A Fast and Efficient Algorithm for the Multiplierless Realization of Linear DSP Transforms

Arda Yurdakul and Günhan Dündar

Abstract

In this paper, a fast algorithm having a pseudopolynomial run-time and memory requirement in the worst-case is developed to generate multiplierless architectures at all wordlengths for constant multiplications in linear DSP transforms. It is also re-emphasized that indefinitely reducing operators for multiplierless architectures is not sufficient to reduce the final chip area. For a major reduction, techniques like resource folding must be used. Simple techniques for improving the results are also presented.

Information About Authors:

Assist. Prof. Dr. Arda Yurdakul
Computer Engineering Department
Boğaziçi University
Bebek 80815, ISTANBUL, TURKEY
e-mail: a.yurdakul@ieee.org

Assoc. Prof. Dr. Günhan Dündar
Electrical and Electronics Engineering Department
Boğaziçi University
Bebek 80815, ISTANBUL, TURKEY
e-mail: dundar@boun.edu.tr

Contact Information for the Corresponding Author

Arda Yurdakul
Computer Engineering Department
Boğaziçi University
Bebek 80815, ISTANBUL, TURKEY
e-mail: a.yurdakul@ieee.org

I. Introduction

There has been an intensive study on the development of algorithms for the multiplierless realization of constant multiplications in DSP algorithms in recent years for smaller hardware area [1]-[9]. This is done by generating trees composed of simple operators like adders and subtractors. It has been proved that forming such a tree in place of multipliers is an NP-complete problem [9].

In literature, most of the algorithms make some assumptions to minimize the number of adders. Among these studies, the RAG-n algorithm [3] is the most notable one, because it can theoretically produce optimal results when all odd coefficients in the system have a single coefficient cost of 1 or more. To achive this, RAG-n algorithm uses two memory-inefficient look-up tables that are generated by the MAG algorithm [4] in exponential time. The main drawback of the look-up-table-based systems is the limited physical memory, which cannot provide infinite storage. RAG-n algorithm also suffers from this well-known fact: The look-up tables can represent the coefficients only up to 12-bit precision. For larger multipliers, fast heuristics are present in RAG-n. As a result, for large integer multiplications, the adder tree produced by RAG-n can employ more adders than necessary due to the limited table-length. All the remaining algorithms use heuristic methods for all wordlengths by using as little memory as possible at a fairly good speed at a cost of near-optimal trees. This is usually sufficient, because it has been proved that minimizing number of adders in multiplications indefinitely is unnecessary because in this case, area and power due to storage elements and adders at the accumulator units begin to dominate [3], [8]. Therefore, other techniques like register minimization, resource folding must be incorporated for minimum area.

The proposed algorithm is developed for the faster generation of synchronous, fully pipelined and multiplierless architectures for realizing multiplications of an input with a set of constants: FIR filters, multirate FIR filters in the same fold, constant multiplications with switching inputs, linear transforms like DCT with a serialized input. The essence of the algorithm lies in blindly realizing all constant multiplications without taking timing into consideration. After this, a careful scheduling must be done to minimize area due to registers and adders at the accumulator units. This paper concentrates merely on multiplierless realization of constant multiplications.

The run-time of the algorithm presented in this paper is rather fast even on a 486

machine. It synthesizes ordinary FIR filters of moderate length (say 100 taps) in less than a second. It synthesizes a 3000-tap chirp filter in less than three minutes on a 386MHz Intel Celeron machine when quantization of coefficients is done using 24 bits [10].

The algorithm explained in this paper uses pattern search as most of the previous algorithms do. If *pattern length* is a term defined as the number of nonzero terms in a pattern, the common trend in previous algorithms is to make the pattern search starting from a pattern length equal to the coefficient wordlength and ending at a pattern length of 2. Our algorithm makes the pattern search at a pattern length of 2 throughout the process, similar to that of [6] which is used only for multiplierless realization of FIR filters. The worst-case run-time and memory requirement of this algorithm is $O(|S|\, m)$ and $O(|S|\, m^2)$, respectively, where m is a number determined by counting the nonzero entries in the Canonic-Signed-Digit (CSD) representation of all constants separately and picking the maximum count, and $|S|$ is the cardinality of the kernel set, S, which is used to generate all coefficients in the system by shifting and/or negating the elements in S.

In the next section, the proposed algorithm is explained with the evaluation of the worst-case run-time and memory requirements of the algorithm. Following it, the refinements that can improve the results found by the basic algorithm, are explained. Section IV is used to demonstrate the performance of the algorithm on a set of experiments. The final section concludes the work .

II. The Algorithm

The new algorithm is based on iteratively combining two nonzero terms in order to form an adder tree which can be used to generate all coefficients in a DSP system. This algorithm runs safely on all systems using any form of binary representation, i.e. two's complement, signed digit (SD) or canonic signed digit (CSD). In this paper canonic-signed-digit (CSD) representation is emphasized, because CSD is proved to be the unique representation with minimum nonzero entries [11]. In CSD and SD representations, a nonzero term means 1 and -1 while it means only 1 in two's complement representation. In our algorithm, the definition of a nonzero term is extended to include a *two-term* or a negative form of the two-term as well. A two-term is defined as the combination of two nonzero terms such that one nonzero term appears at the zeroth bit position and the other nonzero term appears at the c'th bit position. The properties of a two-term can be listed

as follows:

- A two-term has a length of c.
- A two-term has a sign, ς, calculated as

$$\varsigma = \text{sgn}\left(\varsigma_0 \varsigma_c\right), \tag{1}$$

where sgn (.) is the sign function, ς_0 and ς_c are the signs of the nonzero terms at zeroth and c'th bit positions, respectively.

- A two-term has an odd value, v, and calculated as

$$v = v_0 + \varsigma . v_c 2^c, \tag{2}$$

where v_0 and v_c are the values of the nonzero terms at zeroth and c'th bit positions.

- A two-term has an order, i, which is calculated as

$$i = \max\left(i_0, i_c\right) + 1, \tag{3}$$

where i_0 and i_c are the orders of the nonzero terms at zeroth and c'th bit positions. (By definition, the nonzero entries 1 and -1 are of zeroth order.)

Example 1: The two-term $t_1 = 10001$, is of length 4, sign 1, value 17, and order 1.

- The replica, r, of a two-term is the appearance of a two-term in coefficients.

Example 2: In coefficient 1010101, the replicas of two-term t_1 of Example 1 are $R_{t_1} = \{r_{1_1}, r_{1_2}\} = \{t_1, t_1 << 2\}$.

Using the properties of the two-term, an algorithm can be developed to iteratively form an adder tree for the multiplierless realization of a set of coefficients, N. At each iteration, the two-terms are combined to form new two-terms of higher order. The algorithm must realize all coefficients in N which can be reduced to a kernel set S such that for each $n \in N$, there exists exactly one $s \in S$ such that n is the shifted and/or negated version of s. So, it is sufficient for the tree-generation part of the algorithm to run over the set of S rather than N. The complete algorithm is presented in Fig. 1. The operation of the algorithm is explained with the help of below comments by using the numbers shown on the left of each line of Fig. 1 as the individual steps:

Step 1: Constants in N are scaled so that they are odd and positive. These odd numbers form the kernel set S. Note that 1 is not an element of S. Note also that all members of N can be represented using S

Step 2: The solution set, T_F, that will contain the two-terms to realize S, is initialized as an empty set.

Step 3: To find at least one two-term in S, there must be at least two nonzero entries in any $s \in S$. Therefore the condition in "while" loop satisfies the existence of at least one two-term in the loop, i.e. $T \neq \emptyset$. Each iteration determines the order of the two-terms in sets T and T_F.

Step 4: To form set T, all possible pairings of all nonzero terms within each coefficient in S must be examined. The unique one form the set T. containing all available two-terms at that iteration level.

Step 6-7: For each two-term t in T, a replica set R_t is obtained. Each set contains all replicas that can be formed by negating and/or shifting the two-term t.

Step 8: At each iteration within the inner loop ("loop" of Fig. 1), the most desirable two-term at the current iteration is picked. This loop continues until all replica sets are emptied by the innermost loop ("for" of Fig. 1). A two-term $t^* \in T$ if $|R_{t^*}| = r_{\max}$. If there exists more than one t^* satisfying this condition then the one with $\varsigma = -1$ is chosen because hardware realization of a subtraction operation at lower orders (defined by equation 3) is much cheaper than a subtraction operation at the higher orders. If there is no or more than one t^* satisfying this condition, then t^* with minimum length is chosen. These conditions help minimization of the final chip area.

Step 9-12: At each iteration within the innermost algorithm ("for" of Fig. 1) the most desirable replicas of t^* w'll be selected. Replica r from R_{t^*} is selected in a way that maximizes the number of selected replicas from R_{t^*} such that each nonzero term in all coefficients in S is represented by at most by one replica. Since each replica r in R_{t^*} has two nonzero entries, each of these nonzero entries may appear in other replicas of the same two-term (i.e. R_{t^*}) or in the replicas of the other two-terms (i.e. $R_t, \forall t \neq t^*$). So, by selecting r, all other replicas that are implied by r are inherently handled. Therefore they must be removed from related sets, i.e. $R_t, \exists t \in T$. The innermost loop continues until the R_{t^*} is emptied.

Step 13: Include t^* to the solution set T_F. Note that the cardinality of set T_F (i.e. $|T_F|$) will determine the number of additions and/or subtractions to form all numbers in S after the "while" loop is exit.

Step 14: Return T_F and S that will be used to form N. The number of adders to realize the coefficients in N will be given by $|T_F|$.

Example 3: Assume that the set of coefficients for multiplierless realization is $N = \{n_1, n_2, n_3, n_4\} = \{100101, 10101010, 1010101, 1000\}$. Coefficient set N is the input to the algorithm. The operation of the algorithm on this coefficient set is shown in TABLE I. In this figure, the I column stands for the "while" loop, the J column stands for the "loop" loop, the K column stands for the "for" loop in TABLE I. The numbers shown in R_t column needs additional explanation. For example, $R_3 = \{0, 2\}_{s_1}$ means that t_3 has two replicas in number s_1: The two-term t_3 as it is and the two-term t_3 shifted left two times. Note that the outputs of the algorithm are S and T_F and the whole system will be realized by three adders and five shifts.

The worst-case run-time and memory requirement of the algorithm base on two variables: m is a number determined by counting the nonzero entries in the CSD representation of all constants separately and picking the maximum one and $|S|$ is the cardinality of the kernel set S.

Proposition 1: The new algorithm has a worst-case run-time of $O(|S|\, m)$.

Proof: Let's assume that the outmost loop (i.e., the "while" loop in Fig. 1) iterates I times. The following derivation is done for the i'th iteration step of the "while" loop. Each iteration step determines the order of two terms selected during that iteration:

The process of forming two-terms and their related replica sets for each $s \in S$ takes $m_{s,i}$ time where $m_{s,i}$ is the number of nonzero digits in number s at iteration step i. Since this process can be done in parallel for all $s \in S$, it takes m_i time for all $s \in S$ where $m_i = \max_{s \in S} m_{s,i}$. This process produces $|R_{s,i}|$ number of replicas for each $s \in S$:

$$|R_{s,i}| = \left(\begin{array}{c} m_{s,i} \\ 2 \end{array} \right) = \frac{m_{s,i}\,(m_{s,i} - 1)}{2}. \tag{4}$$

Note that

$$\cup_{t \in T} R_{t,i} = \cup_{s \in S} R_{s,i} = R_i \tag{5}$$

Here, $R_{s,i}$ is the replica set of any $s \in S$ and R_i is the universal set for the replicas at the iteration level i.

After this process, the inner loop (i.e. "loop" in Fig. 1) is taken. In this loop, each $r \in R_i$ is handled such that either the replica is selected or it is removed from the replica

set. Since initial steps of "loop" are negligible for the calculation of the iteration time of the "loop", then it can be claimed that the "loop" and the "for" loop in Fig. 1 can be handled as a single loop that iterates J times. Since the algorithm shown in Fig. 1 runs for each $R_{t,i}$, then $J = \sum_{t \in T} J_t$. To calculate J, it can be assumed that the algorithm runs for each $R_{s,i}$ due to the Equation 5. Then

$$J = \sum_{s \in S} J_s. \tag{6}$$

J_s can be derived as follows: At the first iteration of "loop", a replica r is selected such that it realizes an $s \in S$ (i.e. $j = 1$). This process removes $(m_{s,i} - 2)$ replicas from $R_{s,i}$ due to each nonzero digit in r. Since r will be replaced by its related two-term t, it will also be removed from $R_{s,i}$. Then the total number of replicas removed from $R_{s,i}$ is $2\,(m_{s,i} - 2) + 1 = 2m_{s,i} - 3$. This operation also reduces the number of nonzero digits in s by 2. By putting two-term t^* in place of the erased replica, the number of nonzero digits in s is increased by 1. However, this new nonzero digit in s does not appear in any of the replica sets at the current iteration of the "while" loop. Therefore its effect to J_s is null. The inner loop iterates until $R_i = \emptyset$. This can be formulated as follows by using Equation 4:

$$\frac{m_{s,i}\,(m_{s,i} - 1)}{2} - \sum_{j=1}^{J_s} (2m_{s,i,j} - 3) = 0. \tag{7}$$

where $m_{s,i,j} = m_{s,i} - 2(j-1)$. With this fact, the above equation reduces to the following quadratic equation:

$$2J_s^2 - (2m_{s,i} - 1)\,J_s + \frac{m_{s,i}\,(m_{s,i} - 1)}{2} = 0 \tag{8}$$

When the above equation is solved for J_s in terms of $m_{s,i}$, it is obtained that

$$J_s = \left\lfloor \frac{m_{s,i}}{2} \right\rfloor. \tag{9}$$

So, the i'th run-time of the "while" loop can be determined by combining the time for finding of two-terms and forming related replica sets with J which is given by Equation 6:

$$m_i + J = m_i + \sum_{s \in S} \left\lfloor \frac{m_{s,i}}{2} \right\rfloor \tag{10}$$

Since the "while" loop iterates I times, the run-time of the complete algorithm, A, is given as

$$A = \sum_{i=1}^{I} m_i + \sum_{i=1}^{I} \sum_{s \in S} \left\lfloor \frac{m_{s,i}}{2} \right\rfloor \tag{11}$$

At each iteration of the while loop, the number of nonzero digits in all $s \in S$ halves. For example if maximum number of nonzero entries in $s \in S$ is m_s before the first iteration ends, it is $\lceil \frac{m_s}{2} \rceil$ at the end of first iteration. This phenomenon is shown in Fig. 2 for $m_s = 7$. Then at the end of i'th iteration, the number of nonzero digits in $s \in S$ is:

$$\begin{aligned} m_{s,i} &= \left\lceil \frac{m_s}{2^i} \right\rceil, \\ m_i &= \left\lceil \frac{m}{2^i} \right\rceil, \end{aligned} \tag{12}$$

where $m = \max_{s \in S} m_s$. Combining the above equation with Equation 11, the following expression is obtained for the calculation of the run-time:

$$A = \sum_{i=1}^{I} \left\lceil \frac{m}{2^i} \right\rceil + \sum_{i=1}^{I} \sum_{s \in S} \left\| \frac{m_s}{2^{i+1}} \right\| \tag{13}$$

where $\|x\|$ is the integer function which determines the integer part of x. Before working more on the above equation, the number of iterations for the "while" loop, I, must be determined: The "while" loop stops when there is only one nonzero digit in all $s \in S$. Then using Equation 12, I can be found in terms of m.

$$I = \lceil \log_2 m \rceil \tag{14}$$

Note that I is the maximum order of the two-term and the depth of the generated adder tree. Combination of equations 13 and 14 determines A, i.e. the run-time of the whole algorithm. Since run-time of an algorithm is given by the order of growth, i.e. $O(A)$, all ceiling, floor and integer functions can be dropped:

$$\begin{aligned} O(A) &= \textstyle\sum_{i=1}^{\log_2 m} \frac{m}{2^i} + \sum_{s \in S} \left(\sum_{i=1}^{\log_2 m} \frac{m_s}{2^{i+1}} \right) \\ &= \left(1 - \frac{1}{2m}\right) \left(2m + \textstyle\sum_{s \in S} m_s\right) \end{aligned} \tag{15}$$

In the above equation, $\sum_{s \in S} m_s$ determines the $O(A)$. The worst case happens when all s in S have the same number of nonzero entries. However, usually they don't have the same number of nonzero entries. And in the best case, only one s in S is dominant, i.e. all remaining numbers in S has negligible number of nonzero digits. Then it can be easily claimed that

$$O(m) \leq O(A) \leq O(|S|\, m). \tag{16}$$

Proposition 2: The worst-case memory requirement of the new algorithm is $O(|S|\, m^2)$.

Proof: The maximum memory requirement occurs at the first iteration of the "while" loop. This is due to the storage of replica sets. So, the memory requirement of the

algorithm, M can be written using Equation 4 as follows:

$$M = \sum_{s \in S} |R_s| = \sum_{s \in S} \frac{m_s \left(m_s - 1\right)}{2} = O\left(\sum_{s \in S} m_s^2\right) \tag{17}$$

For the worst-case, assume that all numbers in S have the same number of nonzero entries, i.e. m. Then M will be determined using Equation 17 as follows:

$$M = O\left(\sum_{s \in S} m_s^2\right) = O\left(|S|\, m^2\right). \tag{18}$$

III. Refinements

The effective adder count is different than the adders/subtractors count: The cost of a subtraction is bigger than that of an adder and the cost of a negator is smaller than that of an adder. Therefore effective chip area will be less if the following refinements are done:

1. Negative additions can be replaced by regular adders and inverting the sign of the two-term in subsequent two-terms as shown in Fig. 3.

2. If a two-term is a subtraction and it appears as a subtrahend more than it does as a minuend in subsequent two-terms, then the inputs are switched and the sign of the two-term in subsequent two-terms is inverted. This is shown in Fig. 4. This refinement process produces better results if the conversion starts from the highest order two-terms.

3. Subtractions can be replaced by putting a negator at the output of the operator which is used as the subtrahend and converting the subtractor to an adder. This really produces very efficient results if an operator, whose output is used as an subtrahend n times, satisfies following equation:

$$a_{negator} + n \times a_{adder} < n \times a_{subtractor} \tag{19}$$

Here, a_{adder}, $a_{subtractor}$ and $a_{negator}$, stand for area costs of an adder, a subtractor and a negator respectively. This totally depends on the technology used and the architectures of the selected modules to realize the operations. An example of this refinement is shown in Fig. 5.

IV. Experiments

The new algorithm is applied on a number of examples in the literature. In all examples CSD representation is used. The results are presented between TABLE II-TABLE VII. In the first four tables, the term *org* stands for original number of operators (adders, subtractors or negative adders to realize constant multiplications) and shifters, and the

term *New Alg.* stands for the results found by the new algorithm proposed in this paper. The numbers separated from these labels by underscores stand for wordlengths. For example B4L0_8 is the 8-bit realization of B4L0.

In TABLE II, two multirate systems at different wordlengths are tested. Here, the terms B4L0 and B2L3 stand for the four-band block transform whose coefficients are given in [12] and the two-band three-level wavelet transform whose coefficients are given in [13], respectively. In B4L0, all four filters are folded in a single fold. In B2L3, two filters in the first level are folded in a fold and remaining four filters are folded in another fold, hence B2L3 is realized by using two folds so that single clock can be used throughout the system. Therefore the results of the algorithms for B2L3 must be divided by two to find the operator and shifter values in a single fold. As it is observed here, the new algorithm produces better results than [9], because the integer programming model proposed in that paper is equivalent to the single run of the new algorithm. Note that there is a great improvement in terms of operators and shifters with respect to original values.

In TABLE III, POT stands for the first FIR example in [7] and DCT is the one-dimensional 8-point discrete cosine transform. In DCT, all filters are folded in a single fold. Here, again, note that the new algorithm outperforms its precedents in most examples.

TABLE IV and TABLE V are used to compare the performance of recent algorithms at several FIR filters. S1 and S2 are taken from [1] and L1, L2 and L3 are taken from [14]. Note that in all cases the new algorithm and [8] produce equal results. However, there is a strong possibility that our algorithm runs faster than [7] and [8] because they make pattern search starting from N-bits (usually N is the wordlength) down to 2 nonzero terms, but our algorithm always makes the pattern search by using only 2 nonzero terms. The algorithm of [3] can always produce better results because it uses exhaustive search in non-unique signed digit representation of the constants at a cost of huge memory and exponential run-time requirements.

Our tool synthesizes the filters in these experiments in less than a second even on a 33MHz 486 PC. To demonstrate the performance of our tool, a 3000-tap chirp filter of [10] is synthesized at several wordlengths. The tool is recompiled for a 368MHz Intel Celeron PC and the benchmarks of TABLE VI are obtained. It should be noted that the first benchmark also holds on the above-mentioned 486 machine.

TABLE VII is used to demonstrate the effect of refinements. It is not surprising that

the total number of operators before and after refinements 1 and 2 are equal. However the number of adders after refinements 1 and 2 are higher than that is before refinements. This yields the conclusion that the first two refinements really produce a reduction in area cost. This fact usually applies to power reduction also. The last supercolumn of this table (i.e., 'After All Refinements') is prepared according to the fact that the Equation 19 holds for all $n \geq 2$.

V. Conclusion

In this paper, an algorithm which uses two-terms for producing efficient multiplierless architectures for constant multiplications is developed. It is fast, because the pattern search is realized by using only two nonzero terms throughout the whole procedure and redefining the two-terms at the end of each iteration. This algorithm can be considered as the extension of the idea proposed in [9].

This algorithm can be used for constant multiplications in all linear transforms including multirate transforms if the folding, scheduling techniques are carefully used.

The worst-case run-time and memory requirement of the algorithm are pseudopolynomial. This is an important feature of this algorithm because it produces results comparable to previously developed algorithms much more faster than the previously developed algorithms by using less memory.

References

[1] SAMUELI, H.: 'An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients,' *IEEE Transactions on CAS*, 1989, **36** (7) pp. 1044-1047.

[2] OH, W. J. and LEE, Y. H.: 'Implementation of programmable multiplierless FIR filters with powers-of-two coefficients,' *IEEE Transactions on CAS II*, 1995, **42** (8) pp. 553-555.

[3] DEMPSTER, A. G. and MACLEOD, M. D.: 'Use of minimum-adder multiplier blocks in FIR digital filters,' *IEEE Transactions on CAS II*, 1995, **42** (9) pp. 569-577.

[4] DEMPSTER, A. G. and MACLEOD, M. D.: 'Constant integer multiplication using minimum adders,' *IEE Proceedings-Circuits, Devices and Systems*, 1994, **141** (5) pp. 407-413.

[5] MEHENDALE M., SHERLEKAR, S. D. and VENKATESH, G.: 'Synthesis of multiplierless FIR filters with minimum number of additions,' Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design, 1995, Los Alamitos, CA: IEEE Computer Society Press, pp. 668-671.

[6] HARTLEY, R. I.:'Subexpression sharing in filters using canonic signed digit multipliers,' *IEEE Transactions on CAS II*, 1996, **43** (10) pp. 677-688.

[7] POTKONJAK, M., SRIVASTAVA, M. B. and CHANDRAKASAN, A. P.: 'Multiple constant multiplications: Efficient and versatile framework and algorithms exploring common subexpression elimination,' *IEEE Transactions on CAD*, 1996, **15** (2) pp. 151-165.

[8] PASKO, R., SCHAUMONT, P., DERUDDER, V., VERNALDE, S. and DURACKOVA D.:'A new algorithm for elimination of common subexpression,' *IEEE Transactions on CAD*, 1999, **18** (1) pp. 58-68.

[9] YURDAKUL, A. and DÜNDAR, G.: 'Multiplierless realization of linear DSP transforms using common two-term expressions,' *Journal of VLSI Signal Processing*, 1999, **22** (3) pp. 163-172.

[10] COLEMAN, J. O.: 'Cascaded Coefficient Number Systems Lead to FIR Filters of Striking Computational Efficiency,' *Proceedings of The 2001 Int'l IEEE Conf. on Electronics, Circuits, and Systems*, Malta, September 2-5, 2001.

[11] GARNER, H. L.: 'Number Systems and Arithmetic,' *Advanced Computers*, 1965, **6** pp. 131-194.

[12] ALKIN, O. and ÇAĞLAR, H.: 'Design of efficient M-band coders with linear-phase and perfect reconstruction properties,' *IEEE Transactions on Acoustics, Speech, Signal Processing*, 1995, **43** (6) pp. 1579-1590.

[13] DAUBECHIES, I.: 'The wavelet transform, time-frequency localization and signal analysis,' *IEEE Transactions on Information Theory*, 1990, **36** (7) pp. 961-1005.

[14] LIM, Y. C. and PARKER, S. R.: 'Discrete coefficient FIR digital filter design based upon an LMS criteria,' *IEEE Transactions on CAS*, 1983, **30** (10), pp. 723-739.

BEGIN$(N)\{$

1. Using N, form the kernel set S;
2. $T_F = \emptyset$;
3. **while** ($\exists s \in S$ such that more than one nonzero entry is required to represent s) {
4. Form T;
5. **for each** $t \in T$, form R_t;
   loop {
6. $r_{\max} = \max_{t \in T} |R_t|$;
7. **if** ($r_{\max} = 0$) **break;**
8. pick t^* such that $|R_{t^*}| = r_{\max}$;
9. **for each** $r \in R_{t^*}$ {
10. Remove implicants of r from $R_t, \forall t \in T$;
11. Find $s \in S$ such that r exists in s, erase r from s, put t^* in place of r in s;
12. $R_t = R_t \setminus \{r\}$;
    }
13. $T_F = T_F \cup \{t^*\}$;
    }
    }
14. **return** the modified kernel set S and solution set T_F;

}

Fig. 1. The algorithm

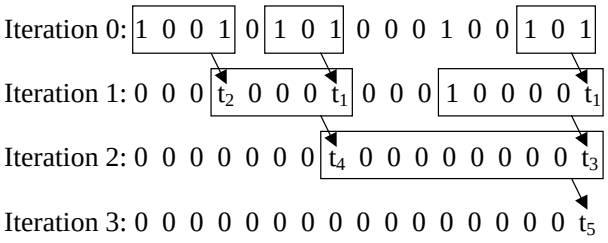
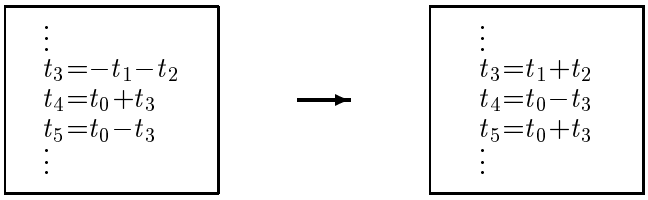


Fig. 2. Step by step demonstration of the constant multiplication realization using two-terms. Note that there are seven nonzero entries in the CSD realization of the number and there are three iterations to realize the system.



(a) (b)

Fig. 3. Refinement 1: (a)Original form: 1 addition, 1 subtraction, 1 negative addition. Since negative addition is an addition followed by a negation, this can be interpreted as 2 additions, 1 subtraction and a negation. (b)After the removal of negative additions: 2 additions, 1 subtraction. Gain = 1 negation. Each $t_.$ stands for a two-term or an input

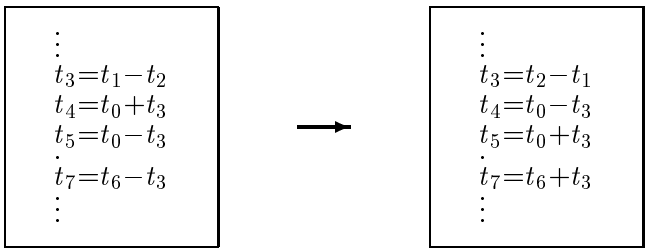
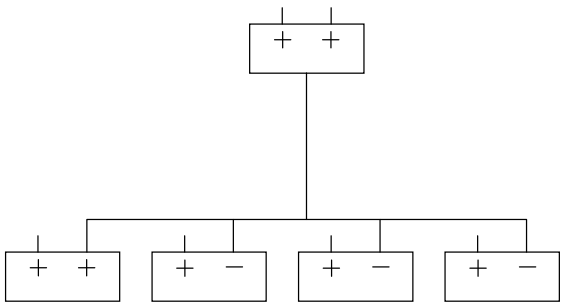
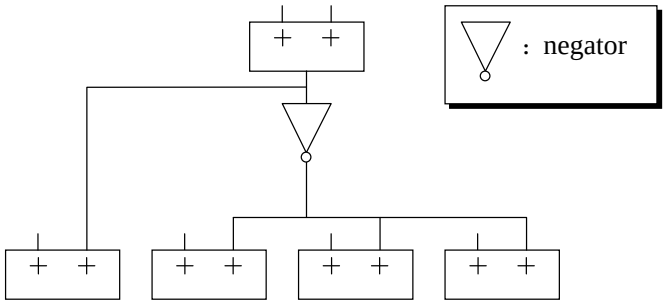


Fig. 4. Refinement 2: (a)Original form: 1 addition, 3 subtractions. (b)After switching the inputs: 2 additions, 2 subtractions. Each $t_.$ stands for a two-term or an input



(a)



(b)

Fig. 5. Refinement 3: (a)Original form, (b) after refinement 3.

I	J	K	S	T	R_t	$\lvert R_t\rvert$	$t*$	r	T_F	Comments
0	0	0	$s_1 : 1010101$ $s_2 : 100101$						$\emptyset$	
1	0	0		$t_1 : 101$	$R_1 = R_{1_{s_1}} \cup R_{1_{s_2}} = \{r_1, r_2, r_3\} \cup \{r_4\} = \{0, 2, 4\}_{s_1"} \cup \{0\}_{s_2}$	4			$\emptyset$	
				$t_1 : 1001$	$R_2 = R_{1_{s_2}} = \{r_5\} = \{2\}_{s_2}$	1				
				$t_1 : 10001$	$R_3 = R_{3_{s_1}} = \{r_6, r_7\} = \{0, 2\}_{s_1}$	2				
				$t_1 : 10001$	$R_4 = R_{4_{s_1}} = \{r_8\} = \{0\}_{s_1}$	1				
				$t_1 : 100001$	$R_5 = R_{5_{s_1}} = \{r_9\} = \{0\}_{s_1}$	1				
1	1	0					t_1		$\emptyset$	
1	1	1	$s_1 : 101000t_1$ $s_2 : 100101$		$R_1 = \{r_3\} \cup \{r_4\} = \{4\}_{s_1"} \cup \{0\}_{s_2}$	2		r_1	$\emptyset$	
					$R_2 = \{r_5\} = \{2\}_{s_2}$	1				
					$R_3 = R_4 = \emptyset$	0				
					$R_5 = \{r_9\} = \{0\}_{s_1}$	1				
1	1	2	$s_1 : 00t_1000t_1$ $s_2 : 100101$		$R_1 = \{r_4\} = \{0\}_{s_2}$	1		r_3	$\emptyset$	
					$R_2 = \{r_5\} = \{2\}_{s_2}$	1				
					$R_3 = R_4 = \emptyset$	0				
					$R_5 = \{r_9\} = \{0\}_{s_1}$	1				
1	1	3	$s_1 : 00t_1000t_1$ $s_2 : 10000t_1$		$R_1 = R_2 = R_3 = R_4 = R_5 = \emptyset$	0		r_4	$\emptyset$	
1	0	0							$\{t_1\}$	exit from J and K loops
2	0	0		$t_6 : t_1000t_1$	$R_6 = R_{6_{s_1}} = \{r_{10}\} = \{0\}_{s_1"}$	1			$\{t_1\}$	
				$t_7 : 10000t_1$	$R_7 = R_{7_{s_2}} = \{r_{11}\} = \{0\}_{s_2}$	1				
2	1	0					t_6		$\{t_1\}$	
2	1	1	$s_1 : 000000t_6$ $s_2 : 10000t_1$		$R_6 = \emptyset$	0		r_10		
					$R_7 = \{r_{11}\} = \{0\}_{s_2}$	1				
2	1	0							$\{t_1, t_6\}$	exit from K loop
2	2	0					t_7		$\{t_1, t_6\}$	
2	2	1	$s_1 : 000000t_6$ $s_2 : 00000t_7$		$R_6 = R_7 = \emptyset$	0		r_11	$\{t_1, t_6\}$	
2	2	0							$\{t_1, t_6, t_7\}$	exit from all loops

TABLE I

ITERATIVE SOLUTION OF EXAMPLE 3 USING ALGORITHM OF FIG. 1

TABLE II

EXPERIMENTAL RESULTS: B4L0-FOUR BAND WAVELET TRANSFORM, B2L3-TWO BAND THREE LEVEL WAVELET TRANSFORM

Experiments	*# of shifts*			*# of operations*		
	org	[9]	*New Alg.*	*org*	[9]	*New Alg.*
B4L0_8	80	6	5	48	4	4
B4L0_12	112	8	7	80	7	6
B4L0_16	160	9	8	128	10	8
B4L0_24	256	15	13	224	17	12
B2L3_8	108	14	14	72	12	10
B2L3_12	180	14	14	144	22	16
B2L3_16	234	36	26	198	38	24
B2L3_24	270	44	24	234	44	24

TABLE III

EXPERIMENTAL RESULTS: POT-EXAMPLE 1 IN [6], DCT-DISCRETE COSINE TRANSFORM

Experiments	*# of shifts*				*# of operations*			
	org	[7]	[9]	*New Alg.*	*org*	[7]	[9]	*New Alg.*
POT	21	7	7	8	21	10	9	9
DCT_8	208	72	10	10	144	94	9	10
DCT_12	272	74	13	12	208	100	14	13
DCT_16	352	107	19	19	288	129	18	19
DCT_24	544	190	30	28	480	212	31	29

TABLE IV

EXPERIMENTAL RESULTS: FIR FILTERS 1

Experiments	*# of operations*			
	org	[8]	[9]	*New Alg.*
S1	11	6	6	6
S2	57	32	30	32
L1	145	58	60	58

TABLE V

EXPERIMENTAL RESULTS: FIR FILTERS 2

Experiments	*# of operations*				
	org	[3]	[8]	[9]	*New Alg.*
L2	49	22	23	23	23
L3	16	5	5	5	5

TABLE VI

EXPERIMENTAL RESULTS: 3000-TAP CHIRP FILTER

Wordlength (bits)	*# of operations*	*Run-time*
8	125	$<$ 1sec
12	987	5sec
16	2496	25sec
20	3598	1min30sec
24	4757	2min45sec

TABLE VII

NUMBER OF OPERATORS PRODUCED AFTER USING THE NEW ALGORITHM: + : ADDERS, − : SUBTRACTORS, + → − : NEGATIVE ADDERS

Experiments	*Before Refinements*			*After Refinements 1 and 2*		*After All Refinements*		
	+	−	+ → −	+	−	+	−	negator
B4L0_8	3	1	0	3	1	3	1	0
B4L0_12	3	3	0	4	2	6	0	1
B4L0_16	2	6	0	4	4	6	2	1
B4L0_24	5	7	0	8	4	11	1	1
B2L3_8	3	2	0	3	2	3	2	0
B2L3_12	3	5	0	5	3	5	3	0
B2L3_16	9	3	0	9	3	9	3	0
B2L3_24	8	4	0	8	4	8	4	0
DCT_8	4	5	1	5	5	7	3	2
DCT_12	6	7	0	6	7	9	4	2
DCT_16	11	7	1	12	7	17	2	2
DCT_24	19	10	0	19	10	22	7	1
POT	6	3	0	6	3	9	0	1
S1	2	3	1	3	3	5	1	1
S2	15	15	2	17	15	31	1	4
L1	25	26	7	32	26	54	4	6
L2	10	9	4	14	9	19	4	1
L3	2	3	0	2	3	5	0	1