HIGH SAMPLE RATE ARCHITECTURES FOR BLOCK ADAPTIVE FILTERS

Srikanth Karkada, Chaitali Chakrabarti and Andreas Spanias
Department of Electrical Engineering and Telecommunications Research Center
Arizona State University, Tempe, AZ 85287-5706, USA
chaitali@asu.edu

ABSTRACT

In this paper we propose a variety of architectures for implementing block adaptive filters in the time-domain. These filters are based on a block implementation of the least mean squares (BLMS) algorithm. First, we present an architecture which directly maps the BLMS algorithm into an array of processors. Next, we describe an architecture where the weight vector is updated without explicitly computing the filter error. Third, we describe an architecture which exploits the redundant computations of overlapping windows. All the architectures have a a significantly smaller sample period compared to frequency domain implementations. Moreover, the sample periods can be reduced even further by applying relaxed look-ahead techniques.

1. INTRODUCTION

Adaptive digital filters have become increasingly popular, largely due to the simplicity of the Least Mean Squares algorithm(LMS). The LMS algorithm updates the weights of a transversal filter using the steepest descent technique [12]. The BLMS algorithm [2] involves calculation of a block of data at a time in order to update the weights. These filters can be realized both in the time-domain and in the frequency-domain. In this paper, we focus on the development of high sample rate architectures for the time-domain implementation of the block adaptive filter.

The sample period of the time-domain implementation is bounded by the delay in the coefficient update loop. In order to reduce the sample period, parallel processing [5],[9], and pipelining techniques [6], [10] have been used. Since direct application of lookahead techniques results in large hardware overhead, the Delayed LMS (DLMS) [8],[7] and more recently relaxed lookahead [11] schemes have been proposed. In both these methods delays are added in the feedback paths to support pipelining. Since the input-output mapping is not exactly maintained, the convergence of these schemes is mildly degraded.

In this paper, we propose three highly modular, high sample rate architectures for implementing the BLMS algorithm. The high sample rates are supported by a relaxed look-ahead implementation of the BLMS algorithm. All these architectures have an area complexity of $O(NM)$, where N is the block size and M is the FIR filter size. Architecture_1 maps the BLMS algorithm directly into an array of $M \times N$ processors. Architecture_2 implements the BLMS algorithm where the weight vector is updated without explicitly computing the filter error. Both these architectures are independent of the value of shift s. Architecture_3 exploits the redundant computations of overlapping windows for the special case when $s = 1$. All the architectures have very small non-pipelined sample periods compared to frequency-domain implementations.

This work was supported in part by a grant from NSF MIP-9309504.

2. PRELIMINARIES

The BLMS algorithm uses a block of errors to estimate the gradient [2]. Let M be the number of FIR filter taps, N be the block length, and j be the block index. Then the output at block time j, $\underline{Y}(j)$, is given by $\underline{Y}(j) = \underline{X}(j)\underline{w}(j)$ where $\underline{w}(j)$ and $\underline{X}(j)$ are matrices of size $M \times 1$ and $N \times M$ respectively. The jth error block $\underline{e}(j)$ is calculated from the jth desired input $\underline{d}(j)$ by $\underline{e}(j) = \underline{d}(j) - \underline{Y}(j)$. The jth block gradient $\nabla(j)$ is found by minimizing the block mean squared error $BMSE = E[\underline{e}(j)^T\underline{e}(j)]$ with respect to the coefficients $w_i(j)$. The resulting weight update equation is given by $\underline{w}(j) = \underline{w}(j-1) + \left(\frac{2\mu}{N}\right)\underline{X}^T(j)\,\underline{e}(j)$.

Relaxed Look-ahead applied to the BLMS:

The relaxed look-ahead technique was introduced in [11] as an approximate technique suitable for high sample rate LMS filters. Here we apply the same technique to the BLMS algorithm. Let the weight vector at block time j be updated by the weight vector at block time $j - D2$. Then
$\underline{e}(j) = \underline{d}(j) - \underline{X}(j)\underline{w}(j-D2)$, and
$\underline{w}(j) = \underline{w}(j-D2) + \left(\frac{2\mu}{ND2}\right)\sum_{m=0}^{D2-1}\underline{X}^T(j-m)\underline{e}(j-m)$
If we assume that the input $x(n)$ is stationary, then instead of summing over $D2$ terms, we can sum over only P terms, where $P \leq D2$. In addition, if we assume that the signal is stationary and slowly time-varying, then the gradient estimate does not change much, and the input can be delayed by $D1$ samples. The weight update equation is then

$$\underline{w}(j) = \underline{w}(j-D2) + \left(\frac{2\mu}{NP}\right)\sum_{m=0}^{P-1}\underline{X}^T(j-m-D1)\underline{e}(j-m-D1) \quad (1)$$

It can be shown theoretically that the relaxed-lookahead BLMS converges [1]. Convergence curves for the BLMS and the relaxed-lookahead BLMS algorithms are given in Fig.1. The curves were generated for a non-time-varying system identification scheme where the underlying system was a 10th order FIR filter with impulse response $h(n) = (-0.95)^n$, $n = 0, 1, \ldots, 10$. The system was excited by white Gaussian noise of zero-mean and unit variance. The blocks were shifted one sample per iteration and a conservative convergence factor of $\mu = 0.005$ was used. While in all the three cases the algorithm converges, the delays in the estimation of the gradient (D1) and in the weight update (D2) result in longer adaptive transients. In summary, relaxed lookahead comes at the expense of some loss in tracking properties. In addition, as P gets larger (more gradient smoothing), the stability bound on μ becomes tighter.

3. ARCHITECTURE_1

Architecture_1 implements the BLMS algorithm in a straightforward manner by an array of $M \times N$ processors. In the jth iteration the following computations take place. First, $\underline{X}(j)\,\underline{w}(j-1)$ and $\underline{e}(j) = \underline{d}(j) - \underline{X}(j)\,\underline{w}(j-1)$ are computed. Then $\underline{X}^T(j)\,\underline{e}(j)$ is computed and $\underline{w}(j-1)$ is updated by $(\frac{2\mu}{N})(\underline{X}^T(j)\,\underline{e}(j))$. For the computation of $\underline{X}(j)\,\underline{w}(j-1)$, the inputs are broadcast diagonally and the weights are broadcast columnwise to the processor array. Each processor computes the product of its input x and weight w. The xw terms along a row are added by a tree of adders, and the sum is subtracted from the desired value d to get the filter error e. Next the error e is broadcast along the row and each processor computes the product of x and e. The xe terms along a column are added by a tree of adders and the sum is multiplied by $\alpha = 2\mu/N$. This is then used to update the weights.

The computation time for one iteration is

$$T_{comp} = 3\,T_{mul} + (1 + \log(MN))T_{add}$$

where T_{mul} and T_{add} are the times required to perform a single multiplication and addition respectively. This is the same as the block pipelining period. For large M and N, broadcasting is impractical. In such a case, Architecture_1 is transformed into a purely systolic architecture with no broadcast. The block pipelining period of the systolic architecture is $(M + N + 1)(T_{mul} + T_{add})$.

Pipelined Implementation: The sampling rate of Architecture_1 can be increased by implementing the relaxed look-ahead realization (eqn 1). Figure 2 describes such an architecture. Here each processor computes the xw terms and the xe terms simultaneously. The 'xw' terms are added horizontally and the 'xe' terms are added vertically. The sum of the 'xe' terms is used to update the delayed coefficient. The block period of this architecture is

$$T_{block} = max\,\{(3T_{mul} + T_{add}\log(MN))/D1\,,\, T_{add}/D2\}\,.$$

Thus this architecture can be sampled at a very high rate by increasing the values of D1 and D2 at the expense of slower convergence.

4. ARCHITECTURE_2

The BLMS algorithm can also be implemented by updating the weight vector in the following way: $\underline{w}(j) = \underline{w}(j-1) - \alpha\underline{X}^T(j)\,\underline{d}(j) + \alpha\underline{X}^T(j)\,\underline{X}(j)\,\underline{w}(j-1)$. Thus the weight vector is updated without explicitly computing the filter error. In the jth iteration, first $\alpha\underline{X}^T(j)\,\underline{X}(j)$ and $\alpha\underline{X}^T(j)\,\underline{d}(j)$ are computed. Then $\alpha\underline{X}^T(j)\,\underline{X}(j)\,\underline{w}(j-1)$ is computed

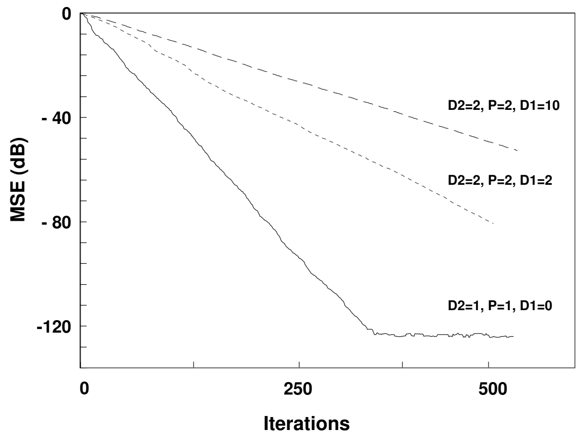

Figure 1: Convergence curves for BLMS and relaxed look-ahead BLMS

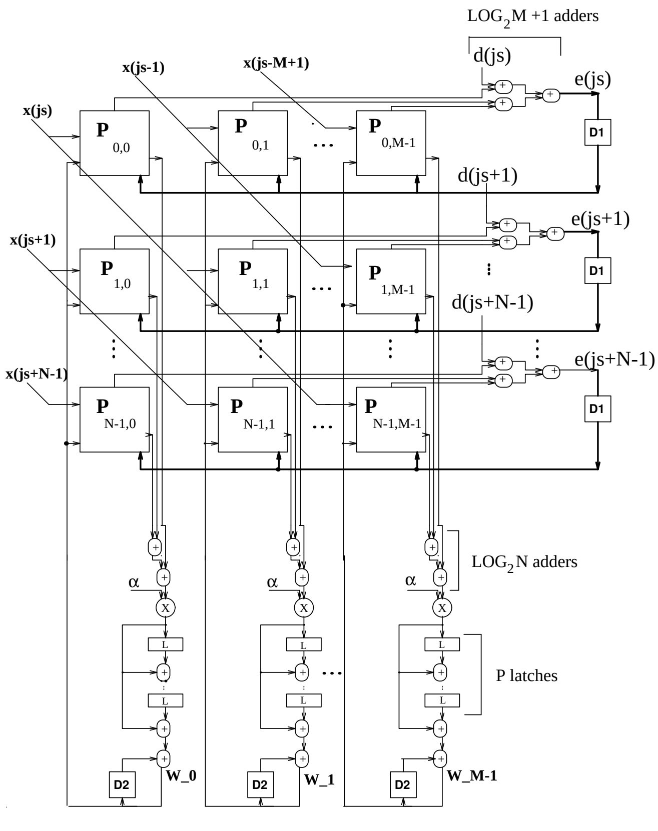

Figure 2: Pipelined implementation of Architecture_1. D1 is the delay in the error feedback loop, and D2 is the delay in the weight update.

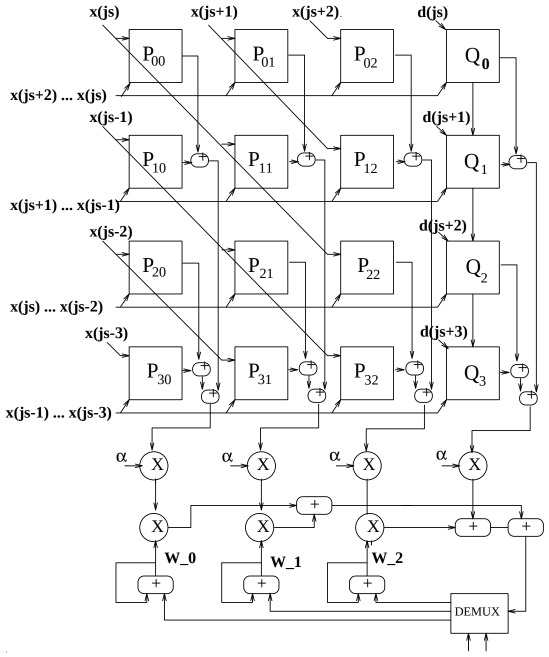

Figure 3: Architecture_2 when M=3 and N=4.

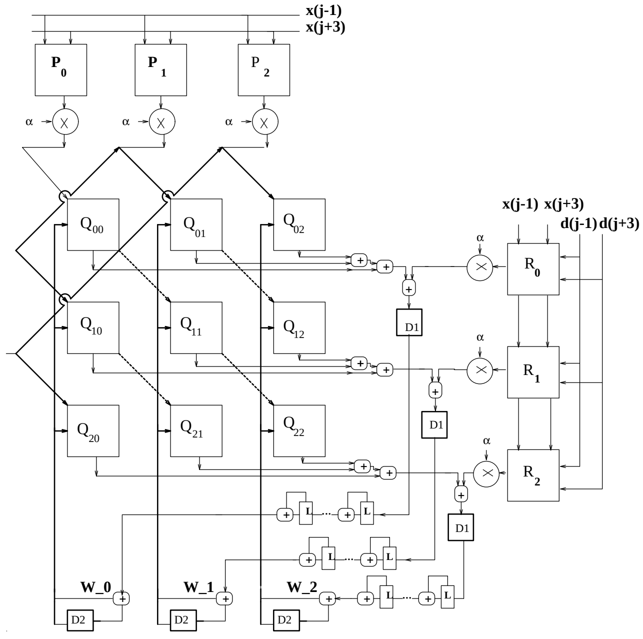

Figure 4: Pipelined implementation of Architecture_3. M=3 and N=4.

and $\underline{w}(j-1)$ is updated. Architecture_2 implements this algorithm.

In this architecture, $\underline{X}^T(j)\underline{X}(j)$ is computed by an array of $(M \times N)$ processors of type P and $\underline{X}^T(j)\underline{d}(j)$ is computed by N processors of type Q. The inputs are broadcast to all the processors along the diagonal as well as along the horizontal directions. Each processor computes the product of its two inputs. The product terms are then added by a tree of adders. The coefficients of the weight vector are updated one after the other. For each coefficient update, the appropriate elements of $\underline{X}^T(j)\underline{X}(j)$ are multiplied with those of $\underline{w}(j-1)$ and the product terms are added by a tree of adders. The computation time for one iteration is given by

$$T_{comp} = (M+2)T_{mul} + (M + \log N + M \log M)T_{add}.$$

This architecture can be pipelined in such a way that while processors of type P operate on the jth set, the weights are updated for the $(j-1)$th set. The block pipelining period of this architecture is then given by the delay in the weight update loop.

$$T_{block} = M(T_{mul} + (1 + \log M)T_{add})$$

For the case when M and N are large, this architecture can be trasnformed into a systolic architecture with no broadcast. The block pipelining period then increases to $M(MT_{mul} + (M+1)T_{add})$.

Pipelined Implementation: The block pipelining period of the semi-systolic implementation can be reduced by applying relaxed look ahead techniques (see eqn.1). If D1 latches are added in the computation path of $\underline{X}^T(j)\underline{X}(j)\underline{w}(j-1)$ and D2 latches are added in the weight update loop, then the block pipelining period of this architecture reduces to

$$T_{block} = max\{M(T_{mul} + \log M T_{add})/D1, T_{add}/D2\}.$$

The total number of computations in this implementation can be significantly reduced by observing that $\underline{X}^T(j)\underline{X}(j)$ is symmetrical about the diagonal, and that the (m,n)th term can be computed from the $(m-1,n-1)$th term, $1 \le m \le M-1$, $m \ge n$. $\underline{X}^T(j)\underline{X}(j)$ can be computed by a linear array of M processors which output one column of the product matrix at a time. The leftmost column of $\underline{X}^T(j)\underline{X}(j)$ is computed from scratch. The remaining columns are computed by using the results of the previous column computations.

The large hardware complexity and the large block pipelining period of Architecture_2 makes it less attractive compared to Architecture_1. However, for the case when the shift s=1, Architecture_2 can be modified to exploit the overlapping computations. We describe such an architecture next.

5. ARCHITECTURE_3

In this section we develop an architecture for the case when $s = 1$. It is known that in this case convergence may improve because data is reused to adapt the weights [3], [4]. For $s = 1$, there is significant overlap between the computations of $\underline{X}^T(j-1)\underline{X}(j-1)$ and $\underline{X}^T(j)\underline{X}(j)$. In fact, given $\underline{X}^T(j-1)\underline{X}(j-1)$, $\underline{X}^T(j)\underline{X}(j)$ can be determined by computing only M new terms. Moreover, these M new terms do not have to be computed from scratch. Instead, each new term can be computed by updating the term that was computed in the previous iteration.

Figure 4 describes the architecture for the case when M=3 and N=4. Architecture_3 consists of (i) M processors of type P which compute the M new terms in $\underline{X}^T(j)\underline{X}(j)$, (ii) $M \times M$ processors of type Q which compute the product of $\alpha\underline{X}^T(j)\underline{X}(j)$ and $\underline{w}(j-1)$, and (iii) M processors of type R which compute $\alpha\underline{X}^T(j)\underline{d}(j)$. The processors of type P compute two multiplications and two additions to update a new term. Specifically, in the jth iteration, P_i

Table 1: Comparison of the three proposed architectures with respect to hardware complexity and block pipelining period

Architecture	*No. of adders*	*No. of multipliers*	*Block pipelining period*
Architecture_1	$N \log M + M(1+\log N)$	$M(N+1)$	$3\,T_{mul} + (1+\log(MN))\,T_{add}$
Pipelined Arch_1			$max\,\{(3T_{mul} + T_{add}\log(MN))/D1, T_{add}/D2\}$
Architecture_2	$M(1+\log N)+\log(MN)$	$M(N+2)+N$	$M(T_{mul} + (1+\log M)T_{add})$
Pipelined Arch_2			$max\{(M(T_{mul} + \log MT_{add}))/D1, T_{add}/D2\}$
Architecture_3	$M(5+\log M)$	$M(5+M)$	$T_{mul} + (2+\lfloor \log M \rfloor)T_{add}$
Pipelined Arch_3			$max\,\{(T_{mul} + (1+\lfloor \log M \rfloor)T_{add})/D1, T_{add}/D2\}$

updates its computation in the previous iteration by $x(j+M)x(j+M-i) - x(j-1)x(j-1-i)$, $0 \le i \le M-1$. The M new terms are multiplied by α and sent to the array of Q processors. The processors of type R also update the results of the previous iteration to compute $\underline{X}^T(j)\,\underline{d}(j)$. In the jth iteration, R_i updates its previous results by $d(j+M)x(j+M-i) - d(j-1)x(j-1-i)$, $0 \le i \le M-1$. The array of processors of type Q compute the product of $\alpha\underline{X}^T(j)\,\underline{X}(j)$ and $\underline{w}(j-1)$ from scratch. Here the weights are broadcast along the columns. Each processor computes the product of its input z and weight w and shifts the input z to its south-east neighbor. The xw terms along a row are added by a tree of adders and the sum is added to $-\alpha\underline{X}^T(j)\,\underline{d}(j)$. The result is used to update the weight vector.

The computation time for one iteration is

$$T_{comp} = 3T_{mul} + (\lfloor \log M \rfloor + 2)T_{add}$$

However, the architecture can be pipelined in such a way that while processors of type P and R operate on the jth set, processors of type Q operate on the $(j-1)$th set. The block pipeline period (or the sample period since s=1) is given by

$$T_{block} = T_{mul} + (\lfloor \log M \rfloor + 2)T_{add}$$

The total number of multipliers and adders in this architecture is $M(5+M)$ and $M(5+\log M)$ respectively.

Pipelined Implementation: This architecture can be easily pipelined using partial look-ahead relaxation (eqn 1). Fig. 4 describes a pipelined implementation of Architecture_3. The block pipeline period of the pipelined implementation is

$$T_{block} = max\,\{(T_{mul} + (1+\lfloor \log M \rfloor)T_{add}/D1, T_{add}/D2\}$$

6. CONCLUSIONS

In this paper we have proposed three high sample rate architectures for implementing block adaptive filters in the time domain. Table 1 compares the three architectures with respect to hardware complexity (number of multipliers and adders) and block pipelining period. While all the time-domain architectures have a large area complexity of $O(MN)$, the sample periods of even the non-pipelined implementations is quite small. In comparison, for $N = M$, the sample period of the frequency-domain implementations is $\approx 12\log N(T_{mul} + T_{add})$, and the hardware complexity is $O(N\log N)$.

7. REFERENCES

[1] S. Karkada, *High Sample Rate Architectures for the Block Least Mean Squares Algorithm*, M.S. thesis, Arizona State University, Dec 93.

[2] G.A.Clark, S.K.Mitra and S.R.Parker. *"Block implementation of adaptive digital filters"*, IEEE Trans. on Acoust., Speech, and Signal Processing, vol. 29, pp. 744-752, June 1981.

[3] Wasfy B. Mikhael and Andreas S. Spanias *"A fast frequency-domain adaptive algorithm"*, Proceedings of IEEE, Vol 76, No. 1, January 1988.

[4] Wasfy B. Mikhael and F. H. Wu, *"Fast algorithms for block FIR adaptive digital filtering,"* IEEE Trans on Circuits and Systems, vol. CAS-34, pp. 1152-1160, Oct 87.

[5] T.K. Miller, S.T. Alexander, and L.J. Faber, *"An SIMD multiprocessor ring architecture for the LMS adaptive algorithm"*, IEEE Trans. on Communications, Vol COM-34, pp 89-92, Jan 1986.

[6] T. H. Meng and D. G. Messerschmitt, *'An arbitrarily high sampling rate adaptive filters"*, IEEE Trans. on Acous., Speech, and Signal Processing, ASSP-35, pp 455-470, April 1987.

[7] M.D. Meyer and D.P. Agarwal, *"A high sampling rate delayed LMS filter architecture,"* IEEE Trans on Circuits and Systems, Nov 1993, pp. 727-729.

[8] G. Long, F. Ling and J. G. Proakis, *'The LMS algorithm with delayed coefficient adaptation"*, IEEE Trans. on Acous., Speech, and Signal Processing, ASSP-37 : pp1397-1405, Sept. 1989.

[9] Seong-Mo Park *"A VLSI architecture for adaptive signal processing"*, IEEE Trans. on Acous., Speech, and Signal Processing, pp 83-85, August 1991.

[10] K.K.Parhi and D.G. Messerschmitt *"An area efficient adapive lattice filter"*, IEEE Trans. on Acous., Speech, and Signal Processing, July 1987, pp 1118-1134.

[11] N.R. Shanbhag and K.K.Parhi *"A pipelined LMS adaptive filter architecture"*, Proc. 25th Asilomar Conf. Signals, Syst., Comput., Nov 1991.

[12] B. Widrow and S. Stearns, *"Adaptive Signal Processing"* , Englewood Cliffs, NJ: Prentice-Hall, Inc., 1985.