A NOVEL HIGH PERFORMANCE DISTRIBUTED ARITHMETIC ADAPTIVE FILTER IMPLEMENTATION ON AN FPGA

Daniel J. Allred, Heejong Yoo, Venkatesh Krishnan, Walter Huang, and David V. Anderson

Center for Signal and Image Processing,
Georgia Institute of Technology, Atlanta, GA 30332 USA

ABSTRACT

In this paper, an FIR adaptive filter implementation using a multiplier-free architecture is presented. The implementation is based on distributed arithmetic (DA) which substitutes multiply-and-accumulate operations with a series of look-up-table (LUT) accesses. This can be achieved at the cost of a moderate increase in memory usage. The proposed design performs an LMS-type adaptation on a sample-by-sample basis. This is accomplished by an innovative LUT update using a matched auxiliary LUT. The system is implemented on an FPGA that enables rapid prototyping of digital circuits. Implementation results are provided to demonstrate that a high-speed, low logic complexity LMS adaptive filter can be realized employing the proposed architecture.

1. INTRODUCTION

Today's consumer electronics such as PDAs, cellular phones and other multi-media and wireless devices often require digital signal processing (DSP) algorithms for several crucial operations. Due to a growing demand for such complex DSP applications, high performance, low-cost system-on-a-chip implementations of DSP algorithms are receiving increased attention among researchers and design engineers.

Linear filtering is one of the fundamental operations that is typically performed in any DSP system. A discrete-time linear finite impulse response (FIR) filter generates the output $y[n]$ as a sum of delayed and scaled input samples $x[n]$ via the equation

$$y[n] = \sum_{k=0}^{K-1} w_k x[n-k]. \tag{1}$$

A typical digital implementation will require K multiply-and-accumulate (MAC) operations, which are expensive to compute in hardware due to logic complexity, area usage, and throughput. Alternatively, the MAC operations may be replaced by a series of look-up-table (LUT) accesses and summations. Such an implementation of the filter, known as distributed arithmetic (DA), achieves higher throughput and lower logic complexity at the cost of increased memory usage. Recent advances in memory design technology have resulted in shrinking memory sizes, making this tradeoff an attractive option. A good tutorial review of DA linear filters is given in [1].

Many DSP applications require linear filters that can adapt to changes in the input signals. The implementation of an adaptive filter [2] based on the DA concept poses several challenges. Since the DA filtering operation is based on an LUT, changes to the filter can require extensive changes to the LUT. This can be impractical for large LUT sizes. Several past attempts have been made to implement adaptive filters using DA [3] [4], but the approximations made to standard adaptation algorithms may be unsuitable for practical applications.

In this paper, we develop and present an implementation of a discrete-time linear FIR adaptive filter based on the well-known LMS algorithm using the DA concept. A novel approach for updating the LUT tables of the DA filter is presented. The details of the proposed design and a description of the constituent modules of the DA-based LMS adaptive filter is provided in Sec. 3. The implementation results and a brief discussion of design trade-offs are presented in Sec. 4.

2. BACKGROUND CONCEPTS

In this section, brief descriptions of DA digital filters and adaptive filtering algorithms suitable for digital hardware implementations are presented.

2.1. Distributed Arithmetic

DA was first introduced by Croisier et al. [5] and further developed by Peled and Lui [6]. DA provides a multiplier-less implementation of FIR filters through a bit-serial computation utilizing all possible combination sums of the filter coefficeints. It is assumed that the inputs to the filter are represented as B bit 2's complement binary numbers with only the sign bit to the left of the radix point. Then

$$x[n-k] = -b_{k0} + \sum_{l=1}^{B-1} b_{kl} 2^{-l} \tag{2}$$

and Eq. (1) can be rewritten as

$$y[n] = -\left[\sum_{k=0}^{K-1} w_k b_{k0}\right] + \sum_{l=1}^{B-1}\left[\sum_{k=0}^{K-1} w_k b_{kl}\right] 2^{-l}. \tag{3}$$

It is noted that the terms in the square brackets may take only one of 2^K possible values; these values, which are all the possible combination sums of the filter coefficents, are stored in an LUT, denoted as the DA filtering LUT (DA-F-LUT). The filtering operation may then be implemented, according to Eq. 3, by B look-up, shift, and accumulate operations.

The block diagram of a typical DA implementation of a four tap ($K = 4$) filter is shown in Fig. 1. The bank of shift registers in Fig. 1 stores four consecutive input samples. The concatenation of the rightmost bits of the shift registers becomes the address of the LUT. The shift registers are shifted right at every clock cycle. The corresponding LUT entries are also shifted and accumulated B consecutive times where B is the precision of the input data. The

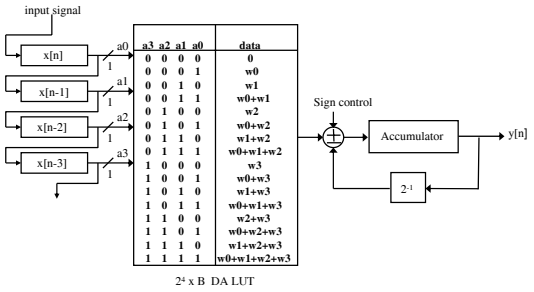

Fig. 1. Block diagram of a four tap ($K = 4$) DA FIR filter. Each coefficient has B bits of precision (ex. B=16).

DA filter can complete the filtering operation in B clock cycles regardless of the size of the filter, K. Thus, a high throughput rate can be obtained using the DA implementation, especially if $K \gg B$.

It must be noted that as the filter size K increases, the memory requirements grow exponentially as 2^K. This problem may be alleviated by breaking up the filter into smaller base DA filtering units that require tractable memory sizes and then summing up the outputs of these units. If the K tap filter is divided into m units of k tap base units ($K = m \times k$), then the total memory requirement would be $m \times 2^k$ memory words. The total number of clock cycles required for this implementation will be $B + \lceil log_2(m) \rceil$; the additional second term is the number of clock cycles required to implement an adder tree to calculate the sums of the units. Thus the decrease in throughput of this implementation is marginal. For instance, if $K = 128$, then instead of 2^{128} in a full LUT implementation, we can choose $k = 4$ and $m = 32$ which would only require 512 memory words. The number of clock cycles required for this implementation would be 21 clock cycles as compared with the full LUT implementation that would require 16 clock cycles.

2.2. LMS Adaptive Filters

An adaptive filter changes its weights w_k with time to match a desired performance objective. Typically, the performance of the adaptive filter is quantified in terms of the mean square value of the error between its output $y[n]$ and a desired signal $d[n]$. The least mean-square (LMS) adaptation algorithm updates the weights to minimize the mean-square error (MSE) of the output. The weight adaptation in an LMS adaptive filter is given by

$$w_k[n+1] = w_k[n] + \mu e[n] x[n-k], \tag{4}$$

where $e[n] = d[n] - y[n]$.

Several approximations of the LMS algorithms are often used for hardware implementations. The sign error LMS (SE-LMS) approximates the error as $e[n] = sgn(d[n] - y[n])$ and the sign data LMS (SD-LMS) replaces the term $x[n-k]$ by $sgn(x[n-k])$. In this paper, an LMS-type algorithm is implemented where the term $\mu e[n]$ is quantized to a power of 2. Implementation results demonstrate that this quantized error LMS (QE-LMS) outperforms the SE-LMS and is comparable to the LMS algorithm in terms of the convergence speed.

3. DA ADAPTIVE FILTER

The LMS adaptation algorithm requires the filter weights $w_k[n]$ to be updated according to Eq. (4) every sample that is filtered. After calculating the updated weights, the entries of the LUT, which are all possible combination sums of the weights, are recalculated and updated. Doing this on a sample-by-sample basis is computationally expensive and time consuming, causing significant reduction in the filter throughput. For example, a brute-force update of the DA-F-LUT could take approximately 1000 clock cycles for a 128 tap FIR filter.

In this paper, a novel adaptation scheme for updating the DA-F-LUT is presented that requires fewer clock cycles than the brute-force LUT recalculation. The overall system level diagram of the proposed implementation is shown in Fig. 2. The DA auxiliary table contains a separate LUT, denoted as the DA auxiliary LUT (DA-A-LUT), which contains all possible combination sums of the K most recent input samples. The table size and structure of the DA-A-LUT are identical to that of the DA-F-LUT. It is this analogous structure that allows the adaptation scheme described below to work efficiently.

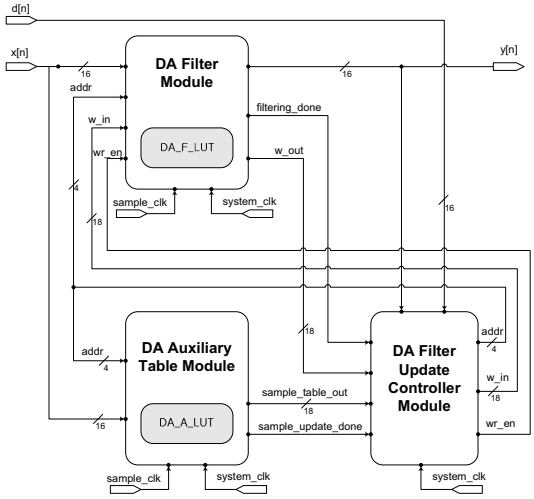

Fig. 2. Proposed DA Adaptive Filter System.

3.1. DA Adaptation Scheme

The brute-force method to update the DA-F-LUT would be to update each new filter weight according to the LMS algorithm described in Section 2, recalculate all possible combination sums of the new weights, and then store the sums in the DA-F-LUT. Instead of updating the weights individually and then using the new weights to regenerate the DA-F-LUT, the proposed method applies the LMS algorithm directly to the contents of the DA-F-LUT. For example, to update the $(2^k - 1)$-th element of the DA-F-LUT, which contains the sum of all k filter weights, the following formula can be used:

$$\sum_{l=0}^{k-1} w_l[n+1] = \sum_{l=0}^{k-1} w_l[n] + \mu e[n] \sum_{l=0}^{k-1} x[n-l]. \tag{5}$$

The DA-F-LUT update then consists of reading the same memory location in both the DA-F-LUT and DA-A-LUT, multiplying the output of the DA-A-LUT by $\mu e[n]$, adding this quantity to the output of the DA-F-LUT, and finally storing the result back in the same memory location of the DA-F-LUT. This process is repeated for addresses 1 to $2^k - 1$ (we can ignore memory location 0 as it will always have a zero value).

Since it is area inefficient to implement $\mu e[n]x[n-k]$ using a hardware multiplier, the following two schemes are considered. The first scheme is the SE-LMS with μ set as a power of two. The multiplication is replaced by a right shift and the result is added or subtracted to the DA-F-LUT output depending on the sign of the error. The second scheme is the QE-LMS, which was mentioned in Sec. 2.2. In this case mu is fixed to some power of two and the error value is quantized to the closest power of two, essentially quantizing the product $\mu e[n]$ to a power of two. Now the magnitude of the error, and not just the sign, plays a role in the rate of adaptation. When the error is large, a large update is applied to the filter weights and faster convergence occurs. When the error is small, a small update is applied thus minimizing the jitter of the weights as they near the optimum values.

3.2. DA Auxiliary Table Module

In this section, the update process of the DA-A-LUT is described. The brute-force method would be to recalculate all possible combination sums of the k most recent samples (after each new sample arrives) and then store the results in the DA-A-LUT. A more efficient update method can be devised by considering how the DA-A-LUT changes as a new sample arrives and the oldest sample is discarded.

Fig. 3 shows the change in the DA-A-LUT from time n to time $n+1$ for $k=4$. The high address locations in the table (addressed by 1000 to 1111) contains sums involving the oldest sample $x[n-3]$, which is the sample being discarded at time $n+1$. The sums in the low address locations of the DA-A-LUT (addressed by 0000 to 0111 at time n) can be reused by mapping these values to the even address locations of the table at time $n+1$, as shown by the arrows in Fig. 3.

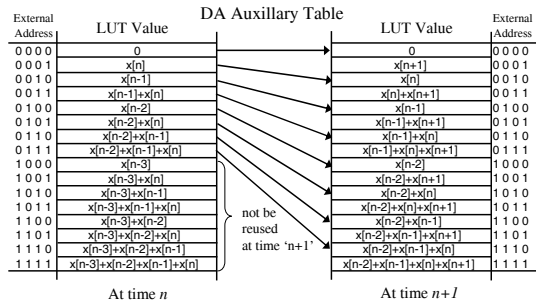

Fig. 3. Change in the content of the DA-A-LUT from time n to time $n+1$.

The values in the odd address locations can then be formed by simply reading the value in the preceding even address location, adding the newest sample $x[n+1]$ and then storing the result at the odd address. This can be represented by

$$\begin{aligned} \text{DA-A-LUT}(2l+1) &= \text{DA-A-LUT}(2l) + x[n+1], \\ & \quad l = 0, \ldots, 2^{k-1}-1. \end{aligned} \tag{6}$$

The low address locations of the DA-A-LUT at time n can be mapped to even address locations at time $n+1$ by a simple rotation of the k address lines. This allows the physical contents of the LUT to remain the same, even as external logic sees the table as shifted. The address rotation from time n to $n+1$ is shown in Table 1. The address line rotation is accomplished via k, k-to-1 multiplexers, whose outputs connect to the DA-A-LUT's internal address and whose select lines are the bits of a counter. This counter is incremented when a new sample arrives, instantaneously remapping the table. The DA auxiliary table module then begins the update of the DA-A-LUT, as described above. When the update is done the *sample_update_done* signal (see Fig. 2) goes high, indicating to the DA filter update controller that the DA-A-LUT is ready to be used in updating the DA-F-LUT.

	Time n	Time $n+1$
Internal Address	External Address	External Address
0000	0000	0000
0001	0001	0010
0010	0010	0100
⋮	⋮	⋮
1101	1101	1011
1110	1110	1101
1111	1111	1111

Table 1. Rotation of address lines from time n to time $n+1$.

3.3. DA Filter Module

The DA FIR filter module performs the filtering operation on the incoming data samples with the current values of the weights. The DA filtering operation has been explained in detail in Sec. 2.1. When filtering is complete, the *filtering_done* signal (see Fig. 2) goes high.

3.4. DA Filter Update Controller Module

The DA filter update controller provides the control logic for the system after the DA-A-LUT is updated and filtering is complete. Once the filtering is done, the error, $e[n]=d[n]-y[n]$, is calculated. The term $\mu e[n]$ is then quantized to the appropriate power of two (see Sec. 3.1). Upon completion of the DA-A-LUT update as described in Sec. 3.2, the update controller accesses the memory locations of the two LUTs and stores the new weight combination sums in the DA-F-LUT.

3.5. Timing Analysis

For a K tap FIR adaptive filter implemented using m base DA filtering units, each of size k, the update of the DA-A-LUT can be done in 2^{k-1} clock cycles as described in Section 3.2. This may be done at the same time (in parallel) as the filtering and adder tree operation, which take a total of $B+\lceil log_2(m) \rceil$ clock cycles. Thus the total number of clock cycles for filtering and updating the DA-A-LUT is $max\big(B+\lceil log_2(m) \rceil, 2^{k-1}\big)$. The updated DA-A-LUT is then used to update the DA-F-LUT; this operation requires 2^k clock cycles. Thus the overall K tap adaptive filter requires $2^k + max\big(B+\lceil log_2(m) \rceil, 2^{k-1}\big)$ clock cycles. The number of clock cycles required for a 128 tap adaptive filter as a function of the size of the base unit k is shown in Fig. 4. It can be observed that the proposed implementation can achieve a much higher throughput than a brute-force implementation if k and m are appropriately chosen.

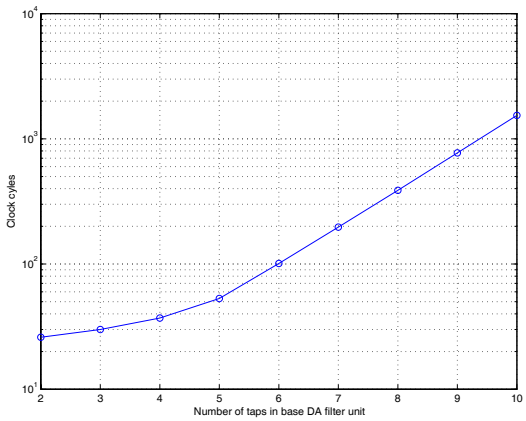

Fig. 4. Number of clock cycles required to filter one time sample as a function of the base DA filter unit size. ($K = 128$)

4. IMPLEMENTATION RESULTS

A DA-based QE-LMS adaptive FIR filter was implemented using using an Altera Stratix FPGA. For illustration purposes, we present the implementation results of a size 32 tap adaptive filter with $k = 4$ and $m = 8$ (to maintain simplicity in the adder tree design, only filters with m as a power of two were considered). White Gaussian noise with zero mean and unit variance was used as the input. The desired signal was generated by filtering the input with an FIR filter whose coefficients are chosen randomly. The number format for the input, desired, and output signal is 2's complement 16 bit Q15. In this implementation $\mu e[n]$ is quantized to four levels.

To contrast the QE-LMS implementation, an SE-LMS FIR filter was designed. Other than the fixed μ for the SE-LMS, the implementation of the two designs are identical. Using the same input and desired signal, the MSE's of each implementation are shown in Fig. 5(a). As illustrated, the QE-LMS filter converges faster than the SE-LMS. These results are also verified by the MATLAB simulation results shown in Fig. 5(b).

For the proposed 32 tap FIR adaptive filter, the entire adaptation may be done in about 37 clock cycles. This is accomplished by employing the DA-A-LUT to avoid the recalculation of all the possible combination sums of the input samples that are required for the weight updates. Thus, compared to the brute-force method of adapting the DA-F-LUT, the proposed method provides significant gains in terms of timing requirements.

5. CONCLUSIONS

A novel multiplier-less implementation of an LMS-type adaptive filter based on distributed arithmetic (DA) has been presented in this paper. The DA concept involves the implementation of a multiply-and-accumulate operation using look-up-tables (LUT). The proposed adaptive filter updates the LUT of all possible combination sums of weights on a sample-by-sample basis using an auxiliary LUT. Such an implementation significantly reduces the number of clock cycles and logic complexity required for the update of the LUT. For the purpose of illustration, a 32 tap DA-based adaptive LMS filter was successfully implemented on an FPGA. It is demonstrated that the system described in this paper may be used for a high-throughput, area efficient implementation of adaptive filters at the cost of a marginal increase in the memory requirements.

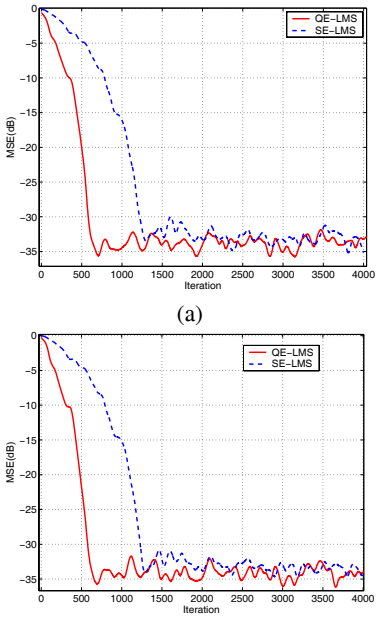

Fig. 5. (a) MSE of 32 tap QE-LMS and SE-LMS filters implemented with Altera's Stratix FPGA chip for $m = 8$ and $k = 4$. (b) MSE of 32 tap QE-LMS and SE-LMS filters simulated in MATLAB.

6. ACKNOWLEDGEMENT

The authors would like to thank Tyson Hall for his valuable discussions and helpful comments.

7. REFERENCES

[1] Stanley A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Magazine*, vol. 6, pp. 4–19, July 1989.

[2] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, Upper Saddle River, NJ, 1996.

[3] C. H. Wei and J. J. Lou, "Multimemory block structure for implementing a digital adaptive filter using distributed arithmetic," *IEE Proceedings*, vol. 133, Pt. G, no. 1, pp. 19–26, February 1986.

[4] C. F. N. Cowan and J. Mavor, "New digital-adaptive filter implementation using distributed-arithmetic techniques," *IEE Proceedings*, vol. 128, Pt. F, no. 4, pp. 225–230, August 1981.

[5] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Rizo, "Digital filter for PCM encoded signals," U.S. Patent No. 3,777,130, issued April, 1973.

[6] A. Peled and B. Lie, "A new hardware realization of digital filters," *IEEE Transactions on A.S.S.P.*, vol. 22, pp. 456–462, December 1974.