2012 International Conference on Information and Computer Applications (ICICA 2012)
IPCSIT vol. 24 (2012) © (2012) IACSIT Press, Singapore

High Performance Architecture for LMS Based Adaptive Filter Using Distributed Arithmetic

Surya Prakash M [+], Rafi Ahamed Shaik

Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati, Guwahati, Assam-781039, India,

Abstract. A new architecture for high throughput implementation of Least Mean Square (LMS) adaptive filter using distributed arithmetic (DA) is presented. DA is a bit-serial computational operation which allows digital filters to be implemented at high throughput rates, regardless of the filter length. However, it poses a problem when implementing adaptive digital filters which requires recalculating the contents of lookup tables (LUTs) that store the filter coefficients. We show that the throughput of the proposed adaptive filter design is the same for filter of any order. Further, we show that the throughput is almost equal to that of the fixed coefficient implementation.

Keywords: Adaptive Filter, Distributed Arithmetic, Least Mean Square (LMS), Look-up-table (LUT).

1. Introduction

Adaptive filters find extensive use in many signal processing applications such as channel equalization, echo processing, noise cancellation etc [5]. Such filters typically involve finite impulse response (FIR) filters whose weights are updated according to some minimizing criteria. FIR filters can be implemented using one or more multiply and accumulate (MAC) units since the output of the filter is the weighted sum of input samples but this would be time consuming and multiple MAC units may increase the system complexity.

Several attempts have been made in order to implement adaptive filters using reduction techniques. A block implementation has been proposed in [3] where the filter weights are updated once for each output block. Later, pipelined architectures [8]-[9] using look-ahead and relaxed look-ahead have been proposed in order to implement high sampling rate adaptive filters. They proved to be effective but the pipelining or parallel implementations either produce adaptation delays or increase the hardware requirement. In order to overcome this, the first frequency domain adaptive filter has been proposed in [6] where a smaller block delay is maintained by using smaller block sizes. In [4] and [7], once again the pipelining technique has been adopted but this time without the adaptation delay and degradation in the convergence performance. With semi-conductor memories becoming more cheaper and much faster since the memory access time is much less than the propagation delay, there is an inclination in the field of research towards the use of semi-conductor memories. Distributed arithmetic [10] is a preferable method since it can realize large filters without hardware multipliers to produce very high throughputs. Few past attempts have been made in order to implement adaptive filters based on DA. Cowan and others [2] tried to implement the same but the approximations made in the process make them impractical. Architecture in [1], proposed by Anderson and others uses auxiliary look-up-tables with special addressing to produce high throughput. The paper is organised as follows. In Section-2, the DA formulation of FIR non-adaptive and adaptive filters is given. Section-3 describes the proposed architecture for DA-based LMS adaptive filter and in the next section; the performance analysis of the proposed design and comparison with the existing implementations is given.

[+] Email: surya@iitg.ac.in. and rafiahamed@iitg.ac.in.

2. DA formulation of fir filter

2.1. Non-adaptive FIR filter

The output y[n] of an *N*-tap FIR filter with each of the input sequence x[n] represented by its *B*-bit *2*'s complement number [1] is given as

$$y[n] = -\left[\sum_{i=0}^{N-1} b_{i,0} w_i\right] + \sum_{p=1}^{B-1}\left[\sum_{i=0}^{N-1} b_{i,p} w_i\right] 2^{-p} \tag{1}$$

The above equation says that for a given set of w_i ($i=0,1,..N-1$) the terms in the square braces can take only 2^N possible values which can be stored in LUT. Hence $y[n]$ now can be computed by accessing the contents of the LUT (the address of which comes from the corresponding bits $b_{i,p}$ of input samples) and adding it with shifted version of the previously accessed content of the LUT. The negative sign in the first square braces of (3) represents the fact that for negative numbers the most significant bit (MSB) would be one and must be subtracted during the computation of $y[n]$. When N is large, DA provides flexibility to use multiple smaller LUTs instead of going for a single large LUT.

2.2. Adaptive FIR filter

A fixed coefficient filter can be implemented using DA efficiently regardless of the filter length. However, problem arises when implementing an adaptive filter whose coefficients are to be adapted which requires that the contents of the LUT are to be updated. For example in-case of a Least Mean Square (LMS) adaptive filter, the weights are updated to minimize the mean squared error ξ according to the update equation

$$w_i[n+1] = w_i[n] + \mu e[n] x[n-i] \tag{2}$$

$d[n]$ being the desired signal and μ being the step size.

In the process, the throughput gain of DA implementation may be eliminated if care is not taken while re-writing the contents of the LUTs. In [1], it is shown that the throughput of the system would remain constant regardless of the filter length for specific values of k. It has been shown that highest throughput values are obtained for $k = 2, 4$ in which case the complexity of the system is high. Choosing $k > 8$ makes it less complex but the throughput of the system is severely limited. Further, it uses auxiliary LUTs with special addressing apart from the LUTs demanded by DA. In the following section, using the framework of [1] we propose a high throughput LMS adaptive filter architecture of low memory based on DA.

3. Proposed Architecture for DA-based LMS Adaptive Filter

The proposed architecture is shown in Fig.1. It uses registers for storing the weights of the filter thereby reducing the storage of redundant partial products in LUTs with the cost of increase in the number of adders. These registers along with the adders are responsible for the generation of partial products. The overall system architecture consists of bank of shift registers to store the bits of input samples, registers to store all the weights of the filter, weight update modules, adder tree and shift and accumulate block. The weight update module shown in Fig.2 consists of a barrel shifter, register, adder and a distributed arithmetic multiplier (DAM) which takes exactly *B* clock cycles assuming all values are taken as *B*-bit. The operation of the filter for one sample of output is as follows:

• Bits of the input samples arriving at a rate equal to the input digital sampling rate are stored into the shift registers which would take exactly $N \times B$ clock cycles.

• Once all the registers are filled, the system clock starts ticking. In the first clock cycle the rightmost bits of all the shift registers are used for the selection of the weights. For example, if the right most bit of the register $x[n-i]$ is *1*, the contents of corresponding register i.e., w_i is transferred to the adder tree. If the bit is *0*, a value of zero is transferred to the adder tree which means that w_i is not required for the computation of the partial product. This can be done by the usage of the AND gates at the output of the weight registers. Similar operation takes place with all the bits of shift registers thereby performing the filtering to produce the output sample $y[n]$ after *B* clock cycles.

• Each of the weight update module which are readily fed with the corresponding input sample performs the weight adjustment as per (2). The product terms $\mu x[n-i], (i = 0,1,2...)$ can be performed using DAMs which operate in parallel with the DA filtering operation. The DAM is basically a DA-module that computes

the sum of products of only one term. Now to perform the product terms $\mu x[n-i]e[n]$ instead of using a traditional hardware multiplier, $e[n]$ is quantized to M levels which are powers of *2* so that the multiplication of $\mu x[n-i]$ with $e[n]$ reduces to shifting the contents of $\mu x[n-i],(i=0,1,2...)$ by a specified amount which can done in one clock cycle using barrel shifters . The correction factors $\mu x[n-i]e[n]$ are then added to $w_i[n+1],(i=0,1,2,...)$ using adders in the weight update modules to produce the new weights $w_i[n+1],(i=0,1,2,...)$. The algorithm describing the operation of the proposed architecture is shown in figure Fig.4.

4. Performance analysis of proposed design

4.1 Throughput Analysis

The throughput comparison of the existing DA implementation and the proposed implementation in terms of the number of clock cycles and the filter order is given in Tab.1. While the existing implementation [1] is throughput efficient for $k=2,4$ and degrades for higher values of *k*, the proposed implementation is a perfect constant for filter of any order which is because of the parallel nature of the system. This is advantageous in high sampling rate applications where high throughput is of major concern. The throughput bar diagram for *8*-tap, *16*-tap and *32*-tap filters in [1] with $k=2,4,8$ and for proposed architecture respectively is shown in Fig.3. It can be seen that the throughput of the proposed design remains constant irrespective of the filter order while the throughput of the existing implementation changes with respect to *k*. The convergence performance of the proposed architecture is same as that of the convergence performance in [1].

Tab.1 Throughput comparison of existing implementation and proposed implementation.

Implementations	**Throughput (Number of clock cycles)**
Existing	$2^k + max(B, 2^{k-1}) + \lceil log_2(m) \rceil$
Proposed	$B+1$

4.2 Complexity Analysis

The proposed design is identical to the architecture in [6] and suffers from one aspect that it demands double the number of adders as there are adders in [14] for the case *k=2*. However, each weight register has its own weight update block and this eliminates the computation time of updating the LUTs as is the case with the existing DA implementation of adaptive filters. Although the number of adders is getting doubled there is a considerable increase in the throughput of the system particularly for higher order filters. Further, it eliminates the usage of auxiliary LUTs which in-turn calls for special addressing control logic. The complexity comparison of the proposed implementation and existing implementation for *k=2* is shown in Tab.2. *k=2* is chosen since the highest throughput values are obtained at this value. It can be seen that the proposed implementation requires only one quarter the number of memory locations as in [14] for *k=2*.

Tab.2 Complexity comparison of the proposed implementation and existing implementation for *k=2*.

Implementations	**Adders**	**2-input AND-gates**	***B*-bit memory locations**
Existing	$N/2$	0	$4N$
Proposed	N	$N \times B$	N

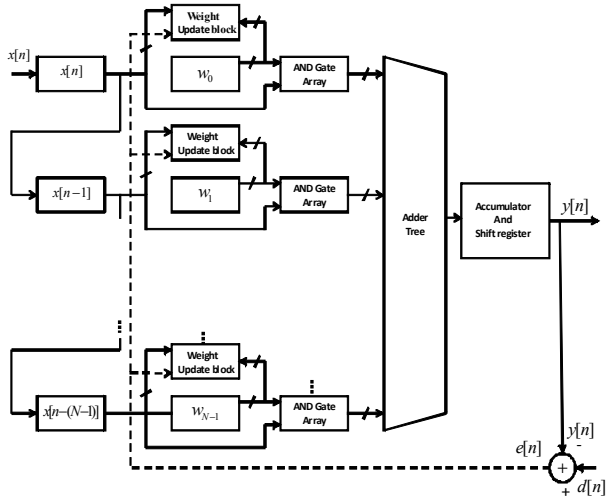

Fig.1 Proposed architecture for DA -based LMS adaptive filter.

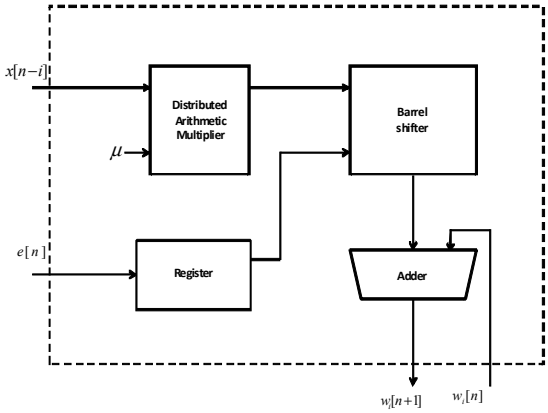

Fig.2 Weight Update module.

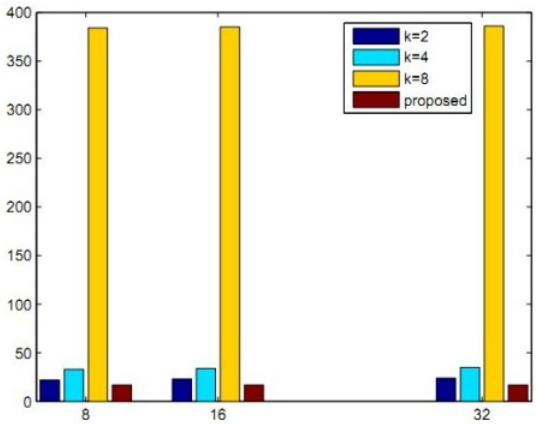

Fig.3 Throughput comparision chart for *8*-tap, *16*-tap and *32*-tap filters in [1] with $k = 2, 4, 8$ and for proposed architecture respectively.

For Continuous Processing

$x[n-i-1]$

label:

for (i=0:B-1)

y = y + addertree_output;

$k = k + b_{\mu, x_{n,i}} 2^{-i}$;

set = 1;

if (set = 1)

$e[n] = d[n] - y[n]$;

$e_{quant}[n] = q * round(e[n]/q)$;

$w_i[n+1] = w_i[n] + k * e_{quant}[n]$;

else

goto label

$n \longleftarrow n+1$

Fig.4 Algorithm describing the operation of the proposed architecture.

5. Conclusion

In this paper, we proposed a high throughput, low memory architecture for LMS adaptive filter based on DA. The proposed implementation uses parallel weight-update blocks thereby reducing the computation time of updating the contents of LUTs as is the case in the existing implementations. This is achieved by the usage of register-adder tree instead of LUTs. This way it utilizes less memory resources and the throughput of the system is a perfect constant regardless of the order of the filter.

6. References

[1] D Allred; W Huang; V Krishnan; H Yoo; DY Anderson. LMS adaptive filters using distributed arithmetic for high throughput. IEEE Transactions on Circuits and Systems, 2005.52, PP1327–1337.

[2] CF Cowan; J Mavour. New digital adaptive-filter implementation using distributed arithmetic techniques. IEEE Proceedings, Communications, Radar and Signal Processing, August 1981.128, PP225–230.

[3] GA Clark; SK Mitra; SR Parker. Block implementation of adaptive digital filters. IEEE Transactions on Acoustics, Speech and Signal Processing, ASSP-29, 1981.3, PP744–752.

[4] SC Douglas; Q Zhu; KF Smith. A pipelined LMS adaptive FIR filter architecture without adaptation delay. IEEE

Transactions on Signal Processing, 1998.46, PP775–779.

[5] S Haykin. Adaptive Filter Theory. Pearson Education, 2008.

[6] Jian-Sein; KK Pang. Multidelay block frequency domain adaptive filter. IEEE Transactions on Acoustics, Speech and Signal Processing, 1990.38, PP373–376.

[7] K Matsubara; K Nishikawa; H Kiya. Pipelined LMS adaptive filter using a new look-ahead transformation. IEEE Transactions on Circuits and Systems-II, 1999.46, PP51–55.

[8] NR Shanbhag; KK Parhi. A pipelined adaptive lattice filter architecture. IEEE Transactions on Signal Processing, 1993.41, PP1925–1939.

[9] NR Shanbhag; KK Parhi. Relaxed look-ahead pipelined LMS adaptive filters and their application to ADPCM coder. IEEE Transactions on Circuits and Systems-II, 1993.40, PP753–766.

[10] SA White. Applications of distributed arithmetic to digital signal processing: A tutorial review. IEEE ASSP Magazine, July 1989.