REFERENCES

[1] U. Grenander and G. Szegö, *Toeplitz Forms and Their Applications.* Berkeley: University of California Press, 1958.

[2] M. Wax and T. Kailath, "Efficient inversion of Toeplitz-block Toeplitz matrix," *IEEE Trans. Acoustics, Speech Signal Processing,* vol. 31, no. 5, pp. 1218–1221, Oct. 1983.

[3] P. A. Voois, "Two-dimensional signal processing for magnetic storage systems," Ph.D. thesis, Stanford Univ., 1993.

[4] R. M. Gray, "On the asymptotic eigenvalue distribution of Toeplitz matrices," *IEEE Trans. Inform. Theory,* vol. IT-18, no. 6, pp. 725–730, Nov. 1972.

[5] ——, "Toeplitz and circulant matrices: A review," Stanford Electron. Lab., Tech. Rep., June 1971.

[6] T. Koga and F. Cheng, "Extension of the Toeplitz theorem and its application," in *Proc. GLOBECOM '87,* Tokyo, Nov. 1987.

[7] F. Cheng and A. N. Venetsanopoulos, "Extension of the Toeplitz theorem to the 2-D case and its application to information theory," in *Proc. 1990 IEEE Int. Symp. Circuits Syst.,* New Orleans, May 1990.

[8] P. J. Davis, *Circulant Matrices, Pure and Applied Mathematics.* New York; Wiley, 1979.

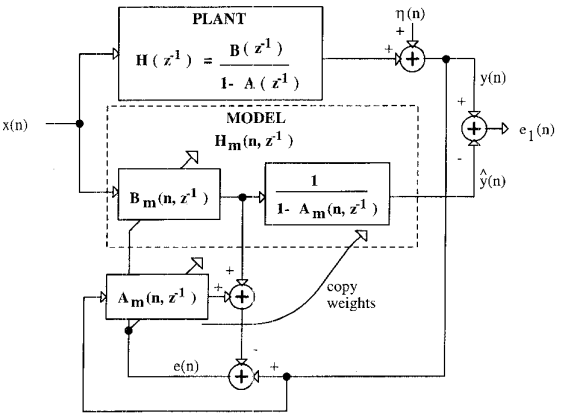

Fig. 1. Conventional system identification scenario.

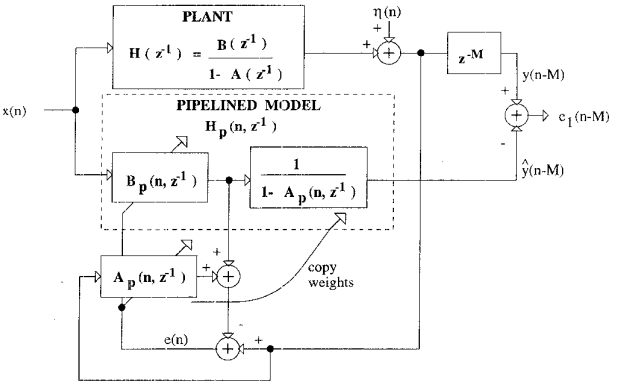

Fig. 2. Pipelined system identification scenario.

Pipelined Adaptive IIR Filter Architectures Using Scattered and Relaxed Look-Ahead Transformations

Naresh R. Shanbhag and Gi-Hong Im

***Abstract*—Fine-grain pipelined architectures for adaptive infinite impulse response (AIIR) filters are presented in this paper. The AIIR filters are *equation error* based. The proposed architectures are developed by employing a combination of *scattered look-ahead* and *relaxed look-ahead* pipelining techniques. First, a pipelined system identification scenario is developed. Then, the scattered look-ahead technique is applied to the nonadaptive (but time-varying) recursive section. Finally, a relaxed look-ahead technique is applied to the adaptive blocks. Convergence analysis of the pipelined architectures is presented and verified via simulations. It is shown that speedups of up to 8 and more can be achieved with marginal or no degradation in performance.**

I. INTRODUCTION

It has been recognized that in addition to algorithmic performance measures (such as signal-to-noise ratio (SNR)), VLSI domain constraints (such as area, speed, and power) need to be addressed during the algorithm design phase. In recent years, *algorithm transformation techniques* [1] such as pipelining [2] have been proposed for high-speed and low-power applications [3]. By combining pipelining with folding [4], it is possible to trade off area with speed. Thus, all the three major parameters of interest in a VLSI implementation (speed, power, and area) can be optimized by the design of pipelined algorithms.

Recently, the *relaxed look-ahead technique* [5] had been proposed for efficient pipelining of adaptive digital filters. This technique is an approximation to the look-ahead technique [2]. In this correspondence, we employ relaxed look-ahead [5] and *scattered look-ahead* [2] to pipeline adaptive IIR (AIIR) filters [6]. While numerous AIIR filter structures have been proposed in the past [7], [8], we will focus on the *equation-error* based [9] approach.

This paper is organized as follows. In Section II, we present some background material. In Section III, the pipelined AIIR architecture is derived via the application of relaxed look-ahead and scattered look-ahead. In Section IV, we present simulation results to verify the convergence analysis results and the performance of the pipelined architectures.

II. PRELIMINARIES

In this section, we describe the scattered look-ahead [2] (for pipelining of fixed coefficient digital filters), relaxed look-ahead [5] (for pipelining of adaptive digital filters), and the properties of relaxed look-ahead pipelined LMS filter [10].

Manuscript received November 29, 1994; revised January 29, 1996. The associate editor coordinating the review of this paper and approving it for publication was Prof. Keshab Parhi..

N. R. Shanbhag is with Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1308 West Main St., Urbana, IL 61801 USA.

G.-H. Im is with AT&T Bell Laboratories, 200 Laurel Avenue, Middletown, NJ 07748 USA.

Publisher Item Identifier S 1053-587X(96)04512-6.

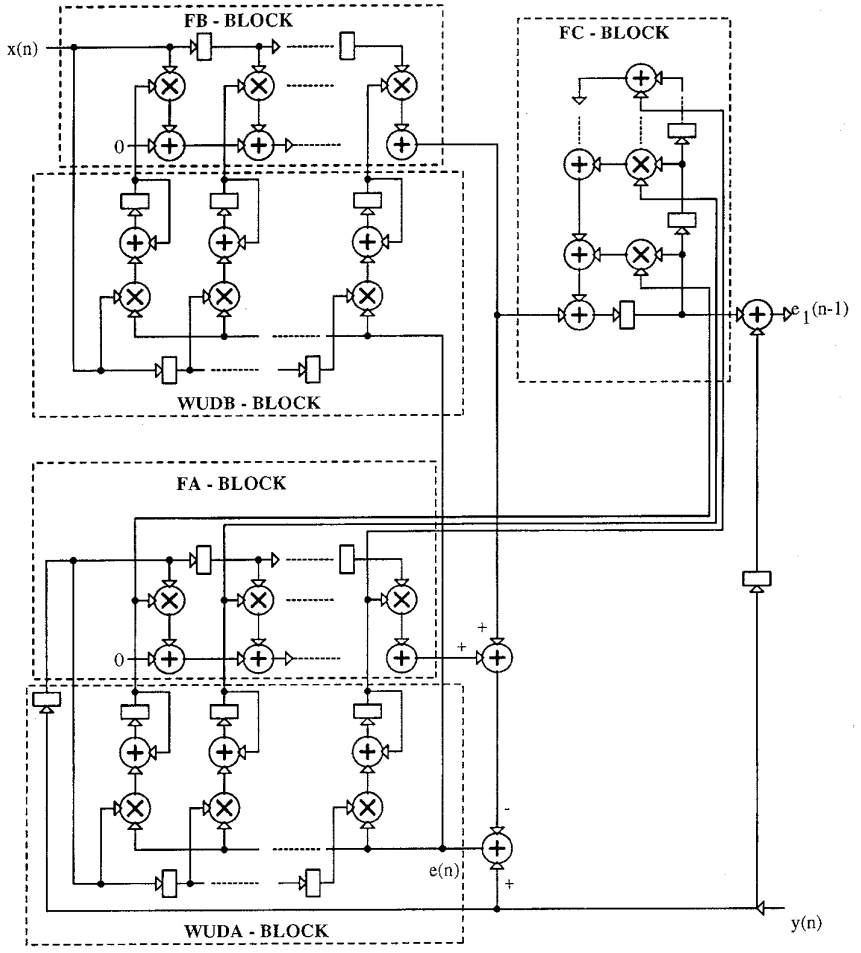

Fig. 3. Serial AIIR (SAIIR) filter architecture.

A. Scattered Look-Ahead

In a serial (or unpipelined) recursive digital filter, the current state $w(n)$ is computed as a function of past states $w(n-1), w(n-2), \cdots, w(n-N)$ and present and past values of input $x(n)$. In other words

$$w(n) = f_{serial}[w(n-1), w(n-2), \cdots, w(n-N), x(n), x(n-1), \cdots, x(n-P)] \tag{2.1}$$

where N is the order of the filter, $f_{serial}(.)$ is a linear function, and n is the time index. On the other hand, in an M-step scattered look-ahead pipelined filter, the current state is computed as follows:

$$w(n) = f_{s,pipe}[w(n-M), w(n-2M), \cdots, w(n-NM), x(n), x(n-1), \cdots, x(n-R)] \tag{2.2}$$

where $f_{s,pipe}(.)$ is the scattered look-ahead function. The hardware overhead due to scattered look-ahead is $O(NM)$, which can be reduced via *decomposition* [2] to $O[N \log_2(M)]$. A significant characteristic of scattered look-ahead is that it preserves stability.

In general, for $H_{serial}(z^{-1})$ given by

$$H_{serial}(z^{-1}) = \frac{\sum_{i=0}^{N_B-1} b_i z^{-i}}{1 - \sum_{i=1}^{N_A} a_i z^{-i}} \tag{2.3}$$

and an M-step scattered look-ahead pipelined transfer function is given by (see [2])

$$\begin{aligned} H_{pipe}(z^{-1}) &= \frac{\left(\sum_{i=0}^{N_B-1} b_i z^{-i}\right) \prod_{k=1}^{M-1} \left(1 - \sum_{i=1}^{N_A} a_i z^{-i} e^{(-j2\pi k)/M}\right)}{\prod_{k=0}^{M-1} \left(1 - \sum_{i=1}^{N_A} a_i z^{-i} e^{(-j2\pi k)/M}\right)} \\ &= \frac{\sum_{i=0}^{N_B-1+(M-1)N_A} b_i' z^{-i}}{1 - \sum_{i=1}^{N_A} a_i' z^{-Mi}}. \end{aligned} \tag{2.4}$$

B. Relaxed Look-Ahead

Consider the first-order recursion

$$w(n) = w(n-1) + a(n)x(n). \tag{2.5}$$

The computation time of (2.5) is lower bounded by a single add time. Next, we apply a M-step look-ahead to (2.5) in the time-domain and obtain

$$w(n) = w(n-M) + \sum_{i=0}^{M-1} a(n-i)x(n-i). \tag{2.6}$$

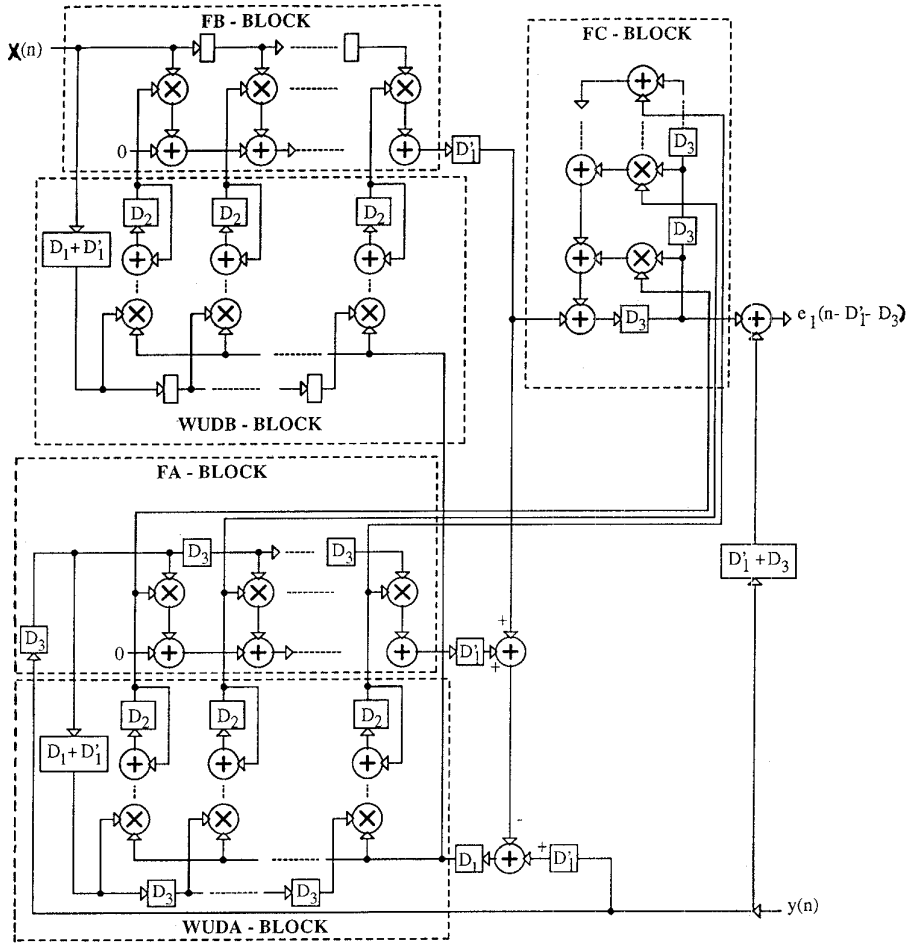

Fig. 4. Pipelined AIIR (PIPAIIR) filter architecture.

This transformation introduces M latches into the recursive loop, which can be retimed [11] to attain M-level pipelining of the add operation. The relaxed look-ahead technique involves approximating architectures such as those described by (2.6), which have been derived via look-ahead technique. *Delay* and *sum* relaxations are two possible approximations.

The delay relaxation involves the use of delayed input $u(n - D_1)$ and delayed coefficient $a(n - D_1)$ in (2.6). If the average value of the product $a(n)u(n)$ is more or less constant over D_1 samples, then (2.6) can be approximated as

$$w(n) = w(n - M) + \sum_{i=0}^{M-1} a(n - D_1 - i)x(n - D_1 - i). \quad (2.7)$$

Note that this approximation results in the "delayed LMS" [12] algorithm when applied to the traditional LMS algorithm. In general, this is a reasonable approximation given that the average value of $a(n)u(n)$ varies slowly in (2.6).

Application of the sum relaxation to (2.6) involves taking LA terms from (2.6), where $LA \leq M$, to get

$$w(n) = w(n - M) + \frac{M}{LA} \sum_{i=0}^{LA-1} a(n - i)x(n - i). \quad (2.8)$$

This relaxation can be justified if the average value of the product $a(n)u(n)$ is slowly varying, and simulations (for both LMS and the AIIR filters) indicate this to be a good approximation.

C. Pipelined LMS (PIPLMS) Architecture

The serial LMS filter is described by the following equations:

$$\mathbf{W}(n) = \mathbf{W}(n - 1) + \mu e(n)\mathbf{U}(n)$$
$$e(n) = y(n) - \mathbf{W}^T(n - 1)\mathbf{U}(n) \quad (2.9)$$

where $\mathbf{W}(n)$ is the weight vector, $\mathbf{U}(n)$ is the input vector, $e(n)$ is the adaptation error, μ is the step-size, and $y(n)$ is the desired signal.

The relaxed look-ahead pipelined LMS architecture (see [10] for details) is given by

$$\mathbf{W}(n) = \mathbf{W}(n - D_2) + \mu \sum_{i=0}^{LA-1} e(n - D_1 - i)\mathbf{U}(n - D_1 - i)$$
$$e(n) = y(n) - \mathbf{W}^T(n - D_2)\mathbf{U}(n) \quad (2.10)$$

where D_1 delays are introduced via the delay relaxation, and D_2 delays are introduced via the sum relaxation. The D_1 and D_2 delays would be employed to pipeline the hardware operators in an actual implementation.

D. Convergence Analysis of PIPLMS

As mentioned before, the relaxed look-ahead technique modifies a given adaptive algorithm, and hence, a convergence analysis needs to be done. The results of the convergence analysis of the mean-squared error (MSE) [10] (for $LA = 1$ and $K = D_1/D_2$) show that

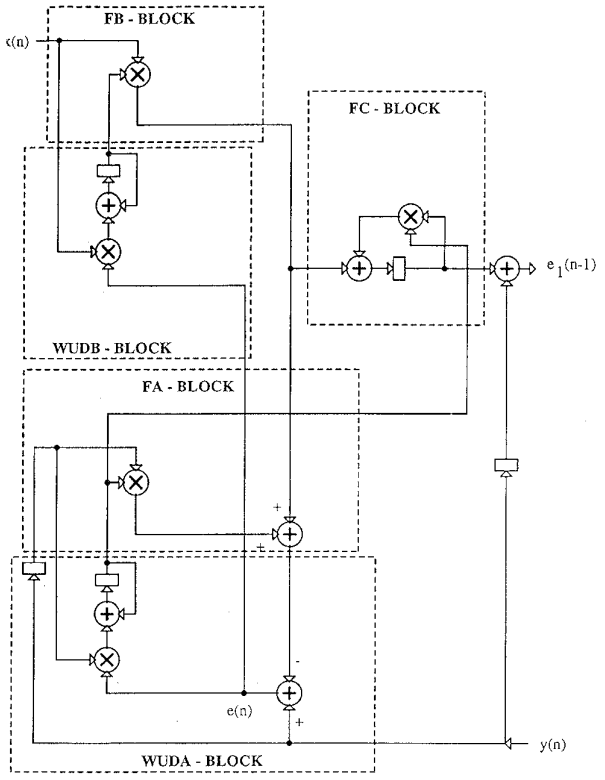

Fig. 5. SAIIR filter with $N_A = 1$ and $N_B = 1$.

the bounds on the step-size are given by

$$0 \le \mu \le \frac{\alpha P + 2K - \sqrt{(\alpha P + 2K)^2 - 8K(K+1)}}{2K(K+1)\sigma^2} \tag{2.11}$$

where

$$\lambda_{rms}^2 = \frac{\sum_{i=1}^{N} \lambda_i^2}{N}; \lambda_{av} = \frac{\sum_{i=1}^{N} \lambda_i}{N} = \sigma^2; \alpha = \frac{\lambda_{rms}^2}{\lambda_{av}^2}$$
$$v = \frac{E\{[u(n)]^4\}}{E\{[u(n)]^2\}^2}; P = N + v - 1; b = \mu\sigma^2 \tag{2.12}$$

where λ_i's ($i = 1, \cdots, N$) are the eigenvalues of $\mathbf{R} = E[\mathbf{U}(n)\mathbf{U}^T(n)]$. This reduction in the upper bound should be kept in mind in applications where fast convergence is important. However, this is not a problem in those applications (such as adaptive equalizers in digital subscriber loops) where the step-size μ is kept very small.

As K decreases from D_1 down to unity, which also corresponds to increasing D_2 from unity to D_1, the convergence speed of PIPLMS slows down. It was shown in [10] that by first choosing appropriate values of D_1 and D_2 for achieving the desired speedup and then adjusting LA provides a solution to this problem. This is also true for the proposed pipelined AIIR filter, as will shown in Section IV.

Finally, the misadjustment for PIPLMS is given by

$$\mathcal{M} = \frac{\alpha N b}{2 - (\alpha P + 2K)b + K(K+1)b^2} \tag{2.13}$$

where b and P are defined in (2.12). From (2.13), as K is increased from unity toward D_1, the misadjustment would increase. In actual practice, the misadjustment does not change substantially as K varies. In the next section, we develop the pipelined AIIR filter architectures.

III. Pipelined AIIR Architectures

In this section, we first formulate a pipelined system identification scenario and then develop the pipelined AIIR architectures.

A. Pipelined System Identification Scenario

In Fig. 1, we show the conventional serial system identification scenario, where $H(z^{-1})$, which is the unknown plant, is given by

$$H(z^{-1}) = \frac{B(z^{-1})}{1 - A(z^{-1})} = \frac{\sum_{i=0}^{NB-1} b_i z^{-i}}{1 - \sum_{i=1}^{NA} a_i z^{-i}} \tag{3.1}$$

with numerator polynomial $B(z^{-1})$ and denominator polynomial $1 - A(z^{-1})$, $\eta(n)$ is the additive noise uncorrelated with the input $x(n)$, $y(n)$ is the plant output (also the desired signal), and $H_m(n, z^{-1})$ is the time-varying model. Note that the numerator polynomial $B_m(n, z^{-1})$ and the polynomial $A_m(n, z^{-1})$ are adaptively computed. The denominator polynomial of the model is formed by taking the inverse $1/[1 - A_m(n, z^{-1})]$ after every coefficient update. If the order of $B_m(n, z^{-1})$ and $A_m(n, z^{-1})$ are sufficiently high and if the adaptation mechanism converges successfully, then $B_m(n, z^{-1})$ will approach $B(z^{-1})$, and $A_m(n, z^{-1})$ will be close to $A(z^{-1})$. One of the disadvantages inherent to the equation error approach is that the final solution may be biased. In this correspondence, however, we will focus only on the problem of providing an efficient pipelined architecture.

We can employ a *relaxed look-ahead* to pipeline the adaptive algorithm in Fig. 1 and scattered look-ahead to pipeline the time-varying recursive section $1/[1 - A_m(n, z^{-1})]$. Therefore, in the pipelined system identification scenario (see Fig. 2), we assume the plant to be in a scattered look-ahead form as in (2.4). Therefore, in Fig. 2, the polynomial $A_p(n, z^{-1})$ (for any time instance n) would have the same form as the denominator in (2.4). In practice, given a plant of type (3.1), we can emulate the behavior of an equivalent pipelined plant by delaying the output of the plant $y(n)$ by M latches (see Fig. 2).

B. Serial AIIR Filter Architecture

The serial AIIR filter is described by the following equations:

$$\begin{aligned} \mathbf{W}(n) &= \mathbf{W}(n-1) + \mu e(n)\mathbf{U}(n) \\ e(n) &= y(n) - \mathbf{W}^T(n-1)\mathbf{U}(n) \\ \hat{y}(n) &= \mathbf{W}^T(n-1)\hat{\mathbf{U}}(n) \end{aligned} \tag{3.2a}$$
$$e_1(n) = y(n) - \hat{y}(n) \tag{3.2b}$$

where

$$\mathbf{W}(n) = \begin{bmatrix} \mathbf{B}(n) \\ \mathbf{A}(n) \end{bmatrix}$$
$$\begin{aligned} \mathbf{B}^T(n) &= [b_0(n), b_1(n), \cdots, b_{N_B-1}(n)] \\ \mathbf{A}^T(n) &= [a_1(n), a_2(n), \cdots, a_{N_A}(n)] \end{aligned} \tag{3.3a}$$
$$\begin{aligned} \mathbf{U}^T(n) = [&x(n), x(n-1), \cdots, x(n-N_B+1), \\ &y(n-1), y(n-2), \cdots, y(n-N_A)] \end{aligned} \tag{3.3b}$$
$$\begin{aligned} \hat{\mathbf{U}}^T(n) = [&x(n), x(n-1), \cdots, x(n-N_B+1), \\ &\hat{y}(n-1), \hat{y}(n-2), \cdots, \hat{y}(n-N_A)]. \end{aligned} \tag{3.3c}$$

Note that $\mathbf{W}(n)$ is the coefficient vector with $\mathbf{B}(n)$ and $\mathbf{A}(n)$ being the coefficient vectors of the numerator and denominator polynomials, respectively.

The serial AIIR (SAIIR) filter architecture is shown in Fig. 3, where blocks **FB** and **FA** do the filtering operation associated with

$B_m(n, z^{-1})$, and $A_m(n, z^{-1})$, respectively. Weight-update blocks **WUDB** and **WUDA** compute the coefficients of **FB** block and **FA** block, respectively. The **FC** block computes the denominator polynomial $1 - A_m(n, z^{-1})$.

Due to the recursive structure of the adaptation and filtering operations, the SAIIR architecture in Fig. 3 has a throughput bottleneck. In particular, it can be seen that the critical path for SAIIR has a computation time of

$$T_{serial} = [\max(N_B, N_A) + 1]T_{A_1} + 2T_m + T_{A_2} \tag{3.4}$$

where N_B and N_A are the number of taps in the **FB** and **FA** blocks, respectively, $\max(\,)$ generates the maximum of its arguments, T_{A_1} is the computation time of a two-operand adder present in the **FB**, **FA**, and **FC** blocks, and T_{A_2} is the computation time of the adder in the **WUDB** and **WUDA** blocks. From (3.4), it is clear that for applications that require large values of N_B and N_A (such as echo cancellation), T_{serial} can be quite large. This problem is solved in the next section, where a pipelined AIIR (PIPAIIR) architecture is presented.

C. The PIPAIIR Architecture

In order to derive the PIPAIIR architecture, we start with the SAIIR equations (3.2). The process of pipelining the SAIIR proceeds in two steps. First, we transform (3.2) such that it is applicable to the scattered look-ahead-based pipelined system identification model of Fig. 2. This step will result in the pipelining of the **FC** block. Next, we apply relaxed look-ahead to the adaptive sections, which will result in the PIPAIIR architecture.

By inspection of (2.4), we transform (3.2) as follows: s

$$\begin{aligned} \mathbf{W}(n) &= \mathbf{W}(n-1) + \mu e(n)\mathbf{U}_1(n); \\ e(n) &= y(n) - \mathbf{W}^T(n-1)\mathbf{U}_1(n) \qquad (3.5a) \\ \hat{y}(n) &= \mathbf{W}^T(n-1)\hat{\mathbf{U}}_1(n); \\ e_1(n) &= y(n) - \hat{y}(n) \qquad (3.5b) \end{aligned}$$

where

$$\begin{aligned} \mathbf{W}(n) &= \begin{bmatrix} \mathbf{B}(n) \\ \mathbf{A}(n) \end{bmatrix} \\ \mathbf{B}^T(n) &= [b_0(n), b_1(n), \cdots, b_{N_{B'}-1}(n)]; \\ \mathbf{A}^T(n) &= [a_1(n), a_2(n), \cdots, a_{N_A}(n)] \qquad (3.6a) \\ \mathbf{U}_1^T(n) &= [x(n), x(n-1), \cdots, x(n-N_{B'}+1), \\ &\quad y(n-D_3), y(n-2D_3), \cdots, y(n-N_AD_3)] \qquad (3.6b) \\ \hat{\mathbf{U}}_1^T(n) &= [x(n), x(n-1), \cdots, x(n-N'_B+1), \hat{y}(n-D_3), \\ &\quad \hat{y}(n-2D_3), \cdots, \hat{y}(n-N_AD_3)]. \qquad (3.6c) \end{aligned}$$

Notice that (3.6c) implies that D_3 latches have been introduced into recursive loop of the **FC** block. In addition, we also observe that the order of $B_p(n, z^{-1})$ is $N_{B'} - 1$, where $N_{B'} = [N_A(D_3 - 1) + N_B]$ is greater than N_B. This is precisely due to the fact that we have assumed a pipelined system identification model based on scattered look-ahead.

Next, we pipeline the adaptive sections via relaxed look-ahead. Applying delay and then sum relaxations to (3.5a) (see also (2.10)), we obtain the following equations that describe the PIPAIIR architecture:

$$\begin{aligned} \mathbf{W}(n) &= \mathbf{W}(n-D_2) + \mu \sum_{i=0}^{LA-1} e(n-D_1-D'_1-i) \\ &\quad \cdot \mathbf{U}_1(n-D_1-D'_1-i), \qquad (3.7a) \\ e(n) &= y(n) - \mathbf{W}^T(n-D_2)\mathbf{U}_1(n); \\ \hat{y}(n) &= \mathbf{W}^T(n-D_2)\hat{\mathbf{U}}_1(n). \qquad (3.7b) \end{aligned}$$

TABLE I
HARDWARE REQUIREMENTS

ARCHITECTURE	No. of Multipliers	No. of Adders	
		Type I	Type II
SAIIR	$3N_A + 2N_B$	$2N_A + N_B + 1$	$N_B + N_A$
PIPAIIR	$3N_A + 2N_{B'}$	$2N_A + N_{B'} + 1$	$N_{B'} + N_A$

In Fig. 4, we show the PIPAIIR architecture, where the D'_1 delays have been retimed such that they have been placed at the output of the **FB** block and **FA** block. This requires delaying $e_1(n)$ by an additional D'_1 delays. Note that the plant output $y(n)$ is delayed by $M = D'_1 + D_3$. This is because (3.6c) introduces a latency of D_3 samples, which is now added to the D'_1 delays due to retiming.

In a practical implementation, we would employ the D'_1 latches to pipeline the filter blocks **FB** and **FA**, whereas the D_1 latches would be used to pipeline the **WUDA** and **WUDB** blocks, and D_3 latches would be employed to pipeline the **FC** block. The hardware requirements for SAIIR and PIPAIIR are shown in Table I. The TYPE I adder, in Table I, refers to the adder in the filter blocks **FB**, **FA**, and **FC**, whereas the TYPE II adder is present in the weight-update blocks **WUDA**, and **WUDB**. It is clear that the increase in hardware is mainly due to the increase in the numerator order.

The convergence behavior of PIPAIIR can be easily obtained by substituting

$$\begin{aligned} N &= N_A + N_{B'}; \\ K &= \frac{D_1 + D'_1}{D_2}; \\ \mathbf{R} &= E[\mathbf{U}_1(n)\mathbf{U}_1^T(n)]; \\ v &= \frac{\frac{N'_B}{N} E\{[x(n)]^4\} + \frac{N_A}{N} E\{[y(n)]^4\}}{\frac{N'_B}{N} (E\{[x(n)]^2\})^2 + \frac{N_A}{N} (E\{[y(n)]^2\})} \end{aligned} \tag{3.8}$$

into (2.11)–(2.13). We will verify the convergence analysis of PIPAIIR via simulations in the next section.

IV. Simulation Results

In this section, we present simulation results to verify the convergence analysis results of Section III-B and the utility of sum relaxation.

A. Experiment A

In this experiment, we consider the problem of identifying the plant

$$H(z^{-1}) = \frac{0.5z^{-1}}{1 - 0.86z^{-1}}. \tag{4.1}$$

The value of D_3 was kept at unity, whereas $D_1 + D'_1$ was varied to obtain different values of K for $D_2 = 1$ and $D_2 = 2$. The plant SNR, which is defined as the power of the input signal $x(n)$ to the power of additive noise $\eta(n)$, was 32 dB. All simulation results were averaged over 100 independent trials, and the final results of this experiment are shown in Table II. The theoretical values of the upper bound on step-size $\mu_{\max}$ and the misadjustment $\mathcal{M}$ (with $\mu = 0.001$) were computed from (2.11) and (2.13), respectively. It can be seen that the theoretical and measured values of $\mathcal{M}$ and $\mu_{\max}$ are quite close and show the same trend as predicted by (2.11) and (2.13). Notice that the degradation in $\mathcal{M}$ and the decrease in the values of $\mu_{\max}$ is quite small for large values of K.

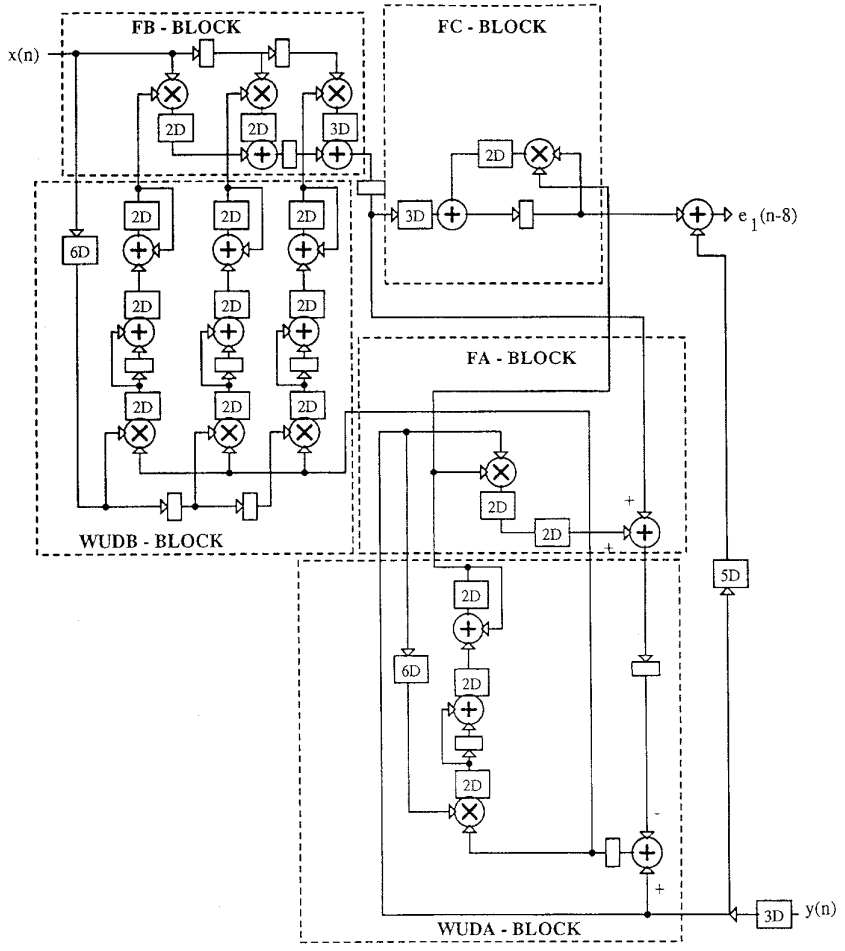

Fig. 6. PIPAIIR filter with $N_A = 1$, $N_B' = 3$, $D_1 = D_1' = 5$, $D_2 = D_2$, $D_3 = D_3$, and $LA = 2$.

TABLE II
SIMULATION RESULTS FOR EXPERIMENT A

	K	μ_{max}		Misadjustment (%)	
		Theory	Measured	Theory	Measured
$D_2 = 1$	1	0.0272	0.0275	1.63	1.06
	2	0.0242	0.0230	1.65	1.15
	4	0.0202	0.0137	1.71	1.22
	8	0.0106	0.0085	1.82	1.27
	16	0.0055	0.0050	2.06	1.42
	32	0.0028	0.0024	2.58	1.92
$D_2 = 2$	1	0.0272	0.0270	1.63	1.10
	2	0.0242	0.0220	1.65	1.12
	4	0.0202	0.0134	1.71	1.15
	8	0.0106	0.0083	1.82	1.26
	16	0.0055	0.0047	2.06	1.70
	32	0.0028	0.0022	2.58	2.32

B. Experiment B

In this experiment, we demonstrate the increase in throughput due to pipelining via an example. Consider the SAIIR architecture with $N_A = 1$ and $N_B = 1$ in Fig. 5. Furthermore, assume that a two-operand multiply takes $T_m = 32$ time units, a two-operand filter block addition time is $T_{A1} = 20$ units, and the weight-update block addition time is $T_{A2} = 40$ units. From (3.4), we can get a value of $T_{serial} = 144$ time units. Employing PIPAIIR with $D_1 = 5$, $D_1' = 5$, $D_2 = 2$, $D_3 = 3$, and $LA = 2$ results in the architecture in Fig. 6. The architecture in Fig. 6 can be obtained from Fig. 4 by filling in appropriate values of D_1, D_1', D_2, and D_3 and retiming [11] the latches. Note that with $D_3 = 3$, the value of N_B' (see Table III) is 3. It can be seen that the PIPAIIR architecture in Fig. 6 can be clocked (assuming uniform pipelining) with a minimum clock period of $T_{pipe} = 20$ time units, which corresponds to a speedup of 7.2. Note that this speedup has been achieved in spite of an increase in the order of the **FB** block.

TABLE III
SIMULATION RESULTS FOR EXPERIMENT B

SU	D_1	D_1'	D_2	D_3	LA	N_B'	SNRE (dB)	SNRE1 (dB)
1	0	0	1	1	1	1	29.36	31.39
2	1	0	1	1	1	1	29.46	31.37
4	3	1	2	2	1	2	29.84	31.17
8	5	4	3	3	1	3	30.18	31.12
8	10	4	3	3	3	3	29.99	30.84

In Table III, we summarize the effect of pipelining on $SNRE$ and $SNRE1$, where $SNRE$ and $SNRE1$ are the SNR's with respect to the adaptation error $e(n)$ and the estimation error $e_1(n)$, respectively. From Table III, it is clear that as speedup increases, both $SNRE$ and $SNRE1$ change only slightly. Even for a speedup of 8,

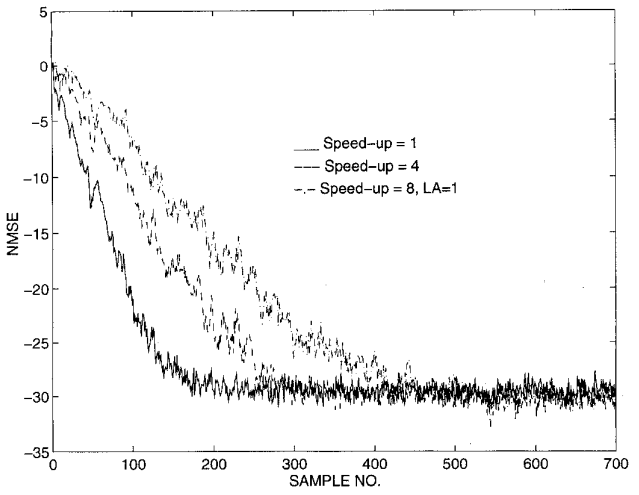

Fig. 7. Normalized MSE plots for different speed-ups.

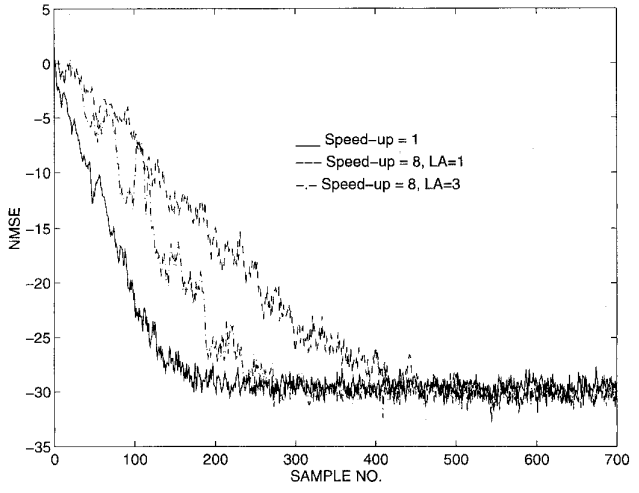

Fig. 8. NMSE plots for different speed-ups with sum relaxation.

the degradation in $SNRE1$ is less than 1 dB. Similar results were observed for higher speedups. Therefore, we can conclude that there is negligible degradation in performance as the level of pipelining increases.

C. Experiment C

As mentioned in Sections II-D and III-B, the convergence speed of PIPAIIR (with $LA = 1$) slows down as D_2 increases from unity to $D_1 + D_1'$. This shown in Fig. 7, where the mean-squared adaptation error (MSE) for speedups of unity, 4, and 8 have been plotted. The values of various parameters were obtained from the first, third, and fourth rows of Table III. Note that as speedup increases, the number of samples required to converge also increases. The slower convergence of PIPAIIR can be rectified by using sum relaxation with $LA = 3$. As $LA = 3$ implies an increase in the computation time of the **WUDA**, and **WUDB** blocks, we need to increase the value of D_1 from 5–10. From Experiment B, we know that the degradation in SNR is negligible for these parameter values. In Fig. 8, we have plotted the MSE curves for a speedup of unity (first row in Table III), eight with $LA = 1$ (fourth row in Table III), and eight with $LA = 3$ (fifth row in Table III). Note the remarkable improvement in convergence speed of PIPAIIR. Thus, PIPAIIR with $D_1 = 10$, $D_1' = 4$, $D_2 = 3$, $D_3 = 3$, and $LA = 3$ achieves a speedup of 8 with no degradation in the misadjustment and some loss in the convergence speed.

REFERENCES

[1] K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. IEEE,* vol. 77, pp. 1879–1895, Dec. 1989.
[2] 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* vol. 37, pp. 1099–1117, July 1989.
[3] A. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE,* vol. 83, no. 4, pp. 498–523, Apr. 1995.
[4] K. K. Parhi, C. Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE J. Solid-State Circuits,* vol. 27, no. 1, pp. 29–43, Jan. 1992.
[5] N. R. Shanbhag and K. K. Parhi, *Pipelined Adaptive Digital Filters.* Boston, MA: Kluwer, 1994.
[6] J. J. Shynk, "Adaptive IIR filtering," *IEEE ASSP Mag.,* pp. 4–21, Apr. 1989.
[7] M. Nayeri and W. K. Jenkins, "Alternate realizations of adaptive IIR filters and properties of their performance surfaces," *IEEE Trans. Circuits Syst.,* vol. 36, pp. 485–496, Apr. 1989.
[8] P. A. Regalia, "Stable and efficient lattice algorithms for adaptive IIR filtering," *IEEE Trans. Signal Processing,* vol. 40, no. 2, pp. 375–388, Feb. 1992.
[9] J. M. Mendel, *Discrete Techniques of Parameter Estimation: The Equation Error Formulation.* New York: Marcel Dekker, 1973.
[10] N. R. Shanbhag and K. K. Parhi, "Relaxed look-ahead pipelined LMS adaptive filters and its application to ADPCM coder," *IEEE Trans. Circuits Syst.,* vol. 40, no. 12, pp. 753–766, Dec. 1993.
[11] C. Leiserson and J. Saxe, "Optimizing synchronous systems," *J. VLSI Comput. Syst.,* vol. 1, pp. 41–67, 1983.
[12] G. Long, F. Ling, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 37, no. 9, pp. 1397–1405, Sept. 1989.

Pipelining in Direct Form Recursive Digital Filters with Periodically Time-Varying Coefficients

Kazys Kazlauskas

***Abstract*—We propose a pipelined realization of direct form recursive digital filters with periodically time-varying coefficients. At first, we present the direct form recursive digital filter as a state-space filter. Afterwards, we derive an equation for block-state update computation. Finally, we use a block processing approach to get the equation for block output computation. The proposed realization is suitable for implementation in the VLSI form.**

Manuscript received May 25, 1995; revised January 12, 1996. The associate editor coordinating the review of this paper and approving it for publication was Prof. Keshab Parhi.

The author is with the Institute of Mathematics and Informatics, Vilnius, Lithuania.

Publisher Item Identifier S 1053-587X(96)04511-4.