Pipeline Interleaving and Parallelism in Recursive Digital Filters—Part II: Pipelined Incremental Block Filtering

KESHAB K. PARHI, MEMBER, IEEE, AND DAVID G. MESSERSCHMITT, FELLOW, IEEE

Abstract—**In the companion paper, we proposed high speed pipelined realizations of recursive digital filters of logarithmic complexity with respect to the number of loop pipeline stages using scattered look-ahead and decomposition techniques. In this paper, we address block implementation and fine-grain pipelined block implementation of recursive digital filters.**

Recently, Wu and Cappello proposed a direct form block filter structure for second-order recursive filters of complexity linear in block size. We extend this linear complexity block filter structure to higher order systems, and refer to it as *incremental block filter*. Block implementation of state space recursive digital filters has been known for a long time. The two existing popular block structures are the block-state structure proposed by Barnes and Shinnaka, and the parallel block-state structure proposed by Nikias. However, the multiplication complexity of these structures is proportional to the square of the block size. The block-state update operation in these filter structures is performed based on the *clustered look-ahead computation*, and requires a linear complexity in block size. But, the output computation of the complete block is done all at once and requires a square complexity in block size. In this paper, we introduce a new technique of *incremental output computation* which requires a linear complexity in block size. Based on the clustered look-ahead and incremental output computation approaches, we derive our *incremental block-state* structure for block implementation of state space filters of multiplication complexity linear in block size. The incremental block-state structure is also extended for the multirate recursive filtering case.

Finally, we combine the techniques of scattered look-ahead, clustered look-ahead, decomposition, and incremental output computation to introduce several pipeline stages inside the recursive loop of the block filter. We derive deeply pipelined block filter structures for implementation of direct form and state space form recursive digital filters. The multiplication complexity of these pipelined block filters is linear with respect to the block size, logarithmic with respect to the number of loop pipeline stages, and the complexities due to pipelining and block processing are additive.

I. INTRODUCTION

IN the companion paper [1], we introduced two new techniques referred to as the *scattered look-ahead* and the *decomposition* techniques. Specifically, we showed that we can use these two techniques to overcome the recursive bottleneck, and pipeline direct form and state space form recursive digital filters with a logarithmic increase in hardware with respect to the number of loop pipeline stages.

Concurrent realization of recursive digital filters can also be achieved by the use of "block processing" [2]–[19]. In block realizations, input samples are processed in the form of nonoverlapping blocks to generate nonoverlapping blocks of output samples. The block of multiple inputs is derived from the single serial input by using a serial-to-parallel converter at the input, and the serial output is derived from the block of outputs by using a parallel-to-serial converter at the output. Because of this serial-to-parallel conversion, the multiple-input–multiple-output (MIMO) or the block system operates at a rate L times slower than that of the converter circuits, where L is the block size [18]. Thus, the clock period of each latch in the MIMO system is L times greater than that of the sample period, or equivalently, each *block delay operator* in the MIMO system is *L-slow* [20]–[22]. Hence, for a given technology, we can increase the block size arbitrarily to obtain arbitrarily high sampling rate filter realizations. Due to this *L-slow* block delay operator, the block-state update operation requires updating the state $\boldsymbol{x}(kL + L)$ based on the past state $\boldsymbol{x}(kL)$. Note that in the block-state update operation, the $(L - 1)$ intermediate states $\boldsymbol{x}(kL + 1), \boldsymbol{x}(kL + 2), \cdots$, and $\boldsymbol{x}(kL + L - 1)$ have been missed, unlike in pipelined realizations where each state is computed. This block-state update operation can be achieved by iterating the original recursion $(L - 1)$ times (using the clustered look-ahead approach) to create a single *L-slow* delay operator in the recursive loop.

In the block filter, for a stable system, the poles (eigenvalues) of the block filter move closer to the origin than that for the word-serial system (since the eigenvalues of the block system are Lth power of those of the word-serial system). These block structures offer several advantages over the word-serial realizations. These include i) a linearly proportional increase (decrease) in the sampling rate (iteration period) with increase in block size, and ii) a linearly proportional decrease in the average roundoff noise at the output.

Manuscript received November 3, 1987; revised November 16, 1988. This work was supported in part by grants from the Advanced Research Project Agency monitored by the Naval Electronics Systems Command under Contract N00039-86-R-0365, the National Science Foundation under Contract DCI-85-17339, an IBM Graduate Fellowship, and a University of California Regents Fellowship.

K. K. Parhi is with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455.

D. G. Messerschmitt is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

IEEE Log Number 8928121.

One approach to implement the direct form block filter is to use the concurrent scattered look-ahead algorithms without decomposition (derived in Part I [1] with replicated parallel hardware instead of pipelined hardware). In this block filter, we can compute $x(kL + L)$ using $x(kL)$, $x(kL - L)$, $\cdots$, and $x(kL - NL + L)$. Note that the state $x(kL - L)$ is derived by delaying $x(kL)$ with unit delay, $x(kL - 2L)$ is obtained by delaying $x(kL - L)$ with unit delay, etc. We can compute L outputs in this manner using their delayed values in an independent manner. The decomposition technique is not applicable, since each delay is L-slow. In this scheme, each output requires an $O(NL)$ multiplication complexity, and the block structure requires a total of $O(NL^2)$ multiplication/addition operations, which is square in block size. Thus, the complexity per output sample is linear in block size L and filter order N. This approach to direct form block processing is discussed in [5] and [23]. We refer to this filter as the *parallel direct form block filter*. Furthermore, all existing approaches to block recursive digital filtering also lead to a square multiplication complexity in block size. It is the objective of this paper to derive block recursive digital filter structures with multiplication complexity linear in block size.

Sung and Mitra recently computed L blocks of outputs (with block size L) and first exploited interblock parallelism and then intrablock parallelism to get a linear speedup with respect to the number of processors at the expense of a larger memory space [13]. They were able to get linear speedup by eliminating the *look-ahead complexity* [1] by deriving the *look-ahead computation* term [1] of each block in a sequential or iterative manner while performing the iterations of L blocks at each instant in parallel (this is referred to as interblock parallelism, see [13]).

Wu and Cappello also proposed a new scheme to implement second-order direct form recursive digital filters [14]. Instead of updating the whole block of L outputs, they updated only N outputs (where N is the filter order) and computed the remaining $(L - N)$ outputs in a nonrecursive or sequential manner using these updated N outputs. This process of computing the outputs is referred to as *incremental output computation* and leads to direct form incremental block filters with linear complexity in block size. In this paper, we extend Wu and Cappello's direct form structure from second-order to higher order case (with a slight variation, we update the first N outputs in each block, whereas the structure in [14] updates the last N outputs). This block filter complexity is linear with respect to block size, and the complexity per output sample is independent of block size. Hence, this structure permits us to achieve a linear speedup with respect to the number of processors with reduced memory space while computing a single block of outputs only.

In the incremental output computation in state space block filters, we compute the outputs incrementally in a sequential manner using the nonrecursively computed intermediate states (which were missed in the block-state update process). As an example, for a block size of 20 and an increment size of 5, we compute $y(20k)$ through $y(20k + 4)$ using the state $\boldsymbol{x}(20k)$, then we compute the intermediate state $\boldsymbol{x}(20k + 5)$ nonrecursively (which was missed due to the block-state update process), and compute the incremental outputs $y(20k + 5)$ through $y(20k + 9)$ using this state. Then we compute the state $\boldsymbol{x}(20k + 10)$ and use this to compute $y(20k + 10)$ through $y(20k + 14)$, and finally compute $\boldsymbol{x}(20k + 15)$ and use this to compute the last incremental output $y(20k + 15)$ through $y(20k + 19)$. A family of filter structures can be described with different values of increment size. In particular, the existing block-state filter structure corresponds to the case where the increment size equals the block size. We derive the optimum increment size as a function of the filter order in a way that minimizes the multiplication complexity of the block filter. The incremental block-state filter is also extended to the multirate filtering case.

It is preferable to use pipelining to the maximum possible extent first, since pipelining exploits concurrency with reduced hardware (i.e., with logarithmic increase as opposed to linear as in block processing). This conclusion is clear from Table I, which compares the number of multiply/add operations for a second-order recursive filter, for direct and state space forms, for scattered look-ahead and decomposition based pipelining and incremental block processing approaches, for typical factors of speedup or increase in the sample rate. Note that all the multipliers in the pipelined filter are pipelined by M levels or stages, and these pipelined multipliers require additional area for the latches. In contrast, the multipliers in the block implementation require single stage pipelining. The latch areas in the pipelined implementation cannot be simply neglected, since each binary latch costs about one-third to one-fourth of a binary adder in terms of silicon area. However, if we compare the complexities of the block filter and the pipelined filter, we observe that the pipelined filter is far more attractive for implementation, even after accounting for the latch areas in the pipelined structure. If sufficient speed cannot be generated by pipelining alone, then we can combine pipelining with block processing (i.e., we can get a speedup by a factor of LM using a block size of L, and M pipeline stages inside the recursive loop of the block filter) [16]–[18].

In this paper, we combine pipeline interleaving and incremental block filtering approaches to derive extensively pipelined direct form and state space form incremental block filters by introducing several pipeline stages inside the recursive loop of the incremental block filter. The pipelined block realizations are derived by using the techniques of scattered look-ahead computation (to introduce several loop pipeline stages), decomposition (to obtain logarithmic complexity realization with respect to pipelining), clustered look-ahead computation (for block-state update operation), and incremental output computation (for linear complexity in block size). The total multiplication complexity of our pipelined block filters is linear

TABLE I
COMPLEXITY OF PIPELINED AND BLOCK SECOND-ORDER FILTERS

Speedup	Direct Form		State Space Form	
	Pipelined	Block	Pipelined	Block
1	5	5	9	9
2	7	11	13	15
4	9	25	17	30
8	11	53	21	67
16	13	109	25	142

in block size, logarithmic with respect to the number of loop pipeline stages, and the complexities due to pipelining and block processing are additive. Because of the scattered look-ahead approach, the distance of the poles or the eigenvalues of the fine-grain pipelined block filter from the origin is identical to that of the block filter with a single latch inside the resursive loop. Thus, the stability of these block filters is not affected due to fine-grain loop pipelining.

The organization of the paper is as follows. The incremental block filter structure is derived in Sections II and III, respectively, for direct form and state space form recursive filters. In Section IV and V, we derive extensively pipelined block filter structures for direct form and state space form recursive filters, respectively.

II. Direct Form Block Filters

Consider the Nth-order direct form filter described by

$$y(n) = \sum_{i=1}^{N} a_i y(n-i) + z(n), \tag{2.1a}$$

$$z(n) = \sum_{i=1}^{N} b_i u(n-i). \tag{2.1b}$$

We can transform the above description to an equivalent block description, where we compute L outputs using L inputs for a block size of L. The state update operation in block filters is based on the clustered look-ahead approach [1]. In the clustered look-ahead approach, the output sample $y(n)$ is computed in terms of past N clustered samples $y(n-M)$, $y(n-M-1)$, $\cdots$, and $y(n-M-N+1)$ (i.e., bypassing $(M-1)$ immediate past outputs), and is given by

$$y(n) = \sum_{j=0}^{N-1} \left[\sum_{k=j+1}^{N} a_k r_{j+M-k} \right] y(n-j-M) + \sum_{j=0}^{M-1} r_j z(n-j). \tag{2.2}$$

This relation was derived in Appendix B of the companion paper [1], and is used here in the context of block filtering. Let a block of samples be denoted by

$$\boldsymbol{y}^{(L)}(i) = \left[y(i), y(i+1), \cdots, y(i+L-1) \right]^T. \tag{2.3}$$

In the existing block structures, the block of outputs $\boldsymbol{y}^{(L)}(kL+L)$ are updated based on the past block of outputs $\boldsymbol{y}^{(L)}(kL)$, with a square complexity in block size.

In the incremental block filter, we update only N states recursively using the clustered look-ahead approach, and compute the remaining $(L-N)$ states in a nonrecursive or sequential manner using these N updated states. The schedule of such an incremental block filter is shown in Fig. 1, where the outputs $y(kL+L)$ through $y(kL+L+N-1)$ are updated recursively, and the remaining $(L-N)$ outputs $y(kL+L+N)$ through $y(kL+2L-1)$ are computed in a nonrecursive manner. This incremental output computation is the key in obtaining a linear complexity implementation in block size. If the block size is less than the filter order, the complexity is inherently proportional to square of the block size, but this is not a problem since the block size is not large. Below, we formulate the incremental block filter for the cases where the block size is greater than and less than the filter order, respectively.

Case I—$L \geq N$: We can update N out of the L outputs using (see Appendix A)

$$\boldsymbol{y}^{(N)}(kL+L) = \boldsymbol{A}(L)\, \boldsymbol{y}^{(N)}(kL) + \boldsymbol{B}(L)\, z^{(L)}(kL+N), \tag{2.4a}$$

where $\boldsymbol{y}^{(L)}(i)$ is an $N \times 1$ column vector, $\boldsymbol{A}(L)$ is an $N \times N$ matrix, and $\boldsymbol{B}(L)$ is an $N \times L$ matrix. The elements of $\boldsymbol{A}(L)$ and $\boldsymbol{B}(L)$ are defined by (see Appendix A)

$$\left[\boldsymbol{A}(L)\right]_{ij} = \sum_{l=1}^{j} a_{N-l+1} r_{L-N+i-j+l-1}, \qquad i, j = 1, 2, \cdots, N \tag{2.4b}$$

$$\left[\boldsymbol{B}(L)\right]_{ij} = r_{L-N+i-j}. \tag{2.4c}$$

The $(L-N)$ outputs $y(kL+N)$, $y(kL+N+1)$, $\cdots$, $y(kL+L-1)$ can be computed nonrecursively in a sequential manner using the past outputs and the corresponding inputs.

The multiplication complexity of the above state update implementation (i.e., for updating N outputs or states) is $(LN + N^2 - (N(N+1)/2))$, of which $(LN - (N(N+1)/2))$ is due to the $\boldsymbol{B}(L)$ matrix and N^2 is due to the $\boldsymbol{A}(L)$ matrix. The computation of $z(kL)$, $z(kL+1)$, $\cdots$, $z(kL+L-1)$ requires $(N+1)L$ multiplications, and the computation of the last $(L-N)$ outputs can be done using the past outputs with $N(L-N)$ multiplications. Thus, the total multiplication complexity of the direct form block filter is $[L(3N+1) - (N(N+1)/2)]$, which is linear in block size L. Another block structure [15] has a multiplication complexity $[2LN + (L(L+1)/2)]$, which is square in block size.

Fig. 2(a) shows a second-order direct form word-serial recursive digital filter, and Fig. 2(b) shows the corresponding block filter for a block size of 5. In this structure, the states $y(5k)$ and $y(5k+1)$ are updated each block, and the outputs $y(5k+2)$, $y(5k+3)$, and $y(5k+4)$ are computed incrementally in a nonrecursive or sequential manner. The multiplication complexity of this second-order filter is $(7L-3)$, and is the same as that of the structure proposed in [14].

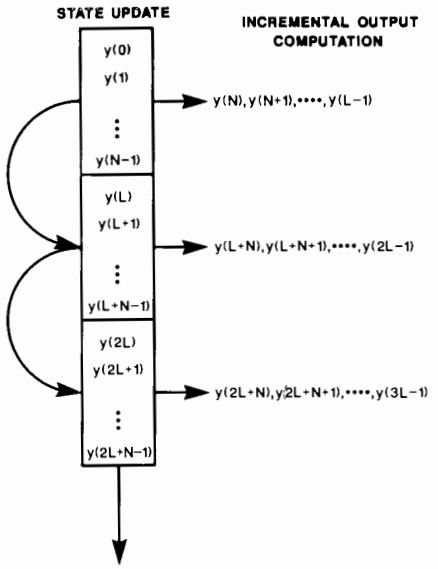
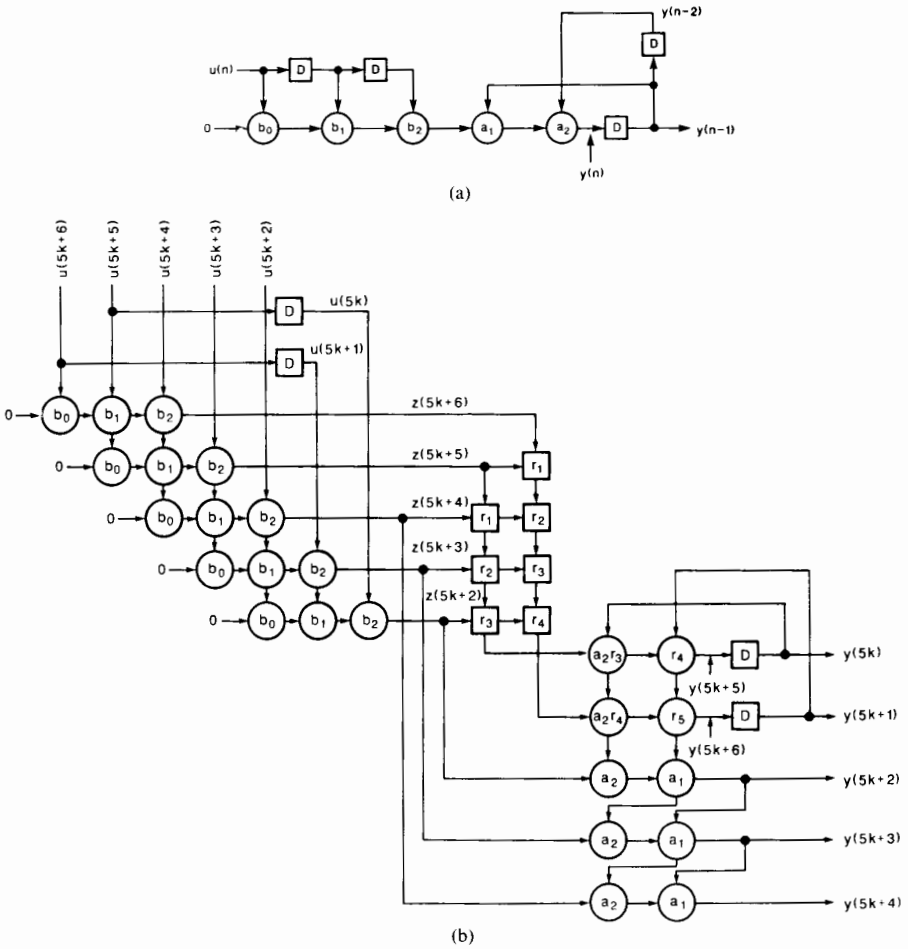


Fig. 1. Incremental output computation in direct form block recursive digital filter.



(a)

(b)

Fig. 2. (a) A second-order word-serial recursive filter. (b) Block implementation of the second-order recursive digital filter for block size of 5.

Case II—$L < N$: For the case when block size L is less than the filter order N, only L states need to be updated, each based on N past states using the clustered look-ahead approach, and the remaining $(N - L)$ states can be derived by delaying the available L states appropriately. For example, if $L = 3$ and $N = 8$, then we can compute the outputs $y(3k + 3)$, $y(3k + 4)$, and $y(3k + 5)$ using $y(3k)$, $y(3k + 1)$, and $y(3k + 2)$ and their delayed samples $y(3k - 1), \cdots, y(3k - 5)$. Note that $y(3k - 1)$ and $y(3k - 4)$ can be obtained by delaying $y(3k + 2)$, $y(3k - 2)$, and $y(3k - 5)$ can be derived by delaying $y(3k + 1)$ and $y(3k - 3)$ can be obtained by delaying $y(3k)$; note that each delay here is 3-slow.

The direct form block filter can be described by (see Appendix A)

$$\boldsymbol{y}^{(L)}(kL + L) = \boldsymbol{A}(L)\,\boldsymbol{y}^{(N)}(kL + L - N) + \boldsymbol{B}(L)\,\boldsymbol{z}^{(L)}(kL + L), \tag{2.5a}$$

where

$$\left[\boldsymbol{A}(L)\right]_{ij} = \sum_{l=1}^{j} a_{N-l+1} r_{i-j+l-1}, \quad i = 1, 2, \cdots, L; \ \ j = 1, 2, \cdots, N \tag{2.5b}$$

$$\left[\boldsymbol{B}(L)\right]_{ij} = r_{i-j} \tag{2.5c}$$

and $\boldsymbol{A}(\boldsymbol{L})$ is $L \times N$, $\boldsymbol{B}(\boldsymbol{L})$ is $L \times L$.

The complexity required for computing $\boldsymbol{z}^{(L)}(kL + L)$ is $L(N + 1)$, for computing $\boldsymbol{B}(L)\,\boldsymbol{z}^{(L)}(kL + L)$ is $L(L - 1)/2$, and that due to $\boldsymbol{A}(L)$ is NL multiplications. Thus, the total multiplication complexity is $(2N + (L + 1/2))L$, which is square in block size. Fortunately, for this case, L is small.

III. State Space Block Digital Filters

Consider the state space recursive filter described by

$$\boldsymbol{x}(n + 1) = \boldsymbol{A}\boldsymbol{x}(n) + \boldsymbol{b}u(n) \tag{3.1a}$$

$$y(n) = \boldsymbol{c}^T\boldsymbol{x}(n) + du(n), \tag{3.1b}$$

where the state $\boldsymbol{x}(n)$ is $N \times 1$, the state update matrix $\boldsymbol{A}$ is $N \times N$, $\boldsymbol{b}$ and $\boldsymbol{c}$ are $N \times 1$ and d, input sample $u(n)$ and output sample $y(n)$ are scalars, and N is the order of the filter. Fig. 3(a) shows a block diagram corresponding to (3.1). For the purposes of this paper, the state update matrix is assumed to be quasi-diagonal (i.e., all real poles of the system are of multiplicity less than or equal to two, and all complex poles are of multiplicity unity), and N_1 is assumed to represent the number of real poles of unity multiplicity. Thus, the quasi-diagonal state update matrix has N_1 blocks of dimension unity, and $(N - N_1)/2$ blocks of dimension 2×2. The total number of nonzero elements in $\boldsymbol{A}$ is $(2N - N_1)$, which is linear in filter order N.

Unlike the direct form representation, the state space representation consists of a state update portion, and an output computation portion. Similarly, the block-state space filter representation also consists of a block-state update portion and a block output computation portion. The block-state update operation has a linear complexity in block size. The block output computation in the existing block-state and parallel block-state structures leads to a square complexity in block size. In this section, we review the existing block-state [7] and the parallel block-state filter structures [11]. Then we present the incremental output computation approach, and using this we derive the incremental block-state filter of complexity linear in block size. The average quantization noise at the output of the incremental block-state structure is the same as that of the block-state structure and less than that of the parallel block-state structure. We also extend the incremental block-state filter structure for the case of multirate recursive filtering.

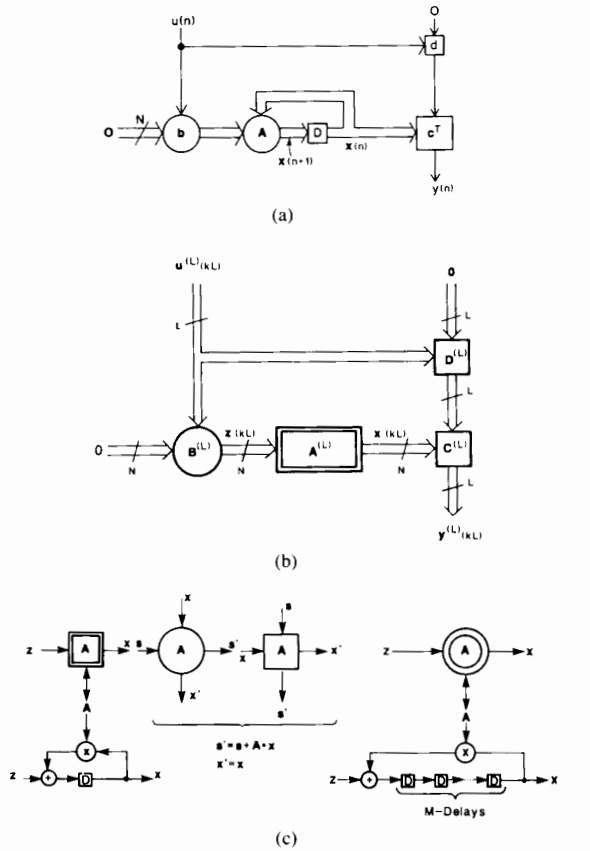


Fig. 3. (a) A word-serial state space recursive filter. (b) Block-state space recursive digital filter. (c) Definition of processing elements.

A. Block-State Implementation

In the block structure with block size L, each implementable latch is *L-slow*, i.e., the clock rate of the latch in the block filter is L times slower than the input sample rate. Hence, the state of the system needs to be updated block by block, i.e., the state $\boldsymbol{x}(kL + L)$ is updated using

$x(kL)$, and the $(L-1)$ intermediate states $x(kL+1)$, $\cdots$, $x(kL+L-1)$ are missed in the block-state update process. The state update representation of the block-state structure [7] can be derived by iterating the single-input–single-output (SISO) state update equation (3.1a) $(L-1)$ times, and is given by

$$x\big((k+1)L\big) = A^{(L)}x(kL) + B^{(L)}u^{(L)}(kL) \quad (3.2a)$$

where

$$A^{(L)} = A^L \quad (3.2b)$$

$$B^{(L)} = (A^{L-1}b \;\; A^{L-2}b \;\cdots\; b) \quad (3.2c)$$

$$u^{(L)}(n) = \big[u(n) \;\; u(n+1) \;\cdots\; u(n+L-1)\big]^T, \quad (3.2d)$$

and $A^{(L)}$ is $N \times N$, $B^{(L)}$ is $N \times L$, $u^{(L)}(kL)$ is $L \times 1$.

In the block-state structure, the block of outputs $y(kL)$, $y(kL+1)$, $\cdots$, and $y(kL+L-1)$ are computed based on the single state $x(kL)$ and the corresponding inputs. This is based on the assumption that the $(L-1)$ intermediate states are not available (since they are missed due to the block-state update process). The block output equation is described by

$$y^{(L)}(kL) = C^{(L)}x(kL) + D^{(L)}u(kL) \quad (3.2e)$$

where

$$C^{(L)} = (c^T \;\; c^TA \;\cdots\; c^TA^{(L-1)})^T \quad (3.2f)$$

$$[D^{(L)}]_{ij} = \begin{cases} 0 & i < j \\ d & i = j \\ c^TA^{(i-j-1)}b & i > j \end{cases} \quad (3.2g)$$

and

$$y^{(L)}(n) = \big[y(n) \;\; y(n+1) \;\cdots\; y(n+L-1)\big]^T. \quad (3.2h)$$

In the output equation, $C^{(L)}$ is $L \times N$, $D^{(L)}$ is $L \times L$ and lower triangular, and $y^{(L)}$ is $L \times 1$. The block diagram of the block-state filter is shown in Fig. 3(b). The blocks marked $B^{(L)}$, $C^{(L)}$, $D^{(L)}$ represent matrix vector multipliers, and the block marked $A^{(L)}$ represents the state update network (see Fig. 3(c) for definition of the processing elements). Fig. 4 shows the block-state implementation of a first-order recursive filter for a block size of three. A partial schedule of the block-state filter is shown in Fig. 5. Thus, in the block-state filter, we use $x(0)$ to compute the block of outputs $y^{(L)}(0)$, and to update $x(L)$. In the next cycle, $x(L)$ is used to compute the next block of outputs $y^{(L)}(L)$, and to update the state $x(2L)$, and the schedule follows in this manner.

For the case of a quasi-diagonal state update matrix, the complexity (in terms of multiplications) of the state update representation is $(NL + 2N - N_1)$, and the output representation is $(NL + (L(L+1)/2))$ where N, L, and N_1, respectively, represent the order of the system, block size, and number of real poles with unity multiplicity.

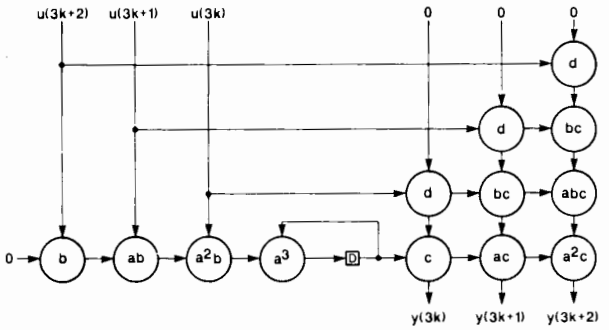


Fig. 4. Block-state implementation of a first-order state space recursive filter for block size of 3.

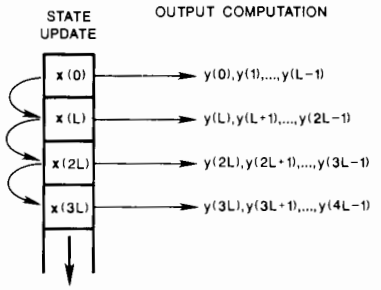


Fig. 5. Partial schedule of a block-state space implementation.

Hence, the total multiplication complexity of the block-state structure (C_b) is given by

$$C_b = 2N(L+1) + \frac{L(L+1)}{2} - N_1, \quad (3.3)$$

and is $O(L^2)$ for a block size of L. The asymptotic complexity per output sample is $(2N + (L + 1/2))$.

B. Parallel Block-State Implementation

In parallel block-state structure [11], the *system state* $x(n)$ is first decomposed into L *section states* $x_1(n)$, $x_2(n)$, $\cdots$, $x_L(n)$, which are related by

$$x(n) = [A^{L-1} \;\; A^{L-2} \;\cdots\; 1] \begin{bmatrix} x_1(n) \\ x_2(n) \\ \vdots \\ x_L(n) \end{bmatrix}. \quad (3.4a)$$

Since each implementable latch of the block structure is *L-slow*, the states in each section are updated block by block, i.e., in section i, the state $x_i(kL+L)$ is computed based on the same $x_i(kL)$. Using each section state, L *partial outputs* are computed and the L *system outputs* are obtained by adding the corresponding L partial outputs. Substituting (3.4a) in (3.2a) and (3.2e), the state update and output representation of the parallel block-state struc-

ture can be derived to be [11]

$$\boldsymbol{x}_i(kL + L) = \boldsymbol{A}^L \boldsymbol{x}_i(kL) + \boldsymbol{b}u(kL + i - 1), \quad i = 1, 2, \cdots, L \tag{3.4b}$$

$$\begin{bmatrix} y(kL) \\ y(kL+1) \\ \vdots \\ y(kL+L-1) \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}^T\boldsymbol{A}^{L-1} & \boldsymbol{c}^T\boldsymbol{A}^{L-2} & \cdots & \boldsymbol{c}^T \\ \boldsymbol{c}^T\boldsymbol{A}^{L} & \boldsymbol{c}^T\boldsymbol{A}^{L-1} & \cdots & \boldsymbol{c}^T\boldsymbol{A} \\ \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{c}^T\boldsymbol{A}^{2L-2} & \boldsymbol{c}^T\boldsymbol{A}^{2L-3} & \cdots & \boldsymbol{c}^T\boldsymbol{A}^{L-1} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x}_1(kL) \\ \boldsymbol{x}_2(kL) \\ \vdots \\ \boldsymbol{x}_L(kL) \end{bmatrix} + \begin{bmatrix} d & 0 & \cdots & 0 \\ \boldsymbol{c}^T\boldsymbol{b} & d & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{c}^T\boldsymbol{A}^{L-2}\boldsymbol{b} & \boldsymbol{c}^T\boldsymbol{A}^{L-3}\boldsymbol{b} & \cdots & d \end{bmatrix} \cdot \begin{bmatrix} u(kL) \\ u(kL+1) \\ \vdots \\ u(kL+L-1) \end{bmatrix}. \tag{3.4c}$$

For the case of a quasi-diagonal state update matrix, the state update complexity is $L(3N - N_1)$ and the output computation complexity is $(NL^2 + (L(L+1)/2))$, leading to a total multiplication complexity of

$$C_p = NL(L+3) + \frac{L(L+1)}{2} - N_1 L. \tag{3.5}$$

The asymptotic complexity of the parallel block-state structure is $O(NL^2)$, which is N times higher than that of the block-state structure. The complexity per output sample is $(N(L+3) + (L + 1/2) - N_1)$. The decomposition of the system state leads to this higher complexity. (Note that in [11], the author represented the multiplication complexity of the parallel block-state structure to be much less than that of the block-state structure. The error resulted from not including a factor of L in the complexity expression.)

C. Incremental Block-State Implementation

In the incremental block-state structure, the state $\boldsymbol{x}(kL + L)$ is updated based on the state $\boldsymbol{x}(kL)$ as in the block-state case. However, the output computation, which is nonrecursive, is done in a different way. Rather than calculating the block of outputs in terms of the updated state every L samples, we first calculate the intermediate states (which were missed due to the block-state update process) every I samples nonrecursively (where I is the increment size), and then calculate the outputs incrementally in a sequential manner using these intermediate states. It is this novel output computation which leads to an $O(L)$ complexity in the incremental block-state structure for a block size of L.

In the incremental block-state structure with increment I, I outputs $y(kL + pI), y(kL + pI + 1), \cdots, y(kL + pI + I - 1)$ are computed using the state $\boldsymbol{x}(kL + pI)$ and the inputs $u(kL + pI), u(kL + pI + 1), \cdots, u(kL + pI + I - 1)$, respectively, and the intermediate state $\boldsymbol{x}(kL + pI + I)$ is nonrecursively computed using $\boldsymbol{x}(kL + pI)$ for computation of the next I outputs. Thus, using $\boldsymbol{x}(kL)$, we can compute $y(kL), \cdots, y(kL + I - 1)$ and can nonrecursively compute $\boldsymbol{x}(kL + I)$ for computation of the next I outputs. The size of the increment I is chosen to minimize the multiplication complexity. Let the block size L correspond to $(PI + Q)$, where I is the increment, and P and Q, respectively, represent the quotient and remainder of L/I. Then the size of the last increment is $(I + Q)$, i.e., the computation of the last $(I + Q)$ outputs is performed based on the state $\boldsymbol{x}(kL + PI - I)$.

The state update representation of the *incremental block-state structure* is identical to that of the block-state structure and is described by (3.2a). The computation of I outputs $y(kL + pI), \cdots, y(kL + pI + I - 1)$ and the intermediate state $\boldsymbol{x}(kL + pI + I)$ is performed using

$$\begin{bmatrix} \boldsymbol{y}^{(I)}(kL + pI) \\ \boldsymbol{x}(kL + pI + I) \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}^{(I)} & \boldsymbol{D}^{(I)} \\ \boldsymbol{A}^{(I)} & \boldsymbol{B}^{(I)} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(kL + pI) \\ \boldsymbol{u}^{(I)}(kL + pI) \end{bmatrix}, \quad p = 0, 1, \cdots, P - 2. \tag{3.6a}$$

The computation of the last $(I + Q)$ outputs is carried out using

$$\boldsymbol{y}^{(I+Q)}(kL + pI - I) = \boldsymbol{C}^{(I+Q)}\boldsymbol{x}(kL + PI - I) + \boldsymbol{D}^{(I+Q)}\boldsymbol{u}^{(I+Q)}(kL + PI - I), \tag{3.6b}$$

where the meaning of the symbols stand as defined in (3.2). A block diagram of the incremental block-state structure is shown in Fig. 6, and its partial schedule is shown in Fig. 7 for $L = 4I$. Fig. 8 shows the incremental block-state implementation of a first-order recursive digital filter for a block size of four and increment size of two.

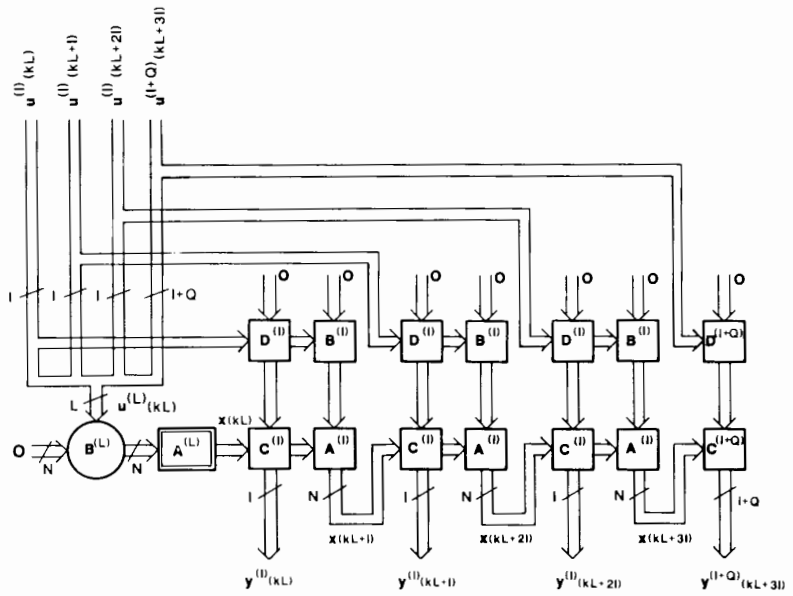


Fig. 6. Incremental block-state implementation of a state space filter ($L = 4I + Q$).

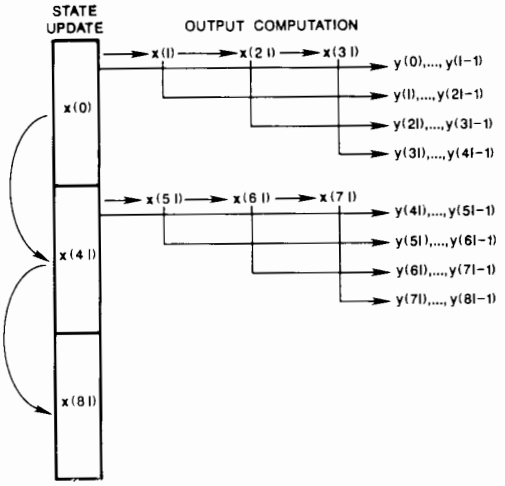


Fig. 7. A partial schedule of an incremental block-state digital filter ($L = 4I$).

A family of block structures can be described by the incremental block-state structure for different values of the increment I. A value of $I \geq L/2$ (i.e., $P = 1$) leads to the block-state structure described in Section III-A. An optimum value of the increment I can be derived to minimize the multiplication complexity of the incremental block-state structure. The optimum increment will depend upon the exact custom VLSI implementation architecture, or scheduling in case of software-programmable parallel-processor realization. For example, if it is not possible to exploit the lower triangular nature of the $\boldsymbol{D}$ matrix, then the complexity of the full matrix will need to be accounted for. In this paper, we assume that it is possible to exploit the lower triangular nature of $\boldsymbol{D}$ matrix, and consider the case of the quasi-diagonal state update matrix. A similar analysis can be carried out for all other cases.

In an incremental block-state realization, the complexity of the state update equation is $(2N - N_1 + NL)$, and is independent of the increment I. The output computation complexity for first $(P - 1)$ increments (of size I each) is $(P - 1)(2IN + 2N - N_1 + (I(I + 1)/2))$, and the last increment (of size $(I + Q)$) is $((I + Q)N + ((I + Q)(I + Q + 1)/2))$. The total complexity of the incremental block-state structure is given by

$$C_i = 2N(P + L) + \frac{L(I + 1)}{2} + \frac{Q(I + Q)}{2} + NI(P - 1) - N_1 P, \tag{3.7}$$

where P and Q are, respectively, quotient and remainder of L/I. In the asymptotic case, we need to minimize the complexity per output in the output equation which is given by

$$C'_{io} = 2N + \frac{2N - N_1}{I} + \frac{I + 1}{2}, \tag{3.8}$$

and is minimized for

$$I = [\![\sqrt{2(2N - N_1)}]\!], \tag{3.9}$$

where $[\![x]\!]$ represents rounding x to the nearest integer. At the optimum increment value, the multiplication complexity associated with the computation of the intermediate state and that of the I outputs are approximately the same. In the asymptotic case (i.e., very large L and $Q = 0$), the complexity of the incremental block-state struc-

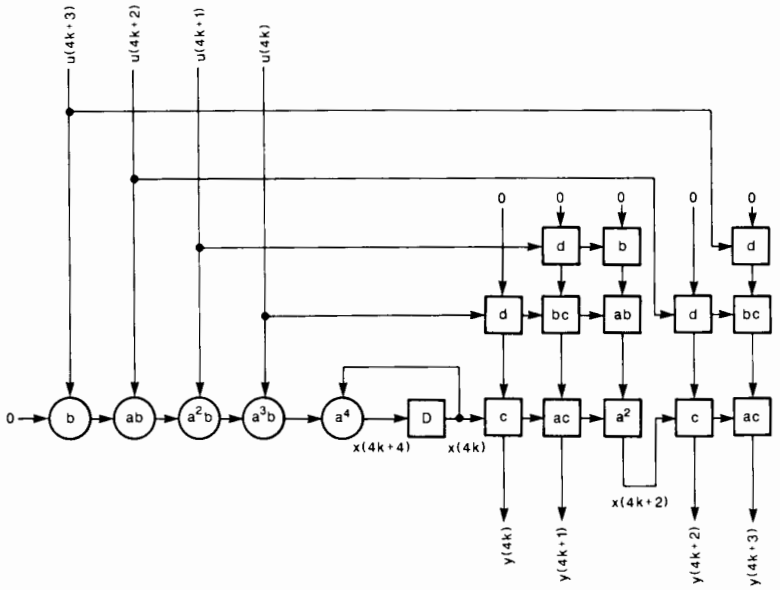


Fig. 8. Incremental block-state implementation of a first-order recursive system for block size of four, and an increment size of two. Each latch in the implementation is 4-*slow*.

ture with optimum I (for a quasi-diagonal state update matrix) is given by

$$C_i = L\left[3N + \frac{2(2N - N_1) + [\![\sqrt{2(2N - N_1)}]\!]^2}{2[\![\sqrt{2(2N - N_1)}]\!]} + \frac{1}{2}\right] - N[\![\sqrt{2(2N - N_1)}]\!], \quad (3.10)$$

and is linear in block size. For the special case of the incremental block-state structure with unity increments ($I = 1$), the complexity per output sample is ($5N - N_1 + 1$).

Table II summarizes the asymptotic complexity (in terms of number of multiplications) per output sample for various structures for the case of a quasi-diagonal state update matrix. Table III summarizes the multiplication complexity (Comp) in terms of number of multiplications for typical filter orders (N) and block sizes (L) for block-state (BS), parallel block-state (PBS), and incremental block-state (IBS) structure with optimum increment (I) for $N_1 = 0$.

For large block sizes, it is possible to achieve linear speedup (in execution time) in case of a software programmable parallel processor implementation (using the incremental block-state recursive digital filter) as the number of processors increases. In the case of a custom VLSI realization, a linear increase in sampling rate can be achieved by using block processing at the expense of a linearly proportional increase in hardware.

Since the recursive state update for block-state and incremental block-state structures are identical, the average roundoff noise at the output is the same for these two structures (under the assumption that the roundoff is performed at the output of state variables and at the system outputs, and the noise sources are white stationary with zero mean and statistically independent). The roundoff noise for the incremental block filter is derived in Appendix B. It has been shown in [11] that the average roundoff noise at the output of the parallel block-state structure is greater than that of the block-state structure (and hence than the incremental block-state structure).

TABLE II
ASYMPTOTIC IMPLEMENTATION COMPLEXITY PER OUTPUT SAMPLE

Implementation	Number of Multiplications per Sample
Direct	$2N + 1$
SISO State Space	$4N - N_1 + 1$
Block-State	$2N + \frac{(L+1)}{2}$
Parallel Block-State	$3N + NL + \frac{(L+1)}{2} - N_1$
Incremental Block-State	$3N + \frac{2(2N-N_1)+[[\sqrt{2(2N-N_1)}]]^2}{2[[\sqrt{2(2N-N_1)}]]} + \frac{1}{2}$

TABLE III
COMPLEXITY COMPARISON FOR DIFFERENT BLOCK STRUCTURES

N	L = 20				L = 40				L = 60			
	BS	PBS	IBS		BS	PBS	IBS		BS	PBS	IBS	
	Comp	Comp	Comp	I	Comp	Comp	Comp	I	Comp	Comp	Comp	I
2	294	1130	179	3	984	4260	366	3	2074	9390	554	3
4	378	2050	310	6	1148	7700	644	4	2318	16950	974	4
6	462	2970	425	7	1312	11140	892	6	2562	24510	1372	7
8	546	3890	527	7	1476	14580	1134	7	2806	32070	1742	7

D. Multirate Incremental Block-State Filter

In this subsection, we study the block realization of multirate recursive digital filters via the incremental block-

state filter. Multirate block recursive filtering based on the block-state structure has been studied in [19].

Any multirate system with sampling rate alteration by a factor of J/K can be realized using a 1-to-J interpolator at the input, and a K-to-1 decimator at the output (where the greatest common divisor of J and K is unity) [see Fig. 9(a) and (b)]. Thus, only one out of J inputs is nonzero and only one out of K outputs needs to be computed. We assume each Jth input to be nonzero and the 1st (of each of the K) outputs to be computed to obtain minimum complexity. Thus, for an input (output) block size of KL (JL), KL nonzero inputs $u(pKJL + J - 1)$, $u(pKJL + 2J - 1)$, $\cdots$, $u(pKJL + KLJ - 1)$ are processed to generate JL nonzero outputs $y(pKJL)$, $y(pKJL + K)$, $\cdots$, $y(pKJL + (JL - 1)K)$ [see Fig. 9(c)]. In the incremental block-state structure with an input (output) increment KI (JI), JI outputs $y(pKJL + qKJI)$, $y(pKJL + qKJI + K)$, $\cdots$, $y(pKJL + qKJI + (JI - 1)K)$ are computed using the state $\boldsymbol{x}(pKJL + qKJI)$, and KI nonzero inputs $u(pKJL + qKJI + J - 1)$, $u(pKJL + qKJI + 2J - 1)$, $\cdots$, $u(pKJL + qKJI + KIJ - 1)$. The state $\boldsymbol{x}(pKJL + (q + 1)KJI)$ is then computed nonrecursively using the state $\boldsymbol{x}(pKJL + qKJI)$ and the KI nonzero inputs (to be used for the computation of the next output increment). The size of the first $(P - 1)$ output increments is JI, and that of the last is $J(I + Q)$ where P and Q are, respectively, the quotient and remainder of L/I.

The state update equation of the incremental block-state recursive multirate filter is described by

$$\boldsymbol{x}((p + 1)KJL) = \boldsymbol{A}^{(KJL)}\boldsymbol{x}(pKJL) + \boldsymbol{B}^{(KJL)}\boldsymbol{u}^{(KJL)}(pKJL). \quad (3.11)$$

and has a multiplication complexity of $(2N - N_1 + KLN)$ (since the number of nonzero inputs is only KL).

The computation of the above J outputs $y(pKJL + qKJI + rKJ)$, $y(pKJL + qKJI + rKJ + K)$, $\cdots$, $y(pKJL + qKJI + rKJ + KJ - K)$ of the $(q + 1)$th increment can be described by

$$\begin{bmatrix} y(pKJL + qKJI + rKJ) \\ y(pKJL + qKJI + rKJ + K) \\ \vdots \\ y(pKJL + qKJI + rKJ + KJ - K) \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{c}^T\boldsymbol{A}^{rKJ} \\ \boldsymbol{c}^T\boldsymbol{A}^{rKJ+K} \\ \vdots \\ \boldsymbol{c}^T\boldsymbol{A}^{rKJ+KJ-K} \end{bmatrix} \boldsymbol{x}(pKJL + qKJI) + \begin{bmatrix} \boldsymbol{c}^T\boldsymbol{A}^{(rK-1)J}\boldsymbol{b} & \boldsymbol{c}^T\boldsymbol{A}^{(rK-2)J}\boldsymbol{b} & \cdots \\ \boldsymbol{c}^T\boldsymbol{A}^{(rK-1)J+K}\boldsymbol{b} & \boldsymbol{c}^T\boldsymbol{A}^{(rK-2)J+K}\boldsymbol{b} & \cdots \\ \vdots & \vdots & \cdots \\ \boldsymbol{c}^T\boldsymbol{A}^{(rK-1)J+(J-1)K} & \boldsymbol{c}^T\boldsymbol{A}^{(rK-2)J+(J-1)K} & \cdots \end{bmatrix}$$

$$\cdot \begin{bmatrix} u(pKJL + qKJI + J - 1) \\ u(pKJL + qKJI + 2J - 1) \\ \vdots \\ u(pKJL + qKJI + rKJ + KJ - 1) \end{bmatrix}$$

$$q = 0, 1, \cdots, (P - 2); \qquad r = 0, 1, \cdots, (I - 1). \quad (3.12a)$$

The complexity associated with the computation of above J outputs is $(JN + rKJ + \Delta)$ multiplications, where

$$\Delta = \left\lfloor \frac{1}{J} \right\rfloor + \left\lfloor \frac{K + 1}{J} \right\rfloor + \cdots + \left\lfloor \frac{(J - 1)K + 1}{J} \right\rfloor, \quad (3.12b)$$

and $\lfloor x \rfloor$ represents the floor function of x. The complexity associated with computation of JI outputs (i.e., for r = 0-to-$(I - 1)$) is $(JIN + I\Delta + (JKI(I - 1)/2))$ multiplications. A partial schedule of a multirate incremental block recursive digital filter is shown in Fig. 10 (for $L = 4I$).

The nonrecursive state update equation for the $(q + 1)$th increment is described by

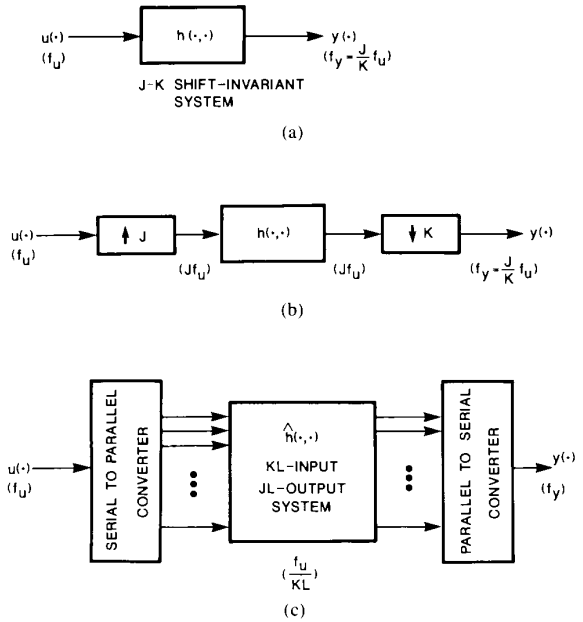


Fig. 9. (a) A multirate recursive digital filter. (b) An equivalent representation of Fig. 9(a). (c) Block implementation of the multirate recursive digital filter.

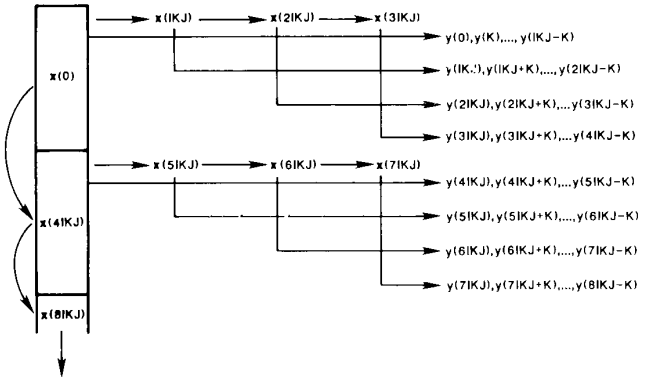


Fig. 10. A partial schedule of an incremental block-state multirate recursive digital filter ($L = 4I$).

$$\begin{aligned} \boldsymbol{x}(pKJL + (q+1)KJI) &= \boldsymbol{A}^{(KJI)}\boldsymbol{x}(pKJL + qKJI) \\ &\quad + \boldsymbol{B}^{(KJI)}\boldsymbol{u}^{(KJI)}(pKJL + qKJI) \end{aligned} \tag{3.13}$$

and leads to a complexity of $(2N - N_1 + KIN)$ multiplications (since only KI inputs are nonzero). Thus, the complexity associated with each (except last) increment (of input size KI and output size JI) is given by

$$JIC'_{io} = I\Delta + \frac{JKI(I-1)}{2} + JIN + (2N - N_1) + KIN, \tag{3.14}$$

where C'_{io} represents the complexity per output sample in the output equation of the incremental block structure. The above complexity is minimized for

$$I = \left[\!\left[\sqrt{\frac{2(2N - N_1)}{KJ}} \right]\!\right]. \tag{3.15}$$

With the above optimum increment, the asymptotic complexity per output sample (in terms of number of multiplications) is approximately given by

$$C_{io} = \frac{2KN}{J} + N + \frac{KJ\left[\!\left[\sqrt{2(2N - N_1)/KJ} \right]\!\right]^2 + 2(2N - N_1)}{2J\left[\!\left[\sqrt{2(2N - N_1)/KJ} \right]\!\right]} - \frac{K}{2} + \frac{\Delta}{J}, \tag{3.16}$$

and is independent of block size. Note that the asymptotic complexity per output sample of the block-state recursive filter in [19] is $[(KN/J) + N + (KL/2) - (K/2) + (\Delta/J)]$, which is linearly proportional to block size, and much larger than that of the incremental block-state structure presented in this paper.

Interpolation and decimation by integer factors are special cases of the general multirate filtering case. A sampling rate increase (interpolation) by factor J can be obtained with unity K, and a sampling rate decrease (decimation) by factor K can be obtained with unity J. For both these cases, the value of Δ is unity.

IV. Direct Form Pipelined Incremental Block Recursive Filters

We can get a speedup by a factor of LM by using a block size L and M stages of pipeline inside the recursive loop [16]-[18]. The pipelined block-state update operations, for the cases when the block size (L) is greater than the filter order (N) and less than the filter order, need to be done differently. We assume M to be a power of 2 to exploit the decomposition of the nonrecursive portion. We first consider the case $L \geq N$ and then the case $L < N$.

Case I—$L \geq N$: The direct form block filter is described by

$$\boldsymbol{y}^{(N)}(kL + L) = \boldsymbol{A}(L)\,\boldsymbol{y}^{(N)}(kL) + \boldsymbol{B}(L)\,\boldsymbol{z}^{(L)}(kL + N). \quad (4.1)$$

In a pipelined block realization with M loop pipeline stages, we need to update $\boldsymbol{y}^{(N)}(kL + ML)$ using the state $\boldsymbol{y}^{(N)}(kL)$. We can derive a pipelined block-state update realization by iterating (4.1) by $(M - 1)$ times to be given by

$$\boldsymbol{y}^{(N)}(kL + ML) = \boldsymbol{A}^M(L)\,\boldsymbol{y}^{(N)}(kL) + \sum_{i=0}^{M-1} \boldsymbol{A}^i(L)\,\boldsymbol{z}_0\big((kL + M - i - 1)L + N\big) \quad (4.2a)$$

where

$$\boldsymbol{z}_0(m) = \boldsymbol{B}(L)\,\boldsymbol{z}^{(L)}(m), \quad (4.2b)$$

and the elements of $\boldsymbol{A}^M(L)$ are derived in Appendix C. The representation of (4.2) can be rewritten (using the decomposition technique) as

$$\boldsymbol{z}_{i+1}(m) = \boldsymbol{z}_i(m) + \boldsymbol{A}^2\boldsymbol{z}_i(m - 2^iL), \quad i = 0, 1, \cdots, (\log_2 M - 1) \quad (4.3a)$$

$$\boldsymbol{y}^{(N)}(kL + ML) = \boldsymbol{A}^M(L)\,\boldsymbol{y}^{(N)}(kL) + \boldsymbol{z}_{\log_2 M}\big((k + M - 1)L + N\big). \quad (4.3b)$$

The above pipelined block-state update operation of (4.2) can also be derived alternatively starting from the block filter representation. We can use a block size of ML in (2.4a) to get

$$\boldsymbol{y}^{(N)}(kL + ML) = \boldsymbol{A}(ML)\,\boldsymbol{y}^{(N)}(kL) + \boldsymbol{B}(ML)\,\boldsymbol{z}^{(ML)}(kL + N), \quad (4.4)$$

which reduces to (4.2) after using (A3.1) and (A3.3).

In the pipelined block implementation, $\boldsymbol{z}^{(L)}((k + M - 1)L + N)$ is computed using the L inputs, and is successively delayed to obtain $\boldsymbol{z}^{(L)}((k + i)L + N)$ for $i = (M - 2)$ through 0. This computation requires $L(N + 1)$ multiplication operations. The computation of $\boldsymbol{z}_0(m)$ requires $(NL - (N(N + 1)/2))$ multiplication operations. The state update implementation of (4.3) requires $N^2(\log_2 M + 1)$ multiplication operations. The final $(L - N)$ outputs are computed nonrecursively in an incremental manner using $N(L - N)$ multiplication operations. Thus, the total multiplication complexity of the pipelined direct form filter is $[L(3N + 1) + N^2 \log_2 M - (N(N + 1)/2)]$, which is linear in block size, logarithmic in loop pipeline stages, and pipelining and block processing complexities are additive. Fig. 11 shows the implementation of a second-order direct form recursive digital filter for a block size of 5, and 4 pipelining stages inside the recursive loop using the decomposition, and incremental output computation techniques.

Case II—$L < N$: For this case, we need to compute (as well as update) only L states, and the $(N - L)$ states can be derived from the L available states. The L outputs $\boldsymbol{y}^{(L)}(kL + ML)$ could be updated using the states $\boldsymbol{y}^{(N)}(kL - (N - L))$ using the clustered look-ahead approach. In this implementation, each of the states $y(kL - (N - L))$ through $y(kL - 1)$ can be derived by delaying the L available states $y(kL)$, $y(kL + 1)$, $\cdots$, and $y(kL + L - 1)$. However, such a realization will contain a single isolated loop delay operator, and hence the decomposition technique cannot be exploited, thus leading to a linear complexity in M. Instead, we can obtain another realization in which all loops contain M delay operators, so that we can exploit the decomposition technique to get a logarithmic complexity with respect to M. In this new pipelined block realization, we express $\boldsymbol{y}^{(L)}(kL + ML)$ in terms of $\boldsymbol{y}^{(L)}(kL)$, $\boldsymbol{y}^{(L)}(kL - ML)$, $\cdots$, and $\boldsymbol{y}^{(L)}(kL - RML)$, where R is $\lfloor N/L \rfloor$. As a special case when block size L is unity, this realization corresponds to the pipelined realization of Section II (case II).

Let $N = RL + S$, where R and S, respectively, correspond to the quotient and remainder of N/L. In the block filter in (2.5), we update the block of states $\boldsymbol{y}^{(L)}(kL + L)$ using $\boldsymbol{y}^{(L)}(kL)$, $\boldsymbol{y}^{(L)}(kL - L)$, $\cdots$, and $\boldsymbol{y}^{(L)}(kL - RL)$. The block filter in (2.5) can be rewritten as

$$\boldsymbol{y}^{(L)}(kL + L) = \sum_{i=0}^{R} \boldsymbol{Q}_{i+1}(1)\boldsymbol{y}^{(L)}\big((k - i)L\big) + \boldsymbol{B}\boldsymbol{z}^{(L)}(kL + L) \quad (4.5a)$$

where

$$[0 \,|\, \boldsymbol{A}(L)] = [\boldsymbol{Q}_{R+1}(1) \,|\, \boldsymbol{Q}_R(1) \,|\, \cdots \,|\, \boldsymbol{Q}_1(1)], \quad (4.5b)$$

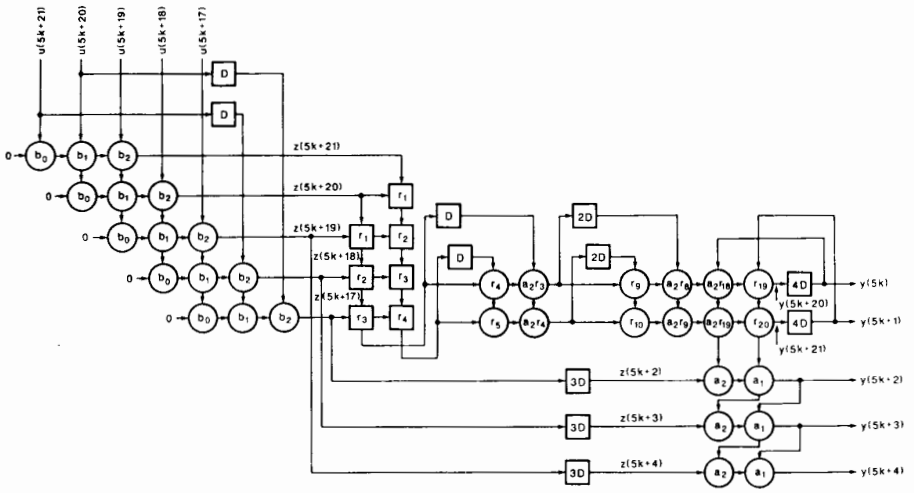


Fig. 11. Block implementation of a second-order direct form recursive digital filter for block size of 5 and 4 loop pipelining stages obtained using the decomposition technique. Each latch in the implementation is 5-*slow*.

and the explicit dependence of the $\boldsymbol{Q}_i$ and $\boldsymbol{B}$ matrices on L has been omitted for simplicity, and $\boldsymbol{Q}_i(M)$'s and $\boldsymbol{B}$ are $L \times L$, and 0 is $L \times (L - S)$. Since only the last S states of $\boldsymbol{y}^{(L)}((k - R)L)$ are needed, the first $(L - S)$ columns of $\boldsymbol{Q}_{R+1}(1)$ correspond to zero. Hence, each matrix vector multiplication $\boldsymbol{Q}_{i+1}(1)\, \boldsymbol{y}^{(L)}((kL - i)L)$ leads to L^2 multiplications for i ranging from 0 to $R - 1$, and SL multiplications for i equal to R. The B matrix has $L(L - 1)/2$ elements which are neither zero nor unity, and leads to $L(L - 1)/2$ multiplication complexity. The derivation of $\boldsymbol{z}^{(L)}(kL + L)$ requires $L(N + 1)$ multiplications. Thus, the total multiplication complexity of the block filter is $RL^2 + SL + L(L - 1)/2 + L(N + 1)$ or $2NL + L(L + 1)/2$.

In the decomposition based pipelined block processing implementation, we compute the L states $\boldsymbol{y}^{(L)}(kL + ML)$ using the states $\boldsymbol{y}^{(L)}(kL)$, $\boldsymbol{y}^{(L)}(kL - ML)$, $\cdots$, $\boldsymbol{y}^{(L)}(kL - RML)$. Thus, each of the states $\boldsymbol{y}^{(L)}(kL - ML)$, $\cdots$, $\boldsymbol{y}^{(L)}(kL - RML)$ can be derived from $\boldsymbol{y}^{(L)}(kL)$ by using M delay operators. By going through the decomposition steps as in Section IV-C of the companion paper [1], we can derive the M-stage pipelined block realization to be given by

$$\boldsymbol{y}^{(L)}(kL + ML) = \sum_{i=0}^{R} \boldsymbol{Q}_{i+1}(M)\, \boldsymbol{y}^{(L)}(kL - iML) + \boldsymbol{z}_{\log_2 M}(kL + ML), \tag{4.6a}$$

where

$$\boldsymbol{z}_{i+1}(kL + ML) = \boldsymbol{z}_i(kL + ML) + \sum_{j=0}^{R} (-1)^j\, \boldsymbol{Q}_{j+1}(2^i) \cdot \boldsymbol{z}_i\big(kL + ML - 2^i(j + 1)L\big), \quad i = 0, 1, \cdots, (\log_2 M - 1) \tag{4.6b}$$

$$\boldsymbol{z}_0(kL + ML) = \boldsymbol{B}\boldsymbol{z}^{(L)}(kL + ML), \tag{4.6c}$$

and $\boldsymbol{Q}_i(2K)$ can be expressed in terms of $\boldsymbol{Q}_i(K)$ using the matrix versions of (A3) of the companion paper [1]. The above can be proved by induction or by following the method outlined in Section IV-C. Notice that the first $(L - S)$ columns of $\boldsymbol{Q}_{R+1}(2^k)$ are zero for any k. The complexity corresponding to the implementation of $\boldsymbol{z}^{(L)}(kL + ML)$ is $L(N + 1)$ multiplications, and that for $\boldsymbol{B}\boldsymbol{z}^{(L)}(kL + ML)$ is $L(L - 1)/2$ multiplications. The multiplication complexity to implement (4.6b) is NL for each i or $NL(\log_2 M)$ for all i's. The complexity of implementing (4.6a) is also NL. Thus, the total multiplication complexity of this implementation is $L[2N + (L + 1/2)] + NL(\log_2 M)$, which is logarithmic with respect to M, linear in L, and is additive with respect to combining pipelining and block processing.

V. State Space Form Pipelined Incremental Block Digital Filters

We can use the techniques of decomposition and incremental output computation to obtain efficient realization of pipelined state space block recursive digital filters. The state update representation in (3.1) can be recaste as

$$\boldsymbol{x}(kL + ML) = \boldsymbol{A}^{ML}\boldsymbol{x}(kL) + \big[\boldsymbol{B}^{(L)} \,\big|\, \boldsymbol{A}^{(L)}\boldsymbol{B}^{(L)} \,\big|\, \cdots \,\big|\, \boldsymbol{A}^{((M-1)L)}\boldsymbol{B}^{(L)}\big] \cdot \begin{bmatrix} \boldsymbol{u}^{(L)}((k + M - 1)L) \\ \boldsymbol{u}^{(L)}((k + M - 2)L) \\ \vdots \\ \boldsymbol{u}^{(L)}(kL) \end{bmatrix} \tag{5.1a}$$

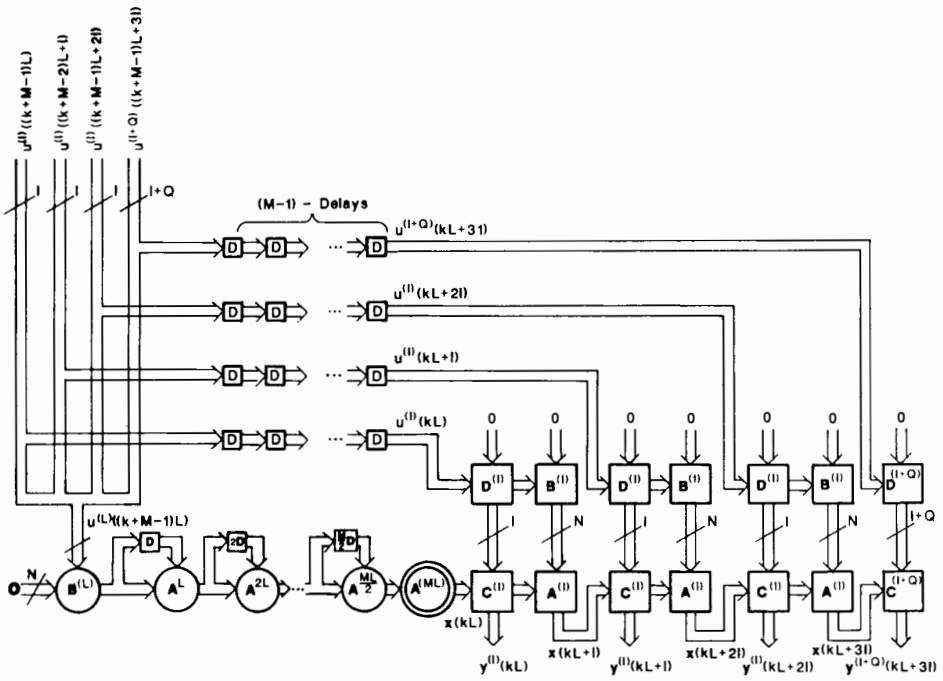


Fig. 12. M-stage pipelined incremental block filter for a block size of $L = 4I + Q$ and increment size of I obtained using the decomposition technique. Each latch in the implementation is *L-slow*.

where

$$A^{(L)} = A^L \tag{5.1b}$$

$$B^{(L)} = \left[A^{L-1}b \mid A^{L-2}b \mid \cdots \mid b\right] \tag{5.1c}$$

$$u^{(L)}(n) = \left[u(n)\, u(n+1) \cdots u(n+L-1)\right]^T, \tag{5.1d}$$

and $A^{(L)}$ is $N \times N$, $B^{(L)}$ is $N \times L$, and $u^{(L)}$ is $L \times 1$. Using the decomposition technique, the state update representation of (5.1) can be rewritten as (for the case where M can be expressed as a power of 2)

$$x(kL + ML) = A^{ML}x(kL) + z_{\log_2 M}\big((k + M - 1)L\big) \tag{5.2a}$$

where

$$z_{i+1}\big((k + M - 1)L\big) = z_i\big((k + M - 1)L\big) + A^{L2^i} z_i\big((k + M - 1 - 2^i)L\big), \quad i = 0, 2, \cdots, \log_2 M - 1 \tag{5.2b}$$

$$z_0\big((k + M - 1)L\big) = B^{(L)}u^{(L)}\big((k + M - 1)L\big). \tag{5.2c}$$

The multiplication complexity to implement (5.2c) is NL, (5.2b) is $(2N - N_1)(\log_2 M)$ for the case of a quasi-diagonal state update matrix, and that for (5.2a) is $(2N - N_1)$. The total state update implementation complexity is $(2N - N_1)(\log_2 M + 1) + NL$ multiplications. The L outputs $y(kL), \cdots, y(kL + L - 1)$ are computed incrementally using (3.6) (exactly in the same manner as described in Section III-C). Adding the output computation complexity of (3.8) to the state update complexity, we can derive the total multiplication complexity of this realization to be

$$C_i = L\left[3N + \frac{2(2N - N_1) + \left[\!\left[\sqrt{2(2N - N_1)}\right]\!\right]^2}{2\left[\!\left[\sqrt{2(2N - N_1)}\right]\!\right]} + \frac{1}{2}\right] + (2N - N_1)\log_2 M - N\left[\!\left[\sqrt{2(2N - N_1)}\right]\!\right], \tag{5.3}$$

which is linear in L, logarithmic in M, and the complexities due to pipelining and block processing are additive. Fig. 12 shows a pipelined block implementation of a pipelined block filter with $L = 4I + Q$, and M loop pipeline stages. Fig. 13 shows an example of a fine-grain pipelined incremental block implementation of a first-order recursive filter, for a block size of four with increment size of two, and for four loop pipeline stages. The roundoff error in pipelined incremental block-state filter is studied in Appendix D, and it is shown that the average roundoff error strictly improves with increase in the number of loop pipeline stages.

To conclude, in a block realization, the eigenvalues of the system move much closer to the origin (the eigenval-

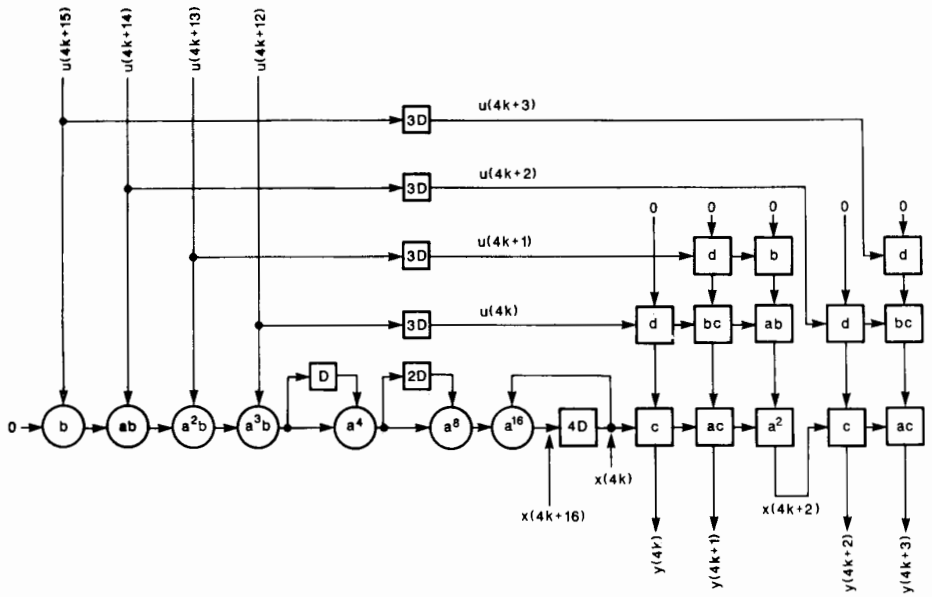


Fig. 13. A 4-stage pipelined incremental first-order block filter for a block size of four, and increment size of two obtained using the decomposition technique. Each latch in the implementation is 4-*slow*.

ues of the block filter are Lth power of that of the original filter for block size L). In a pipelined filter with M loop pipeline stages, the distance of the poles remains the same as that of the original filter, and for each pole in the original filter, $(M-1)$ additional poles are added. In a pipelined block filter with M loop pipeline stages and block size L, the distances of the poles are the same as that of the block filter with block size L (i.e., with $M=1$). As an example, for a first-order system with block size 3, and 4 loop pipeline stages, the eigenvalues are at $\pm a^3$ and $\pm ja^3$, where a is the eigenvalue of the original system.

VI. Conclusion

We have proposed an incremental output computation technique, and using this we have derived incremental block filter structures for direct form and state space form recursive digital filters of complexity linear in block size. We have combined the incremental block filtering and the scattered look-ahead and decomposition based pipelining approaches to derive fine-grain pipelined block filter realizations of direct form and state space form recursive digital filters with complexity linear in block size, logarithmic in number of loop pipeline stages, and additive with respect to combining pipelining and block filtering. These architectures can be appropriately partitioned and implemented using multiple chips. Although the incremental output computation has been proposed in the context of incremental block filters, it is also applicable for implementation of incremental parallel direct form block filters (based on scattered look-ahead, see Section I), and incremental parallel block-state space filters (see Section III). We have not addressed these implementations, since the multiplication complexity of these block filters is larger than our proposed incremental block digital filters.

Appendix A
Direct Form Block Digital Filtering

In this appendix, we derive the block-state update operation of the direct form recursive digital filter.

In Appendix B of the companion paper [1], we derived

$$y(n) = \sum_{j=0}^{N-1} \left[\sum_{l=j+1}^{N} a_l r_{j+M-l} \right] y(n-j-M) + \sum_{j=0}^{M-1} r_j z(n-j), \tag{A1.1}$$

in the context of clustered look-ahead based pipelined realization of recursive digital filters. The above can be rewritten as

$$\begin{aligned} y(n) &= \sum_{j=0}^{N-1} \left[\sum_{l=N-j}^{N} a_l r_{N-1-j+M-l} \right] \cdot y(n-M-N+1+j) + \sum_{j=0}^{M-1} r_{M-j-1} z(n-M+1+j) \\ &= \sum_{j=0}^{N-1} \left[\sum_{l=1}^{j+1} a_{N-l+1} r_{M+l-j-2} \right] \cdot y(n-M-N+1+j) + \sum_{j=0}^{M-1} r_{M-j-1} z(n-M+1+j). \end{aligned} \tag{A1.2}$$

Case I—$L \geq N$: For the case where the block size is greater than the filter order, we need to express the N states $y(kL+L)$ through $y(kL+L+N-1)$ in terms of the N clustered past states $y(kL)$ through $y(kL+N-1)$. Substituting $n = kL+L+i$ and $M = L-N+i+1$ in (A1.2), we have

$$y(kL + L + i) = \sum_{j=0}^{N-1} \left[\sum_{l=1}^{j+1} a_{N-l+1} r_{L-N+l+i-j-1} \right] \cdot y(kL + j) + \sum_{j=0}^{L-N+i} \cdot r_{L-N+i-j} z(kL + N + j). \quad \text{(A1.3)}$$

The matrix formulation for computing the N outputs can be derived by substituting $i = 0, 1, \cdots, N-1$ in the above equation, and is given in (2.4a).

Case II—$L < N$: For the case when the block size is less than the filter order, we need to compute the L outputs $y(kL + L)$ through $y(kL + 2L - 1)$ using the clustered N states $y(kL + L - N)$ through $y(kL + L - 1)$. Substituting $n = kL + L + i$ and $M = i + 1$ in (A1.2), we have

$$y(kL + L + i) = \sum_{j=0}^{N-1} \left[\sum_{l=1}^{j+1} a_{N-l+1} r_{l+i-j-1} \right] \cdot y(kL + L - N + j) + \sum_{j=0}^{i} r_{i-j} z(kL + L + j). \quad \text{(A1.4)}$$

The matrix formulation for computing the L outputs can be derived by substituting $i = 0, 1, \cdots, L-1$ in (A1.4), and is given by (2.5a).

Appendix B

In this appendix, we derive an expression for the roundoff noise error in an incremental block-state filter, and show that the average noise level at the outputs of the incremental block-state filter is the same as that of the block-state filter and less than the parallel block-state filter. For comparison purposes, we assume that the roundoff is performed only at the outputs of the state-variable summing nodes, and at the summing nodes of the filter outputs. All roundoff noise sources are assumed to be stationary white with zero mean and statistically independent.

The roundoff error at the summing nodes of the state variables can be described by

$$\tilde{\boldsymbol{x}}(kL + L) = \boldsymbol{A}^L \tilde{\boldsymbol{x}}(kL) + \boldsymbol{e}_s(kL), \quad \text{(A2.1)}$$

where $\boldsymbol{e}_s$ is of dimension $N \times 1$ and

$$E[\boldsymbol{e}_s \boldsymbol{e}_s^T] = \sigma_0^2 \boldsymbol{I}_N. \quad \text{(A2.2)}$$

The matrix $\boldsymbol{I}_N$ represents the unity matrix of dimension N. The variance of the errors at the summing nodes of the state variables is described by the covariance matrix

$$\boldsymbol{Q} = E[\tilde{\boldsymbol{x}} \tilde{\boldsymbol{x}}^T] = \boldsymbol{A}^L \boldsymbol{Q} (\boldsymbol{A}^T)^L + \sigma_0^2 \boldsymbol{I}_N = \sigma_0^2 \sum_{p=0}^{\infty} \boldsymbol{A}^{pL} (\boldsymbol{A}^T)^{pL}. \quad \text{(A2.3)}$$

The error at the summing node of the ith output is described by

$$\tilde{y}(kL + i) = \boldsymbol{c}^T \boldsymbol{A}^i \tilde{\boldsymbol{x}}(kL) + e(kL + i), \quad \text{(A2.4)}$$

where the last term corresponds to the error at the output summation node and

$$E[e^2(kL + i)] = \sigma_0^2.$$

The variance of the error at the ith output summing node is given by [using (A2.3)]

$$\frac{\sigma_i^2}{\sigma_0^2} = \boldsymbol{c}^T \boldsymbol{A}^i \sum_{p=0}^{\infty} \boldsymbol{A}^{pL} (\boldsymbol{A}^T)^{pL} (\boldsymbol{A}^T)^i \boldsymbol{c} + 1. \quad \text{(A2.5)}$$

The average roundoff noise at the outputs is given by [using (A2.5)]

$$\frac{\sigma_{av}^2}{\sigma_0^2} = \frac{\sum_{i=0}^{L-1} \sigma_i^2}{L \sigma_0^2} = \frac{1}{L} \boldsymbol{c}^T \sum_{p=0}^{\infty} \boldsymbol{A}^p (\boldsymbol{A}^T)^p \boldsymbol{c} + 1. \quad \text{(A2.6)}$$

The average roundoff noise for the block-state filter has been verified to be exactly the same as that given by (A2.6) [8]. In [11], the roundoff noise of the parallel block-state has been derived and shown to be greater than that of the block-state filter. Hence, to conclude, the average roundoff noise of the incremental block-state filter is the same as that of the block-state filter and less than that of the parallel block-state filter.

Appendix C

In this appendix, we derive an expression for the elements of $\boldsymbol{A}^M(L)$ as a function of filter coefficients and the sequence r_i (for the case when M can be expressed as a power of 2). This expression is useful for deriving pipelined block realization of direct form recursive digital filters for the case $L \geq N$. The sequence r_i has been defined in Appendix A of the companion paper [1].

Theorem A3.1: The elements of $\boldsymbol{A}^M(L)$ are given by

$$[\boldsymbol{A}^M(L)]_{ij} = \sum_{k=0}^{j-1} a_{N-k} r_{LM-N+i-j+k}. \quad \text{(A3.1)}$$

Proof (by induction): Assume (A3.1) is true for M, and then prove that it also holds for $2M$. The elements of $\boldsymbol{A}^{2M}(L)$ are given by

$$[\boldsymbol{A}^{2M}(L)]_{ij} = \sum_{l=1}^{N} [\boldsymbol{A}^M(L)]_{il} [\boldsymbol{A}^M(L)]_{lj} = \sum_{l=1}^{N} \left[\sum_{k=0}^{l-1} a_{N-k} r_{LM-N+i-l+k} \right] \cdot \left[\sum_{m=0}^{j-1} a_{N-m} r_{LM-N+l-j+m} \right] = \sum_{m=0}^{j-1} a_{N-m} \sum_{l=1}^{N} \left[\sum_{k=0}^{l-1} a_{N-k} r_{LM-N+i-l+k} \right] \cdot r_{LM-N+l-j+m} = \sum_{m=0}^{j-1} a_{N-m} r_{2LM-N+i-j+m}$$

using theorem (A1.1) in [1]. QED

Corollary: $\boldsymbol{A}^M(KL) = \boldsymbol{A}^{KM}(L)$.

Theorem A3.2:

$$\boldsymbol{B}(ML) = [\boldsymbol{A}^{M-1}(L)\, \boldsymbol{B}(L) \quad \boldsymbol{A}^{M-2}(L) \cdot \boldsymbol{B}(L) \cdots \boldsymbol{B}(L)]. \quad \text{(A3.3)}$$

Proof: From (2.4d), we have

$$[\boldsymbol{B}(ML)]_{i,kL+j} = r_{ML-N+i-kL-j}. \quad \text{(A3.4a)}$$

We need to prove that the above element must be the ijth element of $\boldsymbol{A}^{M-1-k}(L)\,\boldsymbol{B}(L)$. This element is given by

$$\begin{aligned}\left[\boldsymbol{A}^{M-1-k}(L)\,\boldsymbol{B}(L)\right]_{ij} &= \sum_{m=1}^{N}\left[\boldsymbol{A}^{M-1-k}(L)\right]_{im}\left[\boldsymbol{B}(L)\right]_{mj} \\ &= \sum_{m=1}^{N}\left[\sum_{s=0}^{M-1} a_{N-s}\,r_{(M-1-k)L-N+i-m-s}\right] r_{L-N+m-j} \qquad \text{(A3.4b)} \\ &= r_{ML-N+i-kL-j} \qquad \text{using theorem A1.1 in [1].}\end{aligned}$$

QED

Appendix D

In this appendix, we derive the roundoff error in a fine-grain pipelined block-state space filter with block size L and M loop pipeline stages, and show that this error is strictly less than that of a block filter with block size L (i.e., with $M = 1$) under the assumptions stated in Appendix B.

The roundoff error at the summing nodes of the state variables of the pipelined block filter is described by

$$\tilde{\boldsymbol{x}}(kL + ML) = \boldsymbol{A}^{ML}\tilde{\boldsymbol{x}}(kL) + \boldsymbol{e}_s(kL + ML), \quad \text{(A4.1)}$$

where $\boldsymbol{e}_s$ is of dimension $N \times 1$ and

$$E[\boldsymbol{e}_s\,\boldsymbol{e}_s^T] = \sigma_0^2\boldsymbol{I}_N. \quad \text{(A4.2)}$$

The matrix $\boldsymbol{I}_N$ represents the unity matrix of dimension N. The variance of the errors at the summing nodes of the state variables is described by the covariance matrix

$$\begin{aligned}\boldsymbol{Q} = E[\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^T] &= \boldsymbol{A}^{ML}\boldsymbol{Q}(\boldsymbol{A}^T)^{ML} + \sigma_0^2\boldsymbol{I}_N \\ &= \sigma_0^2\sum_{p=0}^{\infty}\boldsymbol{A}^{pML}(\boldsymbol{A}^T)^{pML}. \quad \text{(A4.3)}\end{aligned}$$

The error at the summing node of the ith output of the block filter is described by

$$\tilde{y}(kL + i) = \boldsymbol{c}^T\boldsymbol{A}^i\tilde{\boldsymbol{x}}(kL) + e(kL+ i), \quad \text{(A4.4)}$$

where the last term corresponds to the error at the output summation node and

$$E[e^2(kL + i)] = \sigma_0^2.$$

The variance of the error at the ith output summing node is given by [using (A4.3)]

$$\frac{\sigma_i^2}{\sigma_0^2} = \boldsymbol{c}^T\boldsymbol{A}^i\sum_{p=0}^{\infty}\boldsymbol{A}^{pML}(\boldsymbol{A}^T)^{pML}(\boldsymbol{A}^T)^i\boldsymbol{c} + 1. \quad \text{(A4.5)}$$

The average roundoff noise at the outputs is given by [using (A.5)]

$$\begin{aligned}\frac{\sigma_{av}^2}{\sigma_0^2} &= \frac{\sum_{i=0}^{L-1}\sigma_i^2}{L\sigma_0^2} \\ &= \frac{1}{L}\boldsymbol{c}^T\sum_{j=0}^{L-1}\boldsymbol{A}^j\left[\sum_{p=0}^{\infty}\boldsymbol{A}^{pML}(\boldsymbol{A}^T)^{pmL}\right](\boldsymbol{A}^T)^j\boldsymbol{c} + 1, \\ &\qquad \text{(A4.6)}\end{aligned}$$

which is a strictly decreasing function in M.

References

[1] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters—Part I: Pipelining using scattered look-ahead and decomposition," *IEEE Trans. Acoust., Speech, Signal Processing*, this issue, pp. 1099–1117.

[2] B. Gold and K. L. Jordan, "A note on digital filter synthesis," *Proc. IEEE*, vol. 65, pp. 1717–1718, Oct. 1968.

[3] H. B. Voelcker and E. E. Hartquist, "Digital filtering via block recursion," *IEEE Trans. Audio Electroacoust.*, vol. AU-18, pp. 169–176, June 1970.

[4] C. S. Burrus, "Block implementation of digital filters," *IEEE Trans. Circuit Theory*, vol. CT-18, pp. 697–701, Nov. 1971.

[5] A. L. Moyer, "An efficient parallel algorithm for digital IIR filters," in *Proc. IEEE Conf. Acoust., Speech, Signal Processing*, Apr. 1976, pp. 525–528.

[6] S. K. Mitra and R. Gnanasekaran, "Block implementation of recursive digital filters—New structures and properties," *IEEE Trans. Circuits Syst.*, vol. CAS-25, pp. 200–207, Apr. 1978.

[7] C. W. Barnes and S. Shinnaka, "Block shift invariance and block implementation of discrete-time filters," *IEEE Trans. Circuits Syst.*, vol. CAS-27, pp. 667–672, Aug. 1980.

[8] J. Zeman and A. G. Lindgren, "Fast digital filters with low roundoff noise," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 716–723, July 1981.

[9] D. A. Schwartz and T. P. Barnwell, III, "Increasing the parallelism of filters through transformation to block state variable form," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1984.

[10] H. H. Lu, E. A. Lee, and D. G. Messerschmitt, "Fast recursive filtering with multiple slow processing elements," *IEEE Trans. Circuits Syst.*, vol. CAS-32, pp. 1119–1129, Nov. 1985.

[11] C. L. Nikias, "Fast block data processing via a new IIR digital filter structure," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, Aug. 1984.

[12] K. K. Parhi and D. G. Messerschmitt, "Block digital filtering via incremental block-state structure," in *Proc. IEEE Int. Symp. Circuits Syst.*, Philadelphia, PA, May 1987.

[13] W. Sung and S. K. Mitra, "Efficient multi-processor implementation of recursive digital filters," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Tokyo, Apr. 1986, pp. 257–260.

[14] C. W. Wu and P. R. Cappello, "Application specific CAD of VLSI second-order sections," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 813–825, May 1988.

[15] K. S. Arun, "Ultra-high-speed parallel implementation of low-order digital filters," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1986, pp. 944–946.

[16] K. K. Parhi, W. L. Chen, and D. G. Messerschmitt, "Architecture considerations for high speed recursive filtering," in *Proc. 1987 IEEE Int. Symp. Circuits Syst.*, Philadelphia, PA, May 1987.

[17] K. K. Parhi and D. G. Messerschmitt, "A bit-parallel bit level recursive filter architecture," in *Proc. IEEE Int. Conf. Comput. Design*, New York, 1986.

[18] —, "Concurrent cellular VLSI adaptive filter architecture," *IEEE Trans. Circuits Syst.*, vol. CAS-34, pp. 1141–1151, Oct. 1987.

[19] T. Miyawaki and C. W. Barnes, "Multirate recursive digital filters—A general approach and block structures," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 1148–1154, Oct. 1983.

[20] C. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," presented at the 3rd Caltech Conf. VLSI, Pasadena, CA, Mar. 1983.

[21] S. Y. Kung, "On supercomputing with systolic/wavefront array processors," *Proc. IEEE*, vol. 72, July 1984.

[22] D. A. Schwartz and T. P. Barnwell, III, "A graph theoretic technique for the generation of systolic implementation of shift invariant flow graphs," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, San Diego, CA, Mar. 1984.

[23] D. A. Schwartz, "Synchronous multiprocessor realization of shift invariant flow graphs," Ph.D. dissertation, Georgia Inst. Technol., Rep. DSPL-85-2, July 1985.

Keshab K. Parhi (S'85–M'88), for a photograph and biography, see this issue, p. 1117.

David G. Messerschmitt (S'65–M'68–SM'78–F'83), for a biography, see this issue, p. 1117.