FPGA Implementation of LMS-based FIR Adaptive Filter for Real Time Digital Signal Processing Applications

Carlo Safarian, Tokunbo Ogunfunmi, Walter J. Kozacky
Dept. of Electrical Engineering,
Santa Clara University
Santa Clara, CA 95053, USA
togunfunmi, csafarian, wkozacky@scu.edu

B.K. Mohanty
Department of Electronics and Communication Eng.
Jaypee University of Engineering and Technology
AB-Road, Guna, Madhya Pradesh, INDIA
bk.mohanti@juet.ac.in

Abstract — In this paper, we study existing designs proposed for the FPGA implementation of an LMS adaptive filter. Excess use of multipliers and longer cycle periods are a few of the issues associated with the existing structures. Based on this study, we propose a design for an FPGA implementation of an LMS based adaptive filter using the Xilinx DSP48. The proposed architecture uses one set of multipliers for both filter output and weight-increment term computation. We have simulated the proposed design for a 12-tap adaptive filter in Xilinx system generator and implemented the filter using the Vivado tool set. Implementation results shows that the proposed architecture uses 3 DSP48 units compared to 36 DSP48 units for the existing architecture with the same filter of size 12, and our implementation supports a 75% higher clocking frequency than the existing design. Additionally, the proposed architecture consumes nearly 4.7 times less dynamic power than the existing architecture. Therefore, the proposed architecture is suitable for efficient FPGA realization of an LMS FIR adaptive filter for real-time digital signal processing applications.

Keywords — FPGA; ASIC; LMS Adaptive Filters;

I. INTRODUCTION

Adaptive filters are widely used in many areas of digital signal processing applications such as echo cancelation, channel equalizer, noise cancellation and industrial applications [1]. Least mean square (LMS) algorithm is one the most popular adaptive filter algorithms due to its simplicity for implementation and satisfactory convergence behavior. During last decade, several efficient design schemes have been suggested for efficient realization of LMS-based adaptive filter real-time applications [8]-[13]. Most of these design schemes are proposed for implementation of LMS-based adaptive filter in ASIC platform which offers many useful features such as low-power, higher throughput and small area. However, ASIC implementations are expensive and involve longer design cycles. Besides, ASIC designs cannot be reprogrammed and reused for different applications. Although, there are many efficient ASIC designs available in the literature, they have failed to get the attention of industry community. On the other hand, FPGA involves less design cycle time, reprogrammable and less expensive. With the evolution of high bandwidth and low-power FPGA devices, complex digital circuits can easily be realized in FPGA for various real-time applications. Due to these features, FPGA designs are increasingly popular in the industry community. Currently, many complex digital signal processing algorithms have been implemented in FPGA for consumer electronics, automobile and many others manufacturing sectors.

Few schemes have been proposed and found in the literature for FPGA implementation of LMS-based adaptive filters [2]-[5]. Bai et al. have implemented LMS-based adaptive filter for channel equalizer in FPGA using embedded multiplier [2]. They have used transverse filter structure to implement the filter. Yaqin et al. [3] have implemented LMS-based adaptive filter in FPGA employing extreme DSP and Xilinx smart IP-core. They have considered the direct form FIR filter structure and assigned one multiplier for each filter tap along with a cascaded adder to sum the products. Shenming et al. [4] have proposed parallel structure and scheduling scheme for FPGA implementation of LMS-based adaptive filter used in hearing aid. They have considered normalized LMS (NLMS) where the convergence factor is power normalized. A scheduling scheme is proposed to compute filtering, weight increment vector and the power normalized convergence factor using one set of multipliers. Cai et al. [5] have implemented LMS adaptive filter in FPGA using Quartus II.

One of the main issues related to the existing LMS adaptive filter designs is the use of direct-form structure for FIR filter in the filtering section. The direct form structure uses the current cycle weight-vector for computation of current filter output while the transpose-form structure uses filter coefficients of $N$ past clock cycles to compute the current filter output, where $N$ is the filter length of the adaptive filter. Since, $N$ past errors are required for updating the filter weights for $N$ clock cycles, the LMS algorithm with a transpose-form FIR filter is nothing but the delay LMS algorithm (DLMS). The DLMS is similar to LMS algorithm except that it uses past error for updating the filter weight instead on current error.
Therefore, DLMS has degraded error performance compared to the LMS. But, DLMS algorithm offers pipelining and it has shorter iteration period than the LMS-based adaptive filter.

In this paper, we explore some resource sharing techniques such as multiply and accumulation which can be used to reduce the total number of multiplier usage in design. The Xilinx dedicated DSP48 slices have multiplier accumulator (MAC) feature. The implementation of FIR filter can be done using MAC feature of DSP48. Reducing the total number of DSP48 usage in the design offers area and power savings. We have used Vivado tool set and XSG for rapid prototyping the LMS filter. We used DSP48 features in our implementation for area and delay optimization with reduced power consumption. The remainder of this paper is as follows: Section II gives a brief summary of LMS algorithm and proposed architecture. Section III explains FPGA implementation. Section IV shows the simulation results. Section V is conclusion of paper.

II. LMS ADAPTIVE FILTER ALGORITHM AND PROPOSED ARCHITECTURE

A. LMS Adaptive Algorithm

The LMS adaptive filter has a filtering section and one adaptation section. The filtering section uses a finite impulse response (FIR) filter while the adaptation section uses LMS algorithm. In every iteration, the filtering section compute a filter output from which an error value is computed. The adaptation section uses the error value to compute the weight-increment term for updating the weight-vector for the next iteration

The \( n \)-th iteration weight-vector \( \{ w(n) \} \) of an LMS adaptive filter is updated for the \( (n+1) \)-th iteration using the relation:

\[
w(n+1) = w(n) + \mu . x(n) . e(n)
\]

where, \( \mu \) is the convergence factor.

The \( N \)-point weight-vector \( w(n) \) is defined as

\[
w(n) = [w_0(n), w_1(n), \ldots, w_{N-1}(n)]^T
\]

and the error \( e(n) \) is computed as

\[
e(n) = d(n) - y(n)
\]

where, \( d(n) \) is the desired response and \( y(n) \) is the an estimate of the desired response which is computed as

\[
y(n) = x(n)^T w(n)
\]

where, the \( N \)-point input vector \( x(n) \) is defined as

\[
x(n) = [x(n), x(n-1), \ldots, x(n-N+1)]^T
\]

B. Proposed Architecture

The LMS adaptive filter of length \( N \) can be decomposed into \( K \) vector of size \( L \), where \( N=KL \). The filter computation of (4) can be expressed as

\[
y(n) = \sum_{k=0}^{K-1} w_{kL}(n) x(n-k-L) + e(n)
\]

The small weight vectors \( w_m(n) \) can be updated by following equation

\[
w_m(n+1) = w_m(n) + \mu . x_m(n) . e(n)
\]

where, the \( m \)-th small weight vector \( w_m(n) \) and input vector \( x_m(n) \) are defined as

\[
w_m(n) = [w_{mL}(n), w_{mL+1}(n), \ldots, w_{mL+L-1}(n)]
\]

\[
x_m(n) = [x(n-mL), x(n-mL-1), \ldots, x(n-mL+L-1)]
\]

The proposed structure for LMS adaptive filter is shown in Fig.1. It consists of \( M \) processing elements (PEs), one delay unit, one weight-update unit and adder unit. The delay unit
receives one sample of input sequence \( x(n) \) in every iteration and produce one input-vector of size \( N \) using \((N-1)\) recent past samples. The input-vector split into \( M \) small vectors of size \( L \) each. The \( m \)-th PE receive small input-vector \( x_m(n) \), the small weight-vector \( \mathbf{w}_m(n) \) and error-signal \( e(n) \). It performs \( L \)-point inner-product between \( x_m(n) \), and \( \mathbf{w}_m(n) \) to compute one partial filter output \( y_m(n) \) during first half of each iteration and computes \( L \)-point vector-scalar multiplication during second half of each iteration to compute weight-increment vector \( \{\Delta \mathbf{w}_m(n)\} \) for updating weight-vector \( \{\mathbf{w}_m(n)\} \). The weight-update unit receives the \( M \) number weight-increment vectors from \( M \) PEs in every iteration and update the weight-vector \( \{\mathbf{w}(n)\} \) for \((n+1)\) iteration. M partial filter outputs of \( M \) PEs are added in the adder unit to obtain the filter output \( y(n) \). The internal structure of \( m \)-th PE is shown in Fig. 2. It consists of \( L \) 2:1 multiplexers (MUXes) and \( L \) demultiplexers (DMUXes). During first half of each iteration, the \( m \)-th PE performs \( L \)-point inner product between input-vector \( x_m(n) \), and weight-vector \( \mathbf{w}_m(n) \) to compute one partial filter output \( y_m(n) \) while it performs vector-scalar product between \( x_m(n) \), and \( e(n) \) to compute the weight-increment vector \( \{\Delta \mathbf{w}_m(n)\} \). In the next section we present the FPGA implementation of the proposed architecture using Xilinx DSP48E1 Slice.

III. FPGA IMPLEMENTATION

The proposed architecture is implemented in Xilinx 7K325T FPGA. The logiCORE component DSP48 is used for implementation of multiplication and addition. The design is simulated using Xilinx system generator tool and the corresponding HDL code synthesized and implemented in FPGA using Xilinx Vivado software. The simulation was performed using Vivado simulator.

A. System Generator

The proposed simulated structure in Xilinx system generator [7] is shown in Fig. 3. The design includes Control logic unit, serial to parallel (SP) control unit and PE unit. The SP control unit multiplexes the serial input data to parallel and stores the parallel data to the registers. The outputs of these registers are connected to MUXs and are fed to the PE unit. The finite state machine of the control logic unit is described with Verilog and controls the entire system operations starting from filtering step to the error calculation and filter weight update step.

The PE unit consists of three Xilinx DSP48 cores and an adder tree to generate the filter output. The filter output is subtracted from desired signal to obtain the error. The error signal is scaled by the convergence factor and the result is used for scaling the input-vector to calculate the weight-increment vector for updating the current iteration weight-vector. The filtering computation is performed using parallel set of PE’s to reduce the computation time. The resource sharing method is applied for implementation to reduce FPGA DSP48 usage. The number of DSP48 units used in the implementation depends on the algorithm partitions. The implemented filter has 12 taps and it is partitioned into three sections each one with 4 taps. The classification of taps is according to their distance to the desired signal.

B. FPGA Implementation Result and Comparison

The proposed design and the existing design of [2] are modeled in system generator and both the system generator models are synthesized by Xilinx Vivado software and targeted to 7K325T Xilinx FPGA. Table 1 and Table 2 summarize the post implementation resource usage and the power consumption of the proposed architecture and those of [2]. The FPGA resource comparison shows the proposed architecture uses 3 DSP48 units compared to 36 DSP48 units of [2] for the same filter of size 12. The timing report shows that the proposed design can run with maximum 500MHz FPGA clock.
frequency whereas the design for [2] run at maximum frequency 285 MHz. The proposed design support higher clock frequency is due to better utilization of DSP48 for internal pipelining addition operations. As shown in Table 2, the proposed design consumes nearly 4.7 times less dynamic power than [2]. This is mainly due to huge saving in FPGA resources.

**TABLE I. FPGA RESOURCE COMPARISON.**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Proposed</th>
<th>Yongbo et al [2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>1241</td>
<td>992</td>
</tr>
<tr>
<td>LUT</td>
<td>383</td>
<td>466</td>
</tr>
<tr>
<td>I/O</td>
<td>53</td>
<td>47</td>
</tr>
<tr>
<td>DSP48</td>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>RAMB36/FIFO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RAMB18</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**TABLE II. POWER ESTIMATION COMPARISON.**

<table>
<thead>
<tr>
<th></th>
<th>Proposed</th>
<th>Yongbo et al [2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>0.051W</td>
<td>0.195W</td>
</tr>
<tr>
<td>Clock</td>
<td>0.021W</td>
<td>0.018W</td>
</tr>
<tr>
<td>Signals</td>
<td>0.010W</td>
<td>0.037W</td>
</tr>
<tr>
<td>Logic</td>
<td>0.004W</td>
<td>0.017W</td>
</tr>
<tr>
<td>DSP</td>
<td>0.015W</td>
<td>0.089W</td>
</tr>
<tr>
<td>I/O</td>
<td>0.003W</td>
<td>0.033W</td>
</tr>
<tr>
<td>Device Static</td>
<td>0.158W</td>
<td>0.159W</td>
</tr>
</tbody>
</table>

IV. SIMULATION RESULT

The LMS based adaptive filter is simulated with MATLAB Simulink. The simulation model is shown in Fig. 5. The input signal \( x(n) \) is a sinusoidal signal and corrupted by random noise. The convergence factor is set to be \( \mu = 0.01 \). The filter input \( x(n) \) and filter output \( y(n) \) are shown in Fig. 6 and Fig. 7 respectively. The steady state mean square error (MSE) is shown in Fig. 8.

![Figure 5. Simulink simulation platform.](image)

![Figure 6. LMS filter input \( x(n) \).](image)

![Figure 7. LMS filter output \( y(n) \).](image)

![Figure 8. LMS filter MSE.](image)

V. CONCLUSION

In this paper, we studied existing designs proposed for FPGA implementations of LMS adaptive filters and observed that excess use of multipliers and longer cycle periods are a few of the issues associated with the existing structures. We proposed a design for an FPGA implementation of an LMS based adaptive filter using the Xilinx DSP48. The proposed architecture uses one set of multipliers for the computation of both the filter output and weight-increment term. Furthermore, the filter computation and computation of the weight-increment term are decomposed and mapped to a folded structure to save FPGA multiplier resources. We have
simulated the proposed design for a 12-tap filter in Xilinx system generator and implemented the design using the Vivado tool set. The FPGA implementation results shows that the proposed architecture uses 3 DSP48 units compared to 36 DSP48 units of [2] for the same filter of size 12. Additionally, the proposed design supports a 75% higher clocking frequency than [2], and uses nearly 4.7 times less dynamic power than the architecture of [2]. These characteristics make the proposed design suitable for an efficient FPGA realization of an LMS adaptive filter for real-time digital signal processing applications.

ACKNOWLEDGMENT

We acknowledge the support from Santa Clara University PhD Pilot Research Assistant (RA) Tuition program.

REFERENCES