A Novel Approach to FPGA Realization of LMS Adaptive Filter with Low Adaptation Delay Using Partial Product Generator

R Krishnapriya
M.E VLSI DESIGN
RMK ENGINEERING COLLEGE,
CHENNAI, INDIA
rayekrishnapriya@gmail.com

Dr. D Rukmanidevi
PROFESSOR
RMK ENGINEERING COLLEGE,
CHENNAI, INDIA
drd.ece@rmkec.ac.in

Abstract — In this project, we proposed an efficient architecture design technique for the implementation of delayed least mean square adaptive filter, for achieving lower adaptation-delay and efficient area-delay-power implementation. We use a novel partial product generator and propose a strategy for optimized balanced pipelining across the time-consuming combinational blocks of the structure. The structure of each PPG consists of $L/2$ number of 2-to-3 decoders and the same number of AND/OR cells (AOC). The normal LMS algorithm does not support pipelined implementation because of its recursive behavior; it is modified to a form called the delayed LMS (DLMS) algorithm. This DLMS uses a modified systolic architecture to reduce the adaptation delay where they have used relatively large processing elements (PEs) for achieving a lower adaptation delay with the critical path of one MAC operation. Here we compare the various types of arrays based on power, timing analysis, number of slices, flow summary etc. It is shown that compared to the existing system the proposed system produces a less area, delay, and power-efficient implementation of LMS adaptive filter. It is implemented using Modelsim and Quartus II tool by the hybrid combination of VHDL cores and it is downloaded in FPGA kit and the results are obtained.

Index Terms—least mean square adaptive filter, processing element, partial product generator, systolic architecture.

I. INTRODUCTION

Adaptive Signal Processing is concerned with the design, analysis, and implementation of systems whose structure changes in response to the incoming data. In practice, signals of interest often become contaminated by noise or other signals occupying the same band of frequency. The goal of any filter is to extract useful information from noisy data. There are numerous methods for performing weight update of adaptive filters. Here we are going to see about LMS adaptive filter with fixed point implementations. A modified delayed least mean square is used to reduce adaptation delay.

II. ADAPTIVE ALGORITHMS

There are numerous methods for the performing weight update of an adaptive filter. There is the wiener filter, which is the optimum liner filter in terms of mean squared error, and several algorithms that attempt to approximate it, such as the method of steepest descent. There is also least-mean square algorithm, developed by Windrow and Hoff originally for use in artificial neural networks.
Finally, there are other techniques such as the recursive-least square algorithm and the kalman filter. The choice of algorithm is highly dependent on the signals of interest and the operating environment, as well as the convergence time required and computation power available.

A. LEAST MEAN SQUARE ALGORITHM

The least mean square (LMS) algorithm is similar to the method of steepest-descent in that it adapts the weights by iteratively approaching the MSE minimum. Widrow and Hoff invented this technique in 1960 for use in training neural networks. The key is that instead of calculating the gradient at every time step, the LMS uses a rough approximation to the gradient. The structure of general LMS adaptive filter is shown in figure 1.

\[ W_{n+1} = W_n + \mu \cdot e_n \cdot X_n \]  
(1)

The error at the output can be expressed as

\[ e_n = d_n - y_n \]  
(2)

\[ y_n = W_n^T \cdot X_n \]  
(3)

Where the input vector \( X_n \), and the weight vector \( W_n \) at the nth iterations are, respectively, given by

\[ X_n = [x_n, x_{n-1}, \ldots, x_{n-N+1}]^T \]  
(4)

\[ W_n = [w_n(0), w_n(1), \ldots, w_n(N-1)]^T \]  
(5)

A lot of work has been done to implement the DLMS algorithm in systolic architecture to increase the maximum usable frequency but, they involve an adaptation delay of N cycles for filter length N, which is quite high for larger order filters. Since the convergence performance degrades considerably for a large adaptation delay. We use a modified systolic architecture to reduce the adaptation delay. A transpose form LMS adaptive filter is suggested in, where the filter output at any instant depends on the delayed version of weights and the number of delays shown in figure 2.

The weights of LMS adaptive filter during the nth iteration are updated according to the following equation
\[ W_{n+1} = W_n + \mu \cdot e_{n-m} \cdot X_{n-m} \]

(6)

Assuming that the latency of computation of error is \( n_1 \) cycles, the error computed by the structure at the \( n \)th cycle is

\[ e_{n-n_1} = d_{n-n_1} - y_{n-n_1} \]

(7)

Update equation of the modified DLMS algorithm is given as

\[ W_{n+1} = W_n + \mu \cdot e_{n-n_1} \cdot X_{n_1-n_2} \]

(8)

\[ y_n = W_{n-n_2}^T \cdot X_n \]

(9)

Figure 2 Delayed LMS adaptive filter

Several methods have been adopted for implementing the LMS adaptive filters in the recent years. The multiplier based implementation require more number of MAC blocks as the filter order increases hence the complexity of implementation also increases. The memory based approach has high throughput processing capability and increased regularity which results in cost-effective and area-time efficient computing structures. Here we are going to implement the fixed point implementation and optimization of the proposed DLMS adaptive filter. Adder tree is also proposed to reduce the hardware complexity without noticeable degradation of steady state MSE.

The normalized LMS filter differs from conventional LMS filter in the way in which the step size for controlling the adjustments to the filters tap weight vector is defined. Where as in conventional the step size is a scalar parameter denoted by \( \mu \). The two important advantages are folded below.

1. The normalized LMS filter mitigates the gradient noise amplification problem, which can arise when tap input vector is large.

2. The rate of convergence of the normalized LMS filter is potentially faster than that of conventional LMS filter for both correlated and uncorrelated data.

Adaptive noise canceller operating on a narrowband interfering signal component, a strong coupling develops between tap-weight vector filter and interfering signal, with the result that the mean square error of the LMS filter may be reduced below that of the wiener filter.
III. DLMS ALGORITHM

The generalized DLMS is introduced in order to obtain an efficient LMS algorithm in VLSI circuits. It is well known that a pipelined architecture helps the efficient implementation of the LMS algorithm. This type of architecture requires delays in the processing. In this work, different types of delays are introduced into the architecture of LMS algorithm. The influence of these delays on the algorithm behavior is analyzed theoretically and by simulation.

A. ERROR COMPUTATION BLOCK

The structure for error-computation unit of an N-tap DLMS adaptive filter is shown in fig. 3. It consists of N number of 2-b partial product generators (PPG) corresponding to N multipliers and a cluster of L/2 binary adder trees, followed by a single shift-add tree.

B. PARTIAL PRODUCT GENERATOR

It consists of L/2 number of 2-to-3 decoders and the same number of AND/OR cells (AOC) shown in figure 4. Each of the 2-to-3 decoders takes a 2-b digit \((u_1u_0)\) as input and produces three outputs \(b_0 = u_0 \cdot u_1\), \(b_1 = u_0 \cdot u_1\), and \(b_2 = u_0 u_1\), such that \(b_0 = 1\) for \((u_1u_0) = 1\), \(b_1 = 1\) for \((u_1u_0) = 2\), and \(b_2 = 1\) for \((u_1u_0) = 3\). The decoder output \(b_0\), \(b_1\) and \(b_2\) along with \(w\), \(2w\), and \(3w\) are fed to an AOC, where \(w\), \(2w\), and \(3w\) are in 2's complement representation and sign-extended to have \((W + 2)\) bits each. To take care of the sign of the input samples while computing the partial product corresponding to the most significant digit (MSD), i.e., \((uL−1uL−2)\) of the input sample, the AOC \((L/2 − 1)\) is fed with \(w\), −2w, and −w as input since \((uL−1uL−2)\) can have four possible values 0, 1, −2, and −1.

Figure 3 Error Computation Block

Figure 4 Partial Product Generators

C. AND OR CELLS

Each AOC consists of three AND cells and two OR cells. The structure and function of AND cells and each AND cell takes an n-bit input \(D\) and a single bit input \(b\), and consists of n AND gates. It distributes all the n bits of input \(D\) to its n AND gates as one of the inputs. The other inputs of all the n AND gates are fed with the single bit input \(b\). each OR cell similarly takes a pair of n-bit input words and has n OR gates. A pair of bits in the same bit position in \(B\) and \(D\) is fed to the same OR gate. The output of an AOC is \(w\), 2w, and 3w corresponding to the decimal values 1, 2, and 3 of the 2-b input \((u_1u_0)\), respectively. The decoder along with the AOC performs a multiplication of input operand \(w\) with a 2-b digit \((u_1u_0)\), such that the PPG of fig. 5 performs \(L/2\) parallel multiplications of input word \(w\) with a 2-b digit to produce \(L/2\) partial products of the product word \(wu\).
D. ADDER TREE

Conventionally, the shift-add operation should be performed on each PPG separately to obtain the product value and then added all the $N$ product values to compute the desired inner product. However, the shift-add operation to obtain the product value increases the word length, and consequently increases the adder size of $N-1$ additions of the product values. To avoid such increase in word size of the adders, we add all the $N$ partial products of the same place value from all the $N$ PPGs by one adder tree as shown in figure 6. All the $L/2$ partial products generated by each of the $N$ PPGs are thus added by $(L/2)$ binary adder trees. The outputs of the $L/2$ adder trees are then added by a shift-add tree according to their place values. Each of the binary adder trees require $\log_2 N$ stages of adders to add $N$ partial products, and the shift–add tree requires $\log_2 L - 1$ stages of adders to add $L/2$ output of $L/2$ binary adder trees.

E. WEIGHT UPDATE BLOCK

The weight-update block is shown in fig 7. It performs $N$ multiply-accumulate operations of the form $(\mu \times e) \times x_i + w_i$ to update $N$ filter weights. The step size $\mu$ is taken as a negative power of 2 to realize the multiplication with recently available error only by a shift operation. Each of the MAC units therefore performs the multiplication of the shifted value of error with the delayed input samples $x_i$ followed by the additions with the corresponding old weight values $w_i$. All the $N$ multiplications for the MAC operations are performed by $N$ PPGs, followed by $N$ shift–add trees. Each of the PPGs generates $L/2$ partial products corresponding to the product of the recently shifted error value $\mu \times e$ with $L/2$, the number of $2$-$b$ digits of the input word $x_i$, where the sub expression $3 \mu \times e$ is shared within the multiplier. Since the scaled error $(\mu \times e)$ is multiplied with the entire $N$ delayed input values in the weight-update block, this sub expression can be shared across all the multipliers as well. This leads to substantial reduction of the adder complexity. The final outputs of MAC units constitute the desired updated weights to be used as inputs to the error computation block as well as the weight-update block for the next iteration.
F. ADAPTATION DELAY
As shown in Fig 3, the adaptation delay is decomposed into n1 and n2. The error computation block generates delayed errorbyn1-2 cycles, which is fed to the weight update block as shown in Fig 3.5 after scaling by μ; then the input is delayed by one cycle before the PPG to make the total delay introduced by FIR filtering be n1. The weight update block generates Wn-1-n2, and the weights are delayed by n2+1 clock cycles. However, it should be noted that the delay by one clock cycle is due to latch before the partial product generator, which is included in the delay of error computation block, i.e. n1. Therefore the delay generated in weight update block becomes n2.

G. FIR FILTER
The FIR filter is a circuit that filters a digital signal (samples of numbers) and provides an output that is another digital signal with characteristics that are dependent on the response of the filter. The FIR filter is non-recursive. It uses a finite duration of none zero input values and produces a finite duration of output values which are non-zero. FIR filters use addition to calculate their outputs just like averaging does. The primitive elements used in the design of a FIR filter are delays, multipliers and adders. The FIR filter consists of a series of delays, multiplications and additions to produce the time domain output response. The impulse response of the FIR filter is the multiplication coefficients used. The phase of a FIR filter is linear. This is a dominant feature of the FIR filter. The frequency response of the FIR filter is the DFT (Discrete Fourier Transform) of the filter's impulse response.

H. MULTIPLIERS
A multiplier that accepts two numbers in digital form and gives their product in the same digital form, usually by making repeated additions; the multiplying process is simpler if the numbers are in binary form wherein digits are represented by a 0 or 1. A binary multiplier is an electronic circuit used in digital electronics, such as a computer, to multiply two binary numbers. It is build using binary adders. A variety of computer arithmetic techniques can be used to implement a digital multiplier. Most techniques involve computing a set of partial products, and then summing the partial products together. This process is similar to the method taught to primary school children for conducting long multiplication on base-10 integers, but has been modified here for application to a base-2 (binary) numeral system.

A binary computer does exactly the same, but with binary numbers. In binary encoding each long number is multiplied by one digit (either 0 or 1), and that is much easier than in decimal, as the product by 0 or 1 is just 0 or the same number. Therefore, the multiplication of two binary numbers comes down to calculating partial products, shifting them left, and then adding together (a binary addition).

This is much simpler than in decimal system, as there is no table of multiplication to remember: just shift and addition. Older multiplier architectures employed a shifter and an accumulator to sum each partial product, often one partial product per cycle. The performance of Wallace tree implementation is sometimes improved by modified booth encoding one of the two multiplicands, which reduces the number of partial products that must be summed.

I. MULTIPLIER AND ACCUMULATOR
A multiplier can be divided into three operational steps. The first is encoding in which a partial product is generated from the multiplicand(X) and the multiplier(Y). The second is adder array or partial product compression to add all partial products and convert them into the form of sum and carry. The last is the final addition in which the final multiplication result is produced by adding the sum and the carry.
IV. SIMULATION RESULTS

A novel PPG for efficient implementation of general multiplication and inner product computation have been implemented in delayed least mean square adaptive filter. For achieving lower adaptation delay by using an efficient addition scheme for inner product computation and to reduce critical paths. The simulation results for LMS filter brought out by using modelsim. Fixed point implementation of the proposed architecture, and derived an expression for the steady state error.

Optimized balanced pipelining across time-consuming blocks reduced adaption delay and power consumption. Area-delay product is reduced to 16% and Energy-delay product is reduced to 14%.

The steady state analytical MSE matched with the simulation results. The fixed point we used here is N=8. A novel PPG implementation is made in order to reduce adaptation delay by using an efficient addition scheme for inner product computation. When we enable the SA0 input we get an analog form as shown in the figure 8 for certain movement of cursor. This is a bit level.

Pruning without noticeable degradation of steady state performance. DLMS adaptive filter had a shorter critical path than the traditional system. Efficient addition scheme for inner product computation reduced adaption delay which improved convergence performance and reduced critical path to support high input sample rates.

Thus the wave form for the LMS adaptive filter as shown in figure 9 has been performed using Modelsim.

V. CONCLUSION

Area-delay-power efficient low adaptation delay architecture for fixed-point implementation of LMS adaptive filter, used a novel PPG for efficient implementation of general multiplications and inner-product computation by common sub expression sharing. Besides, proposed an efficient addition scheme for inner-product computation to reduce the adaptation delay significantly in order to achieve faster convergence performance and to reduce the critical path to support high input-sampling rates. Aside from this, a strategy for optimized balanced pipelining across the time-consuming blocks of the structure to reduce the adaptation delay and power consumption, as well. The proposed structure involved significantly less adaptation delay and provided significant saving of...
ADP and EDP compared to the existing structures. Fixed-point implementation of the proposed architecture, and derived the expression for steady-state error. The highest sampling rate that could be supported by the ASIC implementation of the proposed design. When the adaptive filter is required to be operated at a lower sampling rate, one can use the proposed design with a clock slower than the maximum usable frequency and a lower operating voltage to reduce the power consumption further.

REFERENCES