VLSI IMPLEMENTATION OF MULTIPLIER BASED BLOCK LMS ADAPTIVE FILTER

G.Preethi*1, E.Swathi*2 and K.K.Senthilkumar*2

*UG Student, Department of Electronics and Communication Engineering, Prince ShriVenkateshwarapadmavathy Engineering College, Chennai, India
*Associate Professor, Department of Electronics and Communication Engineering, Prince ShriVenkateshwarapadmavathy Engineering College, Chennai, India

Abstract—An analysis is made on the computational complexity of Block Least Mean Square Adaptive Filter where the filter computation is decomposed into M sub filters and M=N/L where N is the filter size, L is the block size. The decomposition is done inorder to favour time-multiplexing of the filter output computation and weight-increment term computation of adaptive filter. This structure supports different filter length reconfiguration and has negligible overhead complexity. This scheme also has improved Hardware Utilization Efficiency and even register complexity is independent of its block size. On comparison with the other recently proposed LMS based architectures for adaptive filters, this architecture has L times more multipliers, less adders, almost same number of registers with L times higher throughput. Different multipliers like Array Multiplier, Wallace tree multiplier, Dadda multiplier and Vedic Multiplier are being compared in the proposed scheme to reduce hardware complexities. Synthesis results shows that proposed scheme has 21.4% lesser area and 3.4 times higher throughput than the existing structures.

Index Terms—Block Convolution, Block Correlation, Time Multiplexing

I. INTRODUCTION

An Adaptive Filter is a linear system with transfer function controlled by variable parameters according to an optimization algorithm. Due to its complexity, most adaptive filters are digital filters. They find their usage in most of the Digital Signal Processing applications like noise cancellation, echo cancellation, system identification and channel equalization [1]. Least Mean Square (LMS) based Finite Impulse Response Filter is most popular due to its simplicity and satisfactory convergence behavior [2]. Conventional LMS based adaptive filter uses direct form structure which does not favour pipelined implementation due to recursive behavior and thus offers less throughput [3]. To favour pipelined implementation, an adaptation delay is introduced into the error value of LMS (DLMS) which lowers the error performance. Modified DLMS has been introduced by Poltmann to reduce performance degradation [4]. Several multiplier based architectures for DLMS have been suggested in the literature over last 15 years[5-14]. For area-delay efficient implementation, Distributed Arithmetic based LMS adaptive filter was developed which used LUT’s to obtain filter output and weight increment computation [15-20]. But these LUT’s have to be updated every iteration which consumes high energy. For fast and computationally efficient ADF’s, Block Least Mean Square Adaptive Filter (BLMS-ADF) is one of the most useful derivatives where convolution and correlation are performed using FFT/IFFT [21]. BLMS favours software implementation due to its practical applications [22]. But it does not favour Very Large Scale Integration (VLSI) implementation due to irregular data flow. Distributed Arithmetic (DA) based BLMS ADF favours hardware implementation [23] and are either bit serial or bit parallel based. Bit serial based have longer iteration period but Bit parallel based have reduced iteration period. Although Bit serial based have reduced iteration period, they involve (BLN/4) number of 16 word RAM LUT’s which is expensive for large sized ADF’s where B is the input word length.

A new approach to the efficient VLSI architecture for BLMS adaptive filter is multiplier based. It is an evolution of multiplier based ADF using LMS or DLMS concepts found in the literature [5-14]. This structure is found to take the advantage of utilizing multiplier based design which has comparable area-delay efficiency as the multiplier-less design [13]. Therefore, a multiplier-based ADF could have been explored prior to DA-based design, but we do not find any specific multiplier based design in the literature for BLMS adaptive filter. The multiplier based ADF architecture is obtained directly from LMS by introducing L-fold parallelism. This structure also supports adaptive filter implementation of variable lengths which provides low cost alternative to fixed length ADF’s as separate implementation of these ADF’s are expensive and time consuming. It also supports the following features: i) Lesser Area ii) Maximum throughput iv) 100% Hardware Utilization Efficiency (HUE) v) Reduced Hardware complexity. The proposed decomposition scheme favours time multiplexing the filter computations to achieve 100% HUE which is not possible with existing schemes and provides an area efficient variable length BLMS FIR filter structure. On use of different multipliers instead of normal multiplication technique it reduces hardware complexity.

II. COMPUTATIONAL COMPLEXITY ANALYSIS

The BLMS ADF computes one block of output yk and one block of error vector ek from one block of input vector xk and
desired vector $d_k$. The BLMS algorithm for weight updation in $(k+1)^{th}$ iteration is given by

$$w_{k+1} = w_k + \Delta w_k(1)$$

$$\Delta w_k = \mu [e_k^T X_k]^T$$

$$e_k = [d_k(0) \quad d_k(1) \quad \ldots \quad d_k(L-1)]^T$$

where $\mu$ is the convergence factor, $w_k$and $e$are the weight vector and error vector of the $k^{th}$iteration.

$X_k$ is given by the followin which is marked as (5)

$$X_k = \begin{bmatrix} x(kL) & x(kL - 1) & \ldots & x(kL - M + 1) \\ x(kL - 1) & x(kL - 2) & \ldots & x(kL - M + 2) \\ \vdots & \vdots & \ddots & \vdots \\ x(L - 1) & x(L - 2) & \ldots & x(L - M + 1) \end{bmatrix}$$

The error vector $e_k$ is given by

$$e_k = [d_k(0) \quad d_k(1) \quad \ldots \quad d_k(L-1)]^T$$

and goes on. Similarly the weight increment vector $\Delta w_k$ is decomposed into M short vectors $\epsilon^t_k$ of size L each. (16) is performed through M independent matrix vector relation using

$$\Delta c_k = S_k e_k(18)$$

$$\Delta c_k$$ is given by which is marked as (19)

$$\begin{bmatrix} \Delta w_k(L) \\ \Delta w_k(L + 1) \\ \ldots \quad \Delta w_k(L + \frac{L}{M} - 2) \\ \Delta w_k(L + \frac{L}{M} - 1) \\ \Delta w_k(L + L - 1) \end{bmatrix}$$

If the decomposition scheme is done, then time-multiplexing of filter computations can be made. If N is the filter length and L is the block size, then the number of sub-filters is given by $M=N/L$.

IV. PROPOSED ARCHITECTURE

The proposed architecture of the Block Least Mean Square Adaptive filter for filter length N=32 and block length L=4 uses 8 identical Processing Element(PE) Blocks, as M=N/L is the general representation. Therefore the proposed structure consists of 8 PE’s, one Error Computation Unit(ECU) and one Adder Unit(AU). Each of the PEs receive 8 parallel inputs at each of the iteration and these 8 PE’s produce the partial filter output at the end of each iteration. These 8 PEs are arranged in a serial structure where the output from a PE is given as the input to the next PE and the output of each PE is the delayed version of the input given to the PE i.e, if $x_k$ is the input given to the first PE, then the $(j+1)^{th}$ PE receives the $(j+1)^{th}$ delayed version of $x_k$, which is $x_{k,j}$. During the $k^{th}$ iteration the $(j+1)^{th}$ PE receives the input from the $j^{th}$ PE i.e, $x_{k,j}$ and the scaled error vector $r_i$ from the ECU. During the first half of the iteration it performs the filter computation using the filter coefficient vector $c_i^t$ and produces the partial filter output $u_i$. And during the second half of the iteration PE performs weight increment vector $\Delta c_i^t$ computation using $r_i$ to update the filter weight $c_i^t$. The 8 sub-filter outputs received during the first half of the iteration from 8 PEs are added in the adder unit to obtain the filter output $y_i$. During the same iteration the operation of ECU is also divided into two half. In the first half of the iteration, the ECU receives the filter output $y_i$ from the Adder Unit and desired response $d_i$ and computes the scaled error vector $r_i$. This $r_i$ is transmitted to all the PEs during the second half of the iteration for computing the weight increment vector $\Delta c_i^t$. The proposed multiplier-based
A. PROCESSING ELEMENT BLOCK

The internal structure of \((j+1)^{th}\) PE is shown in Fig 2. The Processing Element block for a block size of 4 consists of one Register Unit, one MUX Unit, one DMUX Unit, one Inner-Product Unit (IPU) and one Weight Update (WU) Unit. The register unit gets the block of input \(x_{k,j}\) from the previous PE, which is \(j\)th PE for a \((j+1)^{th}\) PE. This RU produces the delayed version of the input block \(x_{k,j+1}\) as the output. And also it produces the four rows of decomposed small input matrix \(S_{k,j}\) \(=\) \(\{S_{0,k}, S_{1,k}, S_{2,k}, S_{3,k}\}\). These input vectors are sent to the IPU to perform the filter output and weight increment computation, whereas the delayed version of the input block is given to the next PE. The IPU consists of \(L=4\) number of \(L=4\) point inner-product cells (IPC). Each IPC consists of four multipliers and an adder tree. The multiplication pattern used in IPC varies depending upon the requirement such as speed, delay etc. In the proposed scheme, the use of four different multipliers have been considered which includes Array multiplier, Wallace tree multiplier, Dadda multiplier and Vedic multiplier.

The IPU receives the weight-vector \(c_{k,j}\) in the first half of the iteration from the MUX unit and computes the partial filter output \(u_{j}\) whereas during the second half of the iteration the IPU receives the scaled error vector \(r_{k}\) from the MUX unit and computes the weight increment-vector \(\Delta c_{k,j}\). The WU receives the weight increment-vector \(\Delta c_{k,j}\) from the DMUX unit and updates the filter weight \(c_{k,j}\). Both the weight increment term computation and the weight update operation are completed during the second half of every iteration.

B. ERROR COMPUTATION UNIT

The ECU consists of one subtractor unit, one barrel shifter (BS) unit and one loop delay unit. The subtractor unit receives the desired response \(d_{k}\) as input along with the filter output \(y_{k}\) from the DMUX unit and produces the error vector \(e_{k}\) as output. The BS unit receives the error vector \(e_{k}\) as input from the subtractor and scales \(e_{k}\) by shifting it right by appropriate number of bits based on the convergence factor \(\mu\) to provide the convergence of the filter coefficients to achieve the desired response. The output of the BS unit is the scaled error-vector \(r_{k}\). The internal structure of ECU for \(L=4\) is shown in Fig 3.

The proposed structure can be configured for variable filter lengths by enabling and disabling of the MUX and DMUX units. Switching ON first \(m\) PEs of the 8 PEs the structure is configured for a filter-length of \(mL\), where \(1 \leq m \leq 8\). Disabling the MUX and DMUX units of a particular PE will switch OFF that particular PE. The ECU and AU do not require circuit reconfiguration to change the filter-length and also the ECU supports an adjustable \(\mu\) which is useful for a variable length ADF.
V. RESULTS AND DISCUSSION

The number of multipliers, adders, registers, MUX and DMUX used in proposed ADF architecture of different lengths and those of the existing architectures are shown in Table 1. It can be found that the proposed method requires the less number of register units with considerable amount of multipliers and adders when compared with the existing schemes.

Table 1: Required number of multipliers, adders, registers and multiplexers/demultiplexers for 32 point adaptive filter

<table>
<thead>
<tr>
<th>Structure</th>
<th>N=32 and L=4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of Multipliers</td>
</tr>
<tr>
<td>Proposed</td>
<td>128</td>
</tr>
<tr>
<td>Straight Forward</td>
<td>256</td>
</tr>
<tr>
<td>Van and Feng [11]</td>
<td>64</td>
</tr>
<tr>
<td>Long et al [5]</td>
<td>64</td>
</tr>
</tbody>
</table>

Hardware complexity is reduced in the ADF architecture by using different multipliers in IPC’s instead of a normal multiplier. Array multiplier, Wallace tree multiplier, Dadda multiplier and Vedic multiplier are considered for comparison.

Table 2: Comparison of different multipliers

<table>
<thead>
<tr>
<th>Type of Multiplier</th>
<th>Number of Full Adders</th>
<th>Number of Half Adders</th>
<th>Number of Gates required</th>
<th>Number of Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bit Array Multiplier</td>
<td>64</td>
<td>96</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>8 bit Wallace Tree Multiplier</td>
<td>64</td>
<td>49</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>8 bit Dadda Multiplier</td>
<td>64</td>
<td>49</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>8 bit Vedic Multiplier</td>
<td>64</td>
<td>49</td>
<td>0</td>
<td>17</td>
</tr>
</tbody>
</table>

The inputs to the proposed architecture of ADF includes input vector $x_n$, desired vector $d_n$, select signal for enabling and disabling mux and dmux to switch on and off the PE’s at desired time intervals. The Figure 4 shows the input vector $x_n$ given to adaptive filter which are processed to keep the difference between the output vector $y_n$ and desired vector $d_n$ to be minimum.

The partial filter outputs $e_2$, $e_3$, $e_4$ are obtained in the first half of iteration and weight update terms $c_2$, $c_3$, $c_4$, and $c_5$ are obtained in the second half of iteration. The weight update terms are used to reduce error based on error vector.

The inputs given to Adaptive Filter

The Figure 5 shows the partial filter outputs of DMUX unit and weight update unit results of Processing Element Block-1.

The Figure 6 shows the scaled error vector result of ECU. The scaled error vector $e_n$ is obtained by shifting the error vector $e_n$ obtained from the subtractor unit using a BS unit based upon the convergence factor $k$. This $e_n$ is transmitted to all the 8 PE’s to obtain the weight-increment vectors.

VI. CONCLUSION

Thus the hardware complexity of the multiplier based BLMS adaptive filter structure is reduced by implementing different kinds of multipliers instead of normal multipliers in IPC’s of PE. The speed of the filter operations and power could be further improved in future by implementing Multiple Constant Multiplication Algorithm in place of multipliers as hardware alternative must be multiplier-less.

REFERENCES


