2 LK
A: Low-pass ($f_p = 0.25, f_s = 0.3, R_p = 3\ dB, R_s = -50\ dB$)						
BW	20	94 (76)	45 (35)	44 (35)	38 (33)	38 (33)
EP	6	40 (28)	19 (17)	19 (17)	13 (11)	13 (11)
PM	28	140 (118)	63 (53)	62 (53)	62 (51)	57 (49)
LS	41	206 (178)	74 (66)	72 (66)	74 (66)	70 (61)
B: Low-pass ($f_p = 0.27, f_s = 0.2875, R_p = 2\ dB, R_s = -50\ dB$)						
BW	71	220 (158)	76 (68)	75 (67)	78 (64)	70 (62)
Elliptic	8	52 (40)	25 (22)	25 (22)	18 (15)	18 (15)
PM	119	578 (500)	151 (134)	145 (130)	154 (140)	153 (138)
LS	172	734 (606)	160 (146)	156 (142)	177 (165)	175 (165)
C: Low-pass ($f_p = 0.27, f_s = 0.29, R_p = 2\ dB, R_s = -100\ dB$)						
PM	189	850 (694)	183 (176)	179 (168)	198 (185)	198 (181)
LS	326	1054 (874)	202 (190)	200 (179)	228 (215)	227 (209)
D: Low-pass ($f_p = 0.25, f_s = 0.2625, R_p = 2\ dB, R_s = -73\ dB$						
PM	165	774 (598)	173 (157)	170 (155)	185 (164)	178 (159)
LS	236	912 (752)	187 (172)	185 (170)	216 (194)	212 (193)
E: Notch ($f_{p1} = 0.3, f_{s1} = 0.32, f_{s2} = 0.68, f_{p2} = .7$)						
PM	131	390 (280)	102 (74)	101 (72)	113 (96)	111 (91)
LS	170	880 (740)	202 (184)	196 (182)	217 (195)	215 (193)
F: Notch [Symmetric] ($f_{p1} = 0.2, f_{s1} = 0.22, f_{s2} = 0.38, f_{p2} = .4$)						
PM	151	428 (356)	111 (104)	110 (103)	127 (113)	126 (111)
LS	217	494 (392)	129 (111)	125 (109)	145 (126)	143 (123)

TABLE I

Total Number of Adders in Multipliers Obtained Using DCMI for minimally scaled $16-bit$ SM and SPT Number Representations, Respectively for Various Example Filters.