

Short-Length FIR Filters and Their Use in Fast Nonrecursive Filtering

Zhi-Jian Mou, *Member, IEEE*, and Pierre Duhamel, *Senior Member, IEEE*

Abstract—This paper provides the basic tools required for an efficient use of the recently proposed fast FIR algorithms. These algorithms not only reduce arithmetic complexity but also partially maintain the multiply-accumulate structure, thus resulting in efficient implementations.

A set of basic algorithms is derived, together with some rules for combining them. Their efficiency is compared with that of classical schemes in the case of three different criteria, corresponding to various types of implementation. It is shown that this class of algorithms (which includes classical ones as special cases) makes it possible to find the best tradeoff corresponding to any criterion.

I. INTRODUCTION

MANY algorithms are known to reduce the arithmetic complexity of FIR filtering. The widely used ones are indirect algorithms, based either on the cyclic convolution or on the aperiodic convolution using fast transforms as an intermediate step. Direct methods without transforms were also proposed by Winograd [1].

Both direct and indirect methods require large block processing: they make use of the redundancy between at least L successive output computations (L is the length of the filter) to reduce the number of operations to be performed per output point.

Furthermore, the structure of the resulting algorithm has changed completely: the initial computation is mainly based on a multiply-accumulate (MAC) structure, while the fast algorithms always involve a global exchange of data inside a large vector with a minimum size of $2L$.

The main reasons why the above fast algorithms are not of wide interest for real-time filtering are as follows. Hardware implementations require pipelining the whole system with many intermediate memories, which results in a large amount of hardware. On the other hand, software implementations on digital signal processors (DSP's) are not very efficient, except for very large L , since those fast algorithms have lost the multiply-accumulate structure for which all DSP's are optimized.

In other words, the usefulness of such algorithms has decreased because the reduction in arithmetic require-

ments per output point was obtained at the expense of structural regularity. But structural regularity is difficult to quantify. Hardware implementations do not require the same kind of regularity as VLSI implementations, and "structural regularity" is still another matter when thinking of DSP implementations. One fact remains, MAC structure is very efficient on any type of implementation, including those on general purpose computers.

Recently, a new class of fast FIR filtering algorithms taking these considerations into account was proposed [2]–[4]: These algorithms retain partially the FIR filter structure, while reducing the arithmetic complexity. They allow various tradeoffs between structural regularity and arithmetic efficiency, including all classical schemes as special cases [10]. This flexibility in the algorithms' derivation allows finding the best possible solution in any type of implementation.

The purpose of this paper is to provide the basic tools required for the derivation of algorithms meeting various tradeoffs in different implementations.

A brief description of these new algorithms is provided in Section II. The structure of the new algorithms is explained: Short-length FIR filters with reduced arithmetic complexity where all multiplications are replaced by decimated subfilters. Since the process can be reiterated on the subfilters, the short-length filters are recognized as the basic building tools of these fast algorithms.

Hence, Section III is concerned with the derivation of a set of algorithms. This section is mostly based on Winograd's work. We bring some improvements in the number of additions by recognizing that FIR filtering, seen as a running process, involves a pseudocirculant matrix [5] instead of a general Toeplitz one. Another advantage of this presentation is the easy understanding of the transposition principle in the context of multi-input multi-output systems, overlapping between blocks being naturally taken into account. Using this pseudocirculant presentation, we can very easily derive the transposed version of all fast FIR filtering algorithms.

Section IV addresses the case of multifactor algorithms. Iterating the basic process raises the question of the best ordering of the short modules and of the length where the decomposition has to be stopped. We provide the rules for obtaining the ordering of factors which results in the lowest arithmetic complexity. A comparison with classical algorithms (FFT-based ones) is also provided in the case of real valued signals.

Manuscript received May 18, 1989; revised June 24, 1990.

Z.-J. Mou was with CNET/PAB/RPE, 92131 Issy-Les-Moulineaux, France. He is now with the Department of Electronics, Telecom Paris University, 75013 Paris, France.

P. Duhamel is with CNET/PAB/RPE, 92131 Issy-Les-Moulineaux, France.

IEEE Log Number 9143797.

Section V concludes the paper and explains some open problems.

II. GENERAL DESCRIPTION OF THE ALGORITHM

Let us consider the filtering of a sequence $\{x_i\}$ by a length- L FIR filter with fixed coefficients $\{h_i\}$:

$$y_n = \sum_{i=0}^{L-1} x_{n-i} h_i, \quad n = 0, 1, 2, \dots, \infty. \quad (1)$$

In z -domain formulation, this convolution becomes a polynomial product

$$Y(z) = H(z) X(z) \quad (2)$$

where X and Y are of infinite degree, while $H(z)$ has degree $L - 1$. In z domain, the filtering equation, seen as a running process, is described by the product of an infinite degree polynomial and a finite degree one.

Let us now decimate each of the three terms in (2) into N interleaved sequences:

$$\begin{aligned} H_j(z) &= \sum_{m=0}^{L/N-1} h_{mN+j} z^{-m}, & j &= 0, 1, \dots, N-1 \\ X_k(z) &= \sum_{m=0}^{\infty} x_{mN+k} z^{-m}, & k &= 0, 1, \dots, N-1 \\ Y_i(z) &= \sum_{m=0}^{\infty} y_{mN+i} z^{-m}, & i &= 0, 1, \dots, N-1. \end{aligned} \quad (3)$$

Equation (2) then becomes

$$\sum_{i=0}^{N-1} Y_i(z^N) z^{-i} = \sum_{j=0}^{N-1} H_j(z^N) z^{-j} \sum_{k=0}^{N-1} X_k(z^N) z^{-k}. \quad (4)$$

Equation (4) is in the form of a polynomial product or an aperiodic convolution. The two polynomials to be multiplied have finite degree $N - 1$, and their coefficients are themselves polynomials, either of finite degree, such as $\{H_j\}$, or of infinite degree, such as $\{X_k\}$ or $\{Y_i\}$.

Let us now forget for a while that the coefficients of the $(N - 1)$ th degree polynomials are also polynomials, and apply a fast polynomial product algorithm to compute the polynomial product in (4). It is well known, since the work of Winograd [1] that the product of two polynomials with N coefficients can be obtained with a minimum of $2N - 1$ general multiplications. This minimum can be reached for small N , while for larger N the optimal algorithm involves too many additions to be of practical interest. In that case, suboptimal ones are often preferred. Hence, the application of these polynomial product algorithms to (4) results in a scheme requiring $2N - 1$ "products," each one being, in fact, the product of a finite degree polynomial by an infinite degree sequence, that is, an FIR filtering of length- L/N .

Compared to the initial situation, the arithmetic complexity is now as follows.

Equation (4) requires N^2 filterings of length L/N , which is about L multiply-accumulates (MAC's) per output (it is

only a rearrangement of the initial equation), while the fast polynomial product based scheme requires $(2N - 1)$ filterings of length L/N , which is about $L(2N - 1)/N^2$ MAC's per output. Thus the improvement in arithmetic complexity is proportional to the length of the filter, and this is obtained at a fixed cost, depending on N . This means that, for large L , this approach will always be of interest. Precise comparisons are provided in Section III for each algorithm.

Slight additional improvements can be obtained by further considering (4). In fact, (4) contains not only the polynomial product, which allows the arithmetic complexity to be reduced, but also the so-called "overlap" in classical FFT-based schemes. By equating both sides of (4), we have

$$\begin{aligned} Y_{N-1} &= \sum_{i=0}^{N-1} X_{N-1-i} H_i \\ Y_k &= z^{-N} \sum_{i=k+1}^{N-1} X_{N+k-i} H_i + \sum_{i=0}^k X_{k-i} H_i, \\ &0 \leq k \leq N-2 \end{aligned} \quad (5)$$

or in matrix form

$$\begin{bmatrix} Y_{N-1} \\ Y_{N-2} \\ \vdots \\ Y_0 \end{bmatrix} = \begin{bmatrix} H_0 & H_1 & \cdots & H_{N-1} \\ z^{-N} H_{N-1} & H_0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & H_1 \\ z^{-N} H_1 & \cdots & z^{-N} H_{N-1} & H_0 \end{bmatrix} \begin{bmatrix} X_{N-1} \\ X_{N-2} \\ \vdots \\ X_0 \end{bmatrix}. \quad (6)$$

The right side of (6) is the product of a pseudocirculant matrix [5] and a vector. Note that $\{X_i\}$ and $\{H_i\}$ play a symmetric role, hence can be exchanged in (6). The equation is clearly in the form of a length- N FIR filter whose coefficients are $\{H_i\}$, of which N outputs $\{Y_i\}$ are computed. Following the notations of Winograd [1], we shall denote in the following an algorithm computing M outputs of a length- N FIR filter by an $F(M, N)$ algorithm. Considering the FIR filter as a whole, and the fast FIR algorithm as the "diagonalization" of the pseudocirculant matrix of (6) results in some circumstances in a reduction of the number of additions involved, compared with the usual approach which separates the polynomial product and the overlap. This will be seen in Section III.

With the explanation above, the proposed algorithms can be understood as a multidimensional formulation of the FIR filtering, where computations along one dimension are performed through an efficient $F(N, N)$ algorithm, resulting in a reduced arithmetic complexity, while

the other dimension uses a direct computation, thus allowing the process to be a running one.

Let us also point out that, if the FIR filter of (5) or (6) is computed through an FFT-based scheme, and with the appropriate choice of N versus L , the usual FFT-based implementation of FIR filters can be seen to be a member of that class of algorithms. This is explained in [10], where it is shown that all fast FIR schemes including FFT-based ones can be expressed as: decimation of the involved sequences (N on input and output, M on the filter); evaluation of the obtained polynomials at $N + M - 1$ "interpolation points" $\{\alpha_i\}$; "dot product" (or filtering); reconstruction of the resulting polynomials and overlap.

All the algorithms differ only by the choice of N , M , and $\{\alpha_i\}$.

Section III provides various short-length FIR filtering algorithms in the case of real valued sequences.

III. SHORT-LENGTH FIR FILTERING ALGORITHMS

Let us first explain in some detail the simplest case, which is an $F(2, 2)$ algorithm, as given in [2]–[4]: Considering $N = 2$, $\{\alpha_i\} = \{0, 1, \infty\}$, we obtain

$$\begin{aligned} a_0 &= x_0 & b_0 &= h_0 \\ a_1 &= x_0 + x_1 & b_1 &= h_0 + h_1 \\ a_2 &= x_1 & b_2 &= h_1 \\ m_i &= a_i b_i; & i &= 0, 1, 2 \\ y_0 &= m_0 + z^{-2} m_2 \\ y_1 &= m_1 - m_0 - m_2. \end{aligned} \quad (\text{a1})$$

When used in (5) above, this algorithm results in the filtering scheme of Fig. 1 where the problem of computing 2 outputs of a length N filter is turned into that of computing one output of three length- $N/2$ filters, at the cost of 4 adds per block of 2 outputs.

Comparison of arithmetic complexities are now as follows: The initial scheme has a cost of L mults and $(L - 1)$ adds per output point, to be compared with $(3/4)L$ mults per output and $2 + (3/2)(L/2 - 1) = (3/4)L + 1/2$ adds per output.

It is seen that, except for very small L , the numbers of both multiplications and additions have been reduced.

Successive decompositions are feasible, up to the point where the desired tradeoff has been obtained. Of course, this tradeoff depends on the implementation. On general purpose computers where a multiplication and an addition require about the same amount of time, the tradeoff allowing the fastest implementation will certainly correspond to a decomposition near the one minimizing the total number of arithmetic operations. On digital signal processors, a multiply-accumulate operation will in general cost one clock cycle, and an appropriate criterion is certainly to count a MAC as a single operation. A third criterion of interest is the minimum number of multiplications. In the

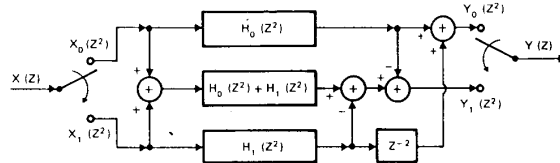


Fig. 1. FIR filtering based on algorithm (a1).

following, tables giving the minimum numbers for these three criteria will be provided.

Nevertheless, in nearly all types of implementations, the situation is much alike: the decomposition provided in Fig. 1 reduces the arithmetic load in a manner proportional to L , the length of the filter, at a fixed cost (initialization of one MAC loop, or one MAC loop plus two adds). Hence the balance between the performances of algorithms depends on the timing spent in the initializations. The precise N by which the splitting becomes of interest therefore depends on the specific machine or circuit. Anyway, this type of splitting will always be of interest for large length filters, or even medium-size ones.

The remaining part of this section provides the simplest $F(M, N)$ algorithms that can be used to reduce the arithmetic complexity of a length- L filtering. Different versions are provided, resulting in various operation counts, and various sensitivities to roundoff noise, a point which will not be dealt with in this paper.

The following algorithm is an $F(2, 2)$ algorithm with interpolation points $\{\alpha_i\} = \{0, -1, \infty\}$. It has exactly the same complexity as the previous one:

$$\begin{aligned} a_0 &= x_0 & b_0 &= h_0 \\ a_1 &= x_0 - x_1 & b_1 &= h_0 - h_1 \\ a_2 &= x_1 & b_2 &= h_1 \\ m_i &= a_i b_i; & i &= 0, 1, 2 \\ y_0 &= m_0 + z^{-2} m_2 \\ y_1 &= m_0 + m_2 - m_1. \end{aligned} \quad (\text{a2})$$

For the sake of completeness, two algorithms computing $F(2, 2)$ with $\{\alpha_i\} = \{0, 1, -1\}$ and $\{\alpha_i\} = \{1, -1, \infty\}$ are provided in Appendix B. They may be of interest as far as roundoff noise is concerned, but they require 3 multiplications and 6 additions, that is, one more addition per output. Note that on some implementations (and especially on DSP's), this is not a real drawback, since these additions are now of the type $a \pm b$, which can be efficiently implemented in many cases.

It is well known in the case of the usual implementation of a digital filter that the transposition of a graph provides a digital filter with the same transfer function. Winograd has proposed an approach which allows obtaining short-length FIR filtering algorithms by transposing polynomial product algorithms. We have shown that the simple overlapping of polynomial products in (5) is sufficient to con-

struct FIR filtering algorithms. The transposition of polynomial products is not necessary. It only provides alternative versions of the algorithms. In the context of pseudocirculant matrix, we can transpose the algorithms in a much easier way, and we can prove that the total arithmetic complexity of all these algorithms is not changed by the transposition operation (the number of both the multiplications and the additions). More details are given in Appendix A.

The following algorithms are the transposed versions of Algorithms (a1) and (a2).

$$\begin{aligned}
 a_0 &= x_0 - x_1 & b_0 &= h_0 \\
 a_1 &= x_0 & b_1 &= h_0 + h_1 \\
 a_2 &= z^{-2}x_1 - x_0 & b_2 &= h_1 \\
 m_i &= a_i b_i; \quad i = 0, 1, 2 & & (a3) \\
 y_0 &= m_1 + m_2 \\
 y_1 &= m_1 - m_0
 \end{aligned}$$

$$\begin{aligned}
 a_0 &= x_0 + x_1 & b_0 &= h_0 \\
 a_1 &= x_0 & b_1 &= h_0 - h_1 \\
 a_2 &= z^{-2}x_1 + x_0 & b_2 &= h_1 \\
 m_i &= a_i b_i; \quad i = 0, 1, 2 \\
 y_0 &= m_1 + m_2 \\
 y_1 &= m_0 - m_1. & & (a4)
 \end{aligned}$$

The use of (a3) in a large FIR filter is provided in Fig. 2.

Iterating the above algorithms results in radix-2 FIR algorithms, with a tree-like structure, as proposed in [4].

Higher radix algorithms can also be derived, and should be more efficient, as seen at the end of Section II, since the ratio $(2N - 1)/N^2$ decreases. However, an optimal $F(3, 3)$ algorithm would require 5 different interpolation points, that is, one more than the simplest ones: $\{\alpha_i\} = \{0, 1, -1, ?, \infty\}$, and the next simplest choices of the last interpolation point $\{\pm 2, \pm 1/2\}$ result in an increased number of additions, and an increased sensitivity to roundoff noise. This is the reason why it is advisable to use a suboptimal $F(3, 3)$ algorithm which provides a better tradeoff between the number of multiplications and the number of additions.

Such an algorithm can be obtained by applying twice Algorithm (a1) as follows. Let us remark that (a1) is based on the following equation:

$$\begin{aligned}
 &(x_0 + x_1 z^{-1})(h_0 + h_1 z^{-1}) \\
 &= x_0 h_0 + x_1 h_1 z^{-2} + [(x_0 + x_1)(h_0 + h_1) \\
 &\quad - x_0 h_0 - x_1 h_1] z^{-1}. & (7)
 \end{aligned}$$

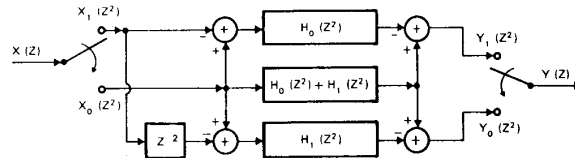


Fig. 2. FIR filtering based on algorithm (a3).

Inspired by (7) we rewrite the radix-3 aperiodic convolution equation as follows:

$$\begin{aligned}
 &[x_0 + (x_1 + x_2 z^{-1})z^{-1}][h_0 + (h_1 + h_2 z^{-1})z^{-1}] \\
 &= (x_0 + Fz^{-1})(h_0 + Gz^{-1}) \\
 &= x_0 h_0 + [(x_0 + F)(h_0 + G) - x_0 h_0 - FG]z^{-1} \\
 &\quad + FGz^{-2}. & (8)
 \end{aligned}$$

Then, the algorithm in (7) is once more applied to the computation of $(x_0 + F)(h_0 + G)$ and FG which are still radix-2 aperiodic convolutions. This results in an aperiodic convolution algorithm requiring 6 mults and 9 adds (an optimal one would require 5 mults and 20 adds [11, p. 86]). Overlap has then to be performed by merging the terms z^0 and z^{-3} , and the term z^{-1} and z^{-4} , with the appropriate delay. Hence, the $F(3, 3)$ algorithm seems to require 2 more adds than the corresponding radix-3 aperiodic convolution algorithm. Nevertheless, considering the redundancy during the overlap in this $F(3, 3)$ algorithm allows a further reduction of the number of additions to 10 adds, the lowest number of operations to our knowledge:

$$\begin{aligned}
 a_0 &= x_0 & b_0 &= h_0 \\
 a_1 &= x_1 & b_1 &= h_1 \\
 a_2 &= x_2 & b_2 &= h_2 \\
 a_3 &= x_0 + x_1 & b_3 &= h_0 + h_1 \\
 a_4 &= x_1 + x_2 & b_4 &= h_1 + h_2 \\
 a_5 &= x_0 + a_4 & b_5 &= h_0 + h_1 + h_2 \\
 m_i &= a_i b_i; \quad i = 0, 1, 2, 3, 4, 5 \\
 t_0 &= m_0 - m_2 z^{-3} \\
 t_1 &= m_3 - m_1 \\
 t_2 &= m_4 - m_1 \\
 y_0 &= t_0 + t_2 z^{-3} \\
 y_1 &= t_1 - t_0 \\
 y_2 &= m_5 - t_1 - t_2. & & (a5)
 \end{aligned}$$

The use of this $F(3, 3)$ algorithm to reduce the arithmetic complexity of a larger FIR filter is provided in Fig. 3, showing that the overall structure is that of a multirate filter bank, where $\{H_1 + H_2, H_1, H_0 + H_1, H_2, H_0, H_0 + H_1 + H_2\}$ are decimated FIR filters.

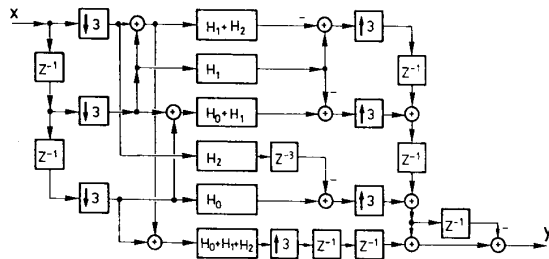


Fig. 3. FIR filtering based on Algorithm (a5).

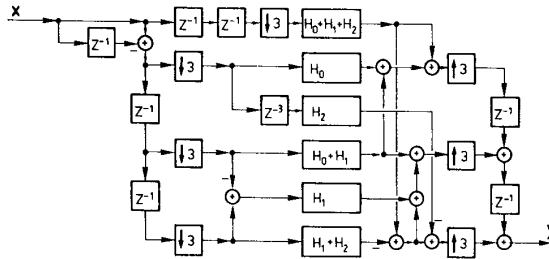


Fig. 4. FIR filtering based on Algorithm (a6).

Transposition of Algorithm (a5) results in the following one, which is depicted in Fig. 4:

$$\begin{aligned}
 a_0 &= x_2 - x_1 & b_0 &= h_0 \\
 a_1 &= (x_0 - x_2 z^{-3}) - (x_1 - x_0) & b_1 &= h_1 \\
 a_2 &= -a_0 z^{-3} & b_2 &= h_2 \\
 a_3 &= (x_1 - x_0) & b_3 &= h_0 + h_1 \\
 a_4 &= (x_0 - x_2 z^{-3}) & b_4 &= h_1 + h_2 \\
 a_5 &= x_0 & b_5 &= h_0 + h_1 + h_2 \\
 m_i &= a_i b_i; \quad i = 0, 1, 2, 3, 4, 5 \\
 y_0 &= m_2 + (m_4 + m_5) \\
 y_1 &= m_1 + m_3 + (m_4 + m_5) \\
 y_2 &= m_0 + m_3 + m_5.
 \end{aligned} \tag{a6}$$

When used in a length- L FIR filter, both schemes (a5) and (a6) require $2L/3$ multiplications and $(2L + 4)/3$ additions per output point, to be compared with L and $L - 1$ operations, respectively, in the direct computation. This means that the computational load has been reduced by nearly $1/3$.

Careful examination of Figs. 1–4 shows that the distribution of the additions between the input samples and the subfilters' outputs is not the same in the initial algorithms and their transposed versions. In all cases, transposed algorithms have more input additions and less output additions. This fact should give them more robustness towards quantization noise.

Another case, which looks interesting at first glance is as follows: why not decimate the filter by a factor of 2, and X and Y by a factor of 3? This would be solved by an $F(3, 2)$ algorithm, which requires $3 + 2 - 1 = 4$ inter-

polating points, which is the very number of the simplest interpolating points $\{0, 1, -1, \infty\}$. This means that $F(3, 2)$ or $F(2, 3)$ are the largest filtering modules that can be computed efficiently with an optimum number of multiplications:

$$\begin{aligned}
 a_0 &= x_3 - x_1 & b_0 &= h_0 \\
 a_1 &= x_1 + x_2 & b_1 &= (h_0 + h_1)/2 \\
 a_2 &= x_1 - x_2 & b_2 &= (h_0 - h_1)/2 \\
 a_3 &= x_2 - x_0 & b_3 &= h_1 \\
 m_i &= a_i b_i; \quad i = 0, 1, 2, 3 \\
 y_0 &= (m_1 + m_2) - m_3 \\
 y_1 &= m_1 - m_2 \\
 y_2 &= m_0 + (m_1 + m_2).
 \end{aligned} \tag{a7}$$

This algorithm looks promising, since the same performance as an $F(3, 3)$ algorithm is obtained with a simpler one: $F(3, 2)$ requires 4 multiplications and 8 additions, which means that it reduces the number of MAC's by $1/3$, at the cost of 8 additions. Nevertheless, problems arise when using this algorithm for speeding up the computation of a large length- L filter. The overall structure is depicted in Fig. 5, where the main computing modules are a kind of $1/3$ FIR decimators. Obtaining 3 successive outputs of the filter requires the computation of 4 length- $L/2$ inner products plus 13 adds, to be compared with Algorithm (a5) which requires 6 length- $L/3$ inner products, plus 10 adds. Therefore, (a7) and (a5) have nearly the same arithmetic complexity. But (a7) requires $3L$ memory registers, instead of $2L$ registers in (a5), and a more complex control system, since the inner products are not true FIR filters any more.

All aperiodic convolution algorithms can be turned into FIR filtering algorithms by appropriate overlap. Suboptimal higher radix (≥ 5) aperiodic convolution algorithms can be derived using the approach in [12]. An $F(5, 5)$ algorithm based on this approach is given in Appendix B. This algorithm, requiring 12 mults and 40 adds, is not optimum as far as the number of multiplications is concerned, but reaches the best tradeoff we could obtain.

All these $F(N, N)$ algorithms are clearly similar to the ones proposed by Winograd [1]. They differ essentially on several points: First, the recognition that, by using decimated sequences, the arithmetic complexity of an FIR filter can be reduced as soon as two outputs of the filter are computed, regardless of the filter's length. Winograd's algorithms always require the computation of at least as many outputs as the filter's order. Second, a straightforward derivation through (4) of the $F(N, N)$ algorithms by polynomial product algorithms which were extensively studied in the literature. Third, a reduction in the number of additions, which is made feasible by naturally taking into account the "overlap" between two consecutive blocks of outputs (hence, the use of the pseudocirculant matrix). Fourth, a new interpretation, in the

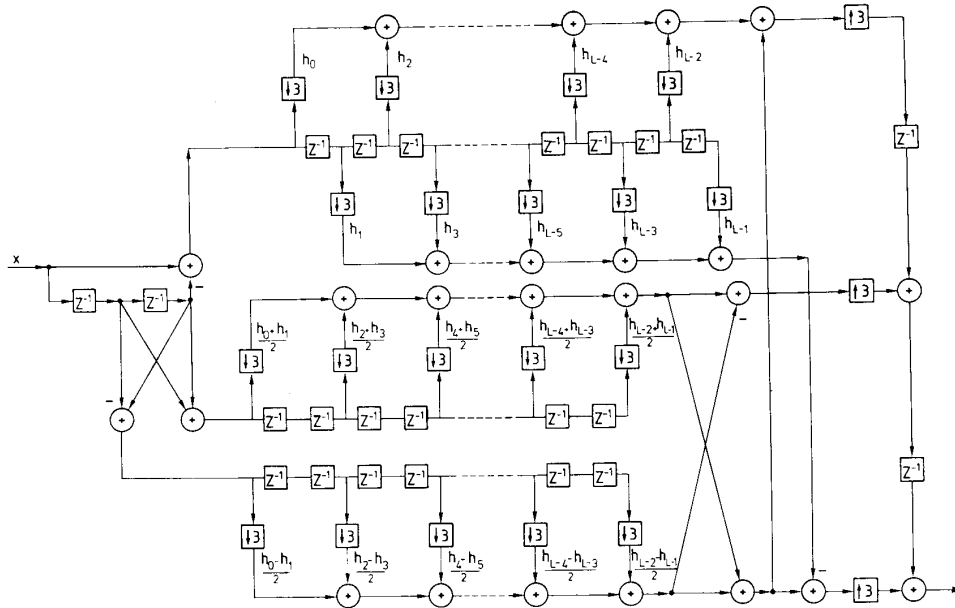


Fig. 5. FIR filtering based on Algorithm (a7).

context of pseudocirculant matrix, of algorithm transposition and a systematic method of obtaining transposed versions. (See Appendix A.)

IV. COMPOSITE LENGTH ALGORITHMS

We explained in Section II that the application of an $F(N, N)$ algorithm could break the computation of a length- L FIR filter into that of several length- L/N filters, in such a manner that the arithmetic complexity is decreased. Nevertheless, the same process can iteratively be applied to the subfilters of length L/N as well, leading to composite length algorithms.

This section is concerned with the problem of finding the best way of combining the small-length filters, depending on various criteria.

We first evaluate the arithmetic complexity of a length $L = N_1 N_2 L_2$ filter, when using two successive decompositions by $F(N_1, N_1)$ first, and then by $F(N_2, N_2)$. Let us assume that, with the notations of Section III, the length- N_i filter requires M_i multiplications and A_i additions.

The first decomposition by $F(N_1, N_1)$ turns the initial problem into that of computing M_1 filters of length $N_2 L_2$ at the cost of A_1 additions. Further decompositions of the length- $N_2 L_2$ subfilters using $F(N_2, N_2)$ require the consideration of a block of N_2 outputs of these subfilters (hence, a block of $N_1 N_2$ outputs of the whole filter). Each of these subfilters is then transformed into M_2 filters of length L_2 at the cost of A_2 additions.

The global decomposition thus has the following arithmetic complexity:

$$N_2 A_1 \text{ additions} \\ + M_1 [M_2 \text{ length-}L_2 \text{ subfilters} + A_2 \text{ additions}].$$

That is,

$$M = M_1 M_2 L_2 \quad (9)$$

$$A = N_2 A_1 + M_1 A_2 + M_1 M_2 (L_2 - 1). \quad (10)$$

Let $m_i = M_i/N_i$ and $a_i = A_i/N_i$ where $i = 1, 2$. m_i and a_i are the numbers of operations (mults and adds, respectively) per output required by $F(N_i, N_i)$. Since (9) and (10) are the arithmetic load for computing $(N_1 N_2)$ outputs, the numbers of operations per output for the whole algorithm are

$$m = m_1 m_2 L_2 \quad (11)$$

$$a = a_1 + m_1 a_2 + m_1 m_2 (L_2 - 1). \quad (12)$$

An application of $F(N_2, N_2)$ first, followed by $F(N_1, N_1)$ would result in the same number of multiplications, and a number of additions which will be higher than (12) as long as

$$a_1 + m_1 a_2 < a_2 + m_2 a_1$$

or

$$(m_1 - 1)/a_1 < (m_2 - 1)/a_2 \quad (13)$$

or, equivalently,

$$(M_1 - N_1)/A_1 < (M_2 - N_2)/A_2. \quad (14)$$

Let us define

$$Q[F(N_i, N_i)] = (m_i - 1)/a_i = (M_i - N_i)/A_i. \quad (15)$$

Q is a parameter specific of an algorithm. Its use has already been proposed in [8] for the cyclic convolution. Equation (14) means that the lowest number of additions is obtained by first applying the short length FIR filter with the smallest Q , and then the one with the second

TABLE I
QUALITY FACTORS FOR SEVERAL
SHORT-LENGTH ALGORITHMS

Algorithms	Q
$F(1, N)$	1
$F(2, 2)$	0.25
$F(3, 3)$	0.3
$F(5, 5)$	0.175

smallest Q and so on. Table I provides $Q[F(N, N)]$ for the most useful short-length filters. Two useful properties of $Q[F(N, N)]$ are as follows.

1) A straightforward FIR filtering "algorithm" has $Q[F(1, N)] = 1$, whatever N is. This means that, in order to minimize the number of additions, they must be located at the "center" of the overall algorithm, as was implicitly assumed up to that point.

2) Iteratively applying an algorithm $F(p, p)$ to obtain $F(p^k, p^k)$ results in an algorithm with the same coefficient Q :

$$Q[F(p^k, p^k)] = Q[F(p, p)]. \quad (16)$$

The demonstration is easily obtained by simply recognizing that applying first $F(p^k, p^k)$ and second $F(p, p)$ or applying them in the reverse order results in the same $F(p^{k+1}, p^{k+1})$.

Hence, as an example, an optimal ordering for $L = 120L_i = 2^3 \times 3 \times 5L_i$ would be

$$F(5, 5), F(2, 2), F(2, 2), F(2, 2), F(3, 3), F(1, L_i).$$

Of course, when it is desired to implement a length- L filter, it is very unlikely that $F(L, L)$ is the most suitable algorithm for a specific type of implementation, even in the case where L is composite. The best thing to do, is to search for the tradeoff minimizing some criteria depending on the implementation.

It is not our propose here to perform such an optimization in a special case, but we shall try to show, in the following, that improvements are feasible whatever the criterion.

Assuming that $L = N_1 \cdots N_i \cdots N_r L_r$, and that a fast algorithm $F(N_i, N_i)$ is used for all N_i , a straightforward one being used for L_r , the general formulas for evaluating the arithmetic complexity per output of a length- L filter are as follows:

$$m = L_r \prod_{i=1}^r m_i \quad (17)$$

$$a = \sum_{i=1}^r a_i \prod_{j=1}^{i-1} m_j + (L_r - 1) \prod_{i=1}^r m_i. \quad (18)$$

A first criterion of interest would be the minimization of the number of multiplications. Examination of (17) shows that such a minimization is performed by fully decomposing L into $N_1 \cdots N_i \cdots N_r L_r$, and this results in the number of operations given in Table II.

All the operation counts are provided assuming that the signal and the filter are real-valued, and the comparison is made with the real-valued FFT-based schemes [6], [9] of twice the filter's length. Table II shows that the proposed approach is more efficient than FFT-based schemes up to $L = 36$, and very competitive up to $L = 64$ which covers most useful lengths.

However, with today's technology, multiplication timings are not the dominant part of the computations any more when a parallel multiplier is built in the computer, and a very useful criterion is the sum of the number of additions and multiplications, i.e., $M + A$.

Table III provides a description of the algorithms requiring the minimum value for such a criterion. It is seen that, although the basic $F(N, N)$ algorithms do not improve the criterion, all the composite ones can be improved, and are even more efficient than FFT-based schemes up to $N = 64$. Consider $N = 16$, for example: FFT-based schemes hardly improve the direct computation (29.5 operations per point versus 31), while the $F(2, 2)$ -based algorithm requires only 19.6 operations per point.

The previous criterion is well suited for general purpose computer implementation, where all operations are performed sequentially, but digital signal processors still state another problem. In fact, DSP's perform generally a whole multiply-accumulate (MAC) operation in a single clock cycle, so that a MAC should be considered as a single operation, which is not more costly than an addition alone.

Table IV provides a description of the algorithm minimizing the corresponding criterion: the sum of the number of MAC's and I/O additions. Of course, this criterion is a rough measure of efficiency, since initialization time of MAC loops is not taken into account. Nevertheless, what Table IV shows is that even this kind of criterion, taking partially into account the structure of DSP's, can be improved using our approach: a length-64 filter can be implemented using this type of algorithms with nearly half the total number of operations (MAC's + I/O adds) per output point, compared to the trivial algorithm. And this is obtained with a block size of only 8 points. This shows that moderate to large length filters can be efficiently implemented on DSP's using these techniques.

Two points should be emphasized here.

1) A lot of systems requiring digital filtering have constraints on the input/output delay, which prevents the use of FFT-based implementation of digital filter. Our approach allows obtaining a reduction of the arithmetic complexity whatever the block size. This means that our approach allows reducing the arithmetic complexity by taking into account such requirements as a constraint on the I/O delay.

2) Another remark, which is not apparent in the tables, is that for a given filter length L , there are often several algorithms providing comparable performance, for a given criterion. It allows us to choose from them the one which is best suited for the specific implementation.

TABLE II
ARITHMETIC COMPLEXITY OF AN $F(N, N)$ ALGORITHM BY SHORT-LENGTH FIR AND BY FFT-BASED CYCLIC CONVOLUTION

Short-Length FIR					FFT-Based FIR				Direct FIR	
N	M	m	A	a	M_{FFT}	m	A_{FFT}	a	m	a
2	3	1.5	4	2					2	1
3	6	2	10	3.3					3	2
4	9	2.25	20	5	15	3.75	43	10.75	4	3
5	12	2.4	40	8					5	4
6	18	3	42	7	33	5.5	83	13.83	6	5
8	27	3.38	76	9.5	43	5.38	131	16.38	8	7
9	36	4	90	10					9	8
10	36	3.6	128	12.8					10	9
12	54	4.5	150	12.5					12	11
15	72	4.8	240	16	112	7.47	353	23.53	15	14
16	81	5.06	260	16.25	115	7.19	355	22.19	16	15
18	108	6	306	17					18	17
20	108	5.4	400	20					20	19
24	162	6.75	498	20.75					24	23
25	144	5.76	680	27.2					25	24
27	216	8	630	23.33					27	26
30	216	7.2	744	24.8	225	7.50	829	27.63	30	29
32	243	7.59	844	26.38	291	9.09	899	28.09	32	31
36	324	9	990	27.5	271	7.53	1084	30.11	36	35
60	648	10.8	2280	38	455	7.58	1955	32.58	60	59
64	729	11.39	2660	41.56	701	10.95	2179	34.05	64	63
128	2187	17.09	8236	64.34	1667	13.02	5123	40.02	128	127
256	6561	25.63	25220	98.52	3483	13.61	11779	46.01	256	255
512	19683	38.44	76684	149.77	8707	17.01	26627	52.01	512	511
1024	59049	57.67	232100	226.66	19459	19.00	59395	58.00	1024	1023

TABLE III
MINIMUM SUM OF OPERATIONS (MULTS + ADDS)

N	Decomposition	$M + A$	$/N$	Block Size
2	direct	6	3	1
3	direct	15	5	1
4	(2 × 2)	26	6.5	2
5	direct	45	9	1
6	(3 × 2)	56	9.33	3
8	(2 × 2 × 2)	94	11.75	4
9	(3 × 3)	120	13.33	3
10	(5 × 2)	152	15.2	5
12	(2 × 3 × 2)	192	16	6
15	(5 × 3)	300	20	5
16	(2 × 2 × 2 × 2)	314	19.63	8
18	(2 × 3 × 3)	396	22	6
20	(5 × 2 × 2)	472	23.6	10
24	(2 × 2 × 3 × 2)	624	26	12
25	(5 × 5)	740	29.6	5
27	(3 × 3 × 3)	810	30	9
30	(5 × 3 × 2)	912	30.4	10
32	(2 × 2 × 2 × 2 × 2)	1006	31.44	16
36	(2 × 2 × 3 × 3)	1260	35	12
60	(5 × 2 × 3 × 2)	2784	46.4	30
64	(FFT)	2880	45.00	64
128	(FFT)	6790	53.05	128
256	(FFT)	15262	59.62	256
512	(FFT)	35334	69.01	512
1024	(FFT)	78854	77.01	1024

($m \times n$) means the decomposition by a fast $F(m, m)$ algorithm using optimal ordering followed by direct length- n FIR filters. (FFT) means using FFT-based schemes.

V. CONCLUSION

We have presented a new class of algorithms for FIR filtering, showing that the basic building tools are short-

TABLE IV
MINIMUM NUMBER OF MACS (LENGTH OF SCALAR PRODUCTS + I/O ADDS)

N	Decomposition	MAC	$/N$	Block Size
2	direct	4	2	1
3	direct	9	3	1
4	direct	16	4	1
5	direct	25	5	1
6	direct	36	6	1
8	direct	64	8	1
9	direct	81	9	1
10	(2 × 5)	95	9.5	2
12	(2 × 6)	132	11	2
15	(3 × 5)	200	13.33	3
16	(2 × 8)	224	14	2
18	(2 × 9)	279	15.5	2
20	(2 × 2 × 5)	325	16.25	4
24	(2 × 2 × 6)	444	18.5	4
25	(5 × 5)	500	20	5
27	(3 × 9)	576	21.33	3
32	(2 × 2 × 8)	736	23	4
36	(2 × 2 × 9)	909	25.25	4
60	(5 × 2 × 6)	2064	34.4	10
64	(2 × 2 × 2 × 8)	2336	36.5	8
128	(2 ⁴ × 8)	7264	56.75	16
256	(2 ⁵ × 8)	22304	87.13	32
512	(2 ⁶ × 8)	67936	132.69	64
1024	(2 ⁷ × 8)	205856	201.03	128

($m \times n$) means the decomposition by a fast $F(m, m)$ algorithm using optimal ordering followed by direct length- n FIR filters.

length FIR modules in which the "multiplications" are replaced by decimated subfilters.

We have also provided the basic tools required for implementing these algorithms.

First, we propose short-length FIR filter modules of the Winograd type with a small number of multiplications and the smallest number of additions. Their transposed versions are also provided.

Second, we give some rules concerning the best way to cascade these short-length FIR modules to obtain composite-length algorithms.

Finally, we show that, for three different criteria, this class of algorithms obtains better tradeoffs than the previously known algorithms, which should make them useful in any kind of implementation.

It is concluded that the presented algorithms not only compute more efficiently moderate- to long-length FIR filtering on DSP's, but also more efficiently compute short-length (< 64) FIR filtering than the FFT-based algorithms, when considering the total number of operations.

These algorithms also suggest efficient multiprocessor implementations due to their inherent parallelism, and efficient realization in VLSI, since their implementations require only local communication, instead of a global exchange of data, as is the case for FFT-based algorithms.

APPENDIX A

1. Transposition of an $F(N, N)$ Algorithm

Let us consider an $F(N, N)$ algorithm as an (N -input, N -output) system shown in Fig. 6(a). Its transmission matrix is pseudocirculant:

$$P(z) = \begin{bmatrix} H_0 & H_1 & \cdots & H_{N-1} \\ z^{-N}H_{N-1} & H_0 & \cdots & \vdots \\ \vdots & \vdots & \ddots & H_1 \\ z^{-N}H_1 & \cdots & z^{-N}H_{N-1} & H_0 \end{bmatrix}. \quad (19)$$

The transposed system [shown in Fig. 6(b)] will have $P^T(z)$, which is the transpose of $P(z)$, as its transmission matrix according to Tellegen's theorem for digital networks [13].

Unlike (1-input, 1-output) systems, an (N -input, N -output) system's transpose will not perform the same function as the initial one unless $P(z)$ is symmetric, i.e., $P(z) = P^T(z)$. Owing to P 's Toeplitz structure, we can have the transposed system perform the same function as the initial one.

The transposed system is as follows:

$$\begin{bmatrix} Y'_{N-1} \\ Y'_{N-2} \\ \vdots \\ Y'_0 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-N}H_{N-1} & \cdots & z^{-N}H_1 \\ H_1 & H_0 & \vdots & \vdots \\ \vdots & \vdots & \ddots & z^{-N}H_{N-1} \\ H_{N-1} & \vdots & \vdots & H_1 & H_0 \end{bmatrix} \begin{bmatrix} X'_{N-1} \\ X'_{N-2} \\ \vdots \\ X'_0 \end{bmatrix}. \quad (20)$$

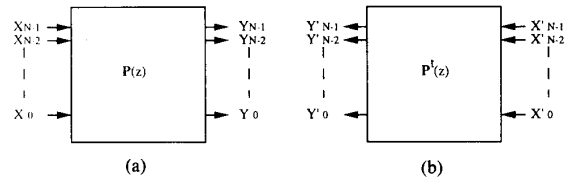


Fig. 6. (a) Initial system; (b) transposed system.

If we permute the input elements $\{X'_i\}$ and the output elements $\{Y'_i\}$ in such a way that their order is reversed, the transposed system will perform the appropriate function:

$$\begin{bmatrix} Y'_0 \\ Y'_1 \\ \vdots \\ Y'_{N-1} \end{bmatrix} = \begin{bmatrix} H_0 & H_1 & \cdots & H_{N-1} \\ z^{-N}H_{N-1} & H_0 & \vdots & \vdots \\ \vdots & \vdots & \ddots & H_1 \\ z^{-N}H_1 & \cdots & z^{-N}H_{N-1} & H_0 \end{bmatrix} \begin{bmatrix} X'_0 \\ X'_1 \\ \vdots \\ X'_{N-1} \end{bmatrix}. \quad (21)$$

The above demonstration is general, so that all (N -input, N -output) systems whose transmission matrix is Toeplitz can be transposed to perform the same function as the initial system after permuting the inputs and outputs in a reversed order. Thus, it can be applied to all kinds of convolution systems. Winograd has already proposed transposing circular convolution algorithms in the same manner [7]. We summarize this principle as the following theorem.

Theorem: The Toeplitz nature of the transmission matrix is sufficient for an (N -input, N -output) system to be transposed and to perform the same function provided that the inputs and outputs be a reverse order.

Hence, we can transpose an $F(N, N)$ algorithm rather easily. In fact, a fast $F(N, N)$ algorithm diagonalizes a pseudocirculant matrix:

$$P(z) = A_{N \times M} H_{M \times M} B_{M \times N} \quad (22)$$

where $H_{M \times M}$ is a diagonal matrix. Then the transposition of $P(z)$ is

$$P'(z) = B'HA'. \quad (23)$$

Permuting the inputs and outputs in a reversed order is equivalent to multiplying $J = \text{antidiag}(1, 1, \dots, 1)$ on the right and the left, respectively. Then the following equation holds:

$$P(z) = (JB')H(A'J). \quad (24)$$

Therefore, we get the transposed version of (22). The new pair of diagonalizing matrices are (JB') and $(A'J)$.

Let us consider as an example an $F(2, 2)$ algorithm, as given in (a1). It is expressed in matrix form as

$$\begin{aligned} \begin{bmatrix} y_1 \\ y_0 \end{bmatrix} &= \begin{bmatrix} h_0 & h_1 \\ z^{-2} h_1 & h_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \\ &= \begin{bmatrix} -1 & 1 & -1 \\ 1 & 0 & z^{-2} \end{bmatrix} \begin{bmatrix} h_0 \\ h_0 + h_1 \\ h_1 \end{bmatrix} \\ &\quad \cdot \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \end{aligned}$$

therefore

$$A = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 0 & z^{-2} \end{bmatrix} \quad B = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

Following (24), we obtain the transposed version of (a1):

$$\begin{aligned} \begin{bmatrix} y_1 \\ y_0 \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} h_0 \\ h_0 + h_1 \\ h_1 \end{bmatrix} \\ &\quad \cdot \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ z^{-2} & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \end{aligned}$$

which is the matrix expression of (a3).

2. Identical Arithmetic Complexity in Initial and Transposed $F(N, N)$ Algorithms

It is clear, following the above explanations, that the multiplicative complexity is not changed by transposition. Let us consider the additive complexity in an (N -input, N -output) system.

A digital network is composed of only branches and nodes. There are two kinds of nodes: a ($M, 1$) summing

node that adds M inputs into 1 output, and a ($1, M$) branching node that branches 1 input into M outputs. A ($M, 1$) summing node can be split into $M - 1$ ($2, 1$) summing nodes. A ($1, M$) branching node can also be split into $M - 1$ ($1, 2$) branching nodes. It is then easy to transform the network into an equivalent one having only ($2, 1$) summing nodes and ($1, 2$) branching nodes. After transposition, the summing nodes become branching nodes and vice versa.

The number of additions in the initial network is equal to that of ($2, 1$) summing nodes, denoted by N_s . The number of additions in the transposed network is equal to that of ($2, 1$) branching nodes in the initial network, denoted by N_b . The additive complexity in initial and transposed algorithms is identical if and only if $N_s = N_b$, and we show in the following that this property holds for (N -input, N -output) systems.

Proof: For an (N -input, N -output) network, if we connect the N inputs to the N outputs graphically, we get a closed network where every branch coming out from a node should enter into another node. Then the number of outputs of all nodes is equal to that of inputs of all nodes. A ($2, 1$) summing node has two inputs and one output while a ($1, 2$) branching node has one input and two outputs. We get

$$\begin{aligned} Nb + 2Ns &= 2Nb + Ns \\ Ns &= Nb. \end{aligned} \quad (\square)$$

APPENDIX B

SEVERAL SHORT-LENGTH FIR ALGORITHMS

$F(2, 2)$ algorithm with (3 multiplications, 6 additions), $\{\alpha_i\} = \{0, 1, -1\}$:

$$\begin{aligned} a_0 &= x_0 & b_0 &= h_0 \\ a_1 &= x_0 + x_1 & b_1 &= (h_0 + h_1)/2 \\ a_2 &= x_0 - x_1 & b_2 &= (h_0 - h_1)/2 \\ m_i &= a_i b_i; \quad i = 0, 1, 2 \\ y_0 &= m_0 + z^{-2}(m_1 + m_2 - m_0) \\ y_1 &= m_1 - m_2. \end{aligned} \quad (\text{a8})$$

$F(2, 2)$ algorithm with (3 multiplications, 6 additions), $\{\alpha_i\} = \{1, -1, \infty\}$:

$$\begin{aligned} a_0 &= x_0 + x_1 & b_0 &= (h_0 + h_1)/2 \\ a_1 &= x_0 - x_1 & b_1 &= (h_0 + h_1)/2 \\ a_2 &= x_1 & b_2 &= h_1 \\ m_i &= a_i b_i; \quad i = 0, 1, 2 \\ y_0 &= m_0 + m_1 - m_2 + z^{-2}m_2 \\ y_1 &= m_0 - m_1. \end{aligned} \quad (\text{a9})$$

$F(5, 5)$ algorithm with (12 multiplications, 40 additions).

This algorithm is based on the approach in [12]:

$$\begin{aligned}
 c_0 &= x_0 + x_3 & g_0 &= (h_0 + h_3)/2 \\
 c_1 &= x_1 + x_4 & g_1 &= (h_1 + h_4)/2 \\
 c_2 &= x_2 & g_2 &= h_2/2 \\
 c_3 &= x_0 - x_3 & g_3 &= (h_3 - h_0)/2 \\
 c_4 &= x_1 - x_4 & g_4 &= (h_4 - h_1)/2 \\
 c_5 &= x_2 & g_5 &= -h_2/2 \\
 a_0 &= c_0 + c_1 + c_2 & b_0 &= (g_0 + g_1 + g_2)/3 \\
 a_1 &= c_0 - c_2 & b_1 &= g_0 - g_2 \\
 a_2 &= c_1 - c_2 & b_2 &= g_1 - g_2 \\
 a_3 &= a_1 + a_2 & b_3 &= (b_1 + b_2)/3 \\
 a_4 &= c_3 - c_4 + c_5 & b_4 &= (g_3 - g_4 + g_5)/3 \\
 a_5 &= c_3 - c_5 & b_5 &= (-2g_3 - g_4 + g_5)/3 \\
 a_6 &= c_3 + c_4 & b_6 &= (g_3 + 2g_4 + g_5)/3 \\
 a_7 &= c_4 + c_5 & b_7 &= (g_3 - g_4 - 2g_5)/3 \\
 a_8 &= x_0 & b_8 &= h_0 \\
 a_9 &= x_0 & b_9 &= h_1 \\
 a_{10} &= x_1 & b_{10} &= h_0 \\
 a_{11} &= x_4 & b_{11} &= h_4 \\
 m_i &= a_i b_i; & i &= 0, 1, \dots, 11 \\
 u_0 &= m_1 - m_3 \\
 u_1 &= m_2 - m_3 \\
 d_0 &= m_0 + u_0 \\
 d_1 &= m_0 - u_0 - u_1 \\
 d_2 &= m_0 + u_1 \\
 d_3 &= m_4 - m_5 + m_7 \\
 d_4 &= -m_4 + m_6 + m_7 \\
 d_5 &= m_4 + m_5 + m_6 \\
 d_6 &= m_9 + m_{10} \\
 f_0 &= d_2 - d_5 \\
 f_1 &= d_1 - d_4 \\
 f_2 &= d_0 - d_3 \\
 f_3 &= d_2 + d_5 \\
 f_4 &= d_1 + d_4 \\
 f_5 &= d_0 + d_3 \\
 y_0 &= m_8 + z^{-5}f_5 \\
 y_1 &= d_6 + z^{-5}(f_0 - m_8) \\
 y_2 &= f_2 - m_{11} + z^{-5}(f_1 - d_6) \\
 y_3 &= f_3 + z^{-5}m_{11} \\
 y_4 &= f_4.
 \end{aligned}$$

REFERENCES

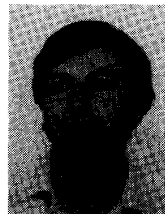
- [1] S. Winograd, "Arithmetic complexity of computations," CBMS-NSF Regional Conf. Series in Applied Mathematics, SIAM Pub. 33, 1980.
- [2] Z. J. Mou and P. Duhamel, "Fast FIR filtering: Algorithms and implementations," *Signal Processing*, Dec. 1987, pp. 377-384.
- [3] M. Vetterli, "Running FIR and IIR filtering using multirate filter banks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, no. 5, pp. 730-738, May 1988.
- [4] H. K. Kwan and M. T. Tsim, "High speed 1-D FIR digital filtering architecture using polynomial convolution," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing* (Dallas, TX), Apr. 1987, pp. 1863-1866.
- [5] P. P. Vaidyanathan and S. K. Mitra, "Polyphase networks, block digital filtering, LPTV systems, and alias-free QMF banks: A unified approach based on pseudocirculants," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, no. 3, pp. 381-391, Mar. 1988.
- [6] P. Duhamel and M. Vetterli, "Improved Fourier and Hartley transform algorithms: Application to cyclic convolution of real data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 35, no. 6, pp. 818-824, June 1988.
- [7] S. Winograd, "Some bilinear forms whose multiplicative complexity depends on the field of constants," *Math. Syst. Theory*, vol. 10, pp. 169-180, Sept. 1977, also in *Number Theory in Digital Signal Processing*, J. H. McClellan and C. M. Rader, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1979.
- [8] R. C. Agarwal and J. W. Cooley, "New algorithms for digital convolution," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, pp. 392-410, Oct. 1977.
- [9] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus, "Real-valued fast Fourier transform algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, no. 6, pp. 849-863, June 1988.
- [10] Z. J. Mou and P. Duhamel, "A unified approach to the fast FIR filtering algorithms," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing* (New York, NY), Apr. 1988, pp. 1914-1917.
- [11] R. E. Blahut, *Fast Algorithms for Digital Signal Processing*. Reading, MA: Addison-Wesley, 1985.
- [12] P. C. Balla, A. Antoniou, and S. D. Morgera, "Higher radix aperiodic convolution algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, no. 1, pp. 60-68, Feb. 1986.
- [13] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1983.



Zhi-Jian Mou (S'87-M'88) was born in China in 1963. He received the B.E. degree in electrical engineering at the University of Science and Technology of China (USTC) in 1984. He received the D.E.A. (Diplôme d'Etudes Approfondies) in 1986 and the Ph.D. degree in 1989 from the University of Paris-Sud.

He was with the French National Telecom Research Center (CNET-Paris B) from 1986 to 1989. He is now with the Department of Electronics, Telecom Paris University. His research interest is

in digital signal processing algorithms/architectures and their implementations.



Pierre Duhamel (M'87-SM'87) was born in France in 1953. He received the Ingenieur degree in electrical engineering from the National Institute for Applied Sciences (INSA), Rennes, France, in 1975, the Dr. Ing. degree in 1978, and the Doctorates Sciences in 1986, both from Sud University of Paris, France.

From 1975 to 1980, he was with Thomson-CSF, Paris, France, where his research interests were in circuit theory and signal processing, including digital filtering and analog fault diagnosis. In 1980, he joined the National Research Center in Telecommunications (CNET), Issy-les-Moulineaux, France, where his activities were first concerned with the design of recursive CCD filters. He is now working on fast Fourier transforms and convolution algorithms, and on the application of similar techniques to adaptive filtering, spectral analysis, and wavelet transforms.

Dr. Duhamel is a member of the DSP Committee and is now serving as an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING.