Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2007, Article ID 57086, 7 pages
doi:10.1155/2007/57086

Research Article

An Efficient Implementation of the Sign LMS Algorithm Using Block Floating Point Format

Mrityunjoy Chakraborty,[1] Rafiahamed Shaik,[1] and Moon Ho Lee[2]

[1] *Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur 721302, India*
[2] *Department of Information and Communication, Chonbuk National University, Chonju 561756, South Korea*

Received 11 July 2005; Revised 31 August 2006; Accepted 24 November 2006

Recommended by Roger Woods

An efficient scheme is presented for implementing the sign LMS algorithm in block floating point format, which permits processing of data over a wide dynamic range at a processor complexity and cost as low as that of a fixed point processor. The proposed scheme adopts appropriate formats for representing the filter coefficients and the data. It also employs a scaled representation for the step-size that has a time-varying mantissa and also a time-varying exponent. Using these and an upper bound on the step-size mantissa, update relations for the filter weight mantissas and exponent are developed, taking care so that neither overflow occurs, nor are quantities which are already very small multiplied directly. Separate update relations are also worked out for the step size mantissa. The proposed scheme employs mostly fixed-point-based operations, and thus achieves considerable speedup over its floating-point-based counterpart.

1. INTRODUCTION

Sufficient signal-to-quantization noise ratio over a large dynamic range is a desirable feature of modern day digital signal processing systems. While the floating point (FP) data format is ideally suited to achieve this due to normalized data representation, the accompanying high processing cost restricts its usage in many applications. This is specially true for resource-constrained contexts like battery-operated low power devices, where custom implementations on FPGA/ASIC are the primary mode of realization. In such contexts, the block floating point (BFP) format provides a viable alternative to the FP scheme. In BFP, a common exponent is assigned to a group of variables. As a result, computations involving these variables can be carried out in simple fixed point (FxP) like manner, while presence of the exponent provides an FP-like high dynamic range.

Over years, the BFP format has been used by several researchers for efficient realization of many signal processing systems and algorithms. These include various forms of fixed coefficient digital filters (see [1–6]), adaptive filters (see [7, 8]), and unitary transforms (see [9–11]) on one hand and several audio data transmission standards like NICAM (stereophonic sound system for PAL TV standard), the audio part of MUSE (Japanese HDTV standard), and DSR (German digital satellite radio system) on the other. Of the various systems studied, adaptive filters pose special challenges to their implementation using the BFP arithmetic. This is mainly because

(i) unlike a fixed coefficient filter, the filter coefficients in an adaptive filter cannot be represented in the simpler fixed point form, as the coefficients in effect evolve from the data by a time update relation;

(ii) the two principal operations in an adaptive filter—filtering and weight updating, are mutually coupled, thus requiring an appropriate arrangement for joint prevention of overflow.

Recently, a BFP-based approach has been proposed for efficient realization of the LMS-based transversal adaptive filters [7], which was later extended to the normalized LMS algorithm [8] and the gradient adaptive lattice [12]. In this paper, we extend the philosophy used in [7] for a BFP realization of the sign LMS algorithm [13]. The sign LMS algorithm forms a popular class of adaptive filters within the LMS family, which considers mainly the sign of the gradient in the weight update process and thus does not require multipliers in the weight update loop. The proposed scheme adopts appropriate BFP format for the filter coefficients which remains invariant as the coefficients are updated in time. Using this and the BFP representation of the data as used in [7], separate time update relations for the filter weight mantissas and the exponent are developed. Unlike [7], the

proposed scheme, however, requires a scaled representation for the step size, which has a time-varying mantissa and also a time-varying exponent. Separate time update relation for the step size mantissa is worked out. It is also shown that in order to maintain overflow free condition, the step size mantissa, at all times, must remain bounded by an upper limit, which is ensured by setting its initial value appropriately. Again, the weight update relation of the sign LMS algorithm is different from the LMS algorithm and thus new steps are needed for the computation of the update term, taking care so that neither overflow occurs, nor are quantities which are already very small multiplied directly. As expected, the proposed scheme employs mostly FxP-based operations and thus achieves considerable speed up over its FP-based counterpart, which is verified both by detailed complexity analysis and from the synthesis report of an FPGA-based realization.

The organization of the paper is as follows: in Section 2, we discuss the BFP arithmetic and present a new block formatting algorithm for FP as well as FxP data. Section 3 presents the proposed BFP realization of the sign LMS algorithm. Complexity issues vis-à-vis an FP-based realization are discussed in Section 4 while finite precision based simulation results as well as the FPGA synthesis summary are presented in Section 5. Variables with an overbar indicate mantissa elements all throughout the paper. Also, boldfaced lowercase letters are used to denote vectors.

2. THE BFP ARITHMETIC AND A BLOCK-FORMATTING ALGORITHM

The BFP representation can be considered as a special case of the FP format, where every nonoverlapping block of N incoming data has a joint scaling factor corresponding to the data sample with the highest magnitude in the block. In other words, given a block $[x_0,\ldots,x_{N-1}]$, we represent it in BFP as $[x_0,\ldots,x_{N-1}] = [\overline{x}_0,\ldots,\overline{x}_{N-1}]2^{\gamma}$ where $\overline{x}_l(= x_l 2^{-\gamma})$ represents the mantissa x_l for $l = 0,1,\ldots,N-1$ and the block exponent γ is defined as $\gamma = \lfloor \log_2 \mathrm{Max} \rfloor + 1 + S$ where $\mathrm{Max} = \max(|x_0|,\ldots,|x_{N-1}|)$, "$\lfloor\cdot\rfloor$" is the so-called floor function, meaning rounding down to the closest integer and the integer S is a scaling factor, used for preventing overflow during filtering operation.

In practice, if the data is given in an FP format, that is, if $x_l = M_l 2^{e_l}$, $l = 0,1,\ldots,N-1$ with $|M_l| < 1$, and the 2's complement system is used, the above block formatting may be carried out by Algorithm 1.

Algorithm 1 (Block-formatting algorithm). First, count the number, say, n_l of binary 0's (if x_l is positive) or binary 1's (if x_l is negative) between the binary point of M_l and the first binary 1 or binary 0 from left, respectively. Compute $e_{\max} = \max\{(e_l - n_l) \mid l = 0,1,\ldots,N-1\}$. Shift each M_l right or left by $(e_{\max} + S - e_l)$ bits depending on whether $(e_{\max} + S - e_l)$ is positive or negative, respectively. Take the block exponent as $e_{\max} + S$.

Note. For cases where x_l is negative with M_l having only binary 0's after the first n_l bits from the binary point, n_l should be replaced by $n_l - 1$ in the above computation.

When the data is given in FxP format, the corresponding block formatting turns out to be a special case of the above, for which $x_l \equiv M_l$, $e_l = 0$, and $e_{\max}$ is given by $\min\{n_l \mid l = 0,1,\ldots,N-1\}$. Note that due to the presence of S, the range of each mantissa is given as $0 \le |\overline{x}_l| < 2^{-S}$. The scaling factor S can be calculated from the inner product computation representing filtering operation [3]. An inner product is calculated in BFP arithmetic as

$$\begin{aligned} y(n) &= \mathbf{w}^t\mathbf{x}(n) \\ &= \left[w_0\overline{x}(n) + \cdots + w_{L-1}\overline{x}(n-L+1)\right]2^{\gamma} \\ &= \overline{y}(n)2^{\gamma}, \end{aligned} \tag{1}$$

where $\mathbf{w}$ is a length L, fixed point filter coefficient vector, and $\mathbf{x}(n)$ is the data vector at the nth index, represented in the aforesaid BFP format. For no overflow in $y(n)$, we need $|\overline{y}(n)| < 1$. Since $|\overline{y}(n)| \le \sum_{k=0}^{L-1} |w_k||\overline{x}(n-k)|$ and $0 \le |\overline{x}(n-k)| < 2^{-S}$, $0 \le k \le L-1$, this implies that it is sufficient to have $S \ge \lceil \log_2(\sum_{k=0}^{L-1} |w_k|)\rceil$ in order to have $|\overline{y}(n)| < 1$ satisfied, where "$\lceil\cdot\rceil$" denotes the so-called ceiling function, meaning rounding up to the closest integer.

3. THE PROPOSED IMPLEMENTATION

Consider a length L sign LMS based adaptive filter [13] that takes an input sequence $x(n)$ and updates the weights as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\mathbf{x}(n)\,\mathrm{sgn}\{e(n)\}, \tag{2}$$

where $\mathbf{w}(n) = [w_0(n)\ \ w_1(n)\ \ \cdots\ \ w_{L-1}(n)]^t$ is the tap weight vector at the nth index, $\mathbf{x}(n) = [x(n)\ \ x(n-1)\ \cdots\ x(n-L+1)]^t$, and $e(n) = d(n) - y(n)$ is the output error corresponding to the nth index. The sequence $d(n)$ is the so-called desired response available during the initial training period and $y(n) = \mathbf{w}^t(n)\mathbf{x}(n)$ is the filter output at the nth index, with μ denoting the so-called step size parameter. The operator $\mathrm{sgn}\{\cdot\}$ is the well known *signum* function which returns values $+1$ or -1 depending on whether the operand is nonnegative or negative, respectively.

The proposed scheme uses a scaled format to represent the filter coefficient vector $\mathbf{w}(n)$ as

$$\mathbf{w}(n) = \overline{\mathbf{w}}(n)2^{\psi_n}, \tag{3}$$

where $\overline{\mathbf{w}}(n)$ and ψ_n are, respectively, the filter mantissa vector and the filter block exponent which are updated separately over n. The chosen format thus normalizes all components of $\mathbf{w}(n)$ by a common factor 2^{ψ_n} at each index n. In our treatment, the exponent ψ_n is a nondecreasing function of n with zero initial value and is chosen to ensure that $|\overline{w}_k(n)| < 1/2$, for all $k \in \mathbb{Z}_L = \{0,1,\ldots,L-1\}$. If the data vector $\mathbf{x}(n)$ is given in the aforesaid BFP format as $\mathbf{x}(n) = \overline{\mathbf{x}}(n)2^{\gamma}$, where $\gamma = \mathrm{ex}+S$, $\mathrm{ex} = \lfloor \log_2 M \rfloor + 1$, $M = \max(|x(n-k)| \mid k \in \mathbb{Z}_L)$ and S is an appropriate scaling factor, then, the filter output $y(n)$ can be expressed as $y(n) = \overline{y}(n)2^{\gamma+\psi_n}$ with $\overline{y}(n) = \overline{\mathbf{w}}^t(n)\overline{\mathbf{x}}(n)$ denoting the output mantissa. To prevent overflow in $\overline{y}(n)$, it is required that $|\overline{y}(n)| < 1$. However, in the proposed scheme, we restrict $\overline{y}(n)$ to lie between $+1/2$ and $-1/2$, that is, $|\overline{y}(n)| < 1/2$.

Since $|\overline{w}_k(n)| < 1/2$, $k \in \mathbb{Z}_L$, from Section 2, this implies that it is sufficient to have $S \geq S_{\min} = \lceil \log_2 L \rceil$, in order to maintain $|\overline{y}(n)| < 1/2$. The two conditions $|\overline{w}_k(n)| < 1/2$, for all $k \in \mathbb{Z}_L$ and $|\overline{y}(n)| < 1/2$ ensure no overflow during updating of $\overline{\mathbf{w}}(n)$ and computation of output error mantissa, respectively, as shown later.

The proposed implementation

The proposed BFP realization consists of the following three stages.

(i) *Buffering*: here, the input sequence $x(n)$ and the desired response $d(n)$ are jointly partitioned into nonoverlapping blocks of length N each, with the ith block given by $\{x(n), d(n) \mid n \in \mathbb{Z}_i'\}$, where $\mathbb{Z}_i' = \{iN, iN+1, \ldots, iN+N-1\}$, $i \in \mathbb{Z}$. For this, $x(n)$ and $d(n)$ are shifted into buffers of size N each. We take $N \geq L-1$, as otherwise, the complexity of implementation would go up. The buffers are cleared and their contents transferred jointly to a block formatter once in every N input clock cycles.

(ii) *Block formatting*: here, the data samples $x(n)$ and $d(n)$ which constitute the ith block, $i \in \mathbb{Z}$, and which are available in either FP or FxP form, are block formatted as per the block formatting algorithm of Section 2, resulting in the BFP representation: $x(n) = \overline{x}(n)2^{\gamma_i}$, $d(n) = \overline{d}(n)2^{\gamma_i}$ $n \in \mathbb{Z}_i'$, where $\gamma_i = \mathrm{ex}_i + S_i$, $\mathrm{ex}_i = \lfloor \log_2 M_i \rfloor + 1$, $M_i = \max\{|x(n)|, |d(n)| \mid n \in \mathbb{Z}_i'\}$. The scaling factor S_i is chosen to ensure that (i) $S_i \geq S_{\min}$, and (ii) $\mathbf{x}(n)$ has a uniform BFP representation during the block-to-block transition phase as well, that is, when part of $\mathbf{x}(n)$ comes from the ith block and part from the $(i-1)$th block. This is realized by the following exponent assignment algorithm (see Algorithm 2).

Algorithm 2. [Exponent assignment algorithm] Assign $S_{\min} = \lceil \log_2 L \rceil$ as the scaling factor to the first block and for any $(i-1)$th block, assume $S_{i-1} \geq S_{\min}$. Then, if $\mathrm{ex}_i \geq \mathrm{ex}_{i-1}$, choose $S_i = S_{\min}$ (i.e., $\gamma_i = \mathrm{ex}_i + S_{\min}$) else (i.e., $\mathrm{ex}_i < \mathrm{ex}_{i-1}$) choose $S_i = (\mathrm{ex}_{i-1} - \mathrm{ex}_i + S_{\min})$, s.t. $\gamma_i = \mathrm{ex}_{i-1} + S_{\min}$.

Note that when $\mathrm{ex}_i \geq \mathrm{ex}_{i-1}$, we can either have $\mathrm{ex}_i + S_{\min} \geq \gamma_{i-1}$ (Case A) implying $\gamma_i \geq \gamma_{i-1}$, or, $\mathrm{ex}_i + S_{\min} < \gamma_{i-1}$ (Case B) meaning $\gamma_i < \gamma_{i-1}$. However, for $\mathrm{ex}_i < \mathrm{ex}_{i-1}$ (Case C), we always have $\gamma_i \leq \gamma_{i-1}$. Additionally, we rescale the elements $\overline{x}(iN-L+1), \ldots, \overline{x}(iN-1)$ by dividing by $2^{\Delta\gamma_i}$, where $\Delta\gamma_i = \gamma_i - \gamma_{i-1}$. Equivalently, for the elements $x(iN-L+1), \ldots, x(iN-1)$, we change S_{i-1} to an effective scaling factor of $S_{i-1}' = S_{i-1} + \Delta\gamma_i$. This permits a BFP representation of the data vector $\mathbf{x}(n)$ with common exponent γ_i during block-to-block transition phase as well.

In practice, such rescaling is effected by passing each of the delayed terms $\overline{x}(n-j)$, $j = 1, \ldots, L-1$, through a rescaling unit that applies $\Delta\gamma_i$ number of right or left shifts on $\overline{x}(n-j)$ depending on whether $\Delta\gamma_i$ is positive or negative, respectively. This is, however, done only at the beginning of each block, that is, at indices $n = iN$, $i \in \mathbb{Z}^+$. Also, note that though for the case (A) above, $\Delta\gamma_i \geq 0$, for (B) and (C), however, $\Delta\gamma_i \leq 0$, meaning that in these cases, the aforesaid mantissas from the $(i-1)$th block are actually scaled up by $2^{-\Delta\gamma_i}$. It is, however, not difficult to see that the effective scaling factor S_{i-1}' for the elements $x(iN-L+1), \ldots, x(iN-1)$ still remains lower bounded by $S_{\min}$, thus ensuring no overflow during filtering operation.

(iii) *Filtering and weight updating*: the block formatter inputs $\overline{x}(n)$, $\overline{d}(n)$, $n \in \mathbb{Z}_i'$, and (b) the rescaled mantissas for $x(iN-k)$, $k = 1, 2, \ldots, L-1$ to the transversal filter, which computes $\overline{y}(n) = \overline{\mathbf{w}}^t(n)\overline{\mathbf{x}}(n)$ for all $n \in \mathbb{Z}_i'$. Since the data in (b), coming from the $(i-1)$th block, are rescaled so as to have the same exponent γ_i, the above computation can be made faster via *overlap and save* method. This employs $(N+L-1)$ point FFT on data frames formed by appending the data in (b) to the left of $[\overline{x}(iN), \ldots, \overline{x}(iN+N-1)]$ and discarding the first $L-1$ output. Since the FFT is FxP-based, it would require much less computational complexities than an FP-based evaluation.

Next, the output error $e(n)$ is evaluated as $e(n) = \overline{e}(n)2^{\gamma_i + \psi_n}$ where the mantissa $\overline{e}(n)$ is given by

$$\overline{e}(n) = \overline{d}(n)2^{-\psi_n} - \overline{y}(n). \tag{4}$$

It is easy to see that $|\overline{e}(n)| < 1$, that is, the computation in (5) above does not produce any overflow, since

$$\begin{aligned} |\overline{e}(n)| &\leq |\overline{d}(n)|2^{-\psi_n} + |\overline{y}(n)| \\ &< 2^{-(S_i+\psi_n)} + \frac{1}{2} \leq \frac{2^{-\psi_n}}{L} + \frac{1}{2} \end{aligned} \tag{5}$$

as $2^{-S_i} \leq 1/L$. Except for $\psi_n = 0$, $L = 1$, the right-hand side is always less than or equal to 1.

For the above description of $e(n)$, $\mathbf{x}(n)$, $\mathbf{w}(n)$ and noting that $\mathrm{sgn}\{e(n)\} = \mathrm{sgn}\{\overline{e}(n)\}$, the weight update equation (2) can now be written as $\mathbf{w}(n+1) = \overline{\mathbf{v}}(n)2^{\psi_n}$, where

$$\overline{\mathbf{v}}(n) = \overline{\mathbf{w}}(n) + \overline{\mu}_n \overline{\mathbf{x}}(n)\,\mathrm{sgn}\,\{\overline{e}(n)\}2^{\gamma_i}, \tag{6}$$

where $\overline{\mu}_n = \mu 2^{-\psi_n}$. In other words, the proposed scheme employs a scaled representation for μ as $\mu = \overline{\mu}_n 2^{\psi_n}$, with $\overline{\mu}_n$ updated from a knowledge of ψ_n and ψ_{n+1} as

$$\overline{\mu}_{n+1} = \overline{\mu}_n 2^{(\psi_n - \psi_{n+1})}. \tag{7}$$

As stated earlier, $\overline{\mathbf{w}}(n+1)$ is required to satisfy $|\overline{w}_k(n+1)| < 1/2$, for all $k \in \mathbb{Z}_L$, which can be realized in several ways. Our preferred option is to limit $\overline{\mathbf{v}}(n)$ so that $|\overline{v}_k(n)| < 1$, for all $k \in \mathbb{Z}_L$. Then, if each $\overline{v}_k(n)$ happens to be lying within $\pm 1/2$, we make the assignments

$$\overline{\mathbf{w}}(n+1) = \overline{\mathbf{v}}(n), \quad \psi_{n+1} = \psi_n. \tag{8}$$

Otherwise, we scale down $\overline{\mathbf{v}}(n)$ by 2, in which case

$$\overline{\mathbf{w}}(n+1) = \frac{1}{2}\overline{\mathbf{v}}(n), \quad \psi_{n+1} = \psi_n + 1. \tag{9}$$

In order to have $|\overline{v}_k(n)| < 1$, for all $k \in \mathbb{Z}_L$ satisfied, we observe from (7) that $|\overline{v}_k(n)| \leq |\overline{w}_k(n)| + \overline{\mu}_n |\overline{x}(n-k)|2^{\gamma_i}$. Since $|\overline{w}_k(n)| < 1/2$, $k \in \mathbb{Z}_L$, it is sufficient to have $\overline{\mu}_n |\overline{x}(n-k)|2^{\gamma_i} \leq 1/2$. Recalling that $|\overline{x}(n-k)| < 2^{-S_i}$, this implies

$$\overline{\mu}_n \leq \frac{2^{-\mathrm{ex}_i}}{2}. \tag{10}$$

It is easy to verify that the above bound for $\overline{\mu}_n$ is valid not only when each element of $\overline{\mathbf{x}}(n)$ in (6) comes purely from the ith block, but also during transition from the $(i-1)$th to the ith block with $\text{ex}_i \geq \text{ex}_{i-1}$, for which, after necessary rescaling, we have $S'_{i-1} \geq S_i = S_{\min}$ implying $|\overline{x}(n-k)| < 2^{-S_i}$. For $\text{ex}_i < \text{ex}_{i-1}$, however, the upper bound expression given by (11) gets modified with ex_i replaced by ex_{i-1}, as in that case, we have $\gamma_i = \text{ex}_{i-1} + S'_{i-1}$ with $S'_{i-1} = S_{\min} < S_i$ meaning $|\overline{x}(n-k)| < 2^{-S'_{i-1}}$.

From above, we obtain a general upper bound for $\overline{\mu}_n$ by replacing ex_i by $\text{ex}_{\max} = \max\{\text{ex}_i \mid i \in \mathbb{Z}^+\}$, which is given by

$$\overline{\mu}_n \leq \frac{2^{-\text{ex}_{\max}}}{2}. \tag{11}$$

In order to satisfy the above upper bound, first note from (8) and (9) that ψ_n is a nondecreasing function of n. This, together with (7), implies that $\overline{\mu}_{n+1} \leq \overline{\mu}_n$ for all n. To satisfy the above upper bound, it is thus enough to fix the initial value of $\overline{\mu}_n$ by setting the first $\text{ex}_{\max}+1$ bits of the corresponding register following the binary point as zero, if $\text{ex}_{\max}+1 \geq 0$. If, however, $\text{ex}_{\max}+1 < 0$, one can retain $|\text{ex}_{\max}+1|$ data bits to the left of the binary point. Note also that since the initial value of ψ_n is zero, the initial value of $\overline{\mu}_n$ actually determines the step size μ.

Finally, for practical implementation of $\overline{\mathbf{v}}(n)$ as given by (6), we need to evaluate the product $\overline{\mu}_n \overline{x}(n-k)2^{\gamma_i}$ in such a way that no overflow occurs in any of the intermediate products or shift operations. At the same time, we need to avoid direct product of quantities which could be very small, as that may lead to loss of several useful bits via truncation. For this purpose, we proceed as follows: if $\text{ex}_i \geq \text{ex}_{i-1}$, then, $S_i = S_{\min}$ and we express 2^{γ_i} as $2^{\gamma_i} = 2^{\text{ex}_i} 2^{S_{\min}}$. If, instead, $\text{ex}_i < \text{ex}_{i-1}$, then, $S'_{i-1} = S_{\min}$, $\gamma_i = \text{ex}_{i-1} + S'_{i-1}$ and we decompose 2^{γ_i} as $2^{\gamma_i} = 2^{\text{ex}_{i-1}} 2^{S_{\min}}$. The factors 2^{ex_i} (or, $2^{\text{ex}_{i-1}}$) and $2^{S_{\min}}$ are then distributed to compute the update term as follows.

Step 1. $\mu_{1,n} = \overline{\mu}_n 2^{\text{ex}_i}$, if $\text{ex}_i \geq \text{ex}_{i-1}$; if $\text{ex}_i < \text{ex}_{i-1}$, $\mu_{1,n} = \overline{\mu}_n 2^{\text{ex}_{i-1}}$.

Step 2. $\overline{x}(n-k)2^{S_{\min}} = \overline{x}_1(n-k)(\text{say}),\ \forall k \in \mathbb{Z}_L$.

Step 3. $\mu_{1,n}\overline{x}_1(n-k),\ \forall k \in \mathbb{Z}_L$.

Note that in Step 2, only the current mantissa $\overline{x}(n)$ is to be shifted by $2^{S_{\min}}$, as the other terms $\overline{x}(n-k)$, $k = 1, 2, \ldots, L-1$ are already shifted at the previous indices. For $n = iN$, that is, the starting index of the ith block, these terms correspond to the last $(L-1)$ mantissas of the $(i-1)$th block, rescaled by $2^{-\Delta\gamma_i}$. Further scaling of these terms by $2^{S_{\min}}$ can be carried out during the block formatting stage, that is, before the processing of the ith block.

The proposed BFP treatment to the sign LMS algorithm is summarized in Table 1. The three units, viz., (i) buffering, (ii) block formatting, and (iii) filtering and weight updating are actually *pipelined* and their relative timing is shown in Figure 1. Also, for the filtering and weight updating unit, the internal processing is illustrated in Figure 2.

Table 1: Summary of the sign LMS algorithm realized in BFP format (initial conditions: $\psi_0 = 0$, $|\overline{w}_k(0)| < 1/2$, $k \in \mathbb{Z}_L$, $\overline{\mu}_0 = \mu$).

(1) Preprocessing:
using the data for the ith block, $x(n)$ and $d(n)$, $n \in \mathbb{Z}'_i$, $i \in \mathbb{Z}^+$ (stored during the processing of the $(i-1)$th block).
(a) Evaluate block exponent γ_i as per the exponent assignment algorithm of Section 3 and express $x(n)$, $d(n)$, $n \in \mathbb{Z}'_i$ as $x(n) = \overline{x}(n)2^{\gamma_i}$, $d(n) = \overline{d}(n)2^{\gamma_i}$.
(b) Rescale the following elements of the $(i-1)$th block: $\{\overline{x}(n) \mid n = iN - L + 1, \ldots, iN - 1\}$ as $\overline{x}(n) \to \overline{x}(n)2^{-\Delta\gamma_i}$, $\Delta\gamma_i = \gamma_i - \gamma_{i-1}$ (also, for Step 2 of Section 3, rescale the same separately by $2^{-\Delta\gamma_i + S_{\min}}$).

(2) Processing for the ith block:
For $n \in \mathbb{Z}'_i = \{iN, iN+1, \ldots, iN+N-1\}$.
(a) Filter output:
$\overline{y}(n) = \overline{\mathbf{w}}^t(n)\overline{\mathbf{x}}(n)$,
$\text{ex\_out}(n) = \gamma_i + \psi_n$
($\text{ex\_out}(n)$ is the filter output exponent at the nth index).
(b) Output error (mantissa) computation:
$\overline{e}(n) = \overline{d}(n)2^{-\psi_n} - \overline{y}(n)$.
(c) Filter weight updating:
compute $\overline{u}_k(n) = \overline{\mu}_n \overline{x}(n-k)2^{\gamma_i}$ for all $k \in \mathbb{Z}_L$
following Steps 1–3 of Section 3.
$\overline{\mathbf{v}}(n) = \overline{\mathbf{w}}(n) + \overline{\mathbf{u}}(n)\,\text{sgn}\{\overline{e}(n)\}$
(where $\overline{\mathbf{u}}(n) = [\overline{u}_0(n), \overline{u}_1(n), \ldots, \overline{u}_{L-1}(n)]^t$).
If $|\overline{v}_k(n)| < 1/2$ for all $k \in \mathbb{Z}_L = \{0, 1, \ldots, L-1\}$
then
$\overline{\mathbf{w}}(n+1) = \overline{\mathbf{v}}(n)$,
$\psi_{n+1} = \psi_n$,
else
$\overline{\mathbf{w}}(n+1) = \frac{1}{2}\overline{\mathbf{v}}(n)$,
$\psi_{n+1} = \psi_n + 1$.
end.
$\overline{\mu}_{n+1} = \overline{\mu}_n 2^{(\psi_n - \psi_{n+1})}$
end
$i = i + 1$.
Repeat Steps 1 to 2.

4. COMPLEXITY ISSUES

The proposed scheme relies mostly on FxP arithmetic, resulting in computational complexities much less than that of their FP-based counterparts. For example, to compute the filter output in Table 1, L "multiply and accumulate (MAC)" operations (FxP) are needed to evaluate $\overline{y}(n)$ and at the most, one exponent addition operation to compute the exponent $\text{ex\_out}(n)$. In FP, this would require L FP-based MAC operations. Note that given three numbers in FP (normalized) format $A = \overline{A}2^{e_a}$, $B = \overline{B}2^{e_b}$, $C = \overline{C}2^{e_c}$, the MAC operation $A + BC$ requires the following steps: (i) $e_b + e_c$, that is, exponent addition (EA), (ii) exponent comparison (EC) between e_a and $e_b + e_c$, (iii) shifting either $\overline{A}$ or $\overline{B}/\overline{C}$, (iv) FxP-based MAC, and finally, (v) renormalization, requiring shift

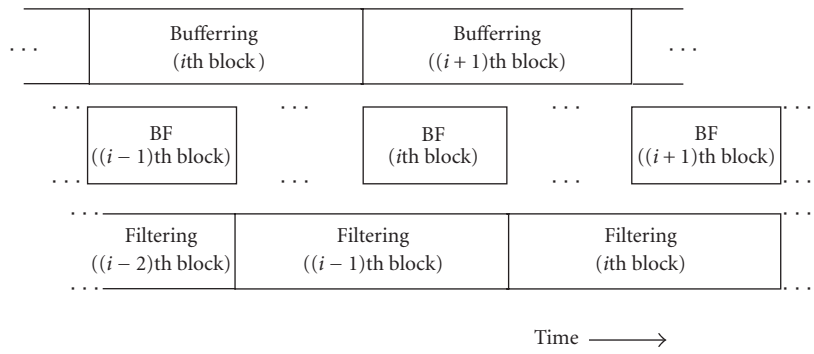

Figure 1: The relative timing of the three units (BF: block formatting).

and exponent addition. In other words, in FP, computation of $y(n)$ will require the following additional operations over the BFP-based realization: (a) $2L$ shifts (assuming availability of single cycle barrel shifters), (b) L EC, and (c) $2L-1$ EA. Similar advantages exist in weight updating also. Table 2 provides a comparative account of the two approaches in terms of number of operations required per iteration. Note that the number of additional operations required under FP increases linearly with the filter length L. It is easy to verify from Table 2 that given a low cost, simple FxP processor with single cycle MAC and barrel shifter units, the proposed scheme is about *three times faster* than an FP-based implementation, for moderately large values of L.

5. SIMULATION AND FINITE PRECISION IMPLEMENTATION

The proposed scheme was implemented in finite precision in the context of a system identification problem. A system modelled by a 3-tap FIR filter was used to generate an output $y(n) = 0.7x(n) + 0.65x(n-1) + 0.25x(n-2) + v(n)$, with $v(n)$ and $x(n)$ being the observation noise and the system input, respectively, with the following variances: $\sigma_v^2 = 0.008$, $\sigma_x^2 = 1$. The variance σ_y^2 of $y(n)(\equiv d(n))$ was found to be 0.935. To calculate the upper bound of $\mu(= 2^{-\mathrm{ex}_{\max}}/2)$, the quantity $M = \{|x(n)|, |y(n)| \mid n \in Z\}$ was calculated, as $1.99\max\{\sigma_x, \sigma_y\}$, so as to contain about 95% of the samples of $x(n)$ and $y(n)$. This gives rise to $\mathrm{ex}_{\max} = 1$ and thus the upper bound of μ to be 0.25. Taking $\mu = 2^{-6}$, block length $N = 20$, and allocating 12 bits $(1+11)$ for the mantissas and $4(1+3)$ bits for the exponents of both the data and the filter coefficients, the proposed scheme was implemented in VHDL. For this, the Xilinx ISE 7.1i software was used for a target device of Xilinx Virtex series FPGA XCV1000bg (speed grade 6). Details of this implementation like hardware requirement, respective gate counts, and execution times are provided later in this section. Here we study the finite precision effects by plotting the learning curves for this as well as for an FP-based realization under same precision for both the exponent and the mantissa. The learning curves, shown in Figure 3 by solid and dashed lines, respectively, demonstrate that both these implementations have similar rates of convergence. However, in the steady state, the BFP scheme has slightly more excess mean square error (EMSE) than the FP scheme, which may be caused by the block formatting of data. This slight increase in EMSE is, however, offset by the speedup that is achieved and verified by comparing the execution times of the proposed realization with its FP counterpart.

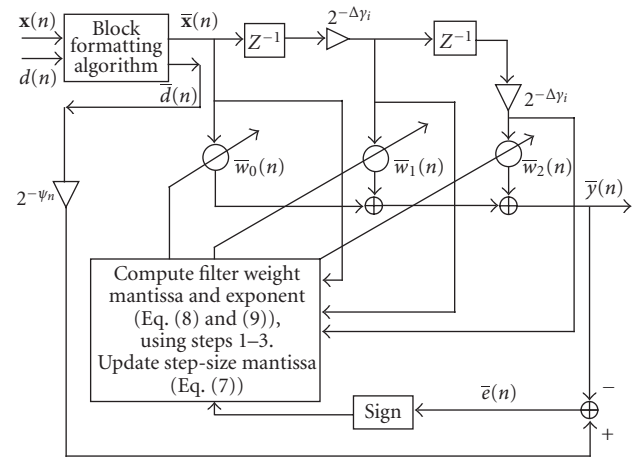

Figure 2: The proposed sign-LMS-based adaptive filter in BFP format. The shifting of $\overline{x}(n-k)$, $k = 1, 2$ by $2^{-\Delta y_i}$ is done only at the starting index of each block, that is, at $n = iN$, $i \in \mathbb{Z}^+$.

Table 2: A comparison between the BFP vis-à-vis the FP-based realizations of the sign LMS algorithm. Number of operations required per iteration for (a) weight updating and (b) filtering is given.

(a)	MAC	Shift	Magnitude check	Exponent comparison	Exponent addition
BFP	L	$L+3$	L	Nil	1
FP	L	$2L$	Nil	L	$2L$

(b)	MAC	Shift	Exponent comparison	Exponent addition
BFP	L	Nil	Nil	1
FP	L	$2L$	L	$2L$

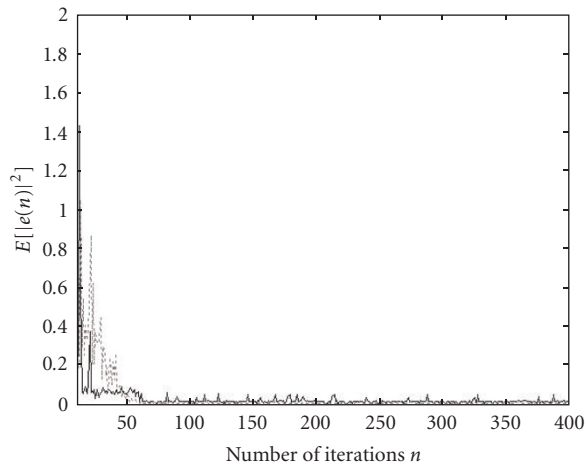

FIGURE 3: Learning curves for the finite precision implementation of (a) the proposed BFP-based scheme (solid line), and (b) an FP-based implementation (dashed line) with identical precision.

FPGA synthesis summary

The proposed scheme as well as the FP-based algorithm are implemented using basic hardware units like adder, multiplier, register, multiplexer, shifter, and so forth. The step size μ is taken to be a power of two as it eliminates the need of multiplier in the weight update loop. For the proposed scheme, the three stages, (a) buffering, (b) block formatting, and (c) filtering and weight updating have the following hardware requirements.

(a) *Buffering*: this stage uses N 16 bit registers, where N is the block length ($N = 20$ for the example considered).

(b) *Block formatting*: this stage first computes $e_{\max} = \max\{(e_l - n_L) \mid l = 0, 1, \ldots, N-1\}$ (see Algorithm 1) by employing a 4 bit subtractor, a 4 bit comparator, and a 4 bit register for each l, $l = 0, 1, \ldots, N-1$. One 4 bit adder is used next to compute the block exponent $e_{\max} + S_i$. Then, for each $l, l = 0, 1, \ldots, N-1$, $e_{\max} + S_i - e_l$ is computed by using one 4 bit subtractor and the lth data mantissa is shifted left/right by $e_{\max} + S_i - e_l$ using two 12 bit shifters. The block formatted mantissas are finally stored in N 12 bit registers.

(c) *Filtering and weight updating*: for filtering, a MAC operation (FxP) is used iteratively L times where L is the filter order ($L = 3$ for the example considered). The MAC unit requires one 12×12 multiplier, one 24 bit adder, and two 24 bit registers, one for holding the result of multiplication and the other for storing the MAC output. This is followed by computation of output error mantissa that uses one 12 bit shifter and one 12 bit subtractor. For updating each tap weight, first note that since μ is a power of 2, that is, $\mu = \overline{\mu}_0 = 2^s$ (say), we have $\overline{\mu}_n = 2^{s_n}$ where $s_n = s - \psi_n$. For updating $\overline{\mu}_n$, it is then enough to update s_n, which requires a 4 bit subtractor and a 4 bit register, but does not require the shifter implied in the general update relation (7). The Steps 1–3 of Section 3 also get simplified, as it is then sufficient to use two 4 bit adders and one 4 bit register to compute $s_n + \text{ex}_i + S_{\min}$, $2L$ 12 bit shifters to shift $\overline{x}(n-k)$, $k \in \mathbb{Z}_L$ left/right by $s_n + \text{ex}_i + S_{\min}$ and L 12 bit adders/subtractors to evaluate $\overline{\mathbf{v}}(n)$ as per (6). Finally, to realize the update relations (8) and (9), we need a 4 bit adder and a 4 bit register to update ψ_n, and L 12 bit shifters as well as L 12 bit registers to compute $\overline{\mathbf{w}}(n)$.

An FP-based realization, on the other hand, has only two operations, namely, filtering and weight updating, both requiring FP addition and multiplication. If two FP numbers having r bit mantissa and m bit exponent each are multiplied, we need one $r \times r$ multiplier, one m bit adder, and two registers of length m bits and $2r$ bits. If, on the other hand, the two numbers are added, we need one m bit comparator, one m bit subtractor, two r bit shifters, two r bit $2:1$ MUX, one r bit adder and for renormalization of the result, two r bit shifters and one m bit adder/subtractor. We also need registers of length m bits and r bits for storing the mantissa and exponent of the result of addition. To realize the filtering operation, an FP-based MAC operation is used iteratively L times that uses one FP multiplication with $r = 12$ and $m = 4$, and an FP addition with $r = 24$ and $m = 4$. For computing the output error, an FP addition with $r = 12$ and $m = 4$ is deployed. For updating each weight, a 4 bit adder is used to add the exponents of the step size and data, followed by an FP addition with $r = 12$ and $m = 4$.

The total equivalent gate count for the proposed scheme with $N = 20$ was found to be 9227, while the same for an FP-based implementation was 12,468. The minimum clock period needed for the FP-based implementation has been 16.052 ns. For the proposed scheme, minimum clock periods required for the three stages, (a) buffering, (b) block formatting, and (c) filtering and weight updating have been 0.232 ns, 4.575 ns, and 6.695 ns. In other words, the minimum clock period needed for the proposed scheme has been 6.695 ns and thus the BFP realization is about 2.39 times faster than the FP-based realization, which also conforms to our observation from Table 2 for $L = 3$.

6. CONCLUSION

The sign LMS algorithm is presented in a BFP framework that ensures simple FxP-based operations in most of the computations while maintaining an FP-like wide dynamic range via a block exponent. The proposed scheme is particularly useful for custom implementations on ASIC or FPGA, where hardware and power efficiency constitute an important factor. For identical resource constraints, the proposed scheme achieves a speed-up in the range of $2:1$ to $3:1$ over an FP-based implementation, as observed both from operational counts and also from a custom implementation on FPGA. Finite precision-based simulations also did not show up any noticeable difference in the convergence characteristics, as one moves from the FP to the BFP format.

ACKNOWLEDGMENT

This work was supported in part by the Institute of Information Technology Assessment (IITA), South Korea.

REFERENCES

[1] K. R. Ralev and P. H. Bauer, "Realization of block floating-point digital filters and application to block implementations," *IEEE Transactions on Signal Processing*, vol. 47, no. 4, pp. 1076–1086, 1999.

[2] S. Sridharan, "Implementation of state-space digital filter structures using block floating-point arithmetic," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '87)*, pp. 908–911, Dallas, Tex, USA, April 1987.

[3] K. Kalliojärvi and J. Astola, "Roundoff errors in block-floating-point systems," *IEEE Transactions on Signal Processing*, vol. 44, no. 4, pp. 783–790, 1996.

[4] S. Sridharan and G. Dickman, "Block floating-point implementation of digital filters using the DSP56000," *Microprocessors and Microsystems*, vol. 12, no. 6, pp. 299–308, 1988.

[5] S. Sridharan and D. Williamson, "Implementation of high-order direct-form digital filter structures," *IEEE Transactions on Circuits and Systems*, vol. 33, no. 8, pp. 818–822, 1986.

[6] F. J. Taylor, "Block floating-point distributed filters," *IEEE Transactions on Circuits and Systems*, vol. 31, no. 3, pp. 300–304, 1984.

[7] A. Mitra, M. Chakraborty, and H. Sakai, "A block floating-point treatment to the LMS algorithm: efficient realization and a roundoff error analysis," *IEEE Transactions on Signal Processing*, vol. 53, no. 12, pp. 4536–4544, 2005.

[8] A. Mitra and M. Chakraborty, "The NLMS algorithm in block floating-point format," *IEEE Signal Processing Letters*, vol. 11, no. 3, pp. 301–304, 2004.

[9] A. C. Erickson and B. S. Fagin, "Calculating the FHT in hardware," *IEEE Transactions on Signal Processing*, vol. 40, no. 6, pp. 1341–1353, 1992.

[10] D. Elam and C. Lovescu, "A block floating point implementation for an N-point FFT on the TMS320C55X DSP," Application Report SPRA948, Texas Instruments, Dallas, Tex, USA, September 2003.

[11] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A fast single-chip implementation of 8192 complex point FFT," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 300–305, 1995.

[12] M. Chakraborty and A. Mitra, "A block floating-point realization of the gradient adaptive lattice filter," *IEEE Signal Processing Letters*, vol. 12, no. 4, pp. 265–268, 2005.

[13] B. Farhang-Boroujeny, *Adaptive Filters—Theory and Application*, John Wiley & Sons, Chichester, UK, 1998.

Mrityunjoy Chakraborty obtained Bachelor of engineering from Jadavpur university, Calcutta, in electronics and telecommunication engineering (1983), followed by Master of Technology and Ph.D. degrees both in electrical engineering from IIT, Kanpur (1985) and IIT, Delhi (1994), respectively. He joined IIT, Kharagpur, as a faculty member in 1994, where he currently holds the position of a Professor in electronics and electrical communication engineering. The teaching and research interests of him are in digital and adaptive signal processing, including algorithm, architecture and implementation, VLSI signal processing, and DSP applications in wireless communications. In these areas, he has supervised several graduate theses, carried out independent research, and has several well-cited publications. He has been an Associate Editor of the IEEE Transactions on Circuits and Systems I during 2004–2005 and also during 2006–2007, he is a Guest Editor for an upcoming special issue of the EURASIP JASP on *distributed space time systems*, has been in the technical committee of many top-ranking international conferences, and has visited many well-known universities overseas on invitation. He is a fellow of the IETE and a Senior Member of IEEE.

Rafiahamed Shaik was born in Mogallur, India, in 1970. He received the B.Tech. and M.Tech. degrees from Sri Venkateswara University, Tirupati, India, in 1991 and 1993, respectively. He is currently working towards the Ph.D. degree at the Indian Institute of Technology, Kharagpur, India, all in electronics and communication engineering. From 1993 to 1995, he has been a faculty member at Deccan College of Engineering and Technology, Hyderabad, India, and from 1995 to 2003 at Bapatla Engineering College, Bapatla, India. His teaching and research interests are in digital and adaptive signal processing, signal processing for communication, microprocessor-based system design, and VLSI signal processing.

Moon Ho Lee received the Ph.D. degrees in electronic engineering from the Chonnam National University, Korea (1984) and the University of Tokyo, Japan (1990). From 1970 to 1980, he was a Chief Engineer with Namyang Moonhwa Broadcasting Corp., Korea. Since 1980, he has been a Professor with the Department of Information and Communication at Chonbuk National University. From 1985 to 1986, he was also with the University of Minnesota, as a Postdoctoral Research Fellow. He has held visiting positions with the University of Hannover (1990), the University of Aachen (1992, 1996), the University of Munich (1998), the University of Kaiserlautern (2001), RMIT (2004), and the University of Wollongong, Australia. He has authored 31 books including *Digital Communication* (Youngil, Korea, 1999), and 1995 SCI international journal papers. His research interests include multidimensional source and channel coding, mobile communication, and heterogeneous network. He is a registered Telecommunication Professional Engineer and a Member of the National Academy of Engineering in Korea. He was the recipient of the Paper Prize Award from the Korean Institute of Communication Science in 1986 and 1997, the Korea Institute of Electronics Engineers in 1987, Chonbuk Province in 1992, and Commendation of Prime Minister, for basic research on jacket matrix theory and applications (2002).