Pipeline Interleaving and Parallelism in Recursive Digital Filters—Part I: Pipelining Using Scattered Look-Ahead and Decomposition

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

Abstract—**The computational latency associated with the internal recursion or feedback in recursive systems limits the opportunities to use a pipelining technique to achieve high sampling rate realizations. Pipelining recursive loops by simply inserting latches is useful for applications requiring moderate sampling rates and where multiple independent computations are available to be interleaved in the pipeline; but not where a single recursive operation needs to be performed at very high sampling rates. In this paper, we introduce a new look-ahead approach (referred to as *scattered look-ahead*) to pipeline recursive loops in a way that guarantees stability. We also propose a new *decomposition technique* to implement the nonrecursive portion (generated due to the scattered look-ahead process) in a decomposed manner to obtain concurrent stable pipelined realizations of logarithmic implementation complexity with respect to the number of loop pipeline stages (as opposed to linear). The upper bound on the roundoff error in these pipelined filters is shown to improve with increase in the number of loop pipeline stages. We study efficient pipelined realizations of both direct form and state space form recursive digital filters. Based on the scattered look-ahead technique, we present fully pipelined and fully hardware efficient linear bidirectional systolic arrays for recursive digital filters. The decomposition technique is also extended to time varying recursive systems.**

I. Introduction

IN order to exploit VLSI for high performance, we need to understand the characteristics of the scaled VLSI technologies. For example, VLSI offers a greater potential for complexity than speed, favors replication of one function, and imposes a high cost in performance for nonlocalized communication. Design costs can be minimized by composing the system as a replication of simple processing elements. These considerations favor implementations which feature arrays of identical or easily parametrized processing elements (since these are easily given a software procedural definition) with mostly localized interconnections (for reduced communication costs). This has led to an interest in systolic- and wavefront-array implementations [1], [2].

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 8928119.

High performance can be achieved by either using exotic high speed technologies, such as bipolar, or GaAs, which allow us to gain performance without modification of the algorithm. On the other hand, we can use a low cost VLSI technology such as CMOS and yet gain impressive performance by exploiting concurrency. Concurrency is usually manifested in the form of pipelining or parallelism or both. Concurrent architectures can be derived by implementing the existing algorithms in new ways. To be more precise, we do not change the transfer function or the input–output characteristics of the algorithm, but we do change the internal structure of the algorithm, thereby impacting the finite precision effects but nothing else. This is referred to as *recasting the structure* of the algorithm. Different forms of recasting a specified algorithm can lead to realizations with entirely different properties and implementation complexities. In this paper, we show that *appropriate* recasting of the structure of an algorithm can have a dramatic effect on the performance of an implementation. When we speak of performance in this context, we are referring specifically to the speed, as reflected in the sampling rate, rather than the functionality of the algorithm.

The challenge in achieving high performance implementations is mostly in recursive systems since, as we will see, the recursion or the internal feedback negates the most obvious ways of improving performance. This is because the computational latency associated with the feedback loop in recursive systems limits the opportunities for pipelining and/or parallel processing. In nonrecursive systems, we can place latches across any *feed-forward cutset* without changing the transfer function (at the expense of latency) and achieve the desired level of pipelining. However, recursive systems cannot be pipelined at an arbitrary level by simply inserting latches, since the pipelining latches would change the number of delay operators in the loop, and hence the transfer function of the implementation. We can overcome this recursive bottleneck by changing the internal structure of the algorithm to *create* additional logical delay operators inside the recursive loop, which can then be used for pipelining.

High sampling rate realizations of recursive digital filters using block processing have been suggested [3]–[18]. In block processing, input samples are processed in the

form of nonoverlapping blocks, and outputs are also generated block by block. Thus, we can increase the block size arbitrarily to achieve arbitrarily high sampling rate recursive system realizations. The best-known block structure reported so far for recursive digital filtering requires a square multiplication complexity with respect to the block size. The block state update operation as well as the pipelined state update operation in recursive filters belong to the class of *look-ahead computation* techniques [19], [20]. In this paper, we only discuss the pipelining approach. The block processing and fine-grain pipelined block processing approaches are addressed in the companion paper [18].

We recently studied look-ahead and associated decomposition algorithms to pipeline recursive loops in the context of a first-order recursive section [i.e., where the state $x(n)$ is expressed as a function of $x(n-1)$] to derive high speed realizations of an adaptive lattice filter [21], [22]. In look-ahead, the algorithm is iterated as many times as desired to create the necessary level of concurrency, and the iterated version is implemented. Specifically, the first-order recursion is iterated to express the state $x(n)$ as a function of $x(n-M)$ to create M delay operators inside the loop so that the loop can be pipelined by M stages. This iteration process contributes to a nonrecursive $O(M)$ multiplication complexity. A *decomposition technique* was also proposed in [21] and [22] to implement the nonrecursive overhead in a decomposed manner to obtain a logarithmic complexity (as opposed to linear). The look-ahead technique has been used in the context of a first-order recurrence system implementation in [23], but without the decomposition technique.

In this paper, we study efficient pipelining of higher order recursive systems. In an Nth-order recursive system, the state $x(n)$ is expressed as a function of past N states $x(n-1), x(n-2), \cdots$, and $x(n-N+1)$. In these higher order systems, look-ahead can be either *clustered* or *scattered*. In *clustered look-ahead*, the algorithm is iterated to express the state $x(n)$ as a function of N consecutive or clustered states $x(n-M), x(n-M-1), \cdots$, and $x(n-M-N+1)$. This look-ahead creates M loop delay operators, which can be used to pipeline the loop by M stages. In this technique, the original Nth-order filter is emulated by an $(N+M-1)$th-order filter [$(M-1)$ cancelling poles and zeros have been added]. The multiplication complexity of the resulting pipelined filter derived is $O(M)$, which is linear with respect to M. This multiplication complexity can be reduced to logarithmic using the *recursive doubling* decomposition algorithm, which was first proposed by Kogge and Stone in the context of first-order linear recurrence systems [24]–[26], and later used for higher order recurrence systems [27]–[29]. Loomis and Sinha recently used this clustered look-ahead approach to derive a pipelined realization of direct form recursive digital filters [30] (without the decomposition). A similar approach was also followed for recursive filter implementation using charge domain devices [31]; however, no systematic transformation was described in this paper. Since the pipelined filter is derived by adding poles and zeros, some of the modes are either uncontrollable or unobservable or both, and hence it is necessary to guarantee stability in the pipelined filter for proper functioning. Unfortunately, for higher order systems, the clustered look-ahead process does not guarantee all the additional poles to lie inside the unit circle, and hence does not guarantee stability.

In this paper, we introduce a new *scattered look-ahead* approach to derive stable pipelined filters [32]. In scattered look-ahead, we express $x(n)$ as a function of past N *scattered* states $x(n-M), x(n-2M), \cdots$, and $x(n-NM)$, thus emulating the original Nth-order filter by an NMth-order filter. Note that the clustered look-ahead and the scattered look-ahead approaches are identical for the first-order case [since in both cases $x(n)$ is expressed as a function of $x(n-M)$], but are quite different for higher order systems; and in that sense the extensions to higher order systems is nontrivial. In scattered look-ahead, for *each* existing pole in the original filter, we add $(M-1)$ additional cancelling poles (and zeros at identical locations) with equal angular spacing at a distance from the origin the same as that of the original pole. The scattered look-ahead process leads to an $O(NM)$ complexity (much larger than that for clustered look-ahead), but guarantees stability. Using a decomposition technique [21], [22], [33], we reduce the multiplication complexity of the nonrecursive portion from $O(NM)$ to $O(N \log_2 M)$. The upper bound on roundoff noise in these pipelined filters improves with increase in M. Based on the scattered look-ahead and the decomposition techniques, we derive pipelined realizations of direct form as well as state space form recursive digital filters. It is useful to note that the form of scattered look-ahead recursion has been used earlier in different contexts. It was used in the direct form block filter representation by Moyer [6], in polyphase network design of multirate recursive filters [34], in parallel implementation of partial differential equations (PDE's) [35]–[40], and for zero-input higher order recurrence systems (not a filtering operation) [41]. For the case where M is a power of two, the decomposition technique presented here is similar to the *cyclic reduction* algorithm used in parallel implementation of PDE's [35]–[40]. A slightly different representation of the decomposition algorithm is also presented in [42].

Several pipelined bidirectional systolic arrays for recursive digital filtering have been proposed in [43]–[47]. However, these structures require interleaving of independent time series when pipelined. In this paper, we present fully pipelined and fully hardware efficient bidirectional systolic arrays for recursive filtering using the scattered look-ahead technique (without requiring interleaving of independent time series) [32]. We have also obtained unidirectional ring systolic architectures for high-speed scattered look-ahead recursive filters, which are fully hardware efficient and fully pipelined. These arrays are presented in [32] and [48], and are not reported here due to lack of space.

The organization of the paper is as follows. The iteration period bound in recursive computations is reviewed in Section II. In Section III, we review the notion of pipeline interleaving in the context of recursive digital filtering. Sections IV and V address fine-grain pipelined realization of direct form and state space form linear time-invariant recursive digital filters, respectively, using scattered look-ahead and the decomposition techniques. The extension of the scattered look-ahead and decomposition techniques to linear time varying recursive systems is addressed in Section VI.

II. Iteration Bound

Let S_l represent the set of loops in the recursive computation graph, D_l represent the latency associated with the computation in loop l, and M_l represent the number of latches or logical delay operators inside the loop l. Let each latch in the computation graph be *L-slow*, i.e., the clock rate of each latch is L times slower than the sample rate, or equivalently, the implementation corresponds to a block implementation with block size L (the block size is assumed to be constant throughout). Then the iteration period bound is given by

$$T_\infty = \frac{1}{L} \underset{l \in S_l}{MAX} \left\lceil \frac{D_l}{M_l} \right\rceil. \tag{2.1}$$

The *maximum achievable sampling rate* for the computation graph is $1/T_\infty$. The loop l_0 for which $T_\infty = D_{l_0}/LM_{l_0}$ is satisfied is called the *critical loop*. For unity block size (i.e., $L = 1$), this definition reduces to the iteration bound definition in [49] and [50].

The iteration period bound can be improved by increasing either the number of pipeline stages inside the recursive loop (M_l) or the block size (L) or both. In this paper, we assume M_l to be the same for all loops and refer to it as M. Thus, by using M pipeline stages inside the recursive loop and a block size of L, the sample rate can be increased by a factor LM. Since pipelined realizations can be achieved with logarithmic increase in hardware (as opposed to linear increase as in block processing), it is hardware efficient to use pipelined algorithms (i.e., with $L = 1$) first for high speed IIR filter implementations, and then combine block processing with pipelining only if sufficient speed cannot be generated by using pipelining alone. Thus, block processing in itself is an inefficient way of implementing high speed custom IIR digital filters. However, block processing is useful for software programmable implementations on general-purpose coarse-grain multiprocessors.

III. Pipeline Interleaving in Digital Filters

The pipeline interleaving notion is an old idea, and has been used in general-purpose computers. Recently, the pipeline interleaving approach has been advocated for programmable implementation of signal processing systems using deeply pipelined programmable digital signal processors [51], and for cyclostatic implementation of signal processing systems [52].

In this section, we review the notion of pipeline interleaving in the context of a simple first-order recursive digital filter. In particular, we discuss three forms of pipeline interleaving: i) inefficient single/multichannel interleaving; ii) efficient single channel interleaving; and iii) efficient multichannel interleaving. In i), the loop is pipelined without changing the structure of the algorithm and thus hardware is not fully utilized, since zero samples need to be interleaved to preserve the integrity of the algorithm. In ii) and iii), the internal structure of the algorithm is changed in a way that the pipeline is maximally or fully utilized.

A. Inefficient Single/Multichannel Interleaving

Consider a first-order linear time-invariant recursion described by

$$x(n+1) = ax(n) + bu(n), \tag{3.1}$$

and shown in Fig. 1(a) in the form of a computation graph. The iteration period of this computation graph is $(T_m + T_a)$, where T_m and T_a, respectively, represent the word-level multiplication time and addition time. Consider obtaining an *M-stage* pipelined version of this implementation by *inserting* $(M - 1)$ additional latches inside the loop as shown in Fig. 1(b) (at the appropriate places). Then the clock period of this implementation can, in principle, be reduced by M times, but the latency associated with the loop computation and the sample period of the implementation will increase to M clock periods. As an example for $M = 5$, if we begin with a state $x^1(0)$ in clock period 0, the next state $x^1(1)$ will be available in clock period 5. For the case of a single time series, this array will be useful for only 20 percent of the time. (Trying to input samples of a single time series each clock period would implement a different algorithm, since the number of logical delays inside the loop has been changed.) Hence, the sampling rate of this implementation is 5 times slower than the clock rate, and is no higher than that of the unpipelined version (in fact, it is worse due to the delay time introduced due to the additional latches). However, if 5 independent time series are available to be filtered by the same hardware, then the hardware can be fully utilized as shown in the schedule of Fig. 1(c), although all the independent time series must be filtered at the slow rate. Independent time series can correspond to outputs of each first- or second-order cascade stage (since these elements can be separated by a feed-forward cutset), or can correspond to independent channels requiring identical filtering operation. As an example, for a 10th-order recursive filter implemented as cascaded second-order sections, the five section outputs are independent and can be interleaved in the pipeline (of course, each at 5-slow rate). Thus, the pipeline interleaving approach is well suited for applications requiring nominal concurrency. To conclude, if a recursive loop with a single delay element is pipelined by M-stages by *inserting* $(M - 1)$ additional delay elements, then the input data must be M-way interleaved, i.e., $(M - 1)$ zero time series or independent

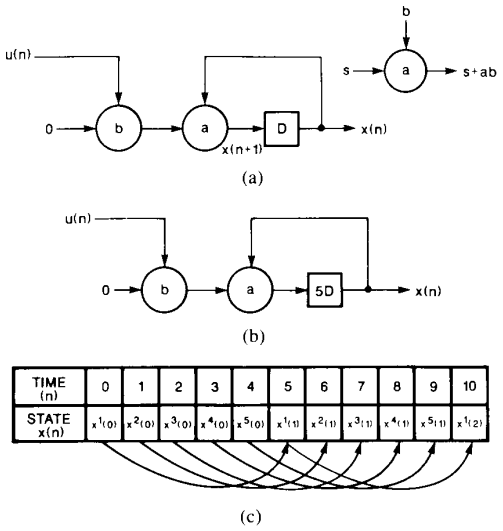

Fig. 1. (a) A simple first-order LTI recursion. (b) The first-order LTI recursion after inserting $(M-1)$ delay operators inside the loop (for $M=5$). This implementation leads to M-way interleaving. (c) A partial schedule for the implementation of Fig. 1(b). The input time series are 5-way interleaved, i.e., 5 independent time series are being filtered simultaneously. The state $x^i(n)$ corresponds to the state of the ith time series at time index n.

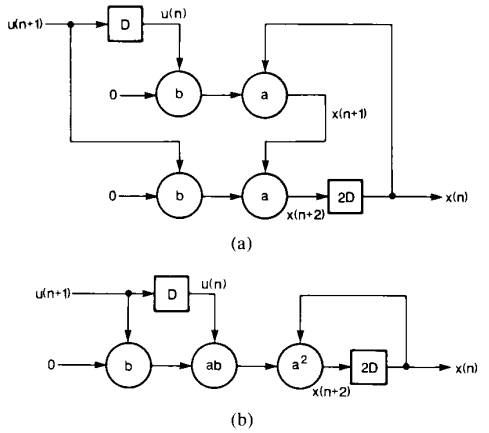

Fig. 2. (a) An equivalent realization of Fig. 1(a) obtained without the use of the look-ahead transformation. The iteration bound is the same as that of Fig. 1(a). (b) An equivalent first-order LTI recursion obtained with the use of look-ahead computation. The iteration bound is improved by a factor of 2 compared to the structure of Fig. 2(a).

time series are interleaved with the given data stream (otherwise, the transfer function of the algorithm will be changed), and nothing has been achieved with respect to the sample rate with which a single time series can be filtered. This implementation has also been referred to as M-slow circuit in the literature [1], [53]–[55]. The hardware in this slow interleaved implementation is inefficiently utilized if M independent computations are not available to be interleaved (which is usually the case).

B. Efficient Single-Channel Interleaving

Ruling out the interleaving of independent time series, the two problems with M-slow implementations are i) a sampling rate M times slower than the clock rate, and ii) inefficient utilization of processing elements. Now we show that both these problems can be overcome by using the look-ahead transformation [19]–[23], in which the given linear recursion is first iterated a few times to *create* additional concurrency.

Consider the first-order LTI recursion of (3.1). By recasting this recursion, we can express $x(n+2)$ as a function of $x(n)$ to obtain

$$x(n+2) = a\left[ax(n) + bu(n)\right] + bu(n+1). \quad (3.2a)$$

A realization of this recursion is shown in Fig. 2(a). The iteration bound of this recursion is $2(T_m + T_a)/2$ and is the same as that of Fig. 1(a). This is because the amount of computation and the number of logical delays inside the recursive loop are both doubled as compared to that in Fig. 1(a) leading to no net improvement. However, another recursion equivalent to that of (3.2a) is

$$x(n+2) = a^2x(n) + abu(n) + bu(n+1), \quad (3.2b)$$

and is shown in Fig. 2(b). The iteration period bound of this realization, $(T_m + T_a)/2$, is a factor of two lower than that of the realizations in Fig. 1(a) and Fig. 2(a)!

Applying $(M-1)$-steps of look-ahead to the iteration of (3.1), we can obtain an equivalent implementation described by

$$x(n+M) = a^Mx(n) + \sum_{i=0}^{M-1} a^ibu(n+M-1-i), \quad (3.2c)$$

and shown in Fig. 3(a). Note that the loop delay corresponds to z^{-M} instead of z^{-1}. This implies that the computation must be completed in M clock cycles rather than 1 clock cycle. The iteration period bound of this computation graph is $(T_m + T_a)/M$, which corresponds to a sampling rate M times higher than that for the original computation graph (although the complexity and system latency are now linearly increased). A portion of the schedule for the realization of Fig. 3(a) is shown in Fig. 3(b) for $M=5$. The terms $ab, a^2b, \cdots, a^{M-1}b, a^M$ in (3.2c) can be precomputed and are referred to as the *precomputation terms*. The second term on the right-hand side of (3.2c) represents the look-ahead computation term, and its complexity is referred to as the *look-ahead complexity*. Since the look-ahead computation term is nonrecursive, it can be pipelined by placing latches at the appropriate feed-forward cutsets.

The steady-state input–output behavior is not altered by the look-ahead technique. By this it is meant that for sufficiently old inputs, the outputs of the transformed system and the original systems will be identical. However, it is also possible to recaste the initial states of the transformed system so that the input–output behavior of the transformed and the original system are identical for *all* inputs, as long as the original system is causal (under infinite precision assumption). Consider the schedule shown in Fig. 3(b) corresponding to the implementation of Fig. 3(a),

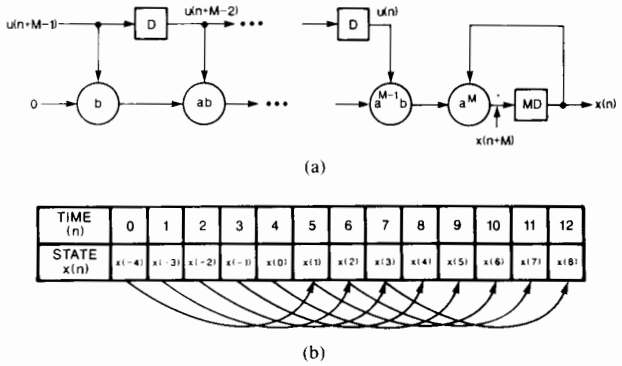

Fig. 3. (a) Equivalent first-order LTI recursion obtained using $(M-1)$ steps of look-ahead. (b) A partial schedule for the structure of Fig. 3(a) for $M = 5$.

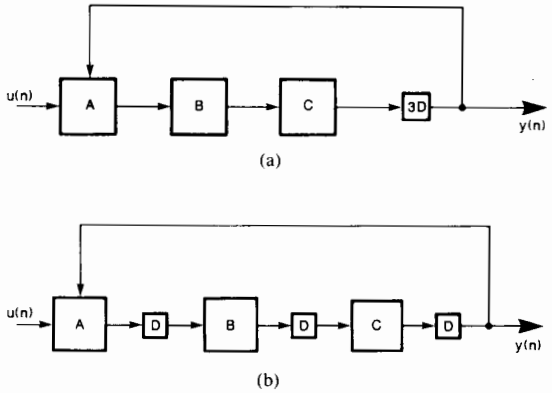

Fig. 4. (a) A computation graph. (b) An equivalent *retimed* computation graph.

where we start with M independent initial states $x(-M+1), x(-M+2), \cdots, x(0)$ (for $M = 5$). In the original system of (3.1), the state $x(1)$ is computed in terms of the initial state $x(0)$,

$$x(1) = ax(0) + bu(0). \tag{3.3a}$$

For the transformed system of (3.2c), the state $x(1)$ is calculated in terms of $x(-M+1)$,

$$x(1) = a^5x(-4) + bu(0), \tag{3.3b}$$

for $M = 5$ (since $u(-4), \cdots, u(-1)$ are all 0 due to causality). From (3.3),

$$x(-4) = a^{-4}x(0). \tag{3.3c}$$

A similar analysis can be carried out to obtain the M initial states

$$x(-i) = a^{-i}x(0), \qquad i = 1, 2, \cdots, (M-1). \tag{3.4}$$

In the transformed system, we start with M initial states and compute the next M states in a pipelined interleaved manner [see Fig. 3(b)]. In this regard, look-ahead computation can be treated as an application of pipeline interleaving. Look-ahead computation has allowed us to transform a single serial computation into M independent concurrent computations, and to pipeline the feedback loop to achieve high speed filtering of a single time series while maintaining full hardware utilization.

Provided the multiplier and the adder can be conveniently pipelined, the iteration bound can be achieved by *retiming* or *cutset transformation* [1], [53]–[55]. The retiming process involves moving the delays around in the feedback loop in such a way that the number of delays in any loop remains unaltered (thereby not affecting the transfer function). A simple example of retiming is illustrated in Fig. 4. The iteration period bound for the realization in Fig. 4(a) is $(T_A + T_B + T_C)/3$, whereas the actual iteration period is $(T_A + T_B + T_C)$, where T_i corresponds to the computation time of block i. The iteration period for an equivalent realization in Fig. 4(b) (obtained after redistributing the delays) is $Max(T_A, T_B, T_C)$. If the computational latencies T_A, T_B, and T_C are identical, then this realization has an iteration period equal to the iteration period bound.

C. Efficient Multichannel Interleaving

We can extend look-ahead to the case where multiple independent channels require identical filtering operation. Consider the same first-order linear recursion of (3.1) for the case of two channels, and six pipeline stages inside the recursive loop. Then, without use of the look-ahead technique, the hardware will be utilized only one-third of the time. To get full utilization of hardware, we iterate the recursion two times, and interleave the computation of two time series. In general, if P independent time series are available, and the loop is pipelined by M-stages (assume $M = PQ$), then the recursion needs to be iterated $(Q-1)$ times. For this example, the iterated recursion corresponds to

$$x^i(n+3) = a^3x^i(n) + a^2bu^i(n) + abu^i(n+1) + bu^i(n+2), \qquad i = 1, 2. \tag{3.5}$$

Fig. 5 shows a partial schedule corresponding to the processing of the states x^1 and x^2 of the two independent time series in an interleaved manner.

IV. Pipelining Direct Form Recursive Digital Filters

The *clustered look-ahead* technique (an approach which has been used in the context of block state update operation in recursive digital filters) has been used to pipeline direct form recursive filters [30]. However, the resulting pipelined realizations require a linear complexity in the number of loop pipeline stages, and are not guaranteed to be stable. In this section, we present a *scattered look-ahead* approach to derive stable pipelined filters of complexity linear with respect to the number of loop pipeline stages. We then propose a *decomposition technique* to implement the nonrecursive portion generated due to the scattered look-ahead process in a decomposed manner to obtain an implementation with logarithmic increase in

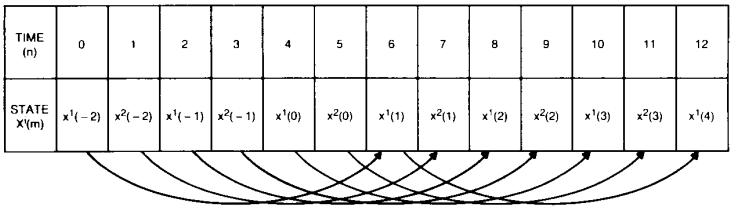

Fig. 5. A partial schedule for a two-channel implementation with six loop pipelining stages obtained using two steps of look-ahead.

hardware with respect to the number of loop pipeline stages. The decomposition technique is the key in obtaining area-efficient implementations, and makes pipelined realizations attractive for high speed VLSI IIR filter implementations. We also present fully pipelined and fully hardware efficient linear bidirectional systolic arrays for recursive filters based on scattered look-ahead.

Let the transfer function of a direct form recursive filter be described by

$$H(z) = \frac{\sum_{i=0}^{N} b_i z^{-i}}{1 - \sum_{i=1}^{N} a_i z^{-i}}. \tag{4.1}$$

Equivalently, the output sample $y(n)$ can be described in terms of the input sample $u(n)$, and the past input and output samples, and is given by

$$y(n) = \sum_{i=1}^{N} a_i y(n-i) + \sum_{i=0}^{N} b_i u(n-i) = \sum_{i=1}^{N} a_i y(n-i) + z(n). \tag{4.2}$$

The sampling rate of this recursive filter realization is limited by the throughput of a single multiplication and N additions, since the critical loop contains a single delay operator or latch.

A. Clustered Look-Ahead Pipelining

We can transform the transfer function of (4.1) such that the coefficients of $z^{-1}, \cdots, z^{-(M-1)}$ in the denominator of the transfer function are zero, i.e., the denominator contains the terms z^{-M}, $z^{-(M+1)}$, $\cdots$, and $z^{-(N+M-1)}$. Such a transfer function corresponds to an M-stage pipelined implementation, since the output sample $y(n)$ can be described in terms of the *cluster* of N past outputs $y(n-M)$, $y(n-M-1)$, $\cdots$, and $y(n-M-N+1)$. A time domain description of such an equivalent filter is given by (see Appendix B)

$$y(n) = \sum_{j=0}^{N-1} \left[\sum_{k=j+1}^{N} a_k r_{j+M-k} \right] y(n-j-M) + \sum_{j=0}^{M-1} r_j z(n-j), \tag{4.3a}$$

where

$$z(n) = \sum_{i=0}^{N} b_i u(n-i) \tag{4.3b}$$

and the sequence r_i is defined in Appendix A. The equivalent transfer function of this pipelined realization is given by (see Appendix B)

$$H(z) = \frac{\sum_{i=0}^{M-1} r_i \sum_{j=0}^{N} b_j z^{-(i+j)}}{1 - \sum_{i=0}^{N-1} \left[\sum_{j=i+1}^{N} a_j r_{i+M-j} \right] z^{-(i+M)}}. \tag{4.4}$$

Note that the coefficients in square brackets in (4.3) and (4.4) are computed off line. This transfer function has been derived by multiplying $\sum_{i=0}^{M-1} r_i z^{-i}$ both in the numerator and the denominator, introducing $(M-1)$ additional cancelling poles and zeros.

Since the critical loop of this implementation contains M delay operators and a single multiplication operation, this loop can be pipelined by M stages, and the sampling rate can be increased by a factor of M. The numerator or the nonrecursive portion of (4.4) can be implemented with $(N+M)$ multiplications, and the denominator or the recursive portion can be implemented with N multiplications. Thus, the total complexity of this pipelined implementation is $(2N+M)$ multiplications, and is linear with respect to number of loop pipeline stages (M) or speedup or increase in the sampling rate.

We illustrate the stability problem in the pipelined recursive filters derived by using the clustered look-ahead approach using an example.

Example 4.1: Consider the example of an all-pole second-order IIR filter with poles at $z = \frac{1}{2}$ and $z = \frac{3}{4}$ [see Fig. 6(a)]. This original filter is described by the transfer function

$$H(z) = \frac{1}{1 - \frac{5}{4}z^{-1} + \frac{3}{8}z^{-2}}. \tag{4.5a}$$

A 2-stage pipelined equivalent recursive digital filter can be derived by multiplying the numerator and denominaotr by $(1 + \frac{5}{4}z^{-1})$, or equivalently, by introducing a pole and a zero at $z = -\frac{5}{4}$ [see Fig. 6(b)], and is given by

$$H(z) = \frac{1 + \frac{5}{4}z^{-1}}{1 - \frac{19}{16}z^{-2} + \frac{15}{32}z^{-3}}. \tag{4.5b}$$

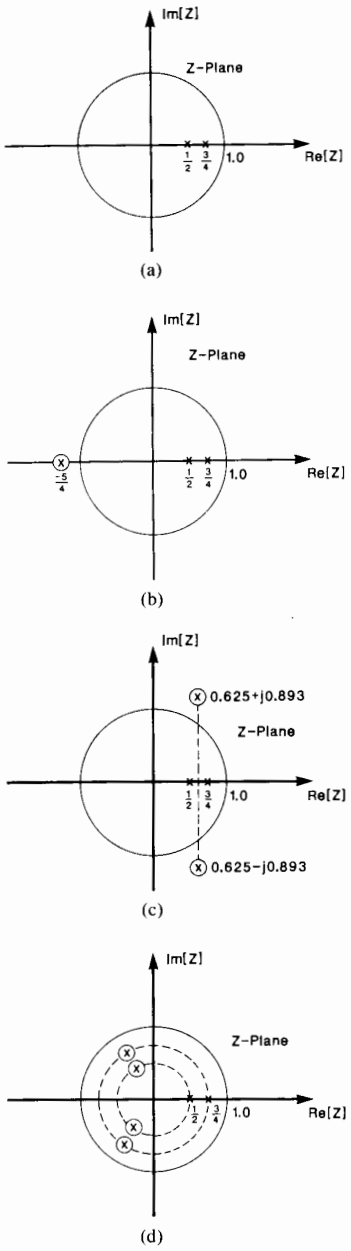

Fig. 6. (a) Pole zero representation of a stable second-order recursive filter. (b) Pole zero representation of a 2-stage pipelined equivalent unstable filter derived using clustered look-ahead approach. (c) Pole zero representation of a 3-stage pipelined equivalent unstable filter derived using clustered look-ahead approach. (d) Pole zero representation of a 3-stage pipelined equivalent stable filter derived using scattered look-ahead approach.

Similarly, a 3-stage pipelined realization can be derived by eliminating the z^{-1} and z^{-2} terms in the denominator of (4.5a), and is given by

$$H(z) = \frac{1 + \frac{5}{4}z^{-1} + \frac{19}{16}z^{-2}}{1 - \frac{65}{64}z^{-3} + \frac{57}{128}z^{-4}}, \tag{4.5c}$$

and has poles at $z = 0.5$, 0.75 and $z = 0.625 \pm j0.893$ [see Fig. 6(c)]. Note that the complex poles are outside the unity circle. Thus, both the 2- and 3-stage equivalent pipelined realizations in (4.5b) and (4.5c) are unstable, even though the original configuration of (4.5a) is stable.

B. Scattered Look-Ahead Pipelining Without Decomposition

In scattered look-ahead, the denominator of the transfer function in (4.1) is transformed in a way that it contains the N terms z^{-M}, z^{-2M}, $\cdots$, and z^{-NM}. Equivalently, the state $y(n)$ is computed in terms of N past *scattered* states $y(n-M)$, $y(n-2M)$, $\cdots$, and $y(n-NM)$. In this look-ahead, for each pole in the original filter, we introduce $(M-1)$ cancelling poles and zeros with equal angular spacing at a distance from the origin the same as that of the original pole. For example, if the original filter has a pole at $z = p$, we add $(M-1)$ poles and zeros at $z = pe^{j2\pi k/M}$ for $k = 1, 2, \cdots, (M-1)$ to derive a pipelined realization with M loop pipeline stages. The pipelining process using scattered look-ahead approach can be described by

$$H(z) = \frac{N(z)}{D(z)} = \frac{N(z)\prod_{k=1}^{M-1} D(ze^{j2\pi k/M})}{\prod_{k=0}^{M-1} D(ze^{j2\pi k/M})} = \frac{N'(z)}{D'(z^M)}. \tag{4.6}$$

Now we consider several examples to illustrate scattered look-ahead based pipelining in recursive filters.

Example 4.2: Consider the first-order filter

$$H(z) = \frac{1}{1 - az^{-1}}, \tag{4.7a}$$

which has a pole at $z = a$. A 3-stage pipelined equivalent stable filter can be derived by adding poles and zeros at $z = ae^{\pm(j2\pi/3)}$, and is given by

$$H(z) = \frac{1 + az^{-1} + a^2z^{-2}}{1 - a^3z^{-3}}. \tag{4.7b}$$

Example 4.3: Consider the second-order filter transfer function

$$H(z) = \frac{1}{1 - a_1z^{-1} - a_2z^{-2}}. \tag{4.8a}$$

A 3-stage equivalent pipelined filter is given by

$$H(z) = \frac{1 + a_1z^{-1} + (a_1^2 + a_2)z^{-2} - a_1a_2z^{-3} + a_2^2z^{-4}}{1 - (a_1^3 + 3a_1a_2)z^{-3} - a_2^3z^{-6}}. \tag{4.8b}$$

Example 4.4: Consider the second-order filter with complex conjugate poles at $z = re^{\pm j\theta}$. The transfer function of the filter is given by

$$H(z) = \frac{1}{1 - 2r\cos\theta z^{-1} + r^2z^{-2}}. \tag{4.9a}$$

We can pipeline this filter by three stages by introducing four additional poles and zeros at $z = re^{\pm j(\theta + 2\pi/3)}$, and $z = re^{\pm j(\theta - 2\pi/3)}$. The equivalent pipelined filter is given by

$$H(z) = \frac{1 + 2r\cos\theta z^{-1} + (1 + 2\cos 2\theta)r^2z^{-2} + 2r^3\cos\theta z^{-3} + r^4z^{-4}}{1 - 2r^3\cos 3\theta z^{-3} + r^6z^{-6}}. \quad (4.9b)$$

Example 4.5: Consider the second-order filter with real poles at $z = r_1$ and $z = r_2$. The transfer function is given by

$$H(z) = \frac{1}{1 - (r_1 + r_2)z^{-1} + r_1r_2z^{-2}}. \quad (4.10a)$$

A 3-stage pipelined realization is derived by adding poles (and zeros) at $z = r_1e^{\pm j2\pi/3}$ and $z = r_2e^{\pm j2\pi/3}$. The pipelined realization is given by

$$H(z) = \frac{1 + (r_1 + r_2)z^{-1} + (r_1^2 + r_1r_2 + r_2^2)z^{-2} + r_1r_2(r_1 + r_2)z^{-3} + r_1^2r_2^2z^{-4}}{1 - (r_1^3 + r_2^3)z^{-3} + r_1^3r_2^3z^{-6}}. \quad (4.10b)$$

The pole zero locations of a 3-stage pipelined second-order filter with poles at $z = 0.5$ and $z = 0.75$ is shown in Fig. 6(d).

The scattered look-ahead approach leads to stable pipelined filters if the original filter is stable, since the distance of the additional poles from the origin is the same as that of the original filter. The multiplication complexity of the nonrecursive portion is (4.6) is $(NM + 1)$, and that of the recursive portion is N, leading to a total complexity $(NM + N + 1)$ pipelined multiplications, a linear complexity with respect to M. Even though this complexity is linear with respect to M, it is much greater than that of clustered look-ahead. Also note that the latch complexity is square in M, since each multiplier is pipelined by M stages.

We now derive another pipelined realization using a *decomposition technique* which leads to a logarithmic increase in hardware with respect to speedup or increase in the sampling rate.

C. Scattered Look-Ahead Pipelining with Power-of-Two Decomposition

In [21] and [22], we proposed a decomposition algorithm to reduce the look-ahead complexity from linear to logarithmic in first-order recursive filters for the case where M is a power of 2. Now we extend power-of-two decomposition to higher order systems.

Let the recursive portion of a digital filter with K pipeline latches inside the critical loop be described by

$$H(z) = \frac{1}{1 - \sum_{i=1}^{N} q_i(K)z^{-iK}}. \quad (4.11)$$

The original transfer function corresponds to a single-stage pipelined implementation for $K = 1$, and hence $q_i(1) = a_i$. We can derive an equivalent $2K$-stage pipelined implementation by multiplying by $(1 - \Sigma_{i=1}^{N}(-1)^i q_i(K)z^{-iK})$ in the numerator and denominator. The equivalent $2K$-stage pipelined implementation is described by

$$H(z) = \frac{1 - \sum_{i=1}^{N}(-1)^i q_i(K)z^{-iK}}{\left[1 - \sum_{i=0}^{N} q_i(K)z^{-iK}\right]\left[1 - \sum_{i=0}^{N}(-1)^i q_i(K)z^{-iK}\right]} = \frac{1 - \sum_{i=0}^{N}(-1)^i q_i(K)z^{-iK}}{1 - \sum_{i=0}^{N} q_i(2K)z^{-2iK}}, \quad (4.12)$$

where the sequence $q_i(2K)$ is derived in terms of the sequence $q_i(K)$ in Appendix C.

We can apply this transformation to the original single-stage pipelined transfer function to obtain a two-stage pipelined implementation, and subsequent transformations lead to four-, eight-, and sixteen-stage pipelined implementations, respectively. Thus, to obtain an M-stage pipelined implementation, we need to apply $\log_2 M$ sets of such transformations. Each transformation leads to an increase in multiplication complexity by N while increasing the speed (or sample rate) or the number of pipeline stages inside the critical recursive loop by a factor 2. A series of such transformations lead to a geometric increase in the number of loop pipeline stages or speed while requiring only an arithmetic increase in hardware complexity!

We apply $(\log_2 M - 1)$ sets of such transformations to derive an equivalent transfer function (with M pipelining stages inside the recursive loop), which is described by

$$H(z) = \frac{\left(\sum_{i=0}^{N} b_iz^{-i}\right)\prod_{k=0}^{\log_2 M - 1}\left[1 - \sum_{i=1}^{N}(-1)^i q_i(2^k)z^{-i2^k}\right]}{1 - \sum_{i=1}^{N} q_i(M)z^{-iM}}, \quad (4.13)$$

and requires a complexity of $(2N + N \log_2 M + 1)$ multiplications, a logarithmic complexity with respect to speedup or M. Note that although the number of multiply operations is logarithmic, the number of delays or latches is linear. The total number of delays or latches is approximately $NM(\log_2 M + 1)$, out of which about NM delays are used for implementation of nonrecursive portions, and about $NM \log_2 M$ delays are required to pipeline each of the $N \log_2 M$ multipliers by M stages. This implementation has been derived by incorporating $N(M - 1)$ additional poles and zeros at identical locations. Instead of implementing the $N(M - 1)$ zeros as a single-stage nonrecursive section, we exploit the symmetry of the coefficients and implement it in a decomposed manner. In the decomposed realization, the first stage implements an Nth-order nonrecursive section, and the subsequent stages, respectively, implement $2N$, $4N$, $\cdots$, $NM/2$-order nonrecursive sections. Due to the symmetry of coefficients, each of these nonrecursive sections requires N multiplications independent of the order of that section. Now we consider examples to illustrate scattered look-ahead and decomposition based pipelining in recursive filters.

Example 4.6—First-Order Section: Consider a first-order recursive filter transfer function described by

$$H(z) = \frac{bz^{-1}}{1 - az^{-1}}. \tag{4.14a}$$

For this transfer function,

$$q(1) = a,\ q(2K) = q^2(K) = a^{2K}. \tag{4.14b}$$

The equivalent pipelined transfer function can be derived using the decomposition technique, and is described by

$$H(z) = \frac{bz^{-1} \prod_{i=0}^{\log_2 M - 1} (1 + q(2^i)z^{-2^i})}{1 - q(M)z^{-M}} = \frac{bz^{-1} \prod_{i=0}^{\log_2 M - 1} (1 + a^{2^i}z^{-2^i})}{1 - a^M z^{-M}}. \tag{4.14c}$$

This pipelined implementation has been derived by adding $(M - 1)$ poles and zeros at identical locations. The original transfer function has a single pole at $z = a$ [see Fig. 7(a)]. The pipelined transfer function has poles at locations a, $ae^{j(2\pi/M)}$, $ae^{j[2(2\pi)/M]}$, $ae^{j[3(2\pi)/M]}$, $\cdots$, $ae^{j[(M-1)(2\pi)/M]}$ (see Fig. 7(b) for $M = 8$). The decomposition of the cancelling zeros is shown in Fig. 7(c). The ith stage of the decomposed nonrecursive portion implements 2^i zeros located at

$$z = ae^{j[(2n+1)\pi/2^i]}, \qquad n = 0, 1, \cdots, (2^i - 1), \tag{4.14d}$$

and requires a single pipelined multiplication operation independent of the stage number i. The total complexity of the pipelined implementation is $(\log_2 M + 2)$ multiplications.

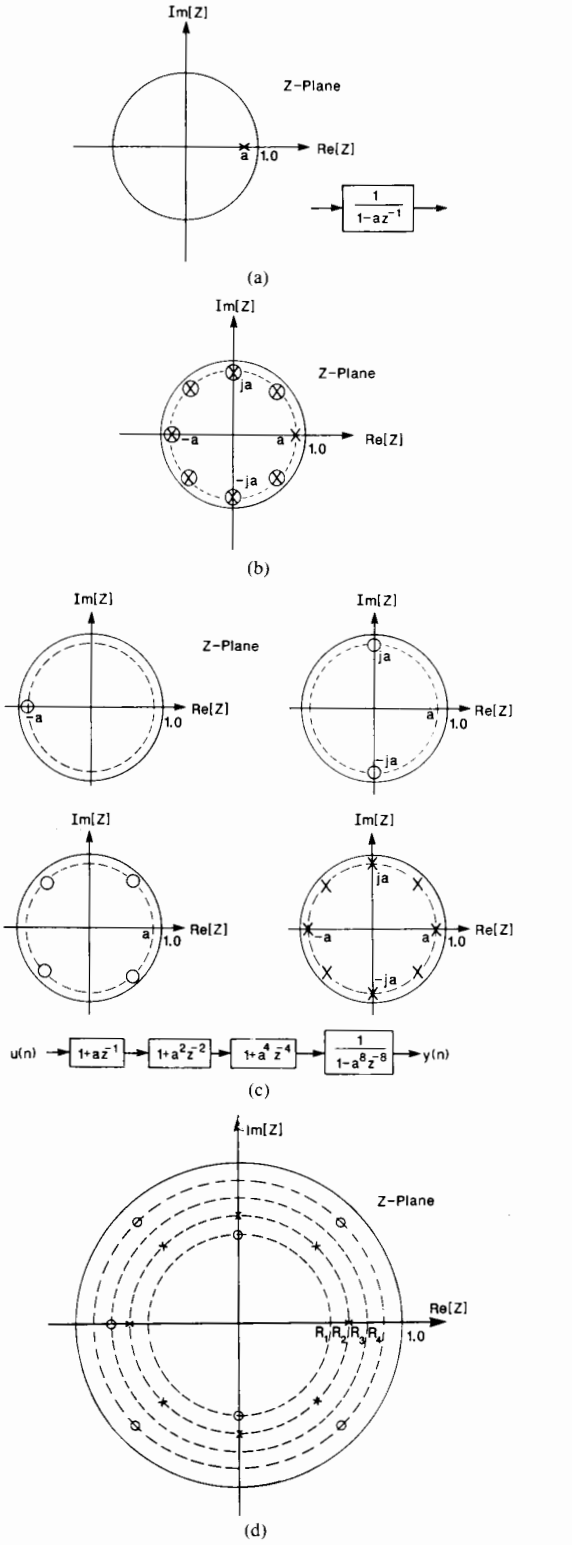

Fig. 7. (a) Pole representation of a first-order recursive filter. (b) Pole zero representation of a first-order LTI recursive system with 8 loop pipelining stages. (c) Decomposition based pipelined implementation of the first-order LTI recursive system for M equal to 8. (d) Inexact pole zero cancellation due to finite precision in a first-order look-ahead filter with eight loop pipeline stages.

The decomposition based pipelined implementation can also be equivalently explained using the time domain approach. The original recursive filter description is given by

$$y(n+1) = ay(n) + bu(n) \tag{4.15a}$$

and the pipelined realization is given by

$$y(n+M) = a^M y(n) + \sum_{i=0}^{M-1} a^i b u(n+M-1-i). \tag{4.15b}$$

As an example, for $M = 8$, we have

$$\begin{aligned} y(n+8) &= a^8 y(n) + \sum_{i=0}^{7} a^i b u(n+7-i) \\ &= a^8 y(n) + \sum_{i=0}^{7} a^i f_0(n+7-i), \\ &\qquad \text{where } f_0(n) = bu(n) \\ &= a^8 y(n) + \sum_{i=0}^{3} a^{2i} f_1(n+7-2i), \\ &\qquad \text{where } f_1(n) = af_0(n-1) + f_0(n) \\ &= a^8 y(n) + \sum_{i=0}^{1} a^{4i} f_2(n+7-4i), \\ &\qquad \text{where } f_2(n) = a^2 f_1(n-2) + f_1(n). \end{aligned} \tag{4.15c}$$

Although the pipelined recursive filter realizations are stable under infinite precision conditions, they are sensitive to filter coefficients under finite precision. In a finite precision implementation, the poles of the first-order M-stage pipelined filter are located at

$$p = (a^M + \Delta)^{1/M} \approx a\left(1 + \frac{\Delta}{Ma^M}\right),$$

where Δ corresponds to the finite precision error in representing a^M. This pole location is more sensitive for smaller values of a (that is, when poles are closer to the origin). Fortunately this is not a problem, since the instability problem for the filter with poles closer to origin is not severe.

In addition to the sensitivity problem, finite precision pipelined filters suffer from inexact pole zero cancellation [see Fig. 7(d)], which leads to magnitude and phase error. These errors can be reduced by increasing word length, but a thorough analysis of this is beyond the scope of this paper.

Example 4.7—Second-Order System: Consider a second-order recursive filter described by

$$H(z) = \frac{Y(z)}{U(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - 2r\cos\theta z^{-1} + r^2 z^{-2}}. \tag{4.16a}$$

The poles of the system are located at $re^{\pm j\theta}$ [see Fig. 8(a)]. For this filter description,

$$q_1(1) = 2r\cos\theta, \qquad q_2(1) = -r^2, \tag{4.16b}$$

and

$$q_1(2K) = q_1^2(K) + 2q_2(K) = 2r^{2K}\cos 2K\theta \tag{4.16c}$$

$$q_2(2K) = -q_2^2(K) = -r^{4K}. \tag{4.16d}$$

The pipelined transfer function can be described by

$$\begin{aligned} H(z) &= \frac{\left(\sum_{i=0}^{2} b_i z^{-i}\right) \prod_{i=0}^{\log_2 M - 1} \left(1 + q_1(2^i) z^{-2^i} - q_2(2^i) z^{-2^{i+1}}\right)}{1 - q_1(M) z^{-M} - q_2(M) z^{-2M}} \\ &= \frac{\left(\sum_{i=0}^{2} b_i z^{-i}\right) \prod_{i=0}^{\log_2 M - 1} \left(1 + 2r^{2^i}\cos 2^i\theta z^{-2^i} + r^{2^{i+1}} z^{-2^{i+1}}\right)}{1 - 2r^M \cos M\theta z^{-M} + r^{2M} z^{-2M}}. \end{aligned} \tag{4.17}$$

The $2M$ poles of the transformed transfer function are located at

$$z = re^{\pm j(\theta + i(2\pi/M))}, \qquad i = 0, 1, 2, \cdots, (M-1),$$

and are shown in Fig. 8(b). The decomposed implementation of the pipelined filter is shown in Fig. 8(c) and (d). The pipelined filter can be implemented with an implementation complexity of $(2\log_2 M + 5)$ multiplications. The quantization error due to the recursive portion of a pipelined second-order section is studied in Appendix D, and it is shown that the upper bound on the quantization error in the pipelined filter decreases with increase in number of loop pipeline stages. Intuitively, this should be

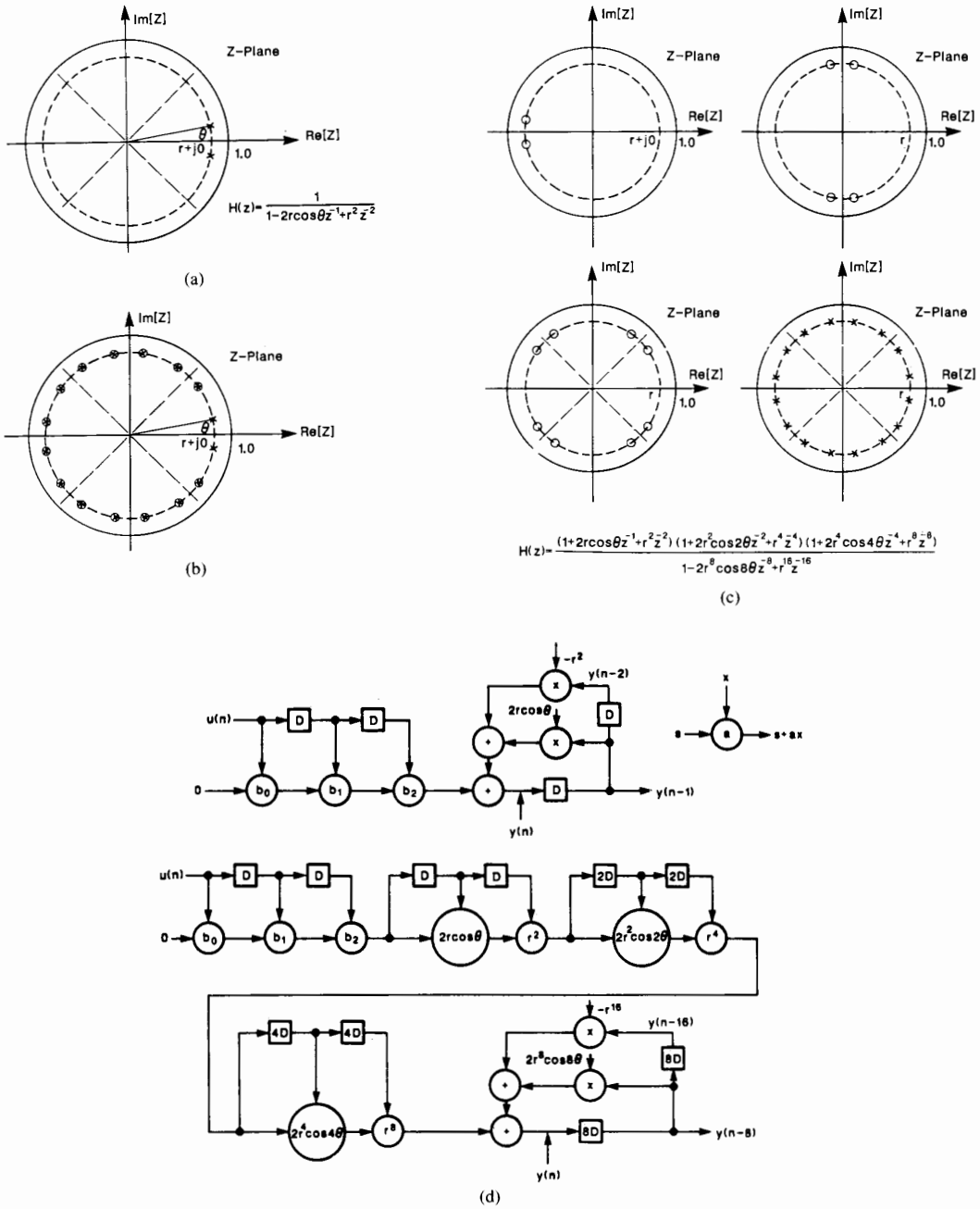

Fig. 8. (a) Pole diagram of the second-order filter. The zeros of the filter have not been shown for clarity. (b) Pole zero representation of the pipelined second-order direct form filter with 8 loop pipelining stages. (c) Decomposition of poles and zeros of the pipelined second-order filter. (d) Implementation of the original second-order filter and the pipelined scattered look-ahead recursive filtering using decomposition technique for 8 pipelining stages inside the recursive loop.

expected, since as M increases, the IIR filter more closely approximates an FIR filter, for which the quantization error is inherently less.

A single-chip implementation of a fourth-order recursive digital filter (organized as two cascaded second-order sections) using four stages of loop pipelining and running at one-hundred-million samples per second sample rate, has been reported in [56]. This chip uses the scattered

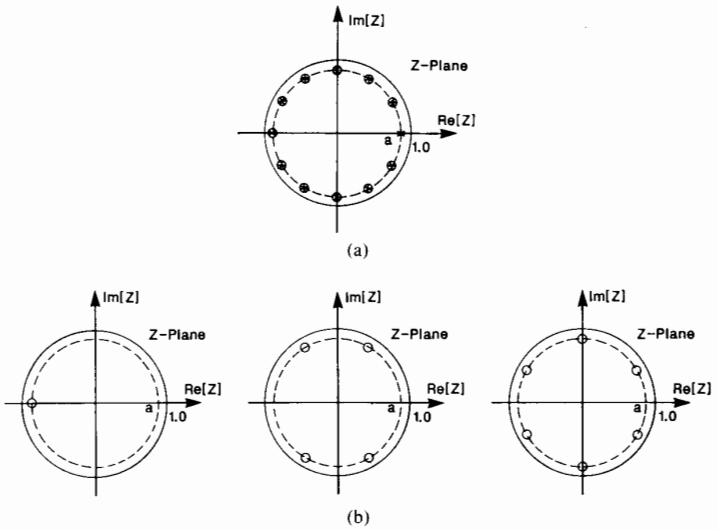

Fig. 9. (a) Pole zero location of a 12-stage pipelined first-order recursive filter. (b) Decomposition of the zeros of the pipelined filter for a 2 × 3 × 2 decomposition.

look-ahead and the decomposition algorithms developed in this paper. The chip is implemented in 0.9 μm double-layer metal CMOS technology at the AT&T Bell Laboratories. It uses a silicon area of 14 mm^2, and has a transistor density of 0.6 million devices per cm^2. The total computing power of the chip is 1.7 billion multiply operations per second. The reader is referred to [56] for details of the integrated circuit chip implementation aspects.

D. Scattered Look-Ahead Pipelining with General Decomposition

We have so far concentrated on power-of-two decompositions only, which leads to hardware-efficient implementations. However, the decomposition of cancelling zeros extends for any arbitrary number of loop pipeline stages. The time-domain interpretation of simple M_1M_2 decomposition was studied in [33]. We now illustrate decomposition of cancelling zeros for arbitrary decomposition of M.

In an Nth-order filter with M-levels of pipelining, there are $N(M-1)$ cancelling zeros. First consider the simple case of $M = M_1M_2$ decomposition [33]. In this implementation, the system has $N(M_1M_2 - 1)$ cancelling zeros. The first stage implements $N(M_1 - 1)$ zeros, and the second stage implements $NM_1(M_2 - 1)$ zeros. In an $M_1M_2M_3$ decomposition, the first stage implements $N(M_1 - 1)$ zeros, the second stage implements $NM_1(M_2 - 1)$ zeros, and the third stage implements $NM_1M_2(M_3 - 1)$ zeros. In general, in an $M = M_1M_2 \cdots M_P$ decomposition, the P nonrecursive stages, respectively, implement $N(M_1 - 1), NM_1(M_2 - 1), \cdots, NM_1M_2 \cdots M_{P-1}(M_P - 1)$ zeros, totaling $N(M-1)$ zeros. The nonrecursive portion of the general decomposition requires about NM delays and $N\sum_{i=1}^{P}(M_i - 1)$ multipliers (each of these multipliers also requires M latches for pipelining).

Example 4.8: Consider the first-order transfer function in [4.7(a)]. A 12-stage pipelined decomposed implementation is given by

$$H(z) = \frac{\sum_{i=0}^{11} a^i z^{-i}}{(1 - a^{12}z^{-12})} = \frac{(1 + az^{-1})(1 + a^2z^{-2} + a^4z^{-4})(1 + a^6z^{-6})}{(1 - a^{12}z^{-12})}. \tag{4.18a}$$

The above implementation corresponds to a 2 × 3 × 2 decomposition. The pole-zero configuration of the 12-stage pipelined filter is shown in Fig. 9(a). The decomposition of 11 cancelling zeros of this filter is shown in Fig. 9(b), where the three sections, respectively, implement 1, 4, and 6 zeros, respectively. Here the first section implements the zero at $-a$, the second section implements four zeros at $ae^{\pm j\pi/3}$ and $ae^{\pm j2\pi/3}$, and the third section implements six zeros at $\pm ja$, $ae^{\pm j\pi/6}$, and $ae^{\pm j5\pi/6}$. Another decomposed transfer function given by

$$H(z) = \frac{(1 + az^{-1})(1 + a^2z^{-2})(1 + a^4z^{-4} + a^8z^{-8})}{(1 - a^{12}z^{-12})} \tag{4.18b}$$

corresponds to 2 × 2 × 3 decomposition. In this implementation, the first nonrecursive section implements one zero at $-a$, the second section implements two zeros at $\pm ja$, and the third section implements eight zeros at $ae^{\pm j\pi/6}$, $ae^{\pm j\pi/3}$, $ae^{\pm j2\pi/3}$, and $ae^{\pm j5\pi/6}$. The 3 × 2 × 2 decomposition is given by

$$H(z) = \frac{(1 + az^{-1} + a^2z^{-2})(1 + a^3z^{-3})(1 + a^6z^{-6})}{(1 - a^{12}z^{-12})}, \tag{4.18c}$$

and the three sections, respectively, implement 2, 3, and 6 zeros. The first section implements two zeros at $ae^{\pm j2\pi/3}$, the second implements three zeros at $-a$ and $ae^{\pm j\pi/3}$, and the third section implements six zeros at $ae^{\pm j\pi/6}$, $\pm ja$, and $ae^{\pm j5\pi/6}$.

Any higher order recursive filter can be factored in terms of first-order sections. Decomposition similar to the above example can be applied to the first-order sections, and then the complex conjugate sections can be combined to obtain the decomposed form in terms of real multiplications.

E. Linear Bidirectional Systolic Array Architectures

All bidirectional systolic array implementations of pipelined recursive digital filters presented so far require many-way interleaving [1], [43]–[47]. In this section, we derive linear bidirectional pipelined systolic arrays for direct form recursive digital filters using the scattered look-ahead algorithm. These arrays are highly concurrent, fully pipelined, and do not require any interleaving of input samples. Since the nonrecursive portion can be implemented with arbitrary level of pipelining, we restrict our attention to only the recursive portion.

Consider the recursive algorithm described by

$$y(n) = \sum_{i=1}^{N} q_i(M)\, y(n - iM) + x(n), \quad (4.19)$$

where $x(n)$ corresponds to the output of the nonrecursive portion. This algorithm corresponds to an M-stage pipelined implementation. A flow graph corresponding to the above algorithm is shown in Fig. 10(a). For $M = 1$, the bidirectional array cannot be fully pipelined without requiring interleaving. However, a pipelined interleaved version can be achieved, which is useful for applications requiring moderate amount of concurrency, and where multiple independent time series need to be filtered similarly in an interleaved manner.

For $M \geq 2$, a fully pipelined systolic array can be implemented. In this implementation, all the processing elements operate in a pipelined manner, and the operations inside each processing element can also be deeply pipelined. The M delays or latches can be moved around the loop to pipeline interstage operations as well as the multiplication/addition operation (intrastage pipeline). Out of M delays, 2 delays are used for interstage pipelining, and the $(M - 2)$ delays are used to pipeline the multiplication/addition operation inside each stage. This technique of moving around delays at will without changing the input–output behavior is referred to as retiming or cutset transformation in the literature [1], [53]–[55]. The pipelined linear systolic array implementation is shown in Fig. 10(b).

F. FIR versus IIR Filters

We can start with frequency domain specifications of a digital filter, and implement the filter as an FIR or an IIR filter. Let the order of an FIR filter be N_{FIR} and the order

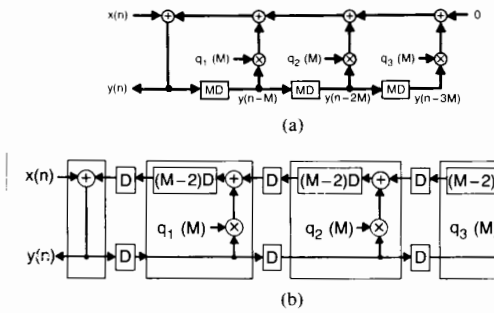

Fig. 10. (a) Linear systolic implementation of a recursive filter. (b) Pipelined linear bidirectional systolic array implementation of a recursive filter.

of an IIR filter to satisfy the same requirement be N_{IIR}. For the same speed (or equivalently, for same level of pipelining), the complexity of the FIR filter in terms of M-stage pipelined multipliers is N_{FIR}, and that for the IIR filter is $(2N_{IIR} + N_{IIR} \log_2 M + 1)$. Hence, the IIR filter realization is preferable if

$$2N_{IIR} + N_{IIR} \log_2 M + 1 < N_{FIR}, \quad (4.20a)$$

or equivalently, if

$$M < 2^{\lfloor (N_{FIR} - 1/N_{IIR}) - 2 \rfloor}, \quad (4.20b)$$

where $\lfloor x \rfloor$ represents the floor function of x. As an example, in Rabiner *et al.* [57], it is shown that a filter spectrum can be implemented as a 6th-order IIR filter or as a 41st-order FIR filter. Then for this filter, M must be less than 16 for the IIR filter to be hardware efficient as compared to its FIR counterpart.

V. Pipelining in State Space Filters

The clustered look-ahead and scattered look-ahead processes are identical for the state space filter. Pipelining in state space filters using the look-ahead computation technique (without the use of decomposition) was introduced in [19] at the expense of a linear increase in complexity with respect to loop pipeline stages. In this section, we derive a decomposition based pipelined realization for state space recursive digital filters of logarithmic complexity with respect to the number of loop pipeline stages.

Consider the state space recursive filter described by

$$\boldsymbol{x}(n + 1) = \boldsymbol{A}\boldsymbol{x}(n) + \boldsymbol{b}u(n) \quad (5.1a)$$

$$y(n) = \boldsymbol{c}^T\boldsymbol{x}(n) + du(n), \quad (5.1b)$$

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. 11(a) shows a block diagram corresponding to (5.1). The transfer function of the state space filter is given by

$$H(z) = \boldsymbol{c}^T(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} + d. \quad (5.1c)$$

However, the state space representation of any transfer function is not unique. The transfer function remains unaltered if the state space representation undergoes a sim-

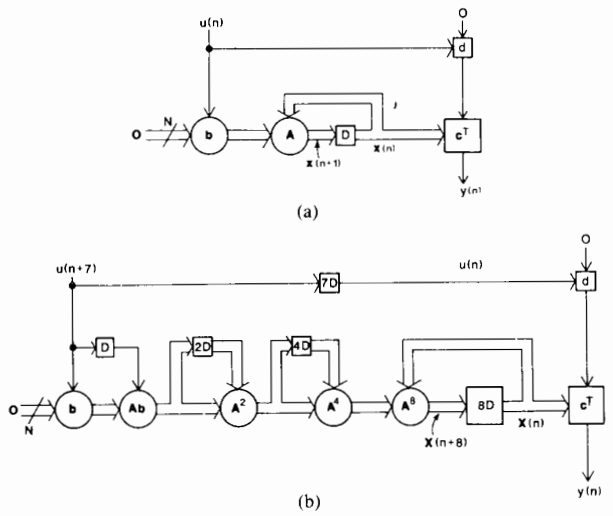

Fig. 11. (a) A state space recursive digital filter. (b) A pipelined state space recursive digital filter with 8 loop pipelining stages obtained using the decomposition algorithm.

ilarity transformation

$$\boldsymbol{x} \rightarrow \boldsymbol{\Lambda}^{-1}\boldsymbol{x};\ \boldsymbol{A},\ \boldsymbol{b},\ \boldsymbol{c},\ d \rightarrow \boldsymbol{\Lambda}^{-1}\boldsymbol{A}\boldsymbol{\Lambda},\ \boldsymbol{\Lambda}^{-1}\boldsymbol{b},\ \boldsymbol{\Lambda}^{T}\boldsymbol{c},\ d. \tag{5.1d}$$

The complexity of the implementation will depend upon the number of nonzero elements in the state update matrix, which in turn depends upon the form of digital filter realization. A parallel realization of first-order sections with real coefficients can be described in terms of a diagonal state update matrix, and a cascaded realization of these sections can be described by a triangular state update matrix. Second-order sections can be described by a quasi-diagonal state update matrix when implemented in a parallel manner, or a quasi-triangular matrix when implemented in a cascade manner. State space representation of lattice filters can be described by a quasi-triangular state update matrix. Full, triangular, and quasi-triangular state update matrices lead to $O(N^2)$ multiplication complexity, whereas diagonal and quasi-diagonal matrices lead to $O(N)$ multiplication complexity, where N is the order of the filter. In what follows, we assume the filter to be described by a quasi-diagonal state update matrix, i.e., the filter has no real pole of multiplicity greater than two, and no complex pole of multiplicity greater than unity. Similar results can be easily derived for all other configurations.

Let the maximum number of nonzero elements among all rows of the state update matrix be N'. Then the iteration period of this implementation corresponds to the time required for a single multiplication and N' additions, and the sample rate corresponds to the reciprocal of the iteration period. Applying M steps of look-ahead, we obtain an equivalent M-stage pipelined algorithm

$$\boldsymbol{x}(n+M) = \boldsymbol{A}^{M}\boldsymbol{x}(n) + \sum_{i=0}^{M-1} \boldsymbol{A}^{i}\boldsymbol{b}\boldsymbol{u}(n+M-1-i), \tag{5.2}$$

which has an iteration period bound (sample rate) M times lower (higher) than the original algorithm. The output equation (5.1b) is nonrecursive, and hence does not require any transformation. Let N be the filter order, and N_1 represent the number of real first-order poles. Then the state update implementation complexity in (5.2) corresponds to $(NM + 2N - N_1)$ multiplications, and the output computation complexity in (5.1b) corresponds to $(N + 1)$ multiplications. Thus, the total complexity is $(NM + 3N + 1 - N_1)$ multiplications, which is linear with respect to the number of pipeline stages M.

We can use decomposition to reduce the implementation complexity. The power-of-two decomposed pipelined state update realization is described by

$$\boldsymbol{z}_1(n+M-1) = \boldsymbol{b}\boldsymbol{u}(n+M-1) + \boldsymbol{A}\boldsymbol{b}\boldsymbol{u}(n+M-2) \tag{5.3a}$$

$$\boldsymbol{z}_{i+1}(n+M-1) = \boldsymbol{z}_i(n+M-1) + \boldsymbol{A}^{2^i}\boldsymbol{z}_i(n+M-1-2^i), \quad i = 1, 2, \cdots, (\log_2 M - 1) \tag{5.3b}$$

$$\boldsymbol{x}(n+M) = \boldsymbol{A}^{M}\boldsymbol{x}(n) + \boldsymbol{z}_{\log_2 M}(n+M-1), \tag{5.3c}$$

and is shown in Fig. 11(b) (for $M = 8$). This pipelined algorithm leads to a multiplication complexity $[2N(\log_2 M + 1) - N_1 \log_2 M]$ for state update implementation, and $(N + 1)$ for output computation; thus leading to a total complexity of $[2N(\log_2 M + \frac{3}{2}) - N_1 \log_2 M + 1]$ multiplications, which is logarithmic with respect to the number of pipeline stages.

The roundoff error in the state space pipelined filter is studied in Appendix E, and is shown to improve monotonically with increase in the number of loop pipeline stages.

VI. Time Varying Recursive Filtering

Pipelined implementation of first-order time varying recursive digital filter using look-ahead and decomposition was studied in [21] and [22]. In this section we illustrate the application of the scattered look-ahead and decomposition techniques in the context of time varying pipelined recursive filter realizations using a second-order filter as an example. Since time varying systems cannot be represented in terms of transfer function descriptions, we resort to the time domain decomposition approach.

Example 6.1—Second-Order Example: Consider the second-order time varying recursive filter example

$$x(n) = a_1(n)\,x(n-1) + a_2(n)\,x(n-2) + u(n). \tag{6.1a}$$

We can use the scattered look-ahead approach to express $x(n)$ as a function of $x(n-M)$ and $x(n-2M)$, and exploit the decomposition technique to obtain a logarithmic complexity. After some manipulation, the M-stage

pipelined filter is

$$x(n+M) = f_{\log_2 M}(n+M)\,x(n) + g_{\log_2 M}(n+M)\,x(n-M) + z_{\log_2 M}(n+M), \quad (6.1b)$$

where

$$f_{i+1}(n+M) = f_i(n+M)\,f_i(n+M-2^i) + g_i(n+M) + \frac{f_i(n+M)\,g_i(n+M-2^i)}{f_i(n+M-2^{i+1})} \quad (6.1c)$$

$$g_{i+1}(n+M) = -\frac{f_i(n+M)\,g_i(n+M-2^i)\,g_i(n+M-2^{i+1})}{f_i(n+M-2^{i+1})} \quad (6.1d)$$

$$z_{i+1}(n+M) = z_i(n+M) + f_i(n+M)\cdot z_i(n+M-2^i) - \frac{f_i(n+M)\,g_i(n+M-2^i)}{f_i(n+M-2^{i+1})}\cdot z_i(n+M-2^{i+1}) \quad (6.1e)$$

$i = 0, 1, \cdots, (\log_2 M - 1)$.

$$f_0(n+M) = a_1(n+M)$$
$$g_0(n+M) = a_2(n+M),$$
$$z_0(n+M) = u(n+M). \quad (6.1f)$$

This realization can be implemented with $(5\log_2 M + 2)$ multipliers (each pipelined by M stages), and $\log_2 M$ pipelined dividers. Note that although the original realization did not require a divider, the pipelined realization does require dividers for the scattered look-ahead technique to be applicable.

A. Conclusions

We have presented new scattered look-ahead and decomposition techniques to derive area-efficient fine-grain pipelined recursive filter realizations. To the best of our knowledge, this is the first successful attempt in the design of stable fine-grain pipelined recursive filters.

Another approach to pipeline recursive filters using signed-digit redundant arithmetic has recently been proposed in [58] and [59]. Our approaches can be combined with the approach in [58] and [59] to pipeline signed-digit recursive filters at bit or digit level with minimum look-ahead. A drawback of the signed-digit representations is that they require a longer internal word length for a specified dynamic range (compared to two's-complement representation), and therefore require higher silicon area. These representations may also suffer from degraded performance due to overflow problems. These issues require further study.

The pipelinability criteria derived in this section can be used to synthesize pipelined filter transfer functions directly from frequency domain specifications thereby eliminating the intermediate transformation procedure. The iterative techniques have been successfully used in the context of traditional recursive filter design [60]–[62]. We hope that by appropriately constraining these iterative techniques, we can satisfy the pipelinability criteria and design pipelined filters directly from spectrum specifications. In this context, earlier techniques used in decimation filter designs (which only contain z^{-kM} terms in the denominator) may also be applicable [63]. Further research is needed for synthesis of pipelined recursive filters using constrained iterative design techniques.

The pipelined algorithm described in this section suffers from inexact cancellation of poles and zeros, which will lead to error in magnitude and phase response of the filter. However, the word length can be increased to minimize this error. The word length and roundoff error tradeoffs in the pipelined filters require further study. The inexact pole zero cancellation may lead to problems for filters with poles close to unit circle. Since the coefficients of the pipelined filter correspond to higher power of the original coefficients, they are too small, and may require a larger word length than the original coefficients (especially for large M). This is another issue that needs further study. Fortunately, using current technologies, we can achieve sample rates up to 100 MHz with M as small as four [56].

The ultimate speed in the pipelined implementations will be limited by practical limitations such as clock skew, packaging delay, etc. Hence, once pipelining is used to maximum possible extent, we need to combine pipelining with block processing to achieve further speedup in the sample rate. In the companion paper [18], we combine pipelining and incremental block filtering approaches to derive area-efficient architectures for direct form and state space form recursive digital filters.

Appendix A

In this appendix, we define the sequence r_i, and study some related recursive relations. The sequence r_i is useful in the context of clustered look-ahead based pipelined and/or block implementation of direct form recursive digital filters. For an Nth-order direct form recursive digital filter, we define

$$r_{-i} = 0 \quad \text{for } i = 1, 2, \cdots, (N-1), \qquad r_0 = 1,$$
$$r_i = \sum_{k=1}^{N} a_k r_{i-k}, \qquad i > 0. \quad (A1.1)$$

For the sequence r_i, we can prove the following theorem.

Theorem A1.1: The values of r_{L+m} can be computed using

$$r_{L+m} = \sum_{k=0}^{N-1} \left[\sum_{j=0}^{k} a_{N-j} r_{m+j-k} \right] r_{L-N+k}. \quad (A1.2)$$

Proof (by induction): Assume the theorem to hold for m, and prove that the theorem also holds for $m + 1$. The value of r_{L+m+1} is given by

$$\begin{aligned} r_{L+m+1} &= \sum_{l=1}^{N} a_l \left[\sum_{k=0}^{N-1} \left\{ \sum_{j=0}^{k} a_{N-j} r_{m-l+j-k+1} \right\} r_{L-N+k} \right] \\ &= \sum_{k=0}^{N-1} \left[\sum_{l=1}^{N} \sum_{j=0}^{k} a_l a_{N-j} r_{m-l+j-k+1} \right] r_{L-N+k} \\ &= \sum_{k=0}^{N-1} \left[\sum_{j=0}^{k} a_{N-j} \sum_{l=1}^{N} a_l r_{m-l+j-k+1} \right] r_{L-N+k}, \end{aligned}$$

from which (A1.2) follows.

Appendix B
Pipelining with Clustered Look-Ahead

In this appendix, we derive a pipelined realization of the direct form recursive digital filter using the clustered look-ahead computation technique. We also study the time domain and frequency domain interpretations of this transformation.

Time Domain Analysis

Theorem A2.1: Any direct form recursive digital filter of the form (4.2) can be equivalently described by

$$\begin{aligned} y(n) = &\sum_{j=0}^{N-1} \left[\sum_{k=j+1}^{N} a_k r_{j+M-k} \right] y(n-j-M) \\ &+ \sum_{j=0}^{M-1} r_j z(n-j), \end{aligned} \tag{A2.1}$$

which corresponds to an M-stage pipelined realization.

Proof (by induction): Assume the above to be true for M, and prove that it also holds for $M + 1$. The above expression can be rewritten as

$$\begin{aligned} y(n) &= r_M y(n-M) + \sum_{j=1}^{N-1} \left[\sum_{k=j+1}^{N} a_k r_{j+M-k} \right] \\ &\quad \cdot y(n-j-M) + \sum_{j=0}^{M-1} r_j z(n-j) \\ &= r_M \left[\sum_{j=1}^{N} a_j y(n-M-j) + z(n-M) \right] \\ &\quad + \sum_{j=1}^{N-1} \left[\sum_{k=j+1}^{N} a_k r_{j+M-k} \right] y(n-M-j) \\ &\quad + \sum_{j=0}^{M-1} r_j z(n-j) \\ &= \sum_{j=1}^{N} \left[\sum_{k=j}^{N} a_k r_{j+M-k} \right] y(n-M-j) \\ &\quad + \sum_{j=0}^{M} r_j z(n-j), \end{aligned} \tag{A2.2}$$

from which (A2.1) follows.

Frequency Domain Analysis

Theorem A2.2: Any transfer function of the form (4.1) is equivalent to the form (4.4).

Proof: Multiply the numerator and denominator by an $(M-1)$-order polynomial $\Sigma_{j=0}^{M-1} r_j z^{-j}$. The denominator $D(z)$ can be expressed as

$$\begin{aligned} D(z) &= \left[1 - \sum_{i=1}^{N} a_i z^{-i} \right] \left[1 + \sum_{j=1}^{M-1} r_j z^{-j} \right] \\ &\quad \text{since } r_0 = 1 \\ &= 1 - \sum_{i=1}^{N} a_i z^{-i} + \sum_{i=1}^{M-1} r_i z^{-i} \\ &\quad - \sum_{i=1}^{N} \sum_{j=1}^{M-1} a_i r_j z^{-(i+j)}. \end{aligned} \tag{A2.3}$$

The last term of the right-hand side of the above equation is given by

$$\begin{aligned} &\sum_{i=1}^{N} \sum_{j=1}^{M-1} a_i r_j z^{-(i+j)} \\ &= \begin{cases} \displaystyle\sum_{i=2}^{M-1} \left[\sum_{j=1}^{i-1} a_j r_{i-j} \right] z^{-i} + \sum_{i=M}^{N} \left[\sum_{j=1}^{M-1} a_{i-j} r_j \right] z^{-i} \\ \quad \displaystyle + \sum_{i=N+1}^{N+M-1} \left[\sum_{j=i-N}^{M-1} a_{i-j} r_j \right] z^{-i}, \qquad N \geq M \\ \displaystyle\sum_{i=2}^{N} \left[\sum_{j=1}^{i-1} a_j r_{i-j} \right] z^{-i} + \sum_{i=N+1}^{M-1} \left[\sum_{j=1}^{N} a_j r_{i-j} \right] z^{-i} \\ \quad \displaystyle + \sum_{i=M}^{N+M-1} \left[\sum_{j=i-M+1}^{N} a_j r_{i-j} \right] z^{-i}, \qquad N < M \end{cases} \end{aligned} \tag{A2.4}$$

from which (4.4) follows.

Appendix C

Theorem A3.1: The sequence $q_i(2K)$ is related to the sequence $q_i(K)$ by the following relations:

$$\begin{aligned} q_1(2K) &= q_1^2(K) + 2q_2(K), \\ q_N(2K) &= (-1)^{N+1} q_N^2(K). \end{aligned} \tag{A3.1}$$

For even filter order N,

$$q_i(2K) = \begin{cases} 2q_{2i}(K) + (-1)^{i+1} q_i^2(K) \\ \quad + 2 \displaystyle\sum_{j=1}^{i-1} (-1)^{j+1} q_j(K) q_{2i-j}(K), \\ \quad i = 2, \cdots, \dfrac{N}{2} \\ (-1)^{i+1} q_i^2(K) + 2 \displaystyle\sum_{j=i+1}^{N} (-1)^{j+1} \\ \quad \cdot q_j(K) q_{2i-j}(K), \\ \quad i = \dfrac{N}{2} + 1, \cdots, N-1. \end{cases} \tag{A3.2}$$

For odd filter order N,

$$q_i(2K) = \begin{cases} 2q_{2i}(K) + (-1)^{i+1} q_i^2(K) + 2\sum_{j=1}^{i-1} (-1)^{j+1} q_j(K)\, q_{2i-j}(K), & i = 2, \cdots, \dfrac{N-1}{2} \\ (-1)^{i+1} q_i^2(K) + 2\sum_{j=1}^{i-1} (-1)^{j+1} \cdot q_j(K)\, q_{2i-j}(K), & i = \dfrac{N+1}{2} \\ (-1)^{i+1} q_i^2(K) + 2\sum_{j=i+1}^{N} (-1)^{j+1} \cdot q_j(K) q_{2i-j}(K), & i = \dfrac{N+3}{2}, \cdots, N-1. \end{cases} \quad \text{(A3.3)}$$

Proof: We have

$$\begin{aligned} 1 - \sum_{i=1}^{N} q_i(2K) z^{-2iK} &= \left[1 - \sum_{i=1}^{N} q_i(K) z^{-iK}\right]\left[1 - \sum_{i=1}^{N} (-1)^i q_i(K) z^{-iK}\right] \\ &= 1 - 2\sum_{i=1}^{\lfloor N/2 \rfloor} q_{2i}(K) z^{-2iK} + \sum_{i=1}^{N} (-1)^i q_i^2(K) z^{-2iK} \\ &\quad + 2\sum_{i=2}^{\lceil N/2 \rceil} \sum_{j=1}^{i-1} (-1)^j q_j(K)\, q_{2i-j}(K) z^{-2iK} \\ &\quad + 2\sum_{i=\lceil N/2 \rceil + 1}^{N-1} \sum_{j=i+1}^{N} (-1)^j q_j(K)\, q_{2i-j}(K) z^{-2iK}. \end{aligned}$$

Matching the powers of z, relations (A3.2) and (A3.3) can be derived.

Appendix D

In this appendix, we study the quantization error due to the recursive portion of a pipelined second-order recursive filters derived by using scattered look-ahead and decomposition techniques. We show that as the number of pipeline stages inside the recursive loop increases, the upper bound on the quantization error of the pipelined filter strictly decreases. Consider the second-order recursive filter transfer function

$$H(z) = \frac{1}{(1 - re^{j\theta} z^{-1})(1 - re^{-j\theta} z^{-1})} = \frac{1}{1 - q_1(1) z^{-1} - q_2(1) z^{-1}}. \quad \text{(A4.1)}$$

In the pipelined filter, $2(M-1)$ poles are introduced on the circle r units apart in the Z-plane. Since the distance of the additional poles remains unaltered from the origin, the pipelined filter remains stable if the original filter is stable. The recursive portion of the pipelined transfer function with M pipeline stages inside the recursive loop is given by

$$H_1(z) = \frac{1}{\prod_{i=0}^{M-1} (1 - re^{j(\theta + (2\pi i/M))} z^{-1})(1 - re^{-j(\theta - (2\pi i/M))} z^{-1})}. \quad \text{(A4.2)}$$

The quantization error of this filter can be derived by evaluating the residues of $H_1(z)\, H_1(z^{-1})/z$ for all poles inside the unit circle, and then taking the sum. We derive the quantization error of this filter in three steps.

Step 1: Consider the pole p_i at $re^{j(\theta + (2\pi i/M))}$, and let its residue be denoted as R_{i1}. Let R_{i2} denote the residue at the complex conjugate pole of p_i. Then we can prove that R_{i2} is the complex conjugate of R_{i1}. This proof is straightforward and is omitted.

Step 2: In the pipelined filter, $(M-1)$ additional poles are introduced for each pole in the original filter. We show that the residue at each of the added poles is identical to that at the corresponding pole of the original filter.

Proof: For the pipelined filter, we have

$$H_1(z)\, H_1(z^{-1}) = \frac{1}{\prod_{i=0}^{M-1} (1 - re^{j(\theta + (2\pi i/M))} z^{-1})(1 - re^{-j(\theta - (2\pi i/M))} z^{-1})} \times \frac{1}{\prod_{i=0}^{M-1} (1 - re^{j(\theta + (2\pi i/M))} z)(1 - re^{-j(\theta - (2\pi i/M))} z)}. \quad \text{(A4.3)}$$

We derive the residue at the pole for $i = I$ and show that this residue is independent of I. After some manipulation, the residue R_{I1} can be derived to be

$$R_{I1} = \frac{1}{\prod_{i=1}^{M-1} (1 - e^{j2\pi i/M}) \prod_{i=0}^{M-1} [(1 - r^2 e^{j(2\theta + (2\pi i/M))})(1 - e^{j(2\theta + (2\pi i/M))})(1 - r^2 e^{j(2\pi i/M)})]} = \frac{1}{M(1 - r^{2M})(1 - r^{2M} e^{j2M\theta})(1 - e^{-j2M\theta})}, \quad \text{(A4.4)}$$

which is independent of I.

Step 3: The total quantization error is proportional to (also referred to as normalized error)

$$E = M(R_{i1} + R_{i2}) = \frac{1 + r^{2M}}{1 - r^{2M}} \frac{1}{1 - 2r^{2M}\cos 2M\theta + r^{4M}}. \quad \text{(A4.5)}$$

In terms of the coefficients of the pipelined filter, the normalized error is given by

$$E = \frac{1 - q_2(M)}{1 + q_2(M)} \frac{1}{1 - q_1(2M) - q_2(2M)}. \quad \text{(A4.6)}$$

The upper bound on the error expression in (A4.5) strictly decreases with increase in M. Furthermore, the expression (A4.6) also holds for the case of two real poles. A similar analysis can be carried out for any arbitrary order recursive digital filter, and the quantization error of the pipelined realization can be shown to be bounded.

Appendix E

In this appendix, we study the roundoff noise error in pipelined state space filters, and show that the roundoff error strictly improves with increase in number of loop pipeline stages M. We assume the roundoff operation to be performed at the output of the state variables and at system outputs. The noise sources are assumed to be white stationary with zero mean, and are assumed to be statistically independent of signals.

The output error at the summing node of the pipelined state space filter with M loop pipeline stages is given by

$$\tilde{\boldsymbol{x}}(n + M) = \boldsymbol{A}^M \tilde{\boldsymbol{x}}(n) + \boldsymbol{e}_s(n), \quad \text{(A5.1)}$$

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

$$E[\boldsymbol{e}_s \boldsymbol{e}_s^T] = \sigma_0^2 \boldsymbol{I}_N. \quad \text{(A5.2)}$$

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

$$\boldsymbol{Q} = E[\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^T] = \boldsymbol{A}^M \boldsymbol{Q} (\boldsymbol{A}^T)^M + \sigma_0^2 \boldsymbol{I}_N = \sigma_0^2 \sum_{p=0}^{\infty} \boldsymbol{A}^{pM} (\boldsymbol{A}^T)^{pM}. \quad \text{(A5.3)}$$

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

$$\tilde{y}(n) = \boldsymbol{c}^T \tilde{\boldsymbol{x}}(n) + e(n), \quad \text{(A5.4)}$$

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

$$E[e^2(n)] = \sigma_0^2.$$

The variance of the error at the output summing node is given by [using (A5.3)]

$$\frac{\sigma_y^2}{\sigma_0^2} = \boldsymbol{c}^T \sum_{p=0}^{\infty} \boldsymbol{A}^{pM} (\boldsymbol{A}^T)^{pM} \boldsymbol{c} + 1, \quad \text{(A5.5)}$$

which is a strictly decreasing function in M.

References

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

[2] H. T. Kung, "Why systolic architectures," *IEEE Computer*, vol. 15, Jan. 1982.

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

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

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

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] T. Meng and D. G. Messerschmitt, "Implementation of arbitrarily fast adaptive lattice filters with multiple slow processing elements," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Tokyo, 1986.

[15] 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.

[16] 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.

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

[18] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters—Part II: Pipelined incremental block filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, this issue, pp. 1118–1134.

[19] ——, "A bit-parallel bit level recursive filter architecture," in *Proc. IEEE Int. Conf. Comput. Design*, New York, 1986.

[20] ——, "Look-ahead computation: Improving iteration bound in linear recursions," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Dallas, TX, Apr. 1987.

[21] ——, "Area-efficient high speed VLSI adaptive filter architectures," in *Proc. IEEE Int. Conf. Commun.*, Seattle, WA, June 1987.

[22] ——, "Concurrent cellular VLSI adaptive filter architectures," *IEEE Trans. Circuits Syst.*, vol. CAS-34, Oct. 1987.

[23] G. R. Gao, "Maximum pipelining linear recurrence on static data flow computers," *Int. J. Parallel Programming*, vol. 15, no. 2, pp. 127–149, 1986.

[24] H. S. Stone, "An efficient parallel algorithm for the solution of a tridiagonal linear system of equations," *J. Assoc. Comput. Machinery*, vol. 20, no. 1, pp. 27–38, Jan. 1973.

[25] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, pp. 786–792, Aug. 1973.

[26] P. M. Kogge, "Parallel solution of recurrence problems," *IBM J. Res. Develop.*, vol. 18, pp. 138–148, Mar. 1974.

[27] S.-C. Chen and D. J. Kuck, "Time and parallel processor bounds for linear recurrence systems," *IEEE Trans. Comput.*, vol. C-24, pp. 701–717, July 1975.

[28] D. D. Gajski, "Processor array for computing linear recurrence systems," in *IEEE Int. Conf. Parallel Processing*, 1978, pp. 246–256.

[29] ——, "An algorithm for solving linear recurrence systems on parallel

and pipelined machines," *IEEE Trans. Computers*, vol. C-30, pp. 190–206, Mar. 1981.
[30] H. H. Loomis and B. Sinha, "High speed recursive digital filter realization," *Circuits, Syst., Signal Processing*, vol. 3, no. 3, pp. 267–294, 1984.
[31] J. J. Tiemann and T. L. Vogelsong, "Charge domain integrated circuits for signal processing," in *Proc. 1984 Int. Symp. Circuits Syst.*
[32] K. K. Parhi and D. G. Messerschmitt, "Pipelined VLSI recursive filters using scattered look-ahead and decomposition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Apr. 1988, New York, pp. 2120–2123.
[33] 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, pp. 374–377.
[34] M. G. Bellanger, G. Bonnerot, and M. Coudreuse, "Digital filtering by polyphase network: Application to sample rate alteration and filter banks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-24, pp. 109–114, Apr. 1976.
[35] R. W. Hockney, "A fast direct solution of Poisson equation using Fourier analysis," *J. Assoc. Comput. Machinery*, vol. 12, no. 1, pp. 95–113, Jan. 1965.
[36] B. L. Buzzbee, G. H. Golub, and C. W. Nielson, "On direct methods for solving Poisson's equations," *SIAM J. Numerical Anal.*, vol. 7, no. 4, pp. 627–656, Dec. 1970.
[37] R. A. Sweet, "A generalized cyclic reduction algorithm," *SIAM J. Numerical Anal.*, vol. 11, no. 3, pp. 506–520, June 1974.
[38] P. N. Swarztrauber, "A direct method for discrete solution of separable elliptic equations," *SIAM J. Numerical Anal.*, vol. 11, no. 6, pp. 1136–1150, Dec. 1974.
[39] ——, "The methods of cyclic reduction, Fourier analysis, and the FACR algorithm for the discrete solution of Poisson's equations on a rectangle," *SIAM Rev.*, vol. 19, no. 3, pp. 490–501, July 1977.
[40] R. W. Hockney and C. R. Jesshope, *Parallel Computers: Architecture, Programming, and Algorithms*. Bristol, U.K.: Adam Hilger, 1981.
[41] W. Oed and O. Lange, "The solution of linear recurrence relations on pipelined processors," in *Proc. Int. Conf. Parallel Processing*, 1983, pp. 545–547.
[42] B. B. Zhou and R. P. Brent, "A two-level pipelined implementation of direct form recursive digital filters," Tech. Rep., Comput. Sci. Lab., The Australian National Univ., 1988.
[43] H. T. Kung, "Special purpose devices for signal and image processing: An opportunity in very large scale integration (VLSI)," *Proc. SPIE*, vol. 241, Real Time Signal Processing III, July 1980, pp. 76–84.
[44] D. E. Heller, "Decomposition of recursive filters for linear systolic arrays," *Proc. SPIE*, vol. 431, Real Time Signal Processing VI, 1983, pp. 55–59.
[45] S. Y. Kung, "From transversal filter to wavefront array," in *Proc. Int. Conf. VLSI*, F. Anceau and E. J. Aas, Eds. Amsterdam, The Netherlands: Elsevier Publishers, IFIP, North-Holland, 1983, pp. 247–261.
[46] S. K. Rao and T. Kailath, "Digital filtering in VLSI," in *Proc. 22nd Allerton Conf.*, 1984.
[47] H. T. Kung and M. S. Lam, "Wafer scale integration and two-level pipelined implementation of systolic arrays," *J. Parallel Distributed Comput.*, vol. 1, pp. 32–63, 1984.
[48] K. K. Parhi, "Algorithm and architecture designs for high speed digital signal processing," Ph.D. dissertation, ERL Memo M88-61, U.C. Berkeley, Sept. 1988.
[49] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constraints," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 196–202, Mar. 1981.
[50] A. Fettweis, "Realizability of digital filter networks," *Arch. Elek. Ubertrangung*, vol. 30, pp. 90–96, Feb. 1976.
[51] E. A. Lee and D. G. Messerschmitt, "Pipeline interleaved programmable DSP's: Architecture," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 1320–1333, Sept. 1987.
[52] D. A. Schwartz, "Synchronous multiprocessor realizations of shift invariant flow graphs," Ph.D. dissertation, Georgia Inst. Technol., Tech. Rep. DSPL-85-2, July 1985.
[53] C. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. 3rd Caltech Conf. VLSI*, Pasadena, CA, Mar. 1983.
[54] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," in *Proc. 22nd Annu. Symp. Foundations Comput. Sci.*, 1981.
[55] D. A. Schwartz and T. P. Barnwell, III, "A graph theoretic technique for the generation of systolic implementations for shift invariant flow graphs," in *Proc. ICASSP-84*, San Diego, CA, Mar. 1984.
[56] K. K. Parhi and M. Hatamian, "A high sample rate recursive digital filter chip," in *Proc. 1988 IEEE VLSI Signal Processing Workshop*, Monterey, CA, Nov. 1988.
[57] L. R. Rabiner, J. F. Kaiser, O. Herrman, and M. T. Dolan, "Some comparisons between FIR and IIR digital filters," *Bell Syst. Tech. J.*, vol. 53, pp. 305–331, Mar. 1981.
[58] R. F. Woods, S. C. Knowles, J. V. McCanny, and J. G. McWhirther, "Systolic IIR filters with bit level pipelining," in *Proc. 1988 Int. Conf. Acoust., Speech, Signal Processing*, New York, Apr. 1988, pp. 2072–2075.
[59] M. D. Ercegovac and T. Lang, "On-line arithmetic: A design methodology and applications in digital signal processing," in *VLSI Signal Processing III*. New York: IEEE Press, Nov. 1988.
[60] K. Steiglitz, "Computer aided design of recursive digital filters," *IEEE Trans. Audio Electroacoust.*, vol. AU-18, pp. 123–129, June 1970.
[61] A. G. Deczky, "Synthesis of recursive digital filters using the minimum p-error criterion," *IEEE Trans. Audio Electroacoust.*, vol. AU-20, pp. 257–263, Oct. 1972.
[62] L. R. Rabiner *et al.*, "Linear programming design of IIR digital filters with arbitrary magnitude function," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 117–123, Apr. 1974.
[63] H. G. Martinez and T. W. Parks, "Design of recursive digital filters with optimum magnitude and attenuation poles on the unit circle," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp. 154–162, Apr. 1978.

Keshab K. Parhi (S'85–M'88) received the B.Tech. (honors) degree from the Indian Institute of Technology, Kharagpur, India, in 1982, the M.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1984, and the Ph.D. degree from the University of California, Berkeley, in 1988.

He held short-term positions at the AT&T Bell Laboratories, Holmdel, NJ, the IBM T. J. Watson Research Center, Yorktown Heights, NY, and the Tata Engineering and Locomotive Company, Jamshedpur, India. He is currently an Assistant Professor of Electrical Engineering at the University of Minnesota, Minneapolis. His research interests include design of concurrent algorithm and architectures for communications, signal and image processing systems, VLSI digital integrated circuits, high speed designs and design methodologies, and computer aided design.

Dr. Parhi received the 1987 Eliahu Jury Award for excellence in research in the areas of systems, communications, and signal processing, the 1987 Demetri Angelakos Award for altruistic activities afforded fellow graduate students, the U.C. Regents Fellowship, and the IBM Graduate Fellowship at the University of California, Berkeley. He is a member of Eta Kappu Nu.

David G. Messerschmitt (S'65–M'68–SM'78–F'83) received the B.S. degree from the University of Colorado in 1967, and the M.S. and Ph.D. degrees from the University of Michigan in 1968 and 1971, respectively.

He is a Professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley. From 1968 to 1977 he was a member of the Technical Staff and later a Supervisor at Bell Laboratories, Holmdel, NJ, where he did systems engineering, development, and research on digital transmission and digital signal processing (particularly relating to speech processing). Current research interests include analog and digital signal processing, adaptive filtering, digital communications (on the subscriber loop and fiber optics), architecture and software approaches to programmable digital signal processing, communication network design and protocols, and computer aided design of communications and signal processing systems. He has published over 70 papers and has 10 patents issued or pending in these fields. Since 1977 he has served as a consultant to a number of companies. He has also organized and participated in a number of short courses and seminars devoted to continuing engineering education.

Dr. Messerschmitt is a member of Eta Kappa Nu, Tau Beta Pi, Sigma Xi, and has several best paper awards. He is currently a Senior Editor of the *Communications Magazine*, and is a past Editor for Transmission of the IEEE TRANSACTIONS ON COMMUNICATIONS, as well as a past member of the Board of Governors of the Communications Society.