Optimized Architecture for an Adaptive FIR Filter using Distributed Arithmetic

Sathyabhama B1, Sahaana Harishankar2, Preetha3
1 Assistant Professor, Department of Electronics and Communication, Panimalar Engineering College, Chennai, India
2 Sahaana Harishankar, Department of Electronics and Communication, Panimalar Engineering College, Chennai, India
3 Preetha, Department of Electronics and Communication, Panimalar Engineering College, Chennai, India

Abstract — the architecture is based on distributed arithmetic in which the partial products of filter coefficients are precomputed. These coefficients are stored in lookup tables (LUTs) and the weighted coefficients are updated using Offset Binary Coding (OBC). The filtering is done by shift and accumulate operations on these partial products. Thus, it results in lesser area comparatively and this technique can be applied to all FIR based adaptive filter application. Results are obtained using modelsim and altera quartus tool.

Index Terms— adaptive filter, Distributed Arithmetic (DA), finite impulse response (FIR), least Mean Square (LMS), lookup table (LUT), Offset Binary Coding (OBC).

I. INTRODUCTION

Adaptive filters are widely used in many signal processing, applications like system identification and modeling, equalization, interference and echo cancellation. However, in many applications such as echo cancellation and system identification, coefficient adaptation is needed. Such filters are generally made of Finite Impulse Response (FIR) filters whose coefficients are updated as per a minimization criterion. The output of a FIR filter is the weighted sum of present and past input samples and, hence they can be realized using multiply and accumulate (MAC) units. If N is the number of filter taps, a single MAC unit would take N clock cycles to produce one sample of output sequence. Although, multiple MAC units can be utilized in order to increase the speed of the system, the system cost increases as the multipliers consume more area.

Distributed Arithmetic (DA) is one of the efficient techniques to realize the higher order filters and it can achieve the high throughputs without using hardware multipliers. Distributed Arithmetic is essentially a bit serial operation that produces the sum of products in a fixed number of clock cycles regardless of the number of products to be summed up. The basic idea behind DA is that, the pre-computed partial products are stored in look up tables (LUT), accessing and shift-accumulating. The right ones will generate the output. Furthermore, the multiplier less architecture of DA makes it the most effective one to realize the higher order filters. A fixed-coefficient filter can be easily realized using DA by storing the partial products of filter coefficients in the LUT. However, the DA treatment to adaptive filters will face certain difficulties not encountered in the fixed-coefficient case namely, the two principal operations – filtering and weight updating are manually coupled and so the partial products of the filter coefficients stored in the LUTs are recalculated before the filtering operation. However, few attempts have been made to realize adaptive filters using DA by approximations to standard adaptation algorithms which will degrade the performance.

II. FIR FILTER DESIGN

A. DA (DISTRIBUTED ARITHMETIC) based FIR Filter

Memory based structures are well-suited for many digital signal processing (DSP) algorithms, which involve multiplication with a fixed set of coefficients. For this Distributed Arithmetic architecture is used in our FIR filter design along with the technique of Offset Binary Coding (OBC). DA is an efficient technique for calculation of sum of products or vector dot product or inner product or multiply and accumulate (MAC). MAC operation is very common in all digital signal processing algorithms. This technique consists of accumulators, look up tables and shift registers. According to this technique, all of the multipliers are totally removed and are replaced by array multipliers. The LUTs can be subdivided into many LUTs so that the size of the higher order filters can be reduced. In DA, the cumulative partial products are precomputed and stored in a look up table (LUT) that is addressed by multiplier bits. The output y[n] of an N-tap fir filter with the present input sample x[n] at any instant n is given by

\[ y[n]=\sum_{i=0}^{N-1} w[i]x[n-i] \]
whose values affect how $y(n)$ is computed. Except for the fact that it contains adjustable parameters whose values affect how $y(n)$ is computed.

Thus for the given set of $w_i$ (i=0,1,…,N-1) the terms $c_{B-1-j}$

Would take only one out of $2^N$ possible combinational, which can be precomputed and stored in LUT. The DA implementation of a four-tap FIR filter is shown in Fig.1. The incoming bits of input samples are stored in the registers in the order that, at any instant the bits of the most recent input sample are stored in the top most register while the bits of the oldest sample are stored in the bottommost register. The least significant bits (LSBs) from each of the registers from the address lines to the LUT containing the partial products. The partial products are then shifted and accumulated for “B” number of clock cycles to produce one sample of the output.

III. ADAPTIVE FILTERING TECHNIQUE

The diagram below shows a block diagram in which a sample from a digital input signal $x(n)$ is fed into a device, called an adaptive filter, that computes a corresponding output signal sample $y(n)$ at time $n$. For the moment, the structure of the adaptive filter is not important, except for the fact that it contains adjustable parameters whose values affect how $y(n)$ is computed.

The output signal is compared to a second signal $d(n)$ called the desired response signal, by subtracting the two samples at time $n$. This difference signal, given by

$$e(n) = d(n) - y(n)$$

is known as the error signal. The error signal is fed into a procedure which alters or adapts the parameters of the filter from time $n$ to time $(n+1)$ in a well-defined manner. This process of adaptation is represented by the oblique arrow that pierces the adaptive filter block in the figure. As the time index $n$ is incremented, it is hoped that the output of the adaptive filter becomes a better and better match to the desired response signal through this adaptation process, such that the magnitude of $e(n)$ decreases over time. In this context, what is meant by “better” is specified by the form of the adaptive algorithm used to adjust the parameters of the adaptive filter.

In the adaptive filtering task, adaptation refers to the method by which the parameters of the system are changed from time index $n$ to time index $(n+1)$. The number and types of parameters within this system depend on the computational structure chosen for the system.

A. DA-Based FIR Filter With the OBC Technique

The ROM size in the before mentioned block can be further reduced using the OBC technique [5] as follows. By rewriting (2) as

$$x_{n-i} = \frac{1}{2} [x_{n-i} - (-x_{n-i})]$$

Fig.3. DA based implementation of four tap fir filter using the OBC scheme

Choose

$$d_{j,j} = \begin{cases} -b_{i,j} - b_{i,j} & j \neq B - 1 \\ -b_{i,j} - b_{i,B-j} & j = B - 1 \end{cases}$$

By substituting (4) and (5) into (1) and rearranging we got

Define

$$p_j = \frac{1}{2} \sum_{i=0}^{N-1} w_i d_{i,j}, \quad 0 \leq j \leq B - 1$$
\[ P_{\text{initial}} = -\frac{1}{2} \sum_{i=0}^{N-1} w_i \] (6)

We get this

\[ y[n] = \sum_{j=0}^{B-1} P_{B-1-j} 2^{-j} + p_{\text{initial}} 2^{-(B-1)} \] (7)

Now, for the given set of \( w_i (i=0,1,\ldots,N-1) \), the terms PB-1-J would take one out of \( 2^n \) combinations, half of which would be the mirror image of the other half[4]. Hence, a \( 2^{N-1} \) sized ROM can be used, the address of which can be obtained through the EXOR operation of all the LSBs with the LSB of the newest sample. Considering an adaptive filter that processes an input signal \( x(n) \) and generates an output signal \( y(n) \) as

\[ Y(n) = w^T x \] (8)

Where \( w^T = [w_0(n), w_1(n), \ldots, w_{N-1}(n)] \) is the coefficient vector and \( x^T = [x(n), x(n-1), \ldots, x(n-N+1)] \) is the input sample vector, where \( N \) is the number of filter coefficients. If the least mean square (LMS) algorithm is chosen, then each of the weights \( w_k (k=0,1,\ldots,N-1) \) can be updated using

\[ w_k(n+1) = w_k(n) + \mu e(n)x(n-k) \] (9)

Where \( e(n) = d(n) - y(n) \) is the error signal and \( \mu \) is the approximate step size.

IV. PROPOSED DA–BLOCK FOR THE ADAPTIVE FILTER

The block schematic of the proposed DA-architecture for efficient implementation of the filtering and weight update operations consists of a register bank to store the incoming input samples, a primary LUT (P-LUT) that stores the combinations of weights that is responsible for the DA filtering operation in every iteration and also a secondary LUT (S-LUT) that will store the combination of input samples. It also consists of a register \( R_0 \) that along with S - LUT aids the weight adaptation process, a shift - and – accumulate block of the DA filtering operation, and a combinational logic block that takes care of the weight adaptation process.
If the vector \( w \) is replaced by the vector \( x \), which is represented as \( x = \{x(n), x(n-1), ..., x(n-N+1)\} \), then a new LUT \( T(n) \) is formed with its entry at address location \( a \) given as...

\[
T_{(a)}(n) = \frac{1}{2} x(n) + \sum_{k=1}^{N-1} x(n-k)(-1)^{q_{N-1}^{(a)+1}},
\]

\[a=0, 1, ..., 2^{N/2}-1\]

(12)

Where \( R_0(n) \) denotes the contents of the register \( R_0 \) at time \( n \) and \( S_{(a)}(n) \) is the entry of the S-LUT at time \( n \) addressed by \( a \).

\[
R_0(n) = \frac{1}{2} x(n)
\]

(13)

\[
S_{(a)}(n) = \sum_{k=0}^{N-2} x(n-k)(-1)^{q_{N-1}^{(a)+1}}
\]

(15)

In Fig-4, the contents of P-LUT and S-LUT at time instant \( n \) for a four-tap filter are shown. It can be observed that S-LUT stores the OBC combinations (lower half) of input samples except for the sample \( x(n) \). The S-LUT update scheme averaging the S-LUT entry with its next consecutive entry would generate a term that is independent of the oldest input sample. The content of register \( R_0 \) is then subtracted and added with the result. For example, when \( N=4 \), the average of the first and second entries of S-LUT would give

\[
\frac{1}{2}[-x(n-1) - x(n-2)],
\]

which is independent of the third term \( x[n-3] \). Now the subtraction and addition of the term of \( x[n] \) with the result will generate the terms of \( [-x(n-1) - x(n-3)] \) and \( \frac{1}{2}(x(n) - x(n-1) - x(n-2)) \), respectively, which are then stored in the same consecutive locations of S-LUT. Similarly, the third and fourth locations of S-LUT are updated by subtracting and adding the term \( 1/2 \left \lfloor x(n) \right \rfloor \) to their average and storing the difference and sum in exactly the same locations. In a similar way, all the entries of S-LUT can be updated using the sum and the difference with the term \( 1/2 x(n) \).

If “1” and “0” are the positive and negative terms in the OBC combinations of input samples, at time \( n \), the contents of S-LUT would look like the binary sequence of the input samples, as shown in Fig-6. At time \( n+1 \), the contents would be a circularly right-shifted version of that binary sequence. Similarly at time \( n+2 \), the contents would be a circularly right-shifted version of the binary sequence at time \( n+1 \) and so on. Hence, for accessing the entries of S-LUT in each iteration, instead of physically moving the contents, the address bits to the S-LUT are circularly left-shifted. At time \( n+3 \), the sequence once again would be a normal binary sequence as shown in Fig-6. Accessing time can be reduced by maintaining two similar S-LUTs namely, ODD-LUT and EVEN-LUT as shown in Fig-7, where the even location entries and odd location entries are stored respectively.
the term $P_{initial}$ is nothing but the additive inverse of the entry of P-LUT at its last address location. The algorithm which explains the overall operation of the proposed filter is shown in fig.8

![Fig. 7. Two LUT storing odd and even location](image)

**IV. SIMULATION RESULTS**

The following simulation results are obtained using altera modelsim and synthesized using quartus.

![Fig. 8. Fir filter using array multiplier](image)

![Fig. 9. Fir filter using Distributed Arithmetic](image)

![Fig. 10. Adaptive fir filter using Distributed Arithmetic simulated results.](image)

**SUMMARY RESULTS (TABLE 1)**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ADAPTIVE FIR FILTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA</td>
<td>Logic elements - 1322</td>
</tr>
<tr>
<td>POWER</td>
<td>327.81mw</td>
</tr>
<tr>
<td>TIMING</td>
<td>4.327ns</td>
</tr>
</tbody>
</table>

**SUMMARY RESULTS (TABLE 2)**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>FIR FILTER WITH ARRAY MULTIPLIER</th>
<th>FIR FILTER WITH DISTRIBUTED ARITHMETIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA</td>
<td>Logic elements - 19</td>
<td>Logic elements - 288</td>
</tr>
<tr>
<td>POWER</td>
<td>324.40mw</td>
<td>320.7mw</td>
</tr>
<tr>
<td>TIMING</td>
<td>15.923ns</td>
<td>4.824 ns</td>
</tr>
</tbody>
</table>

**V. CONCLUSION**

In brief, we have presented an FIR adaptive filter implementation based on DA. In order to reduce the memory requirement this has been implemented. Unlike the technique used in [11], hence the proposed technique uses an S-LUT used to adapt weight process as in [10]. The technique used to store complex combination of input samples and the prev
ious input sample is smartly eliminated in order to update S-LUT. Thus, the proposed structure will utilize very less chip area and can operate at higher throughput when compared to other existing blocks.

REFERENCES


