Two High Performance Adaptive Filter Implementation Schemes Using Distributed Arithmetic

Rui Guo and Linda S. DeBrunner

Abstract—Distributed arithmetic (DA) is performed to design bit-level architectures for vector-vector multiplication with a direct application for the implementation of convolution, which is necessary for digital filters. In this work, two novel DA based implementation schemes are proposed for adaptive FIR filters. Different from conventional DA techniques, our proposed schemes use coefficients as addresses to access a series of look-up tables (LUTs) storing sums of delayed and scaled input samples. Two smart LUT updating methods are developed, and least mean square (LMS) adaptation is performed to update the weights, and minimize the mean square error between the estimated and desired output. Results show that our two high performance designs achieve high-speed, low computation complexities, and low area costs.

Index Terms—adaptive filter, distributed arithmetic (DA), finite impulse response (FIR), least mean square (LMS), look-up table (LUT), multiply-accumulate (MAC), offset-binary coding (OBC).

I. INTRODUCTION

Most portable electronic devices, such as cellular phones, PDAs, and hearing aids, require digital signal processing (DSP) for high performance. Due to the increased demand of implementation of sophisticated DSP algorithms, low-cost designs, i.e., low-area and low-power cost, are needed to make these hand-held devices small with good performance.

Various types of DSP operations are employed in practice. Filtering is one of the most widely used complex signal processing operations [1]. For discrete finite impulse response (FIR) filters, the output $y(n)$ is a linear convolution of weights $w_n$, and inputs. For an $N$-order FIR filter, the generation of each output sample $y(n)$ takes $N + 1$ multiply-accumulate (MAC) operations. Since general purpose multipliers require significant chip area, alternate methods of implementing multiplication are often used particularly when the coefficients values are known prior to implementation. Distributed arithmetic (DA) is one way to implement convolution multiplierless, where the MAC operations are replaced by a series of LUT accesses and summations. Techniques, such as ROM decomposition [2] and offset-binary coding (OBC) [7] can reduce the LUT size, which would otherwise increase exponentially with the filter length $N + 1$ for conventional DA.

However, in many applications, such as echo cancellation and system identification, coefficient adaptation is needed. This adaptation makes it challenging to implement DA-based adaptive filters with low-cost due to the necessity of updating of LUTs. Several approaches have been developed for DA-based adaptive filters, from the point of view of reducing logic complexity [3]–[6], [8]. Recently, a DA-based FIR adaptive filter implementation scheme is presented in [5], [6], [8], which uses extra “auxiliary” LUTs to help in the updating; however, memory usage is doubled.

In this paper, two novel LMS adaptation based DA implementation schemes are proposed for FIR adaptive filter implementation. The first proposed algorithm updates the LUTs in a similar way as described in [5], [6], [8] but without the need for auxiliary LUTs. The second proposed algorithm incorporates an OBC-based LUT updating scheme that further reduces memory usage. It is shown that our two proposed schemes both outperform that described in [5], [6], [8], with the second proposed algorithm requiring less memory usage, but more computation cost than our first proposed algorithm.

This brief is organized as follows. Section II describes the background of DA and OBC. Then, we present our proposed schemes for DA-based FIR adaptive filter in Section III. A performance comparison of different DA-based implementations is made in Section IV. Our conclusions are given in Section V.

II. BACKGROUND

A. Distributed Arithmetic (DA)

DA was first studied by Croisier [9] in 1973, and popularized by Peled and Liu [10]. Quantization effects in DA system were analyzed by Kammeyer [11] and Taylor [12]. Useful tutorials on DA were provided by White [7] and Kammeyer [13]. DA is used to design bit-level architecture for vector multiplications [2]. Traditionally, for filters implemented using DA, the input samples are used as addresses to access a series of LUTs whose entries are sums of coefficients. Consider a discrete $N$-order FIR filter with constant coefficients, and input samples coded as $B$-bit two’s complement numbers with only the sign bit to the left of the binary point

$$x(n - k) = -x_{k0} + \sum_{j=1}^{B-1} x_{kj}2^{-j}$$  \hspace{1cm} (1)

Using (1) to compute the FIR output gives

$$y(n) = -\sum_{k=0}^{N} w_k x_{k0} + \sum_{j=1}^{B-1} \left[ \sum_{k=0}^{N} w_k x_{kj} \right] 2^{-j}$$  \hspace{1cm} (2)

With $C_j = \sum_{k=0}^{N} w_k x_{kj}$, $\forall j \in [1, B - 1]$, and $C_0 = -\sum_{k=0}^{N} w_k x_{k0}$, (2) can be rewritten as

$$y(n) = \sum_{j=0}^{B-1} C_j 2^{-j}$$  \hspace{1cm} (3)

This brief is organized as follows. Section II describes the background of DA and OBC. Then, we present our proposed schemes for DA-based FIR adaptive filter in Section III. A performance comparison of different DA-based implementations is made in Section IV. Our conclusions are given in Section V.
\[ y(n) = \sum_{j=0}^{B-1} C_j 2^{-j} \]  

(3)

The \( C_j \) values can be precomputed and stored in a LUT with the input used as the address. This technique allows the FIR filter with known coefficients to be implemented without general purpose multipliers. This implementation requires a LUT with a size that increases exponentially with the number of taps \( N + 1 \), which results in a large time cost for accessing the LUT for a high order filter. So, reducing the LUT size improves system performance, as well as area cost. One possible way to reduce LUT size, called ROM decomposition, replaces a longer address by shorter addresses, and the data read from smaller LUTs is accumulated to generate the output. For a 64-tap FIR filter, by breaking the LUT with \( 2^{64} \) entries into smaller LUTs with 4-bit addresses, only \( \frac{64}{4} \times 2^4 = 2^8 \) entries are required.

### B. Offset-Binary Coding (OBC)

OBC can be used to reduce the LUT size by a factor of 2 to \( 2^{N-1} \) [7]. By rewriting the input from (1), OBC is derived as follows.

\[ x(n-k) = \frac{1}{2} \{ x(n-k) - \lfloor -x(n-k) \rfloor \} \]  

(4)

\[-x(n-k) = -x_{k0} + \sum_{j=1}^{B-1} x_{kj} 2^{-j} + 2^{-(B-1)} \]  

(5)

Substituting (1) and (5) into (4),

\[ x(n-k) = \frac{1}{2} \{ -(x_{k0} - x_{k0}) + \sum_{j=1}^{B-1} (x_{kj} - x_{k0}) 2^{-j} - 2^{-(B-1)} \} \]  

(6)

By defining \( D_{kj} \) as \( x_{kj} - x_{k0} \), the output from FIR filter can be written as:

\[ y(n) = \sum_{k=0}^{N} \frac{w_k}{2} \left[ -D_{k0} + \sum_{j=1}^{B-1} D_{kj} 2^{-j} - 2^{-(B-1)} \right] = -\sum_{k=0}^{N} \frac{w_k D_{k0}}{2} + \sum_{j=1}^{B-1} \sum_{k=0}^{N} \frac{w_k D_{kj}}{2} 2^{-j} - \sum_{k=0}^{N} \frac{w_k 2^{-j}}{2} - 2^{-(B-1)} \]  

(7)

Defining \( E_j \) as \( \sum_{k=0}^{N} w_k D_{kj} \), and \( E_{\text{extra}} \) as \( \sum_{k=0}^{N} \frac{w_k}{2} \), (7) can be rewritten as

\[ y(n) = -E_0 + \sum_{j=1}^{B-1} E_j 2^{-j} - E_{\text{extra}} 2^{-(B-1)} \]  

(8)

The OBC scheme is described in (4)–(8). The LUT contents for a 4-tap FIR filter are given in Table I. It can be observed that the first and second half of this LUT are mirrored vertically. Therefore, its size can be halved by using \( x_{0j} \) to control the sign of each entry at the cost of a slightly increased hardware complexity. The hardware circuit for implementing a \( K \)-tap filter is shown in Fig. 1.

<table>
<thead>
<tr>
<th>( x_{0j} )</th>
<th>( x_{1j} )</th>
<th>( x_{2j} )</th>
<th>( x_{3j} )</th>
<th>LUT Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1/2(w_0 + w_1 + w_2 + w_3)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1/2(w_0 + w_2 - w_3)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1/2(w_0 + w_1 - w_2 + w_3)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1/2(w_0 - w_2 + w_3)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1/2(w_0 - w_1 - w_2 + w_3)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1/2(w_0 - w_1 + w_2 - w_3)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1/2(w_0 - w_1 + w_2 + w_3)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1/2(w_0 + w_1 - w_2 - w_3)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1/2(w_0 + w_1 + w_2 + w_3)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1/2(w_0 + w_1 + w_2 - w_3)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1/2(w_0 + w_1 + w_2 - w_3)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1/2(w_0 + w_1 + w_2 + w_3)</td>
</tr>
</tbody>
</table>

Figure 1. DA based bit-serial architecture for implementing \( K \)-tap FIR filter with OBC

### III. PROPOSED SCHEMES

For an FIR filter with LMS adaptation, which involves the automatic update of filter weights in accordance with the estimation error, conventional DA performance suffers from the intensive computation required to rebuild LUTs. Work has been done in [3]–[6], [8], [14], [15] to reduce the computation workload for LUT updating by only recomputing a few LUT entries. In [5], [6], [8], the authors proposed an efficient LUT updating method that uses auxiliary LUTs, where only half of the entries need to be recomputed. The techniques proposed in this paper eliminate the need for auxiliary LUTs. Our first scheme uses a similar LUT updating method to [5], [6], [8], but with no need of auxiliary LUTs, since our proposed scheme stores the sums of delayed inputs in LUTs. In our second scheme, OBC is incorporated, and a new updating method is introduced to further reduce the memory usage. Unlike [15], the algorithms proposed in this paper implement the coefficient updating and filter operations concurrently.

#### A. First Proposed Scheme

Conventional DA stores the sums of weights (coefficients) in LUTs and uses the inputs as addresses. This approach works very well for nonadaptive filters with constant coefficients. However, for adaptive filters, adaptation is necessary for the weights and the input registers must be updated. By exploiting the commutative property of convolution, the same filtering operation can be obtained by storing sums of delayed input
samples in LUTs, and using the binary coefficients as addresses. With the inputs and coefficients having a common bit-width, as is often the case, this change does not increase latency; however, computation and memory cost are reduced for a bit-serial DA design. If the inputs and coefficients have different lengths, there is some difference in latency, which we have considered in Section IV. Representing the weights \( w_k \) in two’s complement form using \( B \) bits gives

\[
w_k = -w_{k0} + \sum_{j=1}^{B-1} w_{kj}2^{-j} \quad (9)
\]

Using (9) to calculate the output yields

\[
y(n) = -\sum_{k=0}^{N} x(n-k)w_{k0} + \sum_{j=1}^{B-1} \left[ \sum_{k=0}^{N} x(n-k)w_{kj} \right] 2^{-j} \quad (10)
\]

Similar to (2), the term in square brackets has only \( 2^{N+1} \) possible values, so a LUT can be used. The left table shown in Fig. 2 gives the LUT values for a 4-tap FIR filter. When the time index \( t = n + 1 \), (10) becomes,

\[
y(n+1) = -\sum_{k=0}^{N} x(n-k+1)w_{k0} + \sum_{j=1}^{B-1} \left[ \sum_{k=0}^{N} x(n-k+1)w_{kj} \right] 2^{-j} \quad (11)
\]

Fig. 2 shows graphically how the LUTs can be updated. Specifically, it can be observed from the term in square brackets that the new input sample \( x(n+1) \) is not used for the new entries, whose least significant address bit (LSAB) \( w_{0j} \) is 0. Since all the combinations of inputs \( x(n), x(n - 1) \ldots x(n - N) \) are included in the old LUT at \( t = n \), these new entries with LSAB being 0 can be obtained directly by copying the corresponding entries from the old LUT, as indicated by the arrows in Fig. 2. A closer observation discloses that each of the rest of the new entries with LSAB being 1 can be generated by adding \( x(n+1) \) to the prior entry in the new LUT. Mathematically, the new entries \( T_i(n+1) \) can be obtained from the old entries \( T_i(n) \) by (12) and (13) with the entry index \( i \in [0, 2^{N+1} - 1] \).

\[
T_i(n+1) = T_{i} (n), \forall i \in \{i|i \mod 2 = 0 \} \quad (12)
\]

\[
T_i(n+1) = T_{i-1} (n+1) + x(n+1), \forall i \in \{i|i \mod 2 = 1 \} \quad (13)
\]

In this paper, LMS adaptation is chosen to update the weights \( w_k \), as shown in (14), and (15).

\[
w_k(n+1) = w_k(n) + \mu e(n)x(n-k) \quad (14)
\]

\[
e(n) = d(n) - \mathbf{w}(\mathbf{x}) \quad (15)
\]

where \( d(n) \) is the desired output, \( \mathbf{w}(\mathbf{x}) = [w_0(n) \ w_1(n) \ldots \ w_N(n)] \), \( \mathbf{x}(n) = [x(n) \ x(n-1) \ldots x(n-N)]^T \), and \( e(n) \) is the error between the desired and estimated output.

The top-level circuit diagram of this proposed scheme for an example 4-tap FIR filter is shown in Fig. 3. The method for updating the DA_LUT block in our first scheme is similar to that used for updating the auxiliary LUT in [5], [6], [8], which we refer to as DA0 scheme. For our proposed first scheme, the DA_LUT block could be implemented using a typical DA implementation. Fig. 4 shows more details of the implementation. The Addr Gen block has to generate the addresses in the order as shown in Fig. 4. As shown in the

Figure 2. LUT update from \( t = n \) to \( t = n + 1 \), modified from [6]

Figure 3. Top-level circuit diagram for 4-tap adaptive FIR filter

Figure 4. Detailed DA_LUT block for 4-tap adaptive FIR filter
weights update block in Fig. 3, the multiplication from (14) can be simplified as shifting by scaling the estimation error and assuming the step size to be a power of two. In contrast to \(DA_0\), our scheme uses no auxiliary LUTs, but only main LUTs. The updating of LUTs and weights used as addresses can be performed concurrently, which reduces latency. In \(DA_0\), two types of LUTs are necessary: the auxiliary LUTs need to be updated first, and then the updates of the main LUTs are executed. To compare our scheme with \(DA_0\), synthesis results from 0.18\(\mu\)m standard cell library for implementing FIR filters with 8-bit inputs and weights are presented in Fig. 5 (a). It is shown that since extra auxiliary LUTs are necessary for the \(DA_0\) scheme, significant area savings can be achieved by our proposed scheme. Similar results are obtained in Fig. 5 (b) by implementing an 8-tap FIR filter using FPGA Stratix II EP2S15F672I4. In addition to the area advantage shown in Fig. 5, our proposed algorithm also has an advantage of reduced latency.

B. Second Proposed Scheme

In Section II-B, OBC is shown to reduce the number of LUT entries without increasing the number of LUTs required. In this section, we propose a new scheme that combines OBC with our first proposed scheme. Because of the commutative property, as in our first proposed approach, the sums of delayed input samples are stored in LUTs coded using OBC with binary coefficients as the address, as derived in (16)–(19). (17) indicates that, with \(w_{kj}\) as the address, LUTs can still be used to store \(F_j\), as shown on the left of Fig. 6 for a 4-tap FIR filter. Although the size of LUTs is reduced by applying OBC, the LUT updating still suffers from high computation cost, since the oldest sample is included in every entry, e.g., \(x(n - 3)\) in Fig. 6. The second entry with address 001 at time \(t = n\) needs to be updated by adding \(\frac{x(n - 3) - 2x(n - 2) + x(n + 1)}{2}\). The rest of the entries need to be updated with approximately the same computation cost.

\[
w_k = -w_{k0} + \sum_{j=1}^{B-1} w_{kj}2^{-j} \tag{16}
\]

\[
F_j = \sum_{k=0}^{N} \frac{x(n-k)(w_{kj} - w_{k0})}{2} \tag{17}
\]

To reduce the computation workload, we propose a smart updating algorithm. Fig. 6 shows how the update works for a 4-tap FIR filter, with precomputed \(T_1 = \frac{x(n+1)+x(n-3)}{2}\). For instance, the entry with address 000 on the left for \(t = n\) is used to update the entry with address 111 on the right as follows.

\[
T_1 = \frac{1}{2}[x(n) + x(n - 1) + x(n - 2) + x(n - 3)]
\]

\[
y(n) = -F_0 + \sum_{j=1}^{B-1} F_j 2^{-j} - F_{extra} 2^{-(B-1)} \tag{19}
\]

To reduce the computation workload, we propose a new scheme that combines OBC with our first proposed scheme. Because of the commutative property, as in our first proposed approach, the sums of delayed input samples are stored in LUTs coded using OBC with binary coefficients as the address, as derived in (16)–(19). (17) indicates that, with \(w_{kj}\) as the address, LUTs can still be used to store \(F_j\), as shown on the left of Fig. 6 for a 4-tap FIR filter. Although the size of LUTs is reduced by applying OBC, the LUT updating still suffers from high computation cost, since the oldest sample is included in every entry, e.g., \(x(n - 3)\) in Fig. 6. The second entry with address 001 at time \(t = n\) needs to be updated by adding \(\frac{x(n - 3) - 2x(n - 2) + x(n + 1)}{2}\). The rest of the entries need to be updated with approximately the same computation cost.

\[
w_k = -w_{k0} + \sum_{j=1}^{B-1} w_{kj}2^{-j} \tag{16}
\]

\[
F_j = \sum_{k=0}^{N} \frac{x(n-k)(w_{kj} - w_{k0})}{2} \tag{17}
\]

To update \(F_{extra}\), we could add together all of the first entries from the sub-LUTs used by ROM decomposition, which requires \(num - 1\) additions, where \(num\) is the number of sub-LUTs. Another method is by subtracting half of the oldest input sample from \(F_{extra}\), and adding half of the newest sample to \(F_{extra}\), which requires 2 operations every cycle, since division by 2 can be realized by right shifting. The latter method is chosen to update \(F_{extra}\) for this proposed scheme. Performances for the different schemes are compared in Section IV.

IV. PERFORMANCE COMPARISON

In this section, formulas for certain critical measurements are derived to compare the performances of our proposed two schemes with \(DA_0\). Since these three implementation schemes have a similar clock rate, the memory usage and computation cost for each scheme are derived and compared. The memory usage is measured by the numbers of LUT entries. If step size is assumed and estimation error is scaled to be a power-of-two to simplify multiplication as a shift, the only critical computation used for these three schemes is addition. The calculation cost is estimated as the number of necessary additions per filter cycle, including LUT updating and data filtering.

The memory usages for \(DA_0\), and our two proposed schemes, \(M_{DA_0}\), \(M_{proposed1}\), and \(M_{proposed2}\), are estimated:

\[
M_{DA_0} = (2^m - 1)(\frac{K}{m}) \cdot 2 \tag{21}
\]
where $K$ is the filter length and $m$ is the number of bits required for the LUT address when ROM decomposition is used. $K$ and $m$ are assumed to be powers of two. Since our proposed first scheme does not use any auxiliary LUTs, its memory usage is exactly half of $DA_0$’s. In our second proposed scheme, the memory usage is further reduced by using OBC for the LUTs. With OBC coded LUTs, our second proposed scheme requires the least memory usage, which is less than 30% of that by $DA_0$, since $\frac{M_{\text{proposed2}}}{M_{DA_0}} = \frac{2^3}{(2^{3m-1})} = 26.6\%$. If the coefficients and inputs have different lengths, then the savings ratio will vary somewhat.

If $W$ and $B$ are the bit-width of inputs and coefficients, respectively, then the numbers of necessary additions every filter cycle for these schemes are estimated:

$$A_{DA_0} = 2^{m-1} \cdot K + 2^m \cdot \frac{K}{m} + \left( \frac{K}{m} \right) \cdot W - 1$$  \hspace{1cm} (24)

$$A_{\text{proposed1}} = 2^{m-1} \cdot \frac{K}{m} + K \cdot \left( \frac{K}{m} \right) \cdot B - 1$$  \hspace{1cm} (25)

$$A_{\text{proposed2}} = (2^{m-1} + 1) \cdot \frac{K}{m} + K \cdot \left( \frac{K}{m} \right) \cdot B + 1$$  \hspace{1cm} (26)

The three addends in (24)–(26) count the additions for updating the LUTs, coefficients, and summing up all the entries read from LUTs, respectively. To examine the effects of changing the ratio of the input width $W$ and the coefficient width $B$, we plot the savings of additions for different values in Fig. 7.

It is shown that our two proposed schemes require less addition costs than $DA_0$, while the proposed second scheme needs slightly more additions than our first proposed approach, which is due to the precomputation of $T_1$, and updating of $F_{\text{extra}}$ every cycle.

V. CONCLUSION

In this work, two different DA based schemes are presented for FIR adaptive filter implementation. In contrast to conventional DA based schemes, our schemes store the sums of delayed input samples in LUTs and use the binary coefficients as addresses. It is shown that since no auxiliary LUTs are required, our first proposed scheme needs exactly half the memory usage required by previous work, while the second proposed scheme only needs less than 30% of that required by previous work. In addition, our two proposed schemes both have low computation cost, with the second scheme requiring slightly more addition operations than the first one. Unlike previous work, in our schemes, the updating of LUTs and coefficients can be executed concurrently, which enables low latency.

REFERENCES