Relaxed Look-Ahead Pipelined LMS Adaptive Filters and Their Application to ADPCM Coder

Naresh R. Shanbhag, *Member, IEEE,* and Keshab K. Parhi, *Senior Member, IEEE*

***Abstract*— The *relaxed look-ahead* technique is presented as an attractive technique for pipelining adaptive filters. Unlike conventional look-ahead, the relaxed look-ahead does not attempt to maintain the input-output mapping between the serial and pipelined architectures but preserves the adaptation characteristics. The use of this technique results in a small hardware overhead which would not be possible with conventional look-ahead. The relaxed look-ahead is employed to develop fine-grained pipelined architectures for least mean-squared (LMS) adaptive filtering. Convergence analysis results are presented for the pipelined architecture. The proposed architecture achieves the desired speed-up with marginal or no degradation in the convergence behavior. Past work in pipelined transversal LMS filtering are shown to be special cases of this architecture. Simulation results verifying the convergence analysis results for the pipelined LMS filter are presented. The pipelined LMS filter is then employed to develop a high-speed adaptive differential pulse-code-modulation (ADPCM) codec. The new architecture has a negligible hardware overhead which is independent of the number of quantizer levels, the predictor order and the pipelining level. Additionally, the pipelined codec has a much lower output latency than the level of pipelining. Theoretical analysis indicates that the output signal-to-noise ratio (SNR) is degraded with increase in speed-up. Simulations with image data indicate that speed-ups of up to 44 can be achieved with less than 1 dB loss in SNR.**

I. Introduction

PIPELINING [1]–[2] along with parallel processing [3] are two major techniques for the development of high-speed digital signal processing (DSP) architectures. Recently, pipelining and parallel processing have also been used to design low-power DSP circuits [4]. Thus, it is of interest to develop fine-grain pipelined DSP algorithms. Look-ahead techniques for pipelining of recursive fixed-coefficient filters have been proposed [2], [5]–[6] and have been successfully applied to two-dimensional recursive filtering [7], dynamic programming [8]–[10], algorithms with quantizer loops [11], finite state machines [9], and Huffman decoders [12]. The implementation of a recursive filter chip capable of filtering at 86 million operations per second [13] has clearly demonstrated the feasibility of fine-grain pipelining.

The look-ahead technique transforms a given serial algorithm into an equivalent pipelined algorithm such that the input-output behavior is preserved. This results in a *look-ahead overhead* term of complexity $O(NM)$, where N is the filter order and M is the level of pipelining. This complexity can, however, be reduced to $O(N \log(M))$ via *decomposition techniques* [2]. Nevertheless, the overhead can be quite high for large values of N and M.

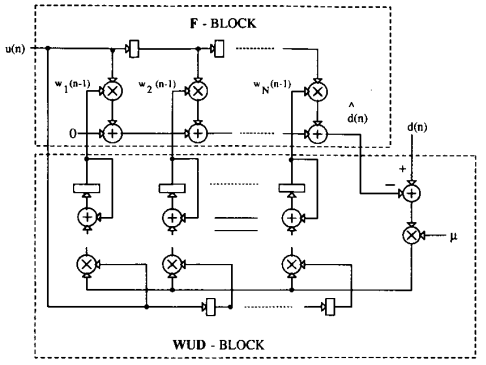


Fig. 1. The SLMS architecture.

Pipelining of adaptive filters is made difficult due to the coefficient update loop. To see this, consider the serial least-mean squared algorithm (SLMS for "serial LMS") described by

$$W(n) = W(n-1) + \mu e(n)U(n) \quad (1)$$

$$e(n) = d(n) - W^T(n-1)U(n) \quad (2)$$

where $W^T(n) = [w_1(n), w_2(n), \cdots, w_N(n)]$ is the vector of weights, $U^T(n) = [u(n), u(n-1), \cdots, u(n-N+1)]$ is the input vector, μ is the adaptation constant, $d(n)$ is the desired signal, $e(n)$ is the error and N is the filter order. The SLMS architecture (Fig. 1) shows that in every iteration the error $e(n)$ needs to be computed by the filter block (F) and the weights updated by the weight-update block (WUD) before the next input can be processed. The clock-period of SLMS is therefore lower bounded by the sum of the computation times of blocks F and WUD.

In order to achieve higher sampling rates we need to introduce pipelining latches in the error-feedback path (EFP) and the recursive loop in WUD (to be referred to as $WUDL$). A direct application of look-ahead would result in a complex architecture. However, we may apply an approximate form of look-ahead if the change in the convergence behavior is not substantial. This approximate form of look-ahead is obtained by relaxing the constraint of maintaining the exact input-output

Manuscript received February 24, 1992; revised January 1993, June 1993. This work was supported by the army research office under contract number DAAL03-90-G-0063. This paper was recommended by Associate Editor G. S. Moschytz.

N. R. Shanbhag is with AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974. K. K. Parhi is with the Department of Electrical Engineering at the University of Minnesota, Minneapolis, MN 55455.

IEEE Log Number 9211971.

mapping. This relaxation can take many forms and may have differing impacts on the convergence behavior. We collectively refer to these relaxations as *relaxed look-ahead*. As the relaxed look-ahead does not attempt to maintain the exact input-output mapping, the resulting hardware savings are enormous. The price we pay for these savings is altered (usually marginally degraded) convergence behavior. For certain applications, the degradation in the convergence behavior of the pipelined filter is either non-existent or can be minimized by tuning the adaptation parameters. This aspect is also illustrated in this paper, where applications of the pipelined adaptive filter to predictive coding of speech and image are demonstrated. In these cases, the advantage due to the gain in speed outweighs the degradation in convergence behavior.

Past work in high-speed LMS adaptive filtering have involved parallel processing [14]–[17] and pipelining [18]–[22]. The look-ahead is applied to design high-speed adaptive lattice filters using parallel processing [18] and pipelining [19]. This is feasible to do as these filters have localized coefficient update (and also better numerical stability). Thus, the architectures in [18], [19] have no degradation in the convergence behavior but still require a substantial amount of hardware.

The architectures in [20]–[22] modify the LMS algorithm in order to pipeline it and therefore are in the same class of architectures presented in this paper. In fact, we shall see later that the delayed LMS (DLMS) [20], [21], which has delays in the EFP, is a special case of the PIPLMS (to be read as 'Pipe LMS') architecture proposed in this paper. A modular implementation of DLMS is given in [22]. The DLMS considers delays only in EFP and has a single delay in the $WUDL$. Therefore, its clock-speed is lower bounded by one add time (or one multiply-add time if leaky LMS is used). For achieving higher sampling rates, finer-grain pipelining is necessary. In addition, for a given speed-up, a fine-grain pipelined architecture can employ arithmetic units with longer word-length. This would enable computation with high precision. Fine-grain pipelining was achieved by the architecture in [23] (referred to as PIPLMS1) which has equal number of delays in the weight-update loop and the error feedback path. As will be shown later, like DLMS, PIPLMS1 is also a special case of PIPLMS. However, PIPLMS1 has a larger convergence time-constant and hence poorer tracking capabilities than DLMS. Thus, increasing the number of delays in $WUDL$ slows down convergence but is necessary for fine-grain pipelining. This has been a problem in the industry for quite some time and it will be shown that the general PIPLMS is a possible solution.

Additional work in pipelined adaptive filtering includes a pipelined adaptive lattice filter [24], where the coefficients are delayed before computing the prediction errors. A class of systolic architectures for LMS adaptive filtering is proposed in [25], where a transpose structure (instead of the conventional tapped-delay line transversal filter) is employed for computing the inner product of the input vector and the coefficients.

In this paper, we employ relaxed look-ahead to pipeline the LMS adaptive filter. In particular, we propose two types of relaxations for pipelining. These include the *delay relaxation* and *sum relaxation* (also referred to as *partial look-ahead*). A major consideration in the formulation of these relaxations was their impact on the hardware requirements. Applying these relaxations we derive a family of architectures referred to as PIPLMS. Convergence analysis of a sub-class of PIPLMS, which we call PIPLMSK, is done by generalizing the analysis in [20]–[21]. Previous work [20]–[21], [23] in pipelined transversal LMS filtering are shown to be special cases of PIPLMSK. Employing the PIPLMS filter, we then develop a hardware-efficient pipelined adaptive differential pulse-code-modulation (ADPCM) codec architecture [26]. The performance of this codec for image and speech compression applications is demonstrated. Note that the relaxed look-ahead technique has also been successfully applied to pipeline the adaptive stochastic gradient lattice filter [27] and the adaptive differential vector quantizer [28]–[29].

This paper is organized as follows. In section II, we describe the relaxed look-ahead. The PIPLMS architecture is derived by the application of the relaxed look-ahead in section III. The convergence properties of the PIPLMS are presented in section IV. In section V, we present the pipelined ADPCM (PIPADPCM) codec architecture. Simulation results to support the convergence analysis results and the performance of the PIPADPCM are presented in section VI.

II. The Relaxed Look-Ahead Technique

In this section, we introduce the relaxed look-ahead as an approximation to the look-ahead. In principle, many types of relaxations are possible. These relaxations are made based on certain assumptions on the input and the error. Depending on the application at hand these assumptions may or may not be valid. Therefore, care should be exercised when making these relaxations. In this paper, we consider two classes of relaxations which we feel are most effective in reducing the hardware. As mentioned before, these two classes are *delay relaxation* and *sum relaxation*. In order to explain these relaxations, we first apply look-ahead to a first-order section and then show the various approximations which constitute the relaxed look-ahead.

Consider the first-order recursion given by

$$x(n) = x(n-1) + a(n)u(n). \tag{3}$$

The computation time of (3) is lower bounded by a single add time. In order to reduce this lower bound, we apply a M-step look-ahead. This is equivalent to expressing $x(n)$ in terms of $x(n-M)$, which leads to

$$x(n) = x(n-M) + \sum_{i=0}^{M-1} a(n-i)u(n-i). \tag{4}$$

This transformation results in M latches being introduced into the recursive loop. These latches can be retimed [30]–[31] to pipeline the add operation to any desired level. Note that this transformation has not altered the input-output behavior. This invariance with respect to the input-output behavior has been achieved at the expense of the look-ahead overhead term (the second term in (4)). This overhead can be very expensive and we will reduce it by applying partial look-ahead to (4).

However, first, we present the delay relaxation, which is a prerequisite for pipelining the LMS filter.

A. Delay Relaxation

The delay relaxation involves the use of delayed input $u(n-D_1)$ and delayed coefficient $a(n-D_1)$ in (4). Clearly, this approximation is based on the assumption that the product $a(n)u(n)$ is more or less constant over D_1 samples. Thus, (4) is approximated as

$$x(n) = x(n-M) + \sum_{i=0}^{M-1} a(n-D_1-i)u(n-D_1-i). \quad (5)$$

Note that this approximation has been used before in case of DLMS [20]–[21] where the gradient $e(n)U(n)$ (see (1), (2)) is assumed to be constant over D_1 samples. In general, this is a reasonable approximation for stationary or slowly varying product $a(n)u(n)$ in (4).

B. Sum Relaxation

As the summation term in (4) is responsible for the hardware overhead we can reduce this overhead by taking LA terms from (4), where $LA \leq M$. Thus, we get

$$x(n) = x(n-M) + \frac{M}{LA}\sum_{i=0}^{LA-1} a(n-i)u(n-i). \quad (6)$$

This relaxation can be justified if the product $a(n)u(n)$ is slowly varying and simulations also indicate this to be a good approximation. The additional correction factor M/LA is necessary to make the average output profile of (6) identical to that of (4) for stationary $a(n)$ and $u(n)$.

Thus, we see that unlike conventional look-ahead, the relaxed look-ahead does not result in a unique final architecture. Application of the two relaxations formulated above (and possibly others), individually or in different combinations, results in a rich variety of architectures. Each of these architectures would have differing convergence characteristics, which depend upon the nature of the approximations made and their validity in a given application.

The relaxed look-ahead should not be viewed as an algorithm transformation technique in the conventional sense [1] as it modifies the input-output behavior of the original algorithm. It may be called a transformation technique in the stochastic sense since the average output profile is maintained. The pipelined architecture resulting from the application of relaxed look-ahead can always be clocked at a higher speed than the original one.

In the next section we apply the delay relaxation and the partial look-ahead to derive PIPLMS.

III. The PIPLMS Architecture

It is desired that the clock period of PIPLMS be less than $1/M$ times that of the serial architecture (SLMS) (Fig. 1), where M denotes the required level of pipelining. The unlabelled boxes, in Fig. 1 and other figures depicting the filter architectures, are delay elements. To derive the pipelined adaptive filter, we start with the SLMS equations and attempt to apply look-ahead directly. Then, we employ relaxed look-ahead to develop PIPLMS.

A. Application of Look-Ahead

The description of SLMS is given by (1), (2). To create D_2 delays in the $WUDL$, we apply a D_2-step look-ahead to (1), (2) and express $W(n)$ in terms of $W(n-D_2)$. We start with (1) and rewrite it as

$$W(n) = W(n-D_2) + \mu\sum_{i=0}^{D_2-1} e(n-i)U(n-i). \quad (7)$$

Note that in order to apply look-ahead exactly, $e(n)$ needs to be expressed as a function of $W(n-D_2)$ and then substituted into (7). However, this would make the resulting equation incredibly complicated and therefore very hardware expensive because (1) and (2) are coupled equations. Hence, we take (7) as our starting equation to which we apply the two relaxations mentioned in the previous section.

B. Application of Relaxed Look-Ahead

The delay relaxation introduces D_1 delays in the EFP. Applying the delay relaxation to (7), we get the following equation

$$W(n) = W(n-D_2) + \mu\sum_{i=0}^{D_2-1} e(n-D_1-i)U(n-D_1-i). \quad (8)$$

The second term in (7) is a summation of past and present gradient estimates. The validity of this relaxation is based on the assumption that the gradient estimate does not change much over D_1 samples. As mentioned in Section II(A), this assumption is not new and has been applied before to derive DLMS [20]–[21].

The hardware overhead in (8) is $N(D_2-1)$ adders. For large values of N and D_2, this overhead may not be acceptable. Hence we proceed further and apply sum relaxation which involves taking LA terms in (8) to obtain

$$W(n) = W(n-D_2) + \mu\sum_{i=0}^{LA-1} e(n-D_1-i)U(n-D_1-i), \quad (9)$$

where LA can vary from 1 to D_2. Finally, we substitute for $W(n-1)$ from (9) in (2) to get

$$\begin{aligned} e(n) &= d(n) - W^T(n-1)U(n) \\ &= d(n) - [W(n-D_2-1) \\ &\quad + \mu\sum_{i=0}^{LA-1} e(n-D_1-i-1)U \\ &\quad \cdot (n-D_1-i-1)]U(n). \end{aligned} \quad (10)$$

Assuming μ is sufficiently small and replacing $W(n-D_2-1)$ by $W(n-D_2)$, we can approximate (10) as

$$e(n) = d(n) - W^T(n-D_2)U(n). \quad (11)$$

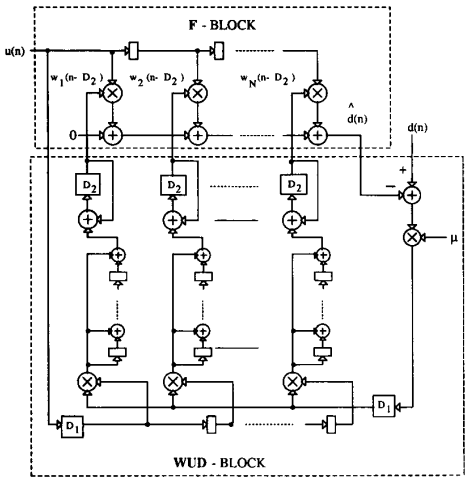


Fig. 2. The PIPLMS architecture.

The PIPLMS is completely described by (9) and (11) and its hardware overhead is $N(LA-1)$ adders. The PIPLMS architecture is shown in Fig. 2. The DLMS is equivalent to PIPLMS with $D_2 = 1$ and $LA = 1$, while PIPLMS1 is equivalent to PIPLMS with $LA = 1$ and $D_1 = D_2$.

IV. Convergence Properties of the PIPLMS

As the relaxed look-ahead technique modifies the input-output behavior, convergence analysis of PIPLMS needs to be carried out. However, this requires the knowledge of $E[U(n)U^T(n-i)]$ for $i > 0$, where $E[.]$ denotes expectation. As this is difficult to express in terms of the input covariance matrix $R = E[U(n)U^T(n)]$, therefore a complete convergence analysis of PIPLMS is intractable. This observation is in accordance with the analysis in [32], where only convergence of the weight vector is analyzed for the average LMS (ALMS) defined by

$$W(n) = W(n-1) + \frac{\mu}{P}\sum_{i=0}^{P-1} e(n-i)U(n-i). \tag{12}$$

However, analysis of convergence in the mean-squared error for ALMS is still difficult. Note that the ALMS is a special case of PIPLMS with $D_2 = 1$, $D_1 = 0$ and $LA = P$.

Keeping in mind the difficulties involved in a complete convergence analysis of PIPLMS, we provide an analysis of a useful sub-class of PIPLMS referred to as PIPLMSK. The PIPLMSK is obtained from PIPLMS with $LA = 1$. In addition, for the tractability of the convergence analysis we place the restriction that D_1 is a multiple of D_2. The equations describing PIPLMSK are therefore

$$W(n) = W(n-D_2) + \mu e(n-D_1)U(n-D_1) \tag{13}$$

where we additionally restrict D_1 to satisfy

$$D_1 = KD_2 \tag{14}$$

with D_2 being at least unity. Note that even though PIPLMSK is a special case of PIPLMS, it is general enough to cover DLMS and PIPLMS1. With $D_2 = 1$ (i.e., $K = D_1$) PIPLMSK is equivalent to DLMS, while $K = 1$ reduces PIPLMSK to PIPLMS1. Thus, DLMS and PIPLMS1 represent two extreme cases of PIPLMSK. The significance of analyzing PIPLMSK lies in the fact that in a practical situation the most appropriate value of K would lie somewhere between 1 and D_1. This is due to the fact that the computation time of $WUDL$ is just an adder time while that of EFP is much larger. Hence, $WUDL$ would need fewer pipelining latches than EFP, i.e., D_2 would be less than D_1 in general.

The convergence analysis of PIPLMSK is a generalization of the DLMS analysis presented in [20]–[21]. We shall also see that as K is varied from 1 to D_1 the convergence behavior changes gradually from that of PIPLMS1 to that of DLMS. In fact, it will be seen that all the analytical expressions derived in [20]–[21] in terms of D_1 are applicable to PIPLMSK by replacing D_1 by K. In this section we present the final analytical expressions, while the details are given in the appendices.

The following definitions are necessary before presenting the analytical expressions.

$$\lambda_{\text{rms}}^2 = \frac{\sum_{i=1}^{N}\lambda_i^2}{N} \tag{15}$$

$$\lambda_{av} = \frac{\sum_{i=1}^{N}\lambda_i}{N} = \sigma^2 \tag{16}$$

$$\alpha = \frac{\lambda_{\text{rms}}^2}{\lambda_{av}^2} \tag{17}$$

$$v = \frac{E[(u(n))^4]}{(E[(u(n))^2])^2} \tag{18}$$

$$P = N + v - 1 \tag{19}$$

$$b = \mu\sigma^2 \tag{20}$$

where λ_i's $(i = 1, \cdots, N)$ are the eigenvalues of R.

A. Bounds on μ for Convergence

The upper bound on μ to guarantee the convergence of the mean-squared error (MSE) of PIPLMSK is found to be tighter as compared to those for SLMS. In particular, the bound on μ for PIPLMSK is given by

$$0 \le \mu \le \frac{\alpha P + 2K - \sqrt{(\alpha P + 2K)^2 - 8K(K+1)}}{2K(K+1)\sigma^2}. \tag{21}$$

By replacing $K = D_1$, we get the corresponding bound for DLMS [eq(c.10), [21]]. In [21] it was shown that as D_1 increases the upper bound in (21) becomes smaller. Similarly, we can expect this bound to decrease as K is increased from 1 to D_1. This fact is confirmed via simulations in section VI. From (21) we also conclude that the upper bound can be kept constant with respect to D_1 if K is kept constant. This is a slight improvement over DLMS, for which the bound becomes less as D_1 is increased.

B. Convergence Speed

In [20]–[21], it was observed that the convergence speed of DLMS slows down slightly as D_1 is increased. On the other hand, the convergence speed of PIPLMS1 [23] was found to

be D_1 times lower than that of SLMS. Thus, as K decreases from D_1 down to unity, the convergence speed of PIPLMSK slows down. This fact has been known in the industry for quite some time and has been a major obstacle in achieving finer-grain pipelining. A major implication of the lower convergence speed is the corresponding degraded tracking capability in a non-stationary environment. It will be shown via simulations that PIPLMS offers a solution to this problem. Therefore, in a practical application, we would choose appropriate values of D_1 and D_2 for achieving the desired speed-up and then adjust LA to prevent the degradation in the convergence speed.

C. Adaptation Accuracy

The adaptation accuracy of an adaptive algorithm is quantified by its misadjustment, which is defined as follows

$$\mathcal{M} = \frac{\epsilon(\infty) - \epsilon_{\min}}{\epsilon_{\min}}, \tag{22}$$

where $\epsilon(n)$ is $E(J(n))$ and $J(n)$ is the mean-squared error at time instance n. The notation $\epsilon_{\min}$ refers to the minimum mean-squared error, which would be obtained if the filter weight vector $W(n)$ equalled the Weiner solution W_o.

The misadjustment for PIPLMSK was found to be

$$\mathcal{M} = \frac{\alpha N b}{2 - (\alpha P + 2K)b + K(K+1)b^2} \tag{23}$$

which reduces to the corresponding expression for DLMS([(c.11), [21]]) if K is replaced by D_1.

In (23), the term in the denominator which is linear in b dominates the quadratic term (since $Kb \ll 1$). Hence, as K is increased from unity towards D_1, the misadjustment would increase. Thus, PIPLMS1 [23] has better adaptation accuracy than DLMS. This fact is in accordance with the conclusions of [23] and also with that of the previous sub-section (i.e., faster convergence should imply worse misadjustment). In actual practice, the misadjustment does not change substantially as K varies and therefore can be considered to be approximately constant. This has been verified through simulations (see section VI).

V. The Pipelined ADPCM Codec

In this section, we use the relaxed look-ahead pipelined adaptive filter as a predictor in an ADPCM coder to demonstrate the usefulness of the adaptive filter in high-speed video applications or low-power speech applications. We first develop the pipelined ADPCM codec and then employ it for image and speech compression. Note that a high-speed architecture for DPCM has been presented [33], where the computation in the critical path is reduced to an amount equal to that of an addition. Our architecture allows much higher speed-ups and is more suited for systems with an adaptive predictor.

We start with a serial ADPCM (SADPCM) architecture [see Fig. 3] described by the following equations

$$\hat{s}(n) = W^T(n-1)\tilde{S}(n-1) \tag{24}$$

$$e(n) = s(n) - \hat{s}(n) \tag{25}$$

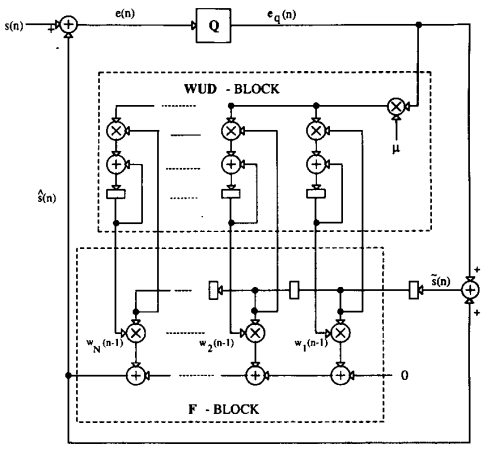


Fig. 3. The SADPCM coder.

$$e_q(n) = Q[e(n)] \tag{26}$$

$$\tilde{s}(n) = \hat{s}(n) + e_q(n) \tag{27}$$

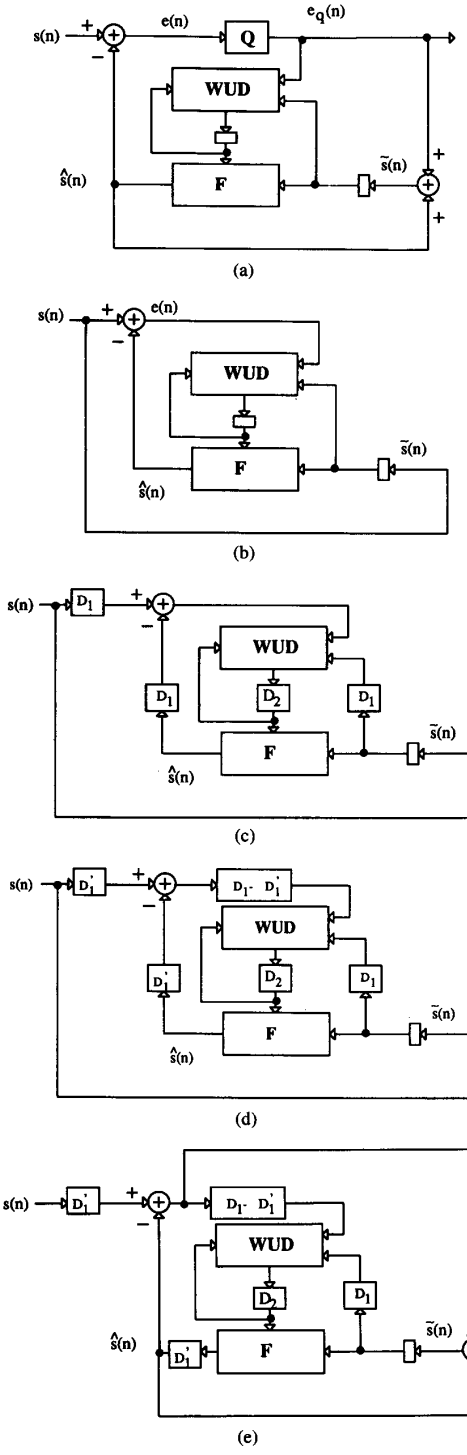
$$W(n) = W(n-1) + \mu e_q(n)\tilde{S}(n-1) \tag{28}$$

where n is the time index, $s(n)$ is the input signal, $\tilde{S}(n) = [\tilde{s}(n), \tilde{s}(n-1), \cdots, \tilde{s}(n-N+1)]^T$ is the vector of reconstructed signal $\tilde{s}(n)$ (N being the order of the predictor), $\hat{s}(n)$ is the predicted value of $s(n)$, $e(n)$ is the prediction error, $e_q(n)$ is the quantized value ($Q[.]$ representing the quantization operator) of $e(n)$, μ is the adaptation parameter and $W(n) = [w_1(n), w_2(n), \cdots, w_N(n)]^T$ is the vector of predictor coefficients. In Fig. 3, F is the predictor block, Q is a $R = \log_2 L$-bit quantizer (L being the number of quantizer levels) and WUD is the weight-update block. Note that the least-mean-squared (LMS) adaptation algorithm has been employed for updating the filter coefficients.

In order to develop a pipelined ADPCM architecture, we make the assumption of fine quantization, i.e., $q(n)$ is very small, where

$$q(n) = e(n) - e_q(n). \tag{29}$$

Under this assumption, the reconstructed signal $\tilde{s}(n) \approx s(n)$ and $e_q(n) = e(n)$. Thus, the SADPCM block diagram (see Fig. 3 for detailed description of blocks F and WUD) in Fig. 4(a) reduces to that of a conventional linear predictor [Fig. 4(b)]. Note that in contrast to Fig. 3, the delay elements are shown outside the WUD block in Fig. 4. The linear predictor was pipelined via relaxed look-ahead in section III. Thus, Fig. 4(c) shows the pipelined linear predictor. An architecture equivalent to Fig. 4(c) is shown in Fig. 4(d), where D_1' is the number of latches required to pipeline the F block, the two adders and the quantizer Q. To convert back to the ADPCM architecture we need to generate $\tilde{s}(n)$ and employ it for future predictions. This is done in Fig. 4(e), where the predictor output is added to the prediction error to obtain $\tilde{s}(n)$. Finally, replacing the quantizer and replacing the D_1' delays at the input with L_c delays, we get the PIPADPCM coder architecture of Fig. 5(a). The PIPADPCM decoder is shown in Fig. 5(b).



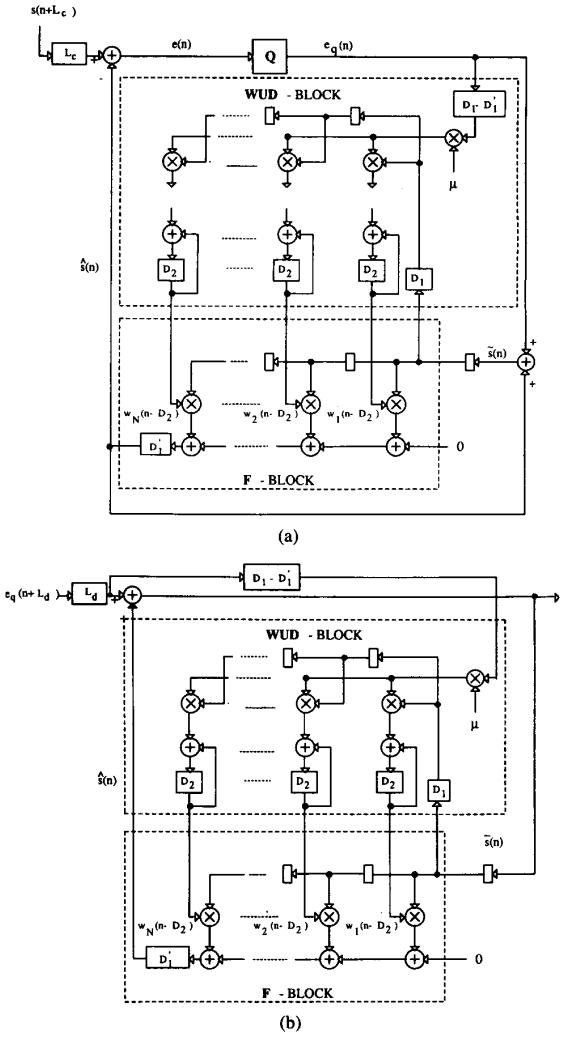


Fig. 5. The PIPADPCM (a) coder and (b) decoder.

Fig. 4. Derivation of PIPADPCM from SADPCM.

It can be seen that the hardware requirements are essentially independent of the number of quantizer levels L and the speed-up M. Note that the latches L_c [see Fig. 5(a)] would be employed to pipeline the adder and the quantizer in the coder. Similarly, the latches L_d [see Fig. 5(b)] would be employed to pipeline the adder in the decoder. This points to an interesting observation about the output latency, which from Fig. 5 is L_c+L_d+1. Assuming that an adder computation time $T_a = 20$ units and a multiplier computation time $T_m = 40$ units, the quantizer computation time $T_q = T_a$, and a predictor order $N = 3$, we have calculated the values of D_1, D_1', D_2, L_c and L_d for different speed-ups M and these are shown in Table I. Clearly, the output latency is much smaller than the level of pipelining.

A. Analysis of PIPADPCM

As certain approximations have been made in the derivation of PIPADPCM, its performance would be degraded as compared to SADPCM. We briefly discuss the nature of this degradation. There are two sources of performance loss in PIPADPCM. The first source of degradation is due to the fact that the PIPADPCM predictor is constrained to perform a

TABLE I
PIPADPCM PARAMETERS FOR DIFFERENT SPEED-UPS

M	D_1	D_1'	D_2	L_c	L_d
1	0	0	1	0	0
2	2	1	1	1	0
5	5	3	1	1	0
11	10	6	1	2	0
20	20	13	2	4	1
44	40	27	4	8	3

$D_1' + 1$-step forward prediction rather than a 1-step prediction. Note that this constraint arose in the transition from Fig. 4(d) to Fig. 4(e).

Under the assumption of fine quantization and a stationary environment it can be shown (see Appendix C) that the misadjustment of PIPADPCM ($\mathcal{M}_P$) is related to that of SADPCM by the following equation

$$\mathcal{M}_P = (\mathcal{M}_S + 1)\tau - 1, \tag{30}$$

where τ is the ratio of the optimal mean-squared prediction error due to $D_1' + 1$-step forward prediction and that due to a 1-step forward prediction. Assuming a fixed uniform R-bit quantizer, the reconstruction error power σ_r^2 of an ADPCM codec is related to its prediction error power σ_e^2 as

$$\sigma_r^2 = \epsilon_* 2^{-2R} \sigma_e^2, \tag{31}$$

where ϵ_* is a constant. Employing (31) to equate the reconstruction powers of SADPCM and PIPADPCM and with (30), we can derive the following relationship between the number of quantizer bits for PIPADPCM (R_P) and for SADPCM (R_S) for the same output SNR

$$R_P = R_s + \frac{1}{2}\log_2 \tau. \tag{32}$$

Thus, τ needs to be at least 4 before we need to increase R_P, for PIPADPCM to have the same SNR as SADPCM. However, in all our simulations we have kept $R_P = R_S$.

The second source of SNR degradation arises when $D_2 > 1$. From the convergence analysis of PIPLMSK (see section IV(C)), we know that if $D_2 > 1$ then the convergence speed is reduced, which would result in additional loss of performance in a non-stationary environment. However, this loss can be reduced by applying sum relaxation.

B. Image Coding

We consider only intra-frame prediction with a fixed uniform quantizer and an adaptive predictor. Our purpose is to compare the relative performance of SADPCM and PIPADPCM under the same quantization scheme. We assume that the input to the codec is in a row-by-row raster scan format. As PIPADPCM makes use of a $D_1' + 1$-step forward prediction, therefore for large values of D_1', the prediction error would increase if one or more of the pixels employed in the prediction are in the same row as the pixel being predicted. However, due to the two-dimensional nature of image signals, we can always employ pixels from the previous line to predict the current pixel. This can be done by including additional latches at the output of the adder [in Fig. 5(a)], which generates the reconstructed signal $\tilde{s}(n)$. Our scheme of prediction of the current pixel using neighboring pixels from the previous line is shown in Fig. 6. Prediction in this fashion results in both SADPCM and PIPADPCM having the same prediction error powers and, therefore, there would be no degradation in SNR due to the $D_1' + 1$-step forward prediction. There would be some loss in SNR if $D_2 > 1$ and in the next section we show that this loss is negligible.

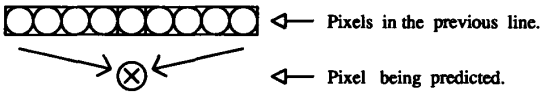


Fig. 6. Pixels employed for prediction.

C. Speech Coding

Unlike images, speech is a one-dimensional signal. Hence, PIPADPCM cannot avoid performing a $D_1' + 1$-step forward prediction. Thus, in speech coding the degradation in SNR is caused by the increase in the prediction error power due to the $D_1' + 1$-step forward prediction. This limits the acceptable values of D_1' and therefore the achievable speed-ups. Nevertheless, substantial speed-ups are possible.

VI. SIMULATION RESULTS

In this section, we present simulation results to verify the performance of PIPLMS and PIPADPCM. In Experiment A, we verify the theoretical expression for the adaptation accuracy (23) and the upper bound on μ (21). In Experiment B, we demonstrate finer-grain pipelining with PIPLMS and show how a desired level of speed-up can be attained without slowing down the convergence speed. In Experiment C, we demonstrate the performance of PIPADPCM in an image coding application, while a speech coding application is presented in Experiment D.

A. Experiment A

In this experiment, we employ linear prediction to verify (21) and (23). For this purpose a 50th order FIR filter was used to generate a moving-average (MA) process with unit variance. Then PIPLMSK, with $N = 20$ and $\mu = 0.01$, was employed to predict the MA process. Simulations were run for 30 independent trials and the results were averaged. Values of $D_1 = 8$ and $D_1 = 16$ were chosen for different values of K. The theoretical and measured values of $\mathcal{M}$ and $\mu_{\max}$ (the maximum value of μ for convergence) are tabulated in Table II.

It can be seen that the theoretical and calculated values of $\mathcal{M}$ and $\mu_{\max}$ are quite close. Also, note the slight degradation in $\mathcal{M}$ and the decrease in $\mu_{\max}$ as K is increased.

B. Experiment B

In Experiment B we illustrate, with a concrete example, the actual increase in clock-frequency due to pipelining. Again we assume that a two operand multiply time T_m is 40 units while a two operand add time T_a is 20 units. For simplicity, we perform a system identification of a 4th order FIR system

TABLE II
EXPERIMENT A

D_1	K	Misadjustment (%) Measured	Theory	μ_{max} Measured	Theory
8	1	13.53	14.10	0.04	0.07
	2	12.30	14.26	0.035	0.066
	4	13.40	14.60	0.03	0.06
	8	12.25	15.29	0.02	0.05
16	1	15.0	14.11	0.04	0.07
	2	14.87	14.26	0.035	0.066
	4	13.68	14.60	0.03	0.06
	8	14.44	15.29	0.02	0.05
	16	14.63	16.77	0.015	0.04

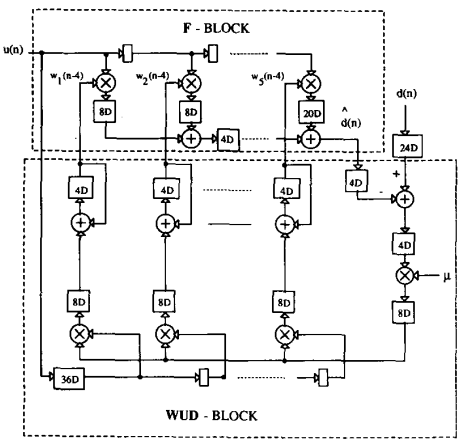


Fig. 7. The PIPLMS11 architecture.

with a 5th order adaptive filter. The simulation results were averaged over 100 independent trials.

The computation time of a 5th order SLMS filter is $T_c = 240$. For a speed-up of $M = 48$, the clock-period of the pipelined system should be $T_p = 5$. This is much less than the adder computation of 20. Hence, the DLMS cannot achieve this speed-up. A simple analysis shows that DLMS can achieve a maximum speed-up of 12, for $N = 5$. However, PIPLMSK with $D_1 = 44$ and $D_2 = 4$ (i.e., $K = 11$) can achieve the desired clock frequency. This is shown in Fig. 7, where the retimed PIPLMS11 (i.e., PIPLMSK with $K = 11$) architecture is depicted (see [30]–31] for details of retiming). The latches lumped at the output of a hardware operator would be used to pipeline that operator. In Fig. 7, each adder is pipelined into 4 stages while each multiplier is pipelined into 8 stages.

As mentioned in section IV, even though PIPLMS11 achieves the desired speed-up, its convergence speed (in number of iterations) would be slow due to the inclusion of latches in $WUDL$. This fact can be seen in Fig. 8, where the mean-squared error (MSE) plots for LMS, DLMS (with $D_1 = 44$) and PIPLMS11 are shown. Clearly, PIPLMS11 converges much slower than either SLMS or DLMS. The latency due to the inclusion of D_1 latches is also visible in the delayed response of DLMS and PIPLMS11.

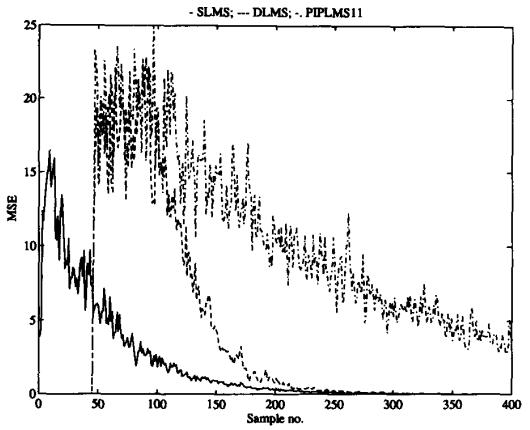


Fig. 8. MSE plots for SLMS, DLMS (with $D_1 = 44$) and PIPLMS11 (with $D_1 = 44$ and $D_2 = 4$).

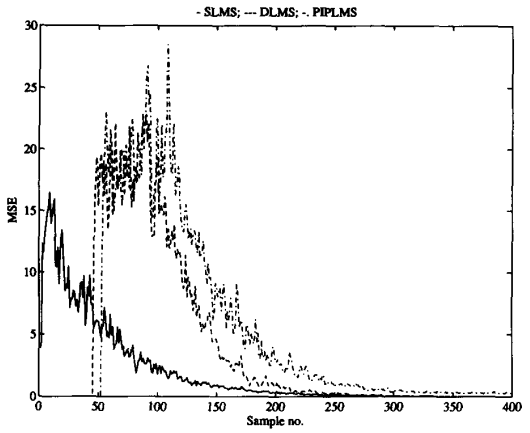


Fig. 9. MSE plots for SLMS, DLMS (with $D_1 = 44$) and PIPLMS (with $D_1 = 51$, $D_2 = 4$, and $LA - 2$.

The slower convergence of PIPLMS11 can be rectified by using sum relaxation in PIPLMS with $LA = 2$. As $LA = 2$ implies increased computation time of EFP, hence we need to increase the value of D_1 from 44 to 51. As was observed in the case of DLMS, increasing the value of D_1 hardly affects the convergence speed. The MSE plots of SLMS, DLMS ($D_1 = 44$) and PIPLMS ($D_1 = 51$, $D_2 = 4$, $LA = 2$) are shown in Fig. 9. Note the remarkable improvement in convergence speed of PIPLMS over that of PIPLMS11. Thus, PIPLMS with $D_1 = 51$, $D_2 = 4$ and $LA = 2$ achieves a speed-up of $M = 48$ with no degradation in the convergence speed and a slight increase in the misadjustment.

The retimed PIPLMS architecture achieving the desired speed-up of 48 is shown in Fig. 10, where the latches lumped at the output of each adder and multiplier would be used for pipelining in an actual implementation.

C. Experiment C

In order to see the increase in clock frequency, we first show a PIPADPCM coder with a speed-up of 44. Employing

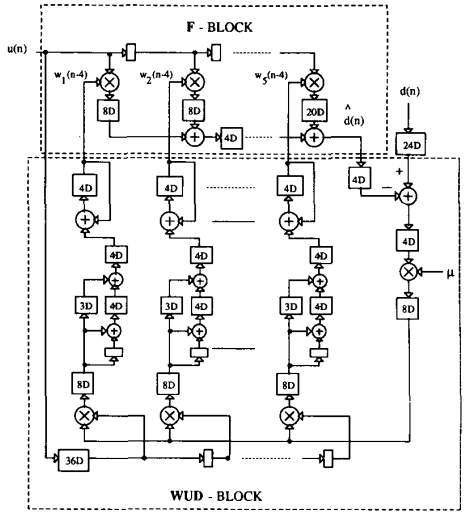


Fig. 10. The PIPLMS architecture with speed-up of 48.

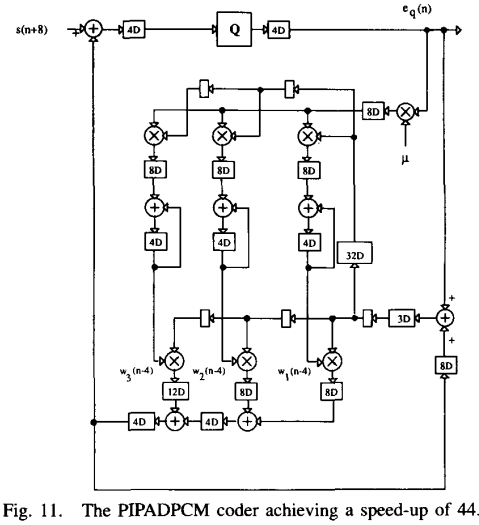


Fig. 11. The PIPADPCM coder achieving a speed-up of 44.

(a)

(b)

(c)

Fig. 12. Image coding example: (a) original, (b) SADPCM output and (c) PIPADPCM output for a speed-up of 44.

the appropriate values of D_1, D_1', D_2 and L_c from Table I, we show the retimed PIPADPCM coder architecture for speed-up $M = 44$ in Fig. 11.

Next, we consider the compression of 256 × 256 input image of "Lenna" [see Fig. 12(a)] with 256 gray levels. All simulations in this experiment were done with a 3-bit uniform and fixed quantizer with a dynamic range of 0.25. The predictor order was $N = 3$ and $LA = 1$. All the parameter values for PIPADPCM were obtained from Table I. The reconstructed image for SADPCM [see Fig. 12(b)] and that of PIPADPCM [see Fig. 12(c)] for a speed-up of 44 are perceptually identical. In Fig. 13, we plot the SNR values for different values of speed-ups. Clearly, the performance of PIPADPCM degrades at speed-ups where $D_1 > 1$. This degradation is approximately 0.97 dB at $M = 44$. However, with $LA = 2$, it was found that the SNR loss was reduced to 0.43 dB. Further SNR improvements are possible by optimizing the dynamic range and the stepsize μ. This is, however, not necessary as in all cases the PIPADPCM output is perceptually very close to that of SADPCM.

D. Experiment D

In this experiment, we use the ADPCM coder in speech coding applications. The input speech waveform that has been coded is shown in Fig. 14. This waveform was obtained by sampling the utterance "zoos" at 8 kHz for a duration of 1 second and then scaling. All simulations are carried out with a 4-bit quantizer. The segmental SNR (segSNR) is employed as performance measure and has been computed by averaging the SNR over speech segments of 128 samples (or 16 ms).

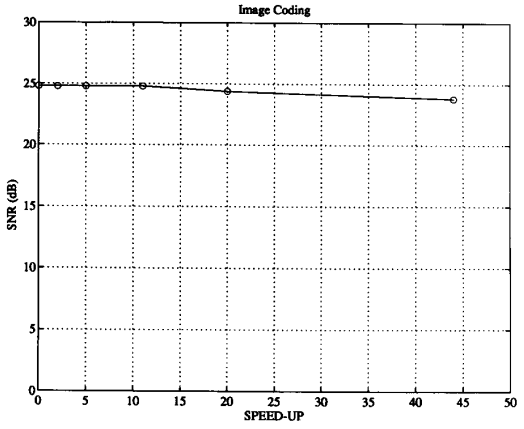


Fig. 13. SNR vs. Speed-up for the image codec.

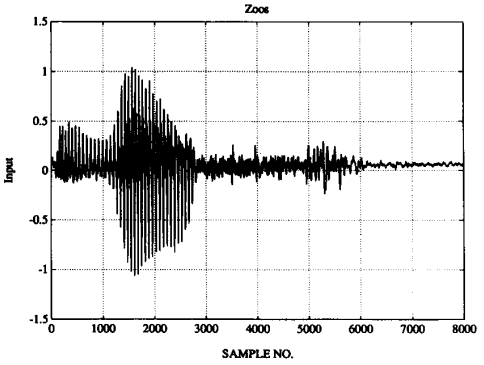


Fig. 14. The input speech signal.

The plot of segSNR *vs.* speed-up is shown in Fig. 15, where we see that as the speed-up is increased, the segSNR for PIPADPCM decreases initially and then levels off for all higher speed-ups. At a speed-up of 44, the SNR drop is about 4.23 dB. At this point it is not clear whether this degradation is perceptually audible or not. It is well known that there is not much correlation between segSNR and subjective measures of performance. Therefore, subjective listening tests need to be done in order to arrive at a definitive evaluation of the PIPADPCM in speech compression. Nevertheless, speed-up of 2 can be achieved with a SNR degradation of less than 1 dB.

VII. Conclusions

The relaxed look-ahead is introduced as an attractive technique for the development of pipelined adaptive filter architectures. This technique has been employed to develop a pipelined LMS adaptive filter architecture (referred to as PIPLMS) and a pipelined ADPCM (PIPADPCM) codec. Both PIPLMS and PIPADPCM have a very low hardware overhead due to pipelining and are therefore attractive from a VLSI implementation point of view. Finite-precision properties of the PIPADPCM have been studied [34] and a VLSI chip has been successfully implemented [35].

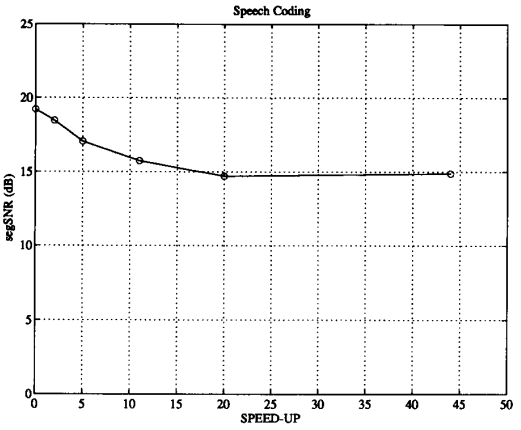


Fig. 15. SNR vs. Speed-up for the speech codec.

The pipelined topologies proposed in this paper can be systematically unfolded [36]–[37] to obtain concurrent parallel processing adaptive filter architectures. These parallel and pipelined architectures can be used either in the context of high-speed applications such as video and radar or low-power applications such as speech and portable communications systems.

In our group, we have also followed similar approaches to design inherently concurrent direct-form and lattice recursive digital filters [38]. Future work is being directed towards design of inherently concurrent algorithms for adaptive decision feedback equalizers [39] and adaptive quantizers.

VIII. Appendix A
Derivation of the Misadjustment Expression and Bounds on μ for Convergence

We rewrite the PIPLMSK equations below

$$W(n) = W(n - D_2) + \mu e(n - D_1)U(n - D_1) \quad (33)$$

$$e(n) = d(n) - W(n - D_2)^T U(n) \quad (34)$$

It is easy to show that

$$\begin{aligned} J(n) &= E[e^2(n) \mid W(n - D_2)] \\ &= \epsilon_{\min} + J_{\text{ex}}(n - D_2) \end{aligned} \quad (35)$$

where $\epsilon_{\min}$ is the minimum mean-squared error and from the independence assumption

$$J_{\text{ex}}(n) = (W(n) - W_o)^T R(W(n) - W_o). \quad (36)$$

with W_o being the optimal weight vector. Define the weight-error vector $C(n)$ as

$$C(n) = W(n) - W_o, \quad (37)$$

and $\nu(n)$ as

$$\nu(n) = Q^T C(n) \quad (38)$$

where Q is the matrix whose columns are composed of the eigenvectors of data covariance matrix R, i.e., $R = Q\Lambda Q^T$ with Λ being the diagonal matrix containing the eigenvalues of R.

Employing (37) and (38), we can rewrite (36) as

$$J_{ex}(n) = \nu^T(n)\Lambda\nu(n). \tag{39}$$

Taking the expectation of (39) gives us

$$\begin{aligned} E[J_{ex}(n)] &= E[\nu^T(n)\Lambda\nu(n)] \\ &= \epsilon_{ex}(n) \end{aligned} \tag{40}$$

From (34), we get

$$\begin{aligned} e(n) &= d(n) - (C(n-D_2) + W_o)^T U(n) \\ &= e_{opt}(n) - \nu^T(n-D_2)Z(n) \end{aligned} \tag{41}$$

where

$$e_{opt} = d(n) - W_o^T U(n) \tag{42}$$

and $Z(n) = Q^T U(n)$. Note that the following is true

$$E[Z(n)Z^T(n)] = \Lambda. \tag{43}$$

Assuming that $u(n)$ is a Gaussian process, we will use the following result from [20],

$$\begin{aligned} E[Z(n-D_1)Z^T(n-D_1)\Lambda Z(n-D_1)Z^T(n-D_1)] \\ = N\lambda_{rms}^2\Lambda + (v-1)\Lambda^3 \end{aligned} \tag{44}$$

where λ_{rms}^2 and v are defined in section IV (see (15) and (18)).

Subtracting W_o from (33) and then premultiplying by Q^T results in

$$\nu(n) = \nu(n-D_2) + \mu e(n-D_1)Z(n-D_1). \tag{45}$$

Substituting (41) into (45), we get

$$\begin{aligned} \nu(n) = \nu(n-D_2) - \mu Z(n-D_1)Z^T(n-D_1)\nu \\ \cdot(n-D) + \mu e_{opt}(n-D_1)Z(n-D_1) \end{aligned} \tag{46}$$

where

$$D = D_1 + D_2. \tag{47}$$

We may now substitute for $\nu(n)$ from (46) into (40). As (46) has three terms, therefore substitution into (40) and multiplying out would give us 9 terms. We consider each of these terms individually.

$T1 = E[\nu^T(n-D_2)\Lambda\nu(n-D_2)]$

From (40) we get

$$T1 = \epsilon_{ex}(n-D_2) \tag{48}$$

$$T2 = -\mu E[\nu^T(n-D_2)\Lambda Z(n-D_1)Z^T(n-D_1)\nu(n-D)]$$

Employing (43) and (15)–(20), we approximate $T2$ as follows

$$\begin{aligned} T2 &= -\mu E[\nu^T(n-D_2)\Lambda^2\nu(n-D)] \\ &\approx -\mu\lambda_{av}E[\nu^T(n-D_2)\Lambda\nu(n-D)] \\ &= -\mu\lambda_{av}a_{D_1}(n-D_2) \\ &= -ba_{D_1}(n-D_2) \end{aligned} \tag{49}$$

where

$$a_k(m) = E[\nu^T(m)\Lambda\nu(m-k)]. \tag{50}$$

We express (50) in terms of $\epsilon_{ex}(n)$ in Appendix B.

$T3 = \mu E[\nu^T(n-D_2)\Lambda e_{opt}(n-D_1)Z(n-D_1)]$.

Knowing that $E[e_{opt}(n)] = 0$, we approximate $T3$ as

$$\begin{aligned} T3 &\approx \mu E[e_{opt}(n-D_1)]E[\nu^T(n-D_2)\Lambda Z(n-D_1)] \\ &= 0. \end{aligned}$$

$T4 = -\mu E[\nu^T(n-D)Z(n-D_1)Z^T(n-D_1)\Lambda\nu(n-D_2)]$.

Comparing $T4$ and definition of $T2$ we get

$$T4 = T2 \approx -ba_{D_1}(n-D_2). \tag{51}$$

$T5 = \mu^2 E[\nu^T(n-D)Z(n-D_1)Z^T(n-D_1)\Lambda Z(n-D_1)Z^T(n-D_1)\nu(n-D)]$.

Employing (40), (44) and (15)–(20), we write $T5$ as

$$\begin{aligned} T5 &= \mu^2 E[\nu^T(n-D)(N\lambda_{rms}^2\Lambda + (v-1)\Lambda^3)\nu(n-D)] \\ &= \mu^2 N\lambda_{rms}^2 E[\nu^T(n-D)\Lambda\nu(n-D)] \\ &\quad + \mu^2(v-1)E[\nu^T(n-D)\Lambda^3\nu(n-D)] \\ &\approx \mu^2 N\lambda_{rms}^2\epsilon_{ex}(n-D) + \mu^2(v-1)\lambda_{rms}^2 E \\ &\quad \cdot[\nu^T(n-D)\Lambda\nu(n-D)] \\ &= \mu^2\lambda_{rms}^2(N+v-1)\epsilon_{ex}(n-D) \\ &= \alpha P b^2\epsilon_{ex}(n-D). \end{aligned} \tag{52}$$

The terms $T6$, $T7$ and $T8$ are approximately equal to zero as $E[e_{opt}(n)] = 0$. This can be shown in exactly the same fashion $T3$ was shown to be zero.

$T9 = \mu^2 E[e_{opt}^2(n-D_1)Z^T(n-D_1)\Lambda Z(n-D_1)]$.

We employ (15)–(20) to approximate $T9$ as follows

$$\begin{aligned} T9 &= \mu^2 E[e_{opt}^2(n-D_1)]E[Z^T(n-D_1)\Lambda Z(n-D_1)] \\ &= \mu^2\epsilon_{min}N\lambda_{rms}^2 \\ &= \epsilon_{min}N\alpha b^2 \end{aligned} \tag{53}$$

where the relation

$$E[Z^T(n-D_1)\Lambda Z(n-D_1)] = N\lambda_{rms}^2 \tag{54}$$

is employed and is obtained from (43).

Therefore, adding all the nine terms together, we can write (40) as follows

$$\begin{aligned} \epsilon_{ex}(n) = \epsilon_{ex}(n-D_2) - 2ba_{D_1}(n-D_2) \\ +\alpha P b^2\epsilon_{ex}(n-D) + \epsilon_{min}\alpha N b^2. \end{aligned} \tag{55}$$

In Appendix B, we show that the three most significant terms in evaluating $a_{D_1}(n-D_2)$ are

$$\begin{aligned} &a_{D1}(n-D_2) \\ &= \epsilon_{\text{ex}}(n-D) - b\sum_{i=1}^{K}\epsilon_{\text{ex}}(n-D-iD_2) \\ &\quad + b^2\sum_{i=1}^{K}\sum_{j=1}^{i}\epsilon_{\text{ex}}(n-2D_1-(j+1)D_2). \end{aligned} \tag{56}$$

Substituting (56) into (55) and then letting $n \to \infty$, we get the desired expression for misadjustment

$$\mathcal{M} = \frac{\epsilon_{\text{ex}}(\infty)}{\epsilon_{\min}} \approx \frac{\alpha N b}{2-(\alpha P+2K)^b + K(K+1)b^2}. \tag{57}$$

Equating the denominator of (57) to zero, we solve for b and get the upper bound on μ for convergence

$$0 \le \mu \le \frac{\alpha P + 2K - \sqrt{(aP+2K)^2 - 8K(K+1)}}{2K(K+1)\sigma^2}. \tag{58}$$

IX. Appendix B

We start with (50) which is reproduced below

$$\begin{aligned} a_k(m) &= E[\nu^T(m)\Lambda\nu(m-k)] \\ &= E[\nu^T(m-k)\Lambda\nu(m)]. \end{aligned} \tag{59}$$

Note that

$$a_0(m) = \epsilon_{\text{ex}}(m). \tag{60}$$

We employ (46) to expand (59) as follows

$$\begin{aligned} a_k(m) &= E[\nu^T(m-D_2)\Lambda\nu(m-k)] \\ &\quad - \mu E[\nu^T(m-D)Z(m-D_1)Z^T \\ &\quad \cdot (m-D_1)\Lambda\nu(m-k)] \\ &\quad + \mu E[\epsilon_{\text{opt}}(m-D_1)Z^T(m-D_1)\Lambda\nu(m-k)]. \end{aligned} \tag{61}$$

The third term in (61) is approximately zero as $E[e_{\text{opt}}(m)] = 0$. Hence, (61) can be written as

$$a_k(m) = a_{k-D_2}(m-D_2) - b a_{D_1-k+D_2}(m-k) \tag{62}$$

where we use the approximation

$$\begin{aligned} &E[\nu^T(m-D)Z(m-D_1)Z^T(m-D_1)\Lambda\nu(m-k)] \\ &\qquad = \lambda_{av}E[\nu^T(m-D)\Lambda\nu(m-k)] \end{aligned} \tag{63}$$

which can be made if the eigenvalues of R are nearly equal.

Now, assume k in (62) is a multiple of D_2, i.e.,

$$k = LD_2. \tag{64}$$

Hence, we recursively substitute for $a_{k-D_2}(m-D_2)$ in (62) to get

$$a_k(m) = a_0(m-LD_2) - b\sum_{i=1}^{L} a_{D_1-k+iD_2}(m-k). \tag{65}$$

We need to find $a_{D_1}(n-D_2)$, hence we substitute $k = D_1$ and $m = n - D_2$ into (65). Note that from (14), $D_1 = KD_2$ and therefore (64) is satisfied. Making these substitutions we get

$$a_{D_1}(n-D_2) = a_0(n-D) - b\sum_{i=1}^{K} a_{iD_2}(n-D). \tag{66}$$

Employ (65) to substitute for the term inside the summation in (66). This can be done as (64) is satisfied. Thus, we get

$$\begin{aligned} &a_{D_1}(n-D_2) \\ &= a_0(n-D) - b\sum_{i=1}^{K} a_0(n-D-iD_2) \\ &\quad + b^2\sum_{i=1}^{K}\sum_{j=1}^{i} a_{D_1-(i-j)D_2}(n-D-iD_2). \end{aligned} \tag{67}$$

Substituting (65) again for the term inside the double summation of (67), which can be done again as (64) is satisfied, we get

$$\begin{aligned} &a_{D_1}(n-D_2) \\ &= a_0(n-D) - b\sum_{i=1}^{K} a_0(n-D-iD_2) \\ &\quad + b^2\sum_{i=1}^{K}\sum_{j=1}^{i} a_0(n-2D_1-(j+1)D_2), \end{aligned} \tag{68}$$

where we have neglected higher order terms containing b^3. This is a valid approximation as kb can be made small as compared to unity. Employing (60) we rewrite (68) as

$$\begin{aligned} &a_{D_1}(n-D_2) \\ &= \epsilon_{\text{ex}}(n-D) - b\sum_{i=1}^{K} \epsilon_{\text{ex}}(n-D-iD_2) \\ &\quad + b^2\sum_{i=1}^{K}\sum_{j=1}^{i} \epsilon_{\text{ex}}(n-2D_1-(j+1)D_2) \end{aligned} \tag{69}$$

to get (56).

X. Appendix C

In the following development, we assume a stationary environment and the fine quantization assumption. Denoting the optimal mean-squared prediction error power for PIPADPCM by $JMIN_{eP}$ and that of SADPCM by $JMIN_{eS}$, we can define the misadjustment of PIPADPCM ($\mathcal{M}_P$) with respect to $JMIN_{eS}$ as follows:

$$\mathcal{M}_P = \frac{\sigma_{eP}^2 - JMIN_{eS}}{JMIN_{eS}}, \tag{70}$$

where σ_{eP}^2 is the steady-state prediction error power of PIPADPCM. Similarly, the misadjustment of SADPCM ($\mathcal{M}_S$)

is defined as

$$\mathcal{M}_S = \frac{\sigma_{eS}^2 - JMIN_{eS}}{JMIN_{eS}}, \tag{71}$$

where σ_{eS}^2 is the steady-state prediction error power of SAD-PCM. Note that conventionally $\mathcal{M}_P$ is defined with respect to $JMIN_{eP}$. However, in the present context we wish to compare $\mathcal{M}_P$ with $\mathcal{M}_S$ and therefore the definition in (70) is used.

We now manipulate (70) as follows

$$\begin{aligned}\mathcal{M}_P &= \frac{\sigma_{eP}^2 - JMIN_{eP} + (JMIN_{eP} - JMIN_{eS})}{JMIN_{es}} \\ &= \tau\left[\frac{\sigma_{eP}^2 - JMIN_{eP}}{JMIN_{eP}} + 1 - \frac{1}{\tau}\right] \\ &= \left[\left(\frac{\sigma_{eP}^2 - JMIN_{eP}}{JMIN_{eP}}\right)\tau + \tau - 1\right] \end{aligned} \tag{72}$$

where $\tau = JMIN_{eP}/JMIN_{eS}$. The term in the round brackets in (72) represents the inherent misadjustment of PIPADPCM. As the inherent misadjustment is completely dependent on the eigenvalue spread of the input correlation matrix therefore the inherent misadjustment of PIPADPCM would be identical to that of PIPLMS in a 1-step predictor mode. From the analysis in section IV, we know that the misadjustment of PIPLMS (in a 1-step forward predictor configuration) is more or less constant with increase in D_1 and equal to that of serial LMS (or SLMS). Thus, we can write (72) as

$$\mathcal{M}_P = (\mathcal{M}_S + 1)\tau - 1. \tag{73}$$

which is the desired equation.

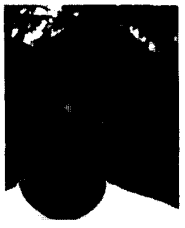
ACKNOWLEDGMENT

The authors would like to acknowledge the suggestions and comments of an anonymous reviewer, which led to substantial improvements in the paper.

REFERENCES

[1] K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proceedings of the 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. on Acoustics, Speech and Signal Proc.*, vol. 37, pp. 1099–1117, July 1989.
[3] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters—Part II: Pipelined incremental block filtering," *IEEE Trans. on Acoustics, Speech and Signal Proc.*, vol. 27, pp. 1118–1134, July 1989.
[4] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power CMOS digital design," *IEEE J. of Solid-State Circuits*, vol. 27, pp. 473–484, April 1992.
[5] P. M. Kogge, "Parallel solution of recurrence problems," *IBM J. Res. Develop.*, vol. 18, pp. 138–148, March 1974.
[6] H. H. Loomis and B. Sinha, "High speed recursive digital filter realization," *Circuits, Syst., Signal Processing*, vol. 3, no. 3, pp. 267–294, 1984.
[7] K. K. Parhi and D. G. Messerschmitt, "Concurrent architectures for two-dimensional recursive digital filtering," *IEEE Trans. on Circuits and Systems*, vol. 36, pp. 813–829, June 1989.
[8] G. Fettweis and H. Meyr, "Parallel Viterbi decoding by breaking the compare-select feedback bottleneck," *IEEE Trans. on Communications*, vol. 37, pp. 785–790, Aug. 1989.
[9] H.-D. Lin and D. G. Messerschmitt, "Finite state machine has unlimited concurrency," *IEEE Trans. on Circuits and Systems*, vol. 38, pp. 465–475, May 1991.
[10] K. K. Parhi, "Pipelining in dynamic programming architectures," *IEEE Trans. on Signal Processing*, vol. 39, pp. 1442–1450, June 1991.
[11] K. K. Parhi, "Pipelining in algorithms with quantizer loops," *IEEE Trans. on Circuits and Systems*, vol. 38, pp. 745–754, July 1991.
[12] K. K. Parhi, "High-speed VLSI architectures for Huffman and Viterbi decoders," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, pp. 385–391, June 1992.
[13] M. Hatamian and K. K. Parhi, "An 85 MHz 4th order programmable IIR digital filter chip," *IEEE J. of Solid-State Circuits*, pp. 175–183, Feb. 1992.
[14] G. A. Clark, S. K. Mitra, and S. R. Parker, "Block implementation of adaptive digital filters," *IEEE Trans. on Acoustics, Speed and Signal Proc.*, vol. 29, pp. 744–752, June 1981.
[15] J. M. Cioffi, "The block-processing FTF adaptive algorithm," *IEEE Trans. on Acoustics, Speech and Signal Proc.*, vol. 34, pp. 77–90, Feb. 1986.
[16] J. M. Cioffi, P. Fortier, S. Kasturia, and G. Dudevoir, "Pipelining the decision feedback equalizer," *IEEE DSP Workshop*, 1988.
[17] A. Gatherer and T. H.-Y. Meng, "High sampling rate adaptive decision feedback equalizer," *IEEE Trans. on Signal Processing*, vol. 41, pp. 1000–1005, Feb. 1993.
[18] T. Meng and D. G. Messerschmitt, "Arbitrarily high sampling rate adaptive filters," *IEEE Trans. on Acoustics, Speech and Signal Proc.*, vol. 35, pp. 455–470, April 1987.
[19] K. K. Parhi and D. G. Messerschmitt, "Concurrent cellular VLSI adaptive filter architectures," *IEEE Trans. on Circuits and Systems*, vol. 34, pp. 1141–1151, Oct. 1987.
[20] 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.
[21] G. Long, F. Ling, and J. G. Proakis, "Corrections to "The LMS algorithm with delayed coefficient adaptation"," *IEEE Trans. Signal Processing*, vol. 40, no. 1, pp. 230–232, Jan. 1992.
[22] M. D. Meyer and D. P. Agrawal, "A modular pipelined implementation of a delayed LMS transversal adaptive filter," *Proc. 1990 IEEE Int. Symp. Circuits and Systems*, New Orleans, pp. 1943–1946.
[23] N. R. Shanbhag and K. K. Parhi, "A pipelined LMS adaptive filter architecture," *Proc. 25th Asilomar Conf. on Sig., Sys., and Comput.*, Calif., Nov. 1991, pp. 668–672.
[24] K. Konstantinides and N. Kanopoulos, "Design alternatives for adaptive digital lattice filters and a new bit-serial architecture," *IEEE Trans. Circuits and Systems*, vol. CAS-34, no. 7, July 1987.
[25] D. Chester, R. Young, and M. Petrowski, "A fully systolic adaptive filter implementation," *Proc. of 1991 Acous., Speech, Signal Processing*, Toronto, pp. 2109–2112.
[26] N. R. Shanbhag and K. K. Parhi, "A high-speed architecture for ADPCM coder and decoder," in *Proc. IEEE Intl. Symp. on Circuits and Systems*, San Diego, May 1992, pp. 1499–1502.
[27] N. R. Shanbhag and K. K. Parhi, "A pipelined adaptive lattice filter architecture," *IEEE Trans. on Signal Processing*, vol. 41, pp. 1925–1939, May 1993.
[28] N. R. Shanbhag and K. K. Parhi, "A pipelined differential vector quantizer architecture for real-time video applications," in *Proc. IEEE Workshop on Visual Signal Processing and Communications*, Raleigh, NC, Sept. 1992, pp. 9–14.
[29] N. R. Shanbhag and K. K. Parhi, "A pipelined adaptive differential vector quantizer for low-power speech coding applications," *IEEE Trans. on Circuits and Systems—II: Analog and Digital Signal Proc.*, May 1993, pp. 347–349.
[30] C. Leiserson and J. Saxe, "Optimizing synchronous systems," *J. of VLSI and Computer Systems*, vol. 1, pp. 41–67, 1983.
[31] S-Y. Kung, "On supercomputing with systolic/wavefront array processors," *Proceedings of the IEEE*, vol. 72, pp. 867–884, July 1984.
[32] J. Kim and L. D. Davisson, "Adaptive linear estimation for stationary M-dependent processes," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 23–31, Jan. 1975.
[33] P. Pirsch, "Design of a DPCM codec for VLSI realization in CMOS technology," *Proceedings of IEEE*, vol. 73, no. 4, pp. 592–598, April 1985.
[34] N. R. Shanbhag and K. K. Parhi, "Roundoff error analysis of the pipelined ADPCM coder," in *Proc. IEEE Intl. Symp. on Circuits and Systems*, Chicago, IL, May 1993, pp. 886–889.

[35] N. R. Shanbhag and K. K. Parhi, "VLSI implementation of a 100 MHz pipelined ADPCM codec chip," in *Proc. IEEE VLSI Signal Processing Workshop*, Veldhoven, The Netherlands, Oct. 1993, pp. 114–122.
[36] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative dataflow programs via optimal unfolding," *IEEE Trans. on Comput.*, vol. 40, no. 2, pp. 178–195, Feb. 1991.
[37] K. K. Parhi, "A systematic approach for the design of digit-serial signal processing architectures," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 4, pp. 358–375, April 1991.
[38] J.-G. Chung and K. K. Parhi, "Design of pipelined lattice IIR digital filters," *Proc. 25th Asilomar Conf. on Sig., Sys., and Comput.*, Calif., Nov. 1991, pp. 1021–1025.
[39] N. R. Shanbhag and K. K. Parhi, "Pipelined adaptive DFE architectures," in *Proc. SPIE Intl. Symp. on Optics, Imaging and Inst., Adv. Algorithms, Architecture, Implementations—IV*, vol. 2027, San Diego, CA, July 1993.



Naresh R. Shanbhag (S'86—M'93) was born on July 2, 1966, in India. He received the B. Tech degree from the Indian Institute of Technology, New Delhi, India, in 1988, M.S. degree from Wright State University, Dayton, OH in 1990, and Ph.D. degree from University of Minnesota, in 1993, all in electrical engineering. In 1993, he joined AT& T Bell Laboratories at Murray Hill, NJ. His research interests are in the general area of VLSI architectures and algorithms for signal processing and communications, which include the design of high-speed algorithms for speech and image compression, adaptive filtering and channel equalization. In addition, he is also interested in efficient VLSI implementation methodologies for digital signal processing and communications algorithms.

Keshab K. Parhi (S'85–M'88–SM'91) 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 has been with the University of Minnesota since 1988 where he is currently an Associate Professor of Electrical Engineering. He has held short term positions at the NEC C& C Research Laboratories, Kawasaki (Japan), the AT& T Bell Laboratories, Holmdel, NJ, and the IBM T. J. Watson Research Center, Yorktown Heights, NY. His research interests include concurrent algorithm and architecture designs for communications, signal and image processing systems, digital integrated circuits, VLSI digital filters, computer arithmetic, high-level DSP synthesis, and multiprocessor prototyping and task scheduling for programmable software systems. He has published over 100 papers in these areas.

Dr. Parhi received the 1993 Guillemin-Cauer award from the IEEE Circuits and Systems society, the 1992 Young Investigator Award of the National Science Foundation, the 1992–1994 McKnight-Land Grant professorship of the University of Minnesota, the 1991 paper award from the IEEE Signal Processing Society, the 1991 Browder Thompson prize paper award of the IEEE, the 1989 research initiation award of the National Science Foundation, and the 1987 Eliahu Jury award for excellence in systems research and the 1987 Demetri Angelakos award for altruistic activities afford fellow graduate students of the University of California (U.C.), Berkeley, where he was a U.C. Regents Fellow and an IBM Graduate Fellow. He is a former associate editor for Image Processing and VLSI Applications of *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS*, is a current associate editor for *IEEE TRANSACTIONS ON SIGNAL PROCESSING, and an editor of JOURNAL OF VLSI SIGNAL PROCESSING, and is a member of the Eta Kappu Nu, and the Association for Computing Machinery.*