Pipelined Adaptive DFE Architectures

Naresh R. Shanbhag and Keshab K. Parhi

Department of Electrical Engineering
University of Minnesota
200 Union Street S.E., Minneapolis, MN-55455

ABSTRACT

Fine-grain pipelined adaptive decision-feedback equalizer (ADFE) architectures are developed using the *relaxed look-ahead* technique. This technique, which is an approximation to the conventional *look-ahead computation*, maintains functionality of the algorithm rather than the input-output behavior. Thus, it results in substantial hardware savings as compared to either parallel processing or look-ahead techniques. The *delay relaxation*, *delay transfer relaxation* and *sum relaxation* are introduced for purposes of pipelining. Both the conventional and the predictor form of ADFE have been pipelined. The performance of the pipelined algorithms for the equalization of a magnetic recording channel is studied. It is demonstrated via simulations that, for a byte error rate of 10^{-7} or less, speed-ups of up to 8 can be easily achieved with the conventional ADFE. The predictor form of ADFE allows much higher speed-ups (up to 32) for less than 1 dB of SNR degradation.

1. INTRODUCTION

In the area of digital communications, there is a growing need for high-speed equalizers for applications such as high-density magnetic storage systems, subscriber loop applications and mobile radio. The adaptive decision-feedback equalizer (ADFE) has been employed successfully for combatting inter-symbol interference (ISI). However, the ADFE has remained difficult to pipeline and in this paper we propose a novel approach for fine-grain pipelining of the ADFE.

Conventionally, algorithm transformation techniques[1] such as *look-ahead* [2] have been employed to introduce concurrency in serial algorithms. The look-ahead technique, however, results in a hardware overhead as it transforms a serial algorithm into an equivalent (in the sense of input-output behavior) pipelined algorithm. In order to reduce this overhead, we have developed the *relaxed look-ahead* technique[3] for the pipelining of adaptive digital filters. The relaxed look-ahead sacrifices the equivalence between the serial and pipelined algorithms at the expense of marginally altered convergence characteristics.

Fine-grain pipelining of the ADFE is known to be a difficult problem due to the fact that the ADFE has a non-linear element (a quantizer) in the decision-feedback loop (**DFL**). The conventional ADFE (to be referred to as ADFE) in Fig.1(a) and the predictor form of ADFE (to be referred to as predictor ADFE) in Fig.1(b), consist of the feedforward filter (**FFF**), the feedback filter (**FBF**), the quantizer (**Q**), and the coefficient update blocks **WUC** (for **FFF**) and **WUD** (for **FBF**). The delays Δ are employed to adjust the position of the main tap of **FFF**, which is usually the center-most tap. In addition to the **DFL**, the presence of the adaptation loop makes it even more difficult to achieve pipelining. Hence, past work [4-6] in high-speed ADFE architectures have almost exclusively adopted parallelization. The use of a transpose structure[7] for the **FFF** and the **FBF** and introduction of delays in the **DFL**[8] in order to obtain a high-speed VLSI implementation of an ADFE can achieve higher speed, but only to a limited extent. In this paper, we present the *delay relaxation*, the *delay transfer relaxation* and the *sum relaxation* as three possible approximations, which can be used for pipelining of the ADFE.

This paper is organized as follows. In section 2, we present the relaxed look-ahead technique, which is then applied to pipeline the ADFE in section 3. In section 4, we analyze the performance of the pipelined algorithms and compare them with that of the serial algorithm. Simulation results are presented for the equalization of a magnetic recording channel in section 5.

0-8194-1276-7/93/$6.00

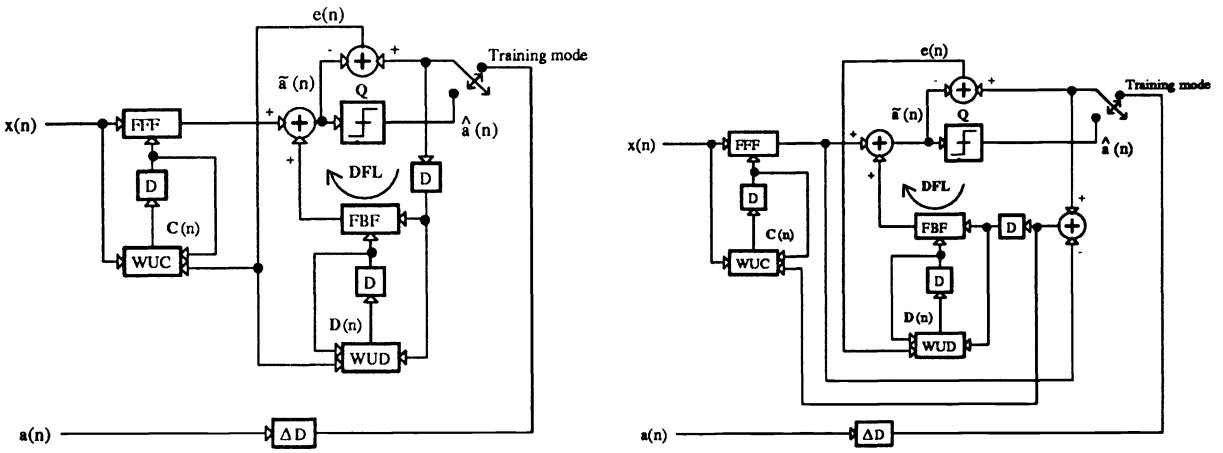


Fig.1 The serial ADFE architecture :(a) conventional and (b) predictor form.

2. THE RELAXED LOOK-AHEAD

In order to introduce relaxed look-ahead, consider the following equations, which describe a linear adaptive estimator with a first-order weight-update recursion

$$\mathbf{W}(n) = \mathbf{W}(n-1) + \mu e(n)\mathbf{X}(n) \qquad (2.1(a))$$

$$e(n) = s(n) - \mathbf{W}^T(n-1)\mathbf{X}(n) \qquad (2.1(b))$$

where $\mathbf{W}(n)$ is a $N \times 1$ vector of coefficients of the filter **FIR** (see Fig.2(a)), μ is the adaptation step-size, $e(n)$ is the estimation error, $\mathbf{X}(n)$ is the $N \times 1$ input vector, and $s(n)$ is the desired signal. The first-order recursion (2.1(a)) also describes the weight-update recursion of the ADFE.

From Fig.2(a) (and (2.1)), we can see that there are two major feedback paths, which present a bottleneck for high throughput applications. The first is called the error feedback path, which consists of the filter **FIR**, the adder, and the weight-update block **WUC**. The second path is the weight-update recursion defined by (2.1(a)). An M-stage pipelined algorithm can be derived from (2.1) by the application of an M-step look-ahead to (2.1). It can be easily checked that the hardware required to do so is quite large because this process involves computing $\mathbf{W}(n)$ from $\mathbf{W}(n-M)$.

However, by considering the error feedback path and weight-update recursion separately, we can pipeline the adaptive estimator in a hardware efficient manner. In particular, we pipeline the error feedback path by the delay relaxation and the delay transfer relaxation, while the weight-update recursion is pipelined by the sum relaxation.

2.1. Delay relaxation

The delay relaxation is shown in Fig.2(b), where the error $e(n)$ and the input $\mathbf{X}(n)$ are delayed by D_1 samples before being employed in the **WUC**. We refer to this transformation as a D_1-step delay relaxation. This transformation is made on the basis of the assumption that the gradient estimate $e(n)\mathbf{X}(n)$ does not change substantially over D_1 clock-cycles. The delay relaxation has been employed[9] to develop the 'delayed LMS' algorithm.

A thorough convergence analysis of this kind of delayed adaptation scheme was also done[9], where it was concluded that the degradation in the convergence speed and especially the adaptation accuracy for delays of upto 32 was very small. However, from an architectural point of view the delay relaxation is an effective method for pipelining. This is because the D_1 delays can now be employed to pipeline the **FIR** and hardware overhead is just the pipelining latches.

2.2. Delay transfer relaxation

A D_1-step delay transfer relaxation (for an adaptive estimator) is shown in Fig.2(c), where the input to the **FIR** is delayed by D_1 samples and then these D_1 delays are transferred from the input of **FIR** to its output. Clearly, this transfer cannot be done via retiming[10] and therefore the two systems in Fig.2(c) are not equivalent. However, after convergence this delay transfer can be justified as the weights do not change much. This implies that some degradation in performance is to be expected while the filter is converging. Note that the D_1 delays would be redistributed to pipeline the **FIR**.

2.3. Sum relaxation

Even though the delay relaxation is sufficient to pipeline the **FIR** and part of the **WUC**, the weight-update loop (2.1(a)) remains to be pipelined. The computation time of (2.1(a)) is lower bounded by a single multiply-add time. In order to reduce this lower bound, we apply a D_2-step look-ahead to (2.1(a)) to give

$$\mathbf{W}(n) = \mathbf{W}(n - D_2) + \mu \sum_{i=0}^{D_2-1} e(n-i)\mathbf{X}(n-i). \tag{2.2}$$

In (2.2), the summation term represents the overhead term. However, instead of taking the sum of D_2 terms in (2.2), we may retain only LA terms to get

$$\mathbf{W}(n) = \mathbf{W}(n - D_2) + \mu \sum_{i=0}^{LA-1} e(n-i)\mathbf{X}(n-i), \tag{2.3}$$

where the partial look-ahead factor LA maybe either less than or equal to D_2. The replacement of D_2 sum terms in (2.2) to LA sum terms in (2.3) is referred to as a D_2-step *sum relaxation*. This relaxation has an overhead of $LA - 1$ adders.

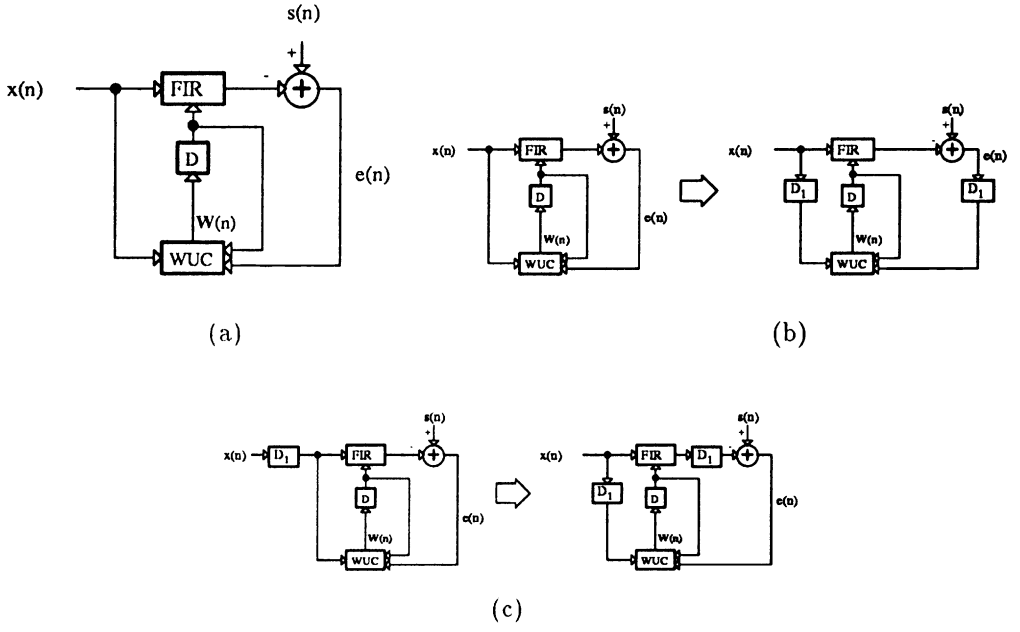


Fig.2 The delay relaxations : (a) original system, (b) delay relaxation and (c) the delay transfer relaxation.

3. PIPELINED ADFE ARCHITECTURES

In this section, we employ the relaxed look-ahead technique, which was described in the previous section, to develop pipelined ADFE architectures. The channel model we consider is shown in Fig.3, where $a(n)$ is the channel

input at time instance n, $\mathbf{h}(n)$ is the channel coefficient vector, $\eta(n)$ is white Gaussian noise and $x(n)$ is the received sample.

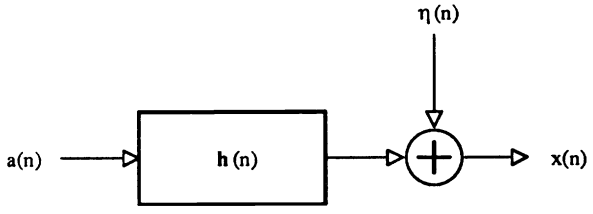


Fig.3 The channel model.

First, we introduce some terminologies to define the equations which describe the serial ADFE of Fig.1(a).

$$\tilde{a}(n) = a_F(n) + a_B(n) \tag{3.1(a)}$$
$$a_F(n) = \mathbf{C}^T(n-1)\mathbf{X}(n) \tag{3.1(b)}$$
$$a_B(n) = \mathbf{D}^T(n-1)\hat{\mathbf{a}}(n-1) \tag{3.1(c)}$$
$$e(n) = \hat{a}(n) - \tilde{a}(n) \tag{3.1(d)}$$
$$\hat{a}(n) = \mathbf{Q}[\tilde{a}(n)] \tag{3.1(e)}$$
$$\mathbf{W}(n) = \mathbf{W}(n-1) + \mu e(n)\mathbf{U}(n), \tag{3.1(f)}$$

where $a_F(n)$ is the output of **FFF**, $a_B(n)$ is the output of the **FBF**, $\mathbf{C}(n)$ is the vector of **FFF** coefficients, $\mathbf{D}(n)$ is the vector of **FBF** coefficients, $\mathbf{X}(n)$ is the vector of received samples, $\hat{\mathbf{a}}(n)$ is the vector of detected symbols, $\tilde{a}(n)$ is the input to quantizer $\mathbf{Q}$ and $\hat{a}(n)$ is the quantizer decision. Note that (3.1(f)) is of the same form as (2.1(a)) and represents the familiar least mean-squared (LMS) algorithm with μ being the adaptation step-size. The vectors $\mathbf{W}^T(n) = [\mathbf{C}^T(n) \quad \mathbf{D}^T(n)]$ and $\mathbf{U}^T(n) = [\mathbf{X}^T(n) \quad \hat{\mathbf{a}}^T(n-1)]$ are the combined coefficient and data vectors, respectively. Note that when correct decisions are made by the quantizer then $\hat{a}(n) = a(n-\Delta)$.

3.1. The PIPADFE1 algorithm

The **FFF** and **FBF** can be pipelined by delaying their inputs by D_1 samples and then applying the delay transfer relaxation (see Fig.2(c)). The weight-update loop can be pipelined by applying a D_2-step sum relaxation. We shall see later that introducing D_1 delays in the **DFL** results in a substantial performance degradation because the **FBF**cannot employ past decisions to cancel the D_1 most-significant ISI terms. These steps result in the PIPADFE1 algorithm (see Fig.4) which is a generalization of a past work[8] . The equations describing PIPADFE1 are

$$\tilde{a}(n) = a_F(n-D_1) + a_B(n-D_1) \tag{3.2(a)}$$
$$a_F(n) = \mathbf{C}^T(n-D_2)\mathbf{X}(n) \tag{3.2(b)}$$
$$a_B(n) = \mathbf{D}^T(n-D_2)\hat{\mathbf{a}}(n-1) \tag{3.2(c)}$$
$$e(n) = \tilde{a}(n) - \hat{a}(n) \tag{3.2(d)}$$
$$\hat{a}(n) = \mathbf{Q}[\tilde{a}(n)] \tag{3.2(e)}$$
$$\mathbf{W}(n) = \mathbf{W}(n-D_2) + \mu \sum_{i=0}^{LA-1} e(n-i)\mathbf{U}_1(n-i), \tag{3.2(f)}$$

where $\hat{a}(n) = a(n - D_1 - \Delta \quad)$ for correct quantizer decisions and $\mathbf{U}_1^T(n) = [\mathbf{X}^T(n-D_1), \hat{\mathbf{a}}^T(n-D_1-1)]$.

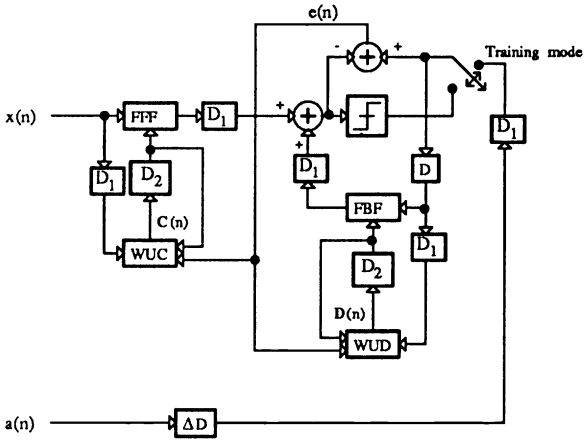


Fig.4 The PIPADFE1 architecture.

In order to demonstrate the increase in throughput due to pipelining, we consider a serial ADFE with the **FFF** and **FBF** having two taps each. Assuming a multiply time of 40 units and an add time of 20 units, it can be easily checked that the critical path in the serial architecture has a delay of 200 units. With $D_1 = 5$ and $D_2 = LA = 1$, we can pipeline the multiply-adds till the critical path delay is reduced to 40 units. The retimed PIPADFE1 architecture, which operates five times faster than the serial ADFE, is shown in Fig.5.

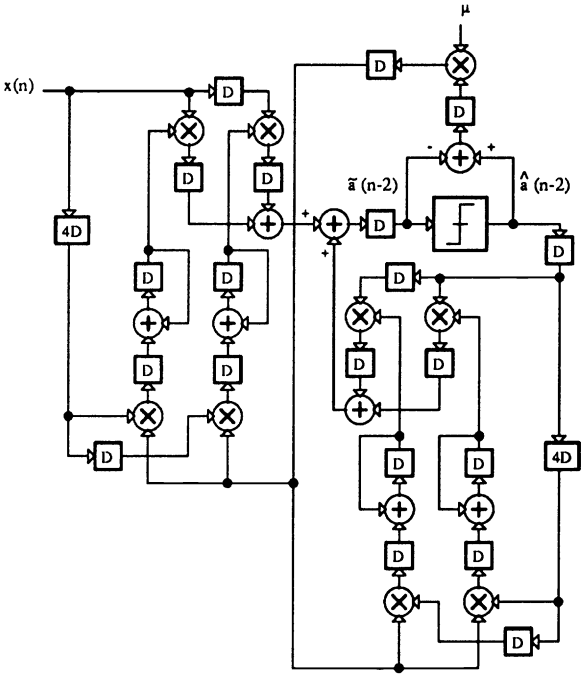


Fig.5 PIPADFE1 with a speed-up of 5.

3.2. The PIPADFE2 algorithm

Instead of introducing D_1 delays in the **DFL** directly as in the case of PIPADFE1, the PIPADFE2 is derived

by pipelining the **DFL** via relaxed look-ahead. The computations of the **DFL** can be written as

$$\tilde{a}(n) = \mathbf{C}^T(n-1)\mathbf{X}(n) + \sum_{i=0}^{N_B-1} d_i(n-1)\hat{a}(n-i-1), \tag{3.3}$$

where $d_i(n)$'s are the **FBF** coefficients and N_B is its order. In order to apply look-ahead, we linearize the **DFL** and then apply the look-ahead technique. Note that the linearization of the **DFL** is simply an intermediate step in the process of developing PIPADFE2. Application of a D_1-step look-ahead to the linearized **DFL** computation results in an equation of the following form

$$\tilde{a}(n) = \mathbf{C}^T A + B \tag{3.4}$$

where A represents the computations of the data preprocessing section (**PP**) defined as

$$\begin{aligned} A = \mathbf{X}(n) + \sum_{i=0}^{D_1-1} d_i\mathbf{X}(n-i-1) + \sum_{i=0}^{D_1-1}\sum_{i_1=0}^{D_1-1} d_i d_{i_1}\mathbf{X}(n-i-i_1-2) \\ + \cdots + \sum_{i=0}^{D_1-1}\sum_{i_1=0}^{D_1-1}\cdots\sum_{i_P=0}^{D_1-1} d_i d_{i_1}\cdots d_{i_P}\mathbf{X}(n-i-i_1-\cdots-i_P-D_1-1), \end{aligned} \tag{3.5}$$

and B represents the computations of the feedback section as shown below

$$\begin{aligned} B = \sum_{i=0}^{N_B-D_1-1} d_{D_1+i}\tilde{a}(n-D_1-i-1) + \sum_{i=0}^{D_1-1}\sum_{i_1=0}^{N_B-D_1-1} d_i d_{i_1+D_1}\tilde{a}(n-D_1-i-i_1-2) \\ + \cdots + \sum_{i=0}^{D_1-1}\sum_{i_1=0}^{D_1-1}\cdots\sum_{i_P=0}^{D_1-1} d_i d_{i_1}\cdots d_{i_P}\tilde{a}(n-i-i_1-\cdots-i_P-D_1-1). \end{aligned} \tag{3.6}$$

We approximate A ((3.5)) as follows

$$A = \mathbf{X}(n) + \sum_{i=0}^{D_1-1} \alpha d_i\mathbf{X}(n-i-1), \tag{3.7}$$

where α is a scalar constant. In making these approximations, we have employed the fact that the **PP** derives all its coefficients (except the one, which is unity) from those of the **FBF** of the serial ADFE. In addition, the coefficients of **FBF**, which appear in **PP**, are d_i, $0 \le i < D_1$. Next, we approximate B as

$$B = \sum_{i=0}^{D_1-1} \beta d_i\tilde{a}(n-P-i-1) + \sum_{i=D_1}^{N_B-1} d_i\tilde{a}(n-P-i-1), \tag{3.8}$$

where β is a scalar constant. This approximation is guided by the observation that (3.6) contains $\tilde{a}(n-D_1-i)$, $i > 1$, which implies that D_1 delays have been introduced at the input to the **FBF**. In addition, all the **FBF** coefficients of the serial ADFE appear in B and the first term in (3.6) indicates that the coefficients d_i $i \ge D_1$ appear unmodulated.

Even with these coarse approximations, it will be shown via simulations that PIPADFE2 outperforms PIPADFE1 substantially. Clearly, the performance of PIPADFE2 can be enhanced further by having better approximations. Next, we apply a D_1-step delay transfer relaxation first to the **PP** and then to **FFF**, which results in the presence of D_1 delays at the output of **FFF**. In a similar fashion, we apply the delay transfer relaxation to **FBF** to transfer D_1 to its output. Finally, employing the sum relaxation as in the case of PIPADFE1, we obtain the following equations

which describe PIPADFE2

$$\tilde{a}(n) = a_F(n - D_1) + a_B(n - D_1) \tag{3.9(a)}$$

$$a_F(n) = \mathbf{C}^T(n - D_2)[\mathbf{X}^T(n) + \sum_{i=0}^{P-1} \alpha d_i(n - D_2)\mathbf{X}^T(n - i - 1)] \tag{3.9(b)}$$

$$a_B(n) = \sum_{i=0}^{D_1-1} \beta d_i(n - D_2)\hat{a}(n - i - 1) + \sum_{i=D_1}^{N_B-1} d_i(n - D_2)\hat{a}(n - i - 1) \tag{3.9(c)}$$

$$e(n) = \tilde{a}(n) - \hat{a}(n) \tag{3.9(d)}$$

$$\hat{a}(n) = \mathbf{Q}[\tilde{a}(n)] \tag{3.9(e)}$$

$$\mathbf{W}(n) = \mathbf{W}(n - D_2) + \mu \sum_{i=0}^{LA-1} e(n - i)\mathbf{U}_2(n - i), \tag{3.9(f)}$$

where

$$\mathbf{U}_2(n) = [\mathbf{X}^T(n - D_1) + \sum_{i=0}^{P-1} \alpha d_i(n - D_2)\mathbf{X}^T(n - D_1 - i - 1), \hat{a}^T(n - D_1 - 1)]^T, \tag{3.10}$$

and $\hat{a}(n) = a(n - D_1 - \Delta)$ for correct quantizer decisions. Optimal values of α and β for a magnetic recording channel (for different values of D_1) were found empirically. As in the case of PIPADFE1, PIPADFE2 also reduces to the serial ADFE if $D_1 = 0$.

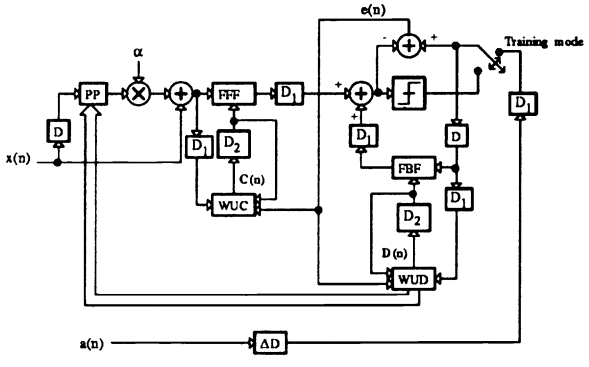


Fig.6 The PIPADFE2 architecture.

The architecture for PIPADFE2 is shown in Fig.6. As compared to PIPADFE1, PIPADFE2 requires an additional D_1 multiplications due to the **PP**, and $LA(N_F + N_B)$ adders (if the weight-update loop is also pipelined). This, however, does not reduce the clock frequency because the D_1 latches can be employed to pipeline **PP** as well.

3.4. The PIPADFE3 Algorithm

In this sub-section, we pipeline the serial predictor ADFE in Fig.1(b), which is described by the following expressions

$$\mathbf{C}(n) = \mathbf{C}(n - 1) + \mu e(n)\mathbf{X}(n) \tag{3.11(a)}$$

$$\mathbf{D}(n) = \mathbf{D}(n - 1) + \mu[\hat{a}(n) - \tilde{a}(n)]\mathbf{E}(n - 1) \tag{3.11(b)}$$

$$e(n) = a(n) - a_F(n) \tag{3.11(c)}$$

$$\tilde{a}(n) = a_F(n) + a_B(n) \tag{3.11(d)}$$

$$a(n) = \mathbf{Q}[\tilde{a}(n)] \tag{3.11(e)}$$

$$a_F(n) = \mathbf{C}^T(n - 1)\mathbf{X}(n) \tag{3.11(f)}$$

$$a_B(n) = \mathbf{D}^T(n - 1)\mathbf{E}(n - 1), \tag{3.11(g)}$$

where $\mathbf{E}^T(n) = [e(n), e(n-1), \ldots, e(n-N_B+1)]$ and $\hat{a}(n) = a(n-\Delta)$ for correct quantizer decisions. From (3.11(a)), (3.11(c)) and (3.11(f)), we see that the **FFF** adapts independently of the **FBF**. This feature of the predictor ADFE allows us to pipeline it to much higher levels than the conventional ADFE.

First, we transform (3.11(b)) into

$$\mathbf{D}(n) = \mathbf{D}(n-1) + \mu[e(n) - a_B(n)]\mathbf{E}(n-1), \tag{3.12}$$

which can be derived by substituting for $\hat{a}(n)$ (from (3.11(c)) and $\tilde{a}(n)$ (from (3.11(d)) in (3.11(b)). Next, we pipeline the predictor ADFE by the following steps

1.) Apply a D_1-step delay relaxation to the **FFF**.

2.) Apply a D_2-step sum relaxation to **FFF** and **FBF**.

3.) Apply a D_3-step delay transfer relaxation to **FFF** and **FBF**.

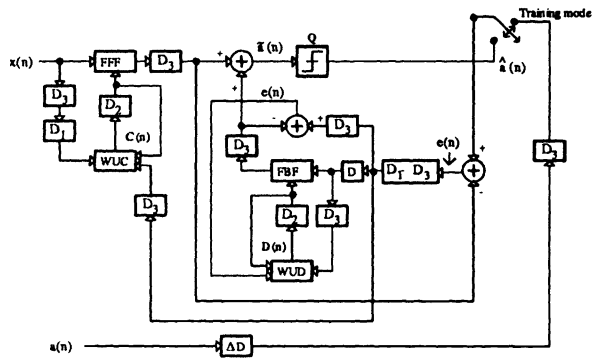


Fig.7 The PIPADFE3 architecture.

Note that step 1 has already introduced D_1 delays in the **DFL**, while step 3 transfers D_3 of these delays to the output of **FBF**. The pipelined predictor ADFE (referred to as PIPADFE3) is shown in Fig.7 and the following equations describe it

$$\mathbf{C}(n) = \mathbf{C}(n-D_2) + \sum_{i=0}^{LA-1} e(n-D_1-i)\mathbf{X}(n-D_1-D_3-i) \tag{3.13(a)}$$

$$\mathbf{D}(n) = \mathbf{D}(n-D_2) + \sum_{i=0}^{LA-1} [e(n-D_1-i) - a_B(n-D_3)]\mathbf{E}(n-D_1-1-i) \tag{3.13(b)}$$

$$e(n) = \hat{a}(n) - a_F(n-D_3) \tag{3.13(c)}$$

$$\tilde{a}(n) = a_F(n-D_3) + a_B(n-D_3) \tag{3.13(d)}$$

$$a(n) = \mathbf{Q}[\tilde{a}(n)] \tag{3.13(e)}$$

$$a_F(n) = \mathbf{C}^T(n-D_2)\mathbf{X}(n) \tag{3.13(f)}$$

$$a_B(n) = \mathbf{D}^T(n-D_2)\mathbf{E}(n-D_1+D_3-1), \tag{3.13(g)}$$

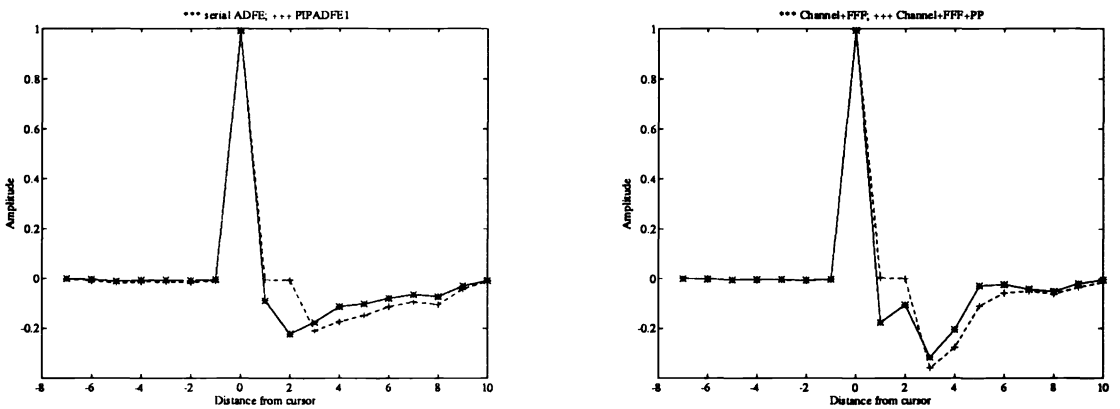
with $\hat{a}(n) = a(n-D_3-\Delta)$ for correct quantizer decisions.

In Fig.1(b), we can easily confirm that the critical path consists of the **FBF**, adder, quantizer, adder and **WUC**. From Fig.7, we find that this critical path contains $D_1 + D_3$ latches. Assuming uniformly pipelined stages, the speed-up achieved by PIPADFE3 over the serial architecture in Fig.1(b) is $D_1 + D_3$. The hardware overhead for PIPADFE3 are the pipelining latches and $(LA - 1)[N_F + N_B]$ adders.

4. PERFORMANCE ANALYSIS

In this section, we will analyze and compare, in qualitative terms, the performance of the serial ADFE, PIPADFE1, PIPADFE2, and PIPADFE3. To do this we have simulated the performance of the equalizers for a magnetic recording channel with a channel SNR of 20 dB. The channel coefficients $([0.2, 0.6, 1.0, -1.0, -0.6, -0.2])$ were obtained from a Lorentzian pulse model[11] with the symbol period being one-half of the width of channel step response pulse at a height of 50% of the maximum.

The **FFF** in all the equalizers attempts to cancel the pre-cursor ISI. To see this, we plot (see Fig.8(a)) the pulse response of the combined channel and **FFF** for a serial ADFE. The non-zero postcursor ISI is cancelled by the **FBF** coefficients. In the case of PIPADFE1 (and PIPADFE2), the **FBF** output is delayed by D_1 samples. Therefore, the **FBF** cannot cancel the first D_1 postcursor ISI terms and the burden of cancelling them falls on the **FFF**. This is also indicated in Fig.8(a), where the combined channel and **FFF** pulse response for PIPADFE1 with $D_1 = 2$ is shown. Clearly, as D_1 increases the performance of PIPADFE1 degrades and approaches that of a linear equalizer.



(a) (b)

Fig.8 Combined pulse response of (a) the channel+**FFF** for the serial ADFE (solid) and PIPADFE1 (dash-dot) with $D_1 = 2$, and (b) channel+**FFF** (solid) and channel+**FFF** + **PP** (dash-dot) for PIPADFE2 with $D_1 = 2$.

From the discussion above it is clear that the performance of a pipelined ADFE algorithm can be improved substantially (especially for large D_1) if the **FFF** does not have to cancel the postcursor ISI. This is exactly what PIPADFE2 achieves. In Fig.8(b), we show the combined channel and **FFF** for PIPADFE2 with $D_1 = 2$. Just as in the case of the serial ADFE, the **FFF** in PIPADFE2 only cancels the precursor ISI. However, in this case the first D_1 postcursor ISI are cancelled by the **PP**, whose coefficients are derived from those of **FBF** (see (3.7) and (3.8)). This can be confirmed by plotting the pulse response of the system comprising the channel, **PP** and **FFF**. Finally, the remaining postcursor ISI are cancelled by the **FBF**.

Even though it can be shown[12] that the predictor ADFE (see Fig.1(b)) is equivalent to the conventional ADFE (see Fig.1(a)), in actual practice the predictor ADFE will perform worse than the conventional form. This is because the predictor form the **FFF** and the **FBF** minimize two different error signals. Our interest, in this paper, is in comparing the performance of the serial predictor ADFE and PIPADFE3.

In section 5, we will further confirm the conclusions of this sub-section by comparing the performance of PIPADFE1, PIPADFE2 and PIPADFE3 as D_1 increases.

5. SIMULATION RESULTS

All the simulations, in this section, have been performed with a Lorentzian model for a magnetic recording channel, whose coefficients are defined in section 4. The order of the **FFF** was 13 for the conventional ADFE and 20 for the predictor ADFE. The order of the **FBF**was 10 for all cases.

After initialization, the first 700 samples were employed for training purposes. The final results were obtained by averaging over 30 independent trials. We have chosen the output SNR, which is the ratio of the signal power at the channel input to the noise power across the quantizer, as a measure of performance. It has been observed[11] that for storage channels an output SNR of 16 dB results in byte error rate of 10^{-7} or less. Hence, we take this value of output SNR as the lower limit of acceptable performance.

5.1. Performance vs. speed-up

The purpose of this experiment is to determine how the performance of PIPADFE1 and PIPADFE2 degrade as the speed-up is increased. These simulations were done for different channel SNR's. The channel SNR is defined as the ratio of the channel input power to the additive noise power. As the nominal measured channel SNR is 21 dB[11], therefore we consider channel SNR's varying from 18 dB to 30 dB.

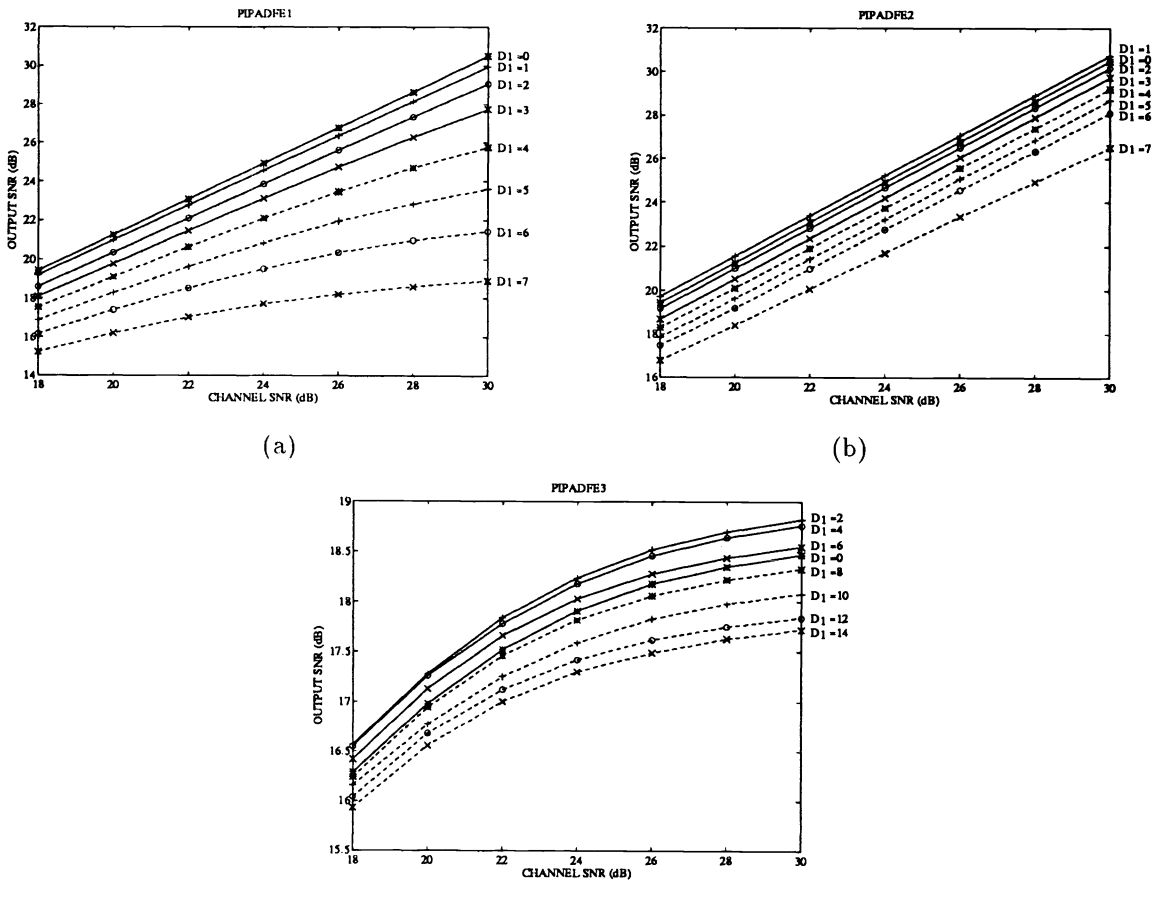


Fig.9 Output SNR vs. channel SNR for different speed-ups : (a) PIPADFE1, (b) PIPADFE2 and (c) PIPADFE3.

In Fig.9, we plot the output SNR as a function of the channel SNR for different values of D_1. In case of

PIPADFE1 (see Fig.9(a)), for a given channel SNR, the output SNR drops as D_1 (which corresponds to the speed-up) increases. In addition, for the same speed-up, this SNR drop increases for high channel SNR's. However, Fig.9(a) also indicates that for a channel SNR of 20 dB (which is slightly conservative as compared to the measured SNR of 21 dB) a speed-up of $D_1 = 7$ results in an output SNR of 16.2 dB, which is an acceptable level of performance for storage channels. From Fig.9(a), we can see that if SNR degradation of 1 dB or less are desired, then PIPADFE1 should be employed in those situations where the desired speed-up is small (say $D_1 = 1$ or 2) and the channel quality is low.

For PIPADFE2 (see Fig.9(b)), not only is the output SNR drop (for a given channel SNR) smaller but this drop remains more or less constant as the channel SNR increases. This is a substantial improvement over PIPADFE1 and it clearly indicates the importance of employing good approximations while using relaxed look-ahead. The SNR advantage of PIPADFE2 over PIPADFE1 ranges from 0.5 dB (with $D_1 = 1$ and channel SNR of 18 dB) to 7.6 dB (with $D_1 = 7$ and channel SNR of 30 dB). Clearly, the slightly higher computational cost of PIPADFE2 is more than compensated for by its superior performance. In addition, unlike PIPADFE1, the output SNR of PIPADFE2, with $D_1 = 7$, is approximately 2.4 dB above the lower limit of acceptable performance (16 dB).

In Fig.9(c), we plot the performance of PIPADFE3 with $N_F = 20$, $N_B = 10$, and $D_1 = D_3$. Recall that the actual speed-up is $D_1 + D_3$. Hence, from Fig.9(c), we find that PIPADFE3 can achieve speed-ups of the order of 28 with less than 1dB of performance loss as compared to the serial predictor ADFE. Note also that for speed-ups of up to 12 there is even a performance gain. As PIPADFE3 can be pipelined at very high-speeds, therefore, we can afford to have higher number of taps for the **FBF** in order to improve its SNR performance and yet achieve significant speed-ups.

5.3. Performance with sum relaxation

The sum relaxation, defined in (2.3), was employed to pipeline the weight-update equations for PIPADFE1 (3.2(f)) and PIPADFE2 (3.9(f)). In this experiment, we show the effect of sum relaxation in improving the output SNR as D_2 (the pipelining level of the weight-update loop) increases.

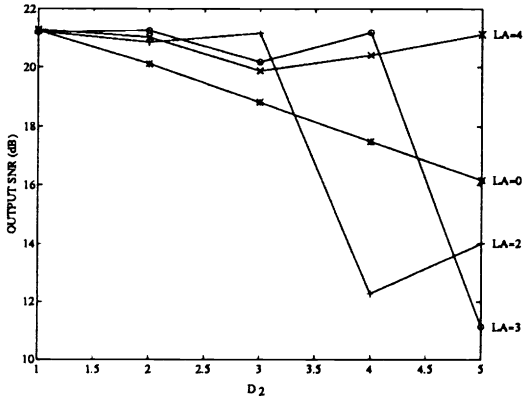


Fig.10 Output SNR vs. D_2 for different values of LA.

The simulation results for a channel SNR of 20 dB and $D_1 = 0$ are shown in Fig.10, where we have plotted the output SNR as a function of D_2 for different values of the look-ahead factor LA. With $LA = 0$, the output SNR keeps decreasing monotonically as D_2 increases. However, note that for each value of D_2 under consideration, there exists a value of LA at which the SNR drop is less than 0.1 dB. For example, in a practical implementation, we would use $LA = 2$ with $D_2 = 3$ and $LA = 4$ with $D_2 = 5$. This clearly implies that very fine pipelining of the weight-update loop is possible. Similar results were obtained for PIPADFE3.

6. CONCLUSIONS

The pipelined algorithms presented in this paper can be improved further by incorporating sophisticated relaxations, especially in case of PIPADFE2. In addition, combining coding with equalization, could also be exploited for the development of better pipelined equalization algorithms. Convergence analysis of the pipelined algorithms is currently in progress.

7. ACKNOWLEDGMENT

This research was supported by the army research office by contract number DAAL-90-G-0063.

8. 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. 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.

4. 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.

5. J. M. Cioffi, P. Fortier, S. Kasturia, and G. Dudevoir, "Pipelining the decision feedback equalizer", *IEEE DSP Workshop*, 1988.

6. K. J. Raghunath and K. K. Parhi, "Parallel adaptive decision feedback equalizers", *IEEE Trans. on Signal Processing*, vol.41, pp. 1956-1961, May 1993.

7. F. Lu and H. Samueli, "A 60-MBd, 480-Mb/s, 256-QAM decision-feedback equalizer in 1.2μ CMOS", *IEEE Journal of Solid-State Circuits*, vol. 28, pp. 330-338, March 1993.

8. M. Schobinger, et. al., "CMOS digital adaptive decision feedback equalizer chip for multilevel QAM digital radio modems", *Proc. IEEE Intl' Symp. Circ. and Syst.*, pp. 574-557, May 1990.

9. 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.

10. C. Leiserson and J. Saxe, "Optimizing synchronous systems", *J. of VLSI and Computer Systems*, vol. 1, pp. 41-67, 1983.

11. J. M. Cioffi, W. L. Abbot, H. K. Thapar, C. M. Melas, and K. D. Fisher, "Adaptive equalization in magnetic-disk storage channels", *IEEE Communications Magazine*, pp. 14-29, February 1990.

12. C. A. Belfiore and J. H. Park, Jr., "Decision feedback equalization", *Proc. IEEE*, vol. 67, pp. 1143-1156, Aug. 1979.