ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

AN EFFFICIENT VLSI ARCHITECTURE OF FIXED-POINT LMS ADAPTIVE FILTER FOR DSP APPLICATION

C.Sreevidya [1], V.Tamizharasan [2]
Student M.E. , Applied Electronics [1]
Assistant Professor, ECE SIG(1)[2]
Erode sengunthar engineering college

ABSTRACT

In this paper we propose adaptive filter implementing multiplication cell with efficient adder's tree to minimize the critical path during the inner product computation and also minimizing the area without increasing the adaptation delays. And also comparing the existing system with the proposed system which gives better performance characteristics.

INTRODUCTION

The filter is an important component in the communication world. It can eliminate unwanted signals from useful information.However, to obtain an optimal filtering performance; it requires 'a priori' knowledge of both the signal and its embedded noise statistical information. The classical approach to this problem is to design frequency selective filters, which approximate the frequency band of the signal of interest and reject those signals outside this frequency band. The removal of unwanted signals through the use of optimization theory is becoming popular, particularly in the area of adaptive filtering. These filters minimize the mean square of the error signal, which is the difference between the reference signal and the estimated filter output, by removing unwanted signals according to statistical parameters. In adaptive filters concepts,the most commonly used concept is least mean square (LMS) adaptive filter due to its simple and satisfactory convergence performance. The LMS adaptive filter has a long critical path in its direct form because of its inner product computations for obtaining the filter flawless output. This path have to be reduced by implementing a technique called pipelined structure whenever it exceeds the given sample period. The recursive behaviour of our conventional LMS algorithm does not support the use of pipelined concept; hence it is modified to a form called delayed least mean square algorithm adaptive filter.

In order by implementing the DLMS algorithm in the system architecture to increase the maximum usable frequency results in adaptation delays approximately N cycles for filter length N, which increases with higher order accordingly.The adaptation delay is inversely proportional to the convergence performance since increase in

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

IJARMATE
Your uth-MATE Research Paper !!!

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

adaptation delay degrades convergence performance. Many proposed some modified architecture to overcome the drawbacks like using a transpose-form LMS adaptive filter, using large processing elements, fine grained pipelined design . A 2-bit multiplication cell is used with efficient adder's tree to minimize the critical path during the inner product computation and also minimizing the area without increasing the adaptation delays.

SYSTEM DESCRIPTION

1. EXISISTING SYSTEM

The DLMS adaptive algorithm is introduced to achieve lower adaptation-delay. It can be implemented using pipelining. But it can be used only for large order adaptive filters. Typical DSP Programs with highly real-time, design hardware and or software to meet the application speed constraint. It also deals with 3-Dimensional Optimization (Area, Speed, and Power) to achieve required speed, area power trade-offs and power consumption. An efficient scheme is presented for implementing the LMS-based transversal adaptive filter in block floating-point (BFP) format, which permits processing of data over a wide dynamic range, at temporal and hardware complexities significantly less than that of a floating-point processor.

1.1. REVIEW OF DLMS ALGORITHM

For every input sample, the LMS algorithm calculates the filter output and finds the difference between the computed output and the desired response. Using this difference the filter weights are updated in every cycle. During the n-th iteration, LMS algorithm updates the weights as follows:

$$W_{n+1} = W_n + \mu \cdot e(n) \cdot x(n) \qquad (1)$$

Where, μ is the convergence-factor.

$$e(n) = d(n) - y(n) \quad y(n)= w_n^T \cdot x(n) \qquad (2)$$

Here, x(n) is the input vector,d(n) is the desired response, and y(n) is the filter output of the nth iteration w(n) is the weight vector of an Nth order LMS adaptive filter at the nth iteration, respectively, given by,

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

$$w_n = [w_n(0), w_n(1), \ldots, w_n(N-1)]^T$$ e(n) denotes the error computed in the nth iteration which is used to update the weights.

The DLMS algorithm uses the delayed error e(n−m), (i.e.) the error corresponding to (n−m) -iteration for updating the current weight. The weight-update equation of DLMS algorithm is given by,

$$W_{n+1} = W_n + \mu \cdot e(n-m) \cdot x(n-m) \qquad (3)$$

where,

The structure of conventional delayed LMS adaptive filter is shown in Fig1. It can be seen that the adaptation-delay 'm' is the number of cycles required for the error corresponding to any given sampling instant to become available to the weight adaptation circuit.

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

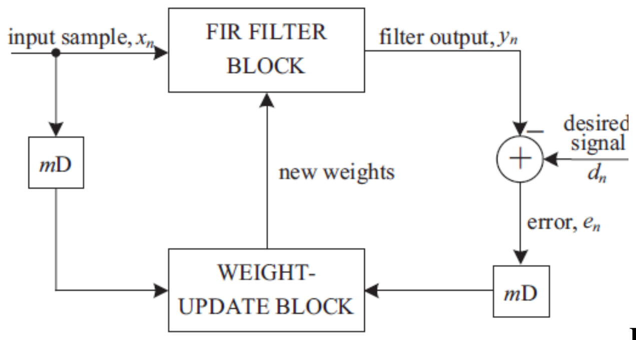


Fig. 1.1.Structure of conventional delayed LMS adaptive filter

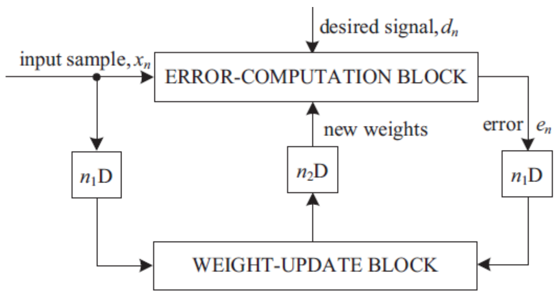


Fig. 1.2.Structure of modified delayed LMS adaptive filter

The block diagram of the DLMS adaptive filter is shown Fig. 1. , where the adaptation delay of m cycles amounts to the delay introduced by the whole of adaptive filter structure consisting of finite impulse response (FIR) filtering and the weight-update process.

It is shown in that the adaptation delay of conventional LMS can be decomposed into two parts: one part is the delay introduced by the pipeline stages in FIR filtering, and the other part is due to the delay involved in pipelining the weight update process. Based on such a decomposition of delay, the DLMS adaptivefilter canbe implemented by a structure shown Fig. 2. . Assuming that the latency of computation of error is n1 cycles, the error computed by the structure at the nth cycle is en−n1, which is used with the input samples delayed by n1 cycles to generate the weight-increment term.

The weight update equation of the modified DLMS algorithm is given by

wn+1 =wn +µ·en−n1 ·xn−n1 (3a)

where

en−n1 =dn−n1 −yn−n1 (3b)

yn =wT n−n2 ·xn. (3c)

We notice that, during the weight update, the error with n1 delays is used, while the filtering unit uses the weights delayed by n2 cycles. The modified DLMS algorithm decouples computations of theerror-computationblockandthe weight-update block and allows us to perform optimal pipelining by feedforward cut-set retiming of both these sections separately to minimize the number of pipeline stages and adaptation delay. The adaptive filters with different n1 and n2 are simulated for a system identificationproblem.The 10-tap band-passfilter with impulse response

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

hn = sin(wH(n−4.5))/ π(n−4.5) − sin(wL(n−4.5))/ π(n−4.5)

for n =0,1,2,...,9,

otherwisehn =0 (4)

is used as the unknown system . wH and wL represent the high and low cut-off frequencies of the pass band, and are set to wH = 0.7π and wL = 0.3π, respectively. The step size μ is set to 0.4. A 16-tap adaptive filter identifies the unknown system with Gaussian random input xn of zero mean and unit variance. In all cases, outputs of known system are of unity power, and contaminated with white Gaussian noise of −70 dB strength. Fig. 3 shows the learning curve of MSE of the error signal en by averaging 20 runs for the conventional LMS adaptive filter (n1 =0,n2 =0) and DLMS adaptive filters with (n1 = 5,n2 = 1) and (n1 = 7,n2 = 2). It can be seen that, as the total number of delays increases, the convergence is slowed down, while the steady-state MSE remains almost the same in all cases. In this example, the MSE difference between the cases (n1 = 5,n2 = 1) and (n1 = 7,n2 = 2) after 2000 iterations is less than 1 dB, on average.

1.2. DLMS ADAPTIVE FILTER ARCHITECTURE

There are two main computing blocks in the adaptive filter architecture:

1) The error-computation block

2) The weight-update block.

In this Section, we discuss the design strategy of the proposed structure to minimize the adaptation delay in the error-computation block, followed by the weight-update block.

1.2.1. Pipelined Structure of the Error-Computation Block

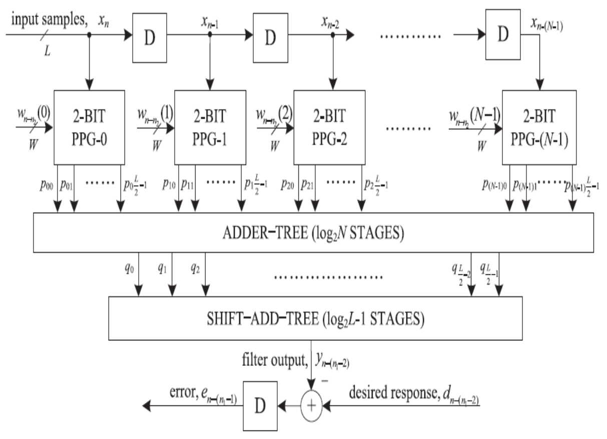

input samples, x_n
L
D
x_{n-1}
D
x_{n-2}
D
$x_{n-(N-1)}$
$w_{n-n_2}(0)$
W
2-BIT PPG-0
$w_{n-n_2}(1)$
2-BIT PPG-1
$w_{n-n_2}(2)$
2-BIT PPG-2
$w_{n-n_2}(N-1)$
2-BIT PPG-(N-1)
p_{00} p_{01} $p_{0\frac{L}{2}-1}$
p_{10} p_{11} $p_{1\frac{L}{2}-1}$
p_{20} p_{21} $p_{2\frac{L}{2}-1}$
$p_{(N-1)0}$ $p_{(N-1)1}$ $p_{(N-1)\frac{L}{2}-1}$
ADDER−TREE ($\log_2 N$ STAGES)
q_0 q_1 q_2 $q_{\frac{L}{2}-2}$ $q_{\frac{L}{2}-1}$
SHIFT−ADD−TREE ($\log_2 L$-1 STAGES)
filter output, $y_{n-(n_1-2)}$
error, $e_{n-(n_1-1)}$
D
desired response, $d_{n-(n_1-2)}$

Fig. 1.3. Structure of error-computation block .

The proposed structure for error-computation block unit of an N-tap DLMS adaptive filter is shown , 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. Each sub block is described in detail.

1.2.1.1. Structure of PPG:

The structure of each PPG is shown Fig. 4. , It consists of L/2 number of 2-to-3 decoders and the same number of AND/OR cells(AOC).1 Each of the 2-to-3 decoders takes a 2-b digit (u1u0) as input and produces three outputs b0 =u0·¯ u1, b1 =¯ u0·u1,and b2 =u0·u1, such that b0 =1 for(u1u0)=1, b1 =1 for(u1u0) = 2, and b2 = 1 for(u1u0) = 3. The decoder output b0,b1 and b2 along with w,2w, and

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

3 w are fed to an AOC, where w,2 w, and 3 w 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 productcorrespondingto 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.

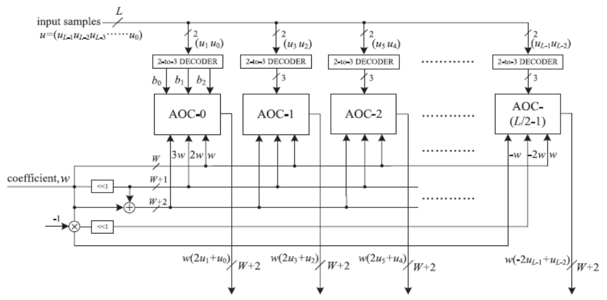


Fig.1.4.Structure of PPG.

1.2.1.2. Structure of AOCs:

Thestructureand functionof an AOC are depicted . Each AOC consists of three AND cells and two OR cells. The structure and function of AND cells and OR cells are depicted by, respectively. 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. As shown in , 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 sameOR gate. The output of an AOC is w,2 w, and 3 w corresponding to the decimal values 1, 2, and 3 of the 2-b input (u1u0), respectively. The decoder along with the AOC performs a multiplication of input operand w with a 2-b digit (u1u0), such that the PPG 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.

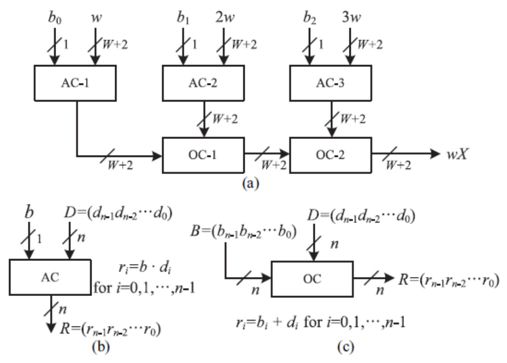

b_0 w b_1 $2w$ b_2 $3w$
AC-1
AC-2
AC-3
OC-1
OC-2
W+2
wX
(a)
$D=(d_{n-1}d_{n-2}\cdots d_0)$
AC
$r_i=b\cdot d_i$ for $i=0,1,\cdots,n-1$
$R=(r_{n-1}r_{n-2}\cdots r_0)$
(b)
$B=(b_{n-1}b_{n-2}\cdots b_0)$
OC
$r_i=b_i+d_i$ for $i=0,1,\cdots,n-1$
(c)

Fig.1.5. Structure and function of AND/OR Cells.

1.3.STRUCTURE OF ADDER TREE

Conventionally, we should have performed the shift-add operation on the partial products of 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-addoperationto 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. 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

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

a shift-add tree according to their place values. Each of the binary adder trees require log2 N stages of adders to add N partial product, and the shift–add tree requires log2 L –1 stages of adders to add L/2 output of L/2 binary adder trees.2 The addition scheme for the error-computation block for a four-tap filter and input word size L =8 is shown . For N =4 andL =8, the adder network requires four binary adder trees of two stages each and a two-stage shift–add tree. In this figure, we have shown all possible locations of pipeline latches by dashed lines, to reduce the critical path to one addition time.

If we introduce pipeline latches after every addition,it would require L(N –1)/2+L/2–1 latches in log2 N +log2 L–1 stages, which would lead to a high adaptation delay and introduce a large overhead of area and power consumption for large values of N and L. On the other hand, some of those pipeline latches are redundant in the sense that they are not required to maintain a critical path of one addition time.

The final adder in the shift–add tree contributes to the maximum delay to the critical path. Based on that observation, we have identified the pipeline latches that do not contribute significantly to the critical path and could exclude those without any noticeable increase of the critical path. The location of pipeline latches for filter lengths N =8, 16, and 32 and for input size L =8 are shown in Table I. The pipelining is performed by a feedforward cut-set retiming of the error-computation block.

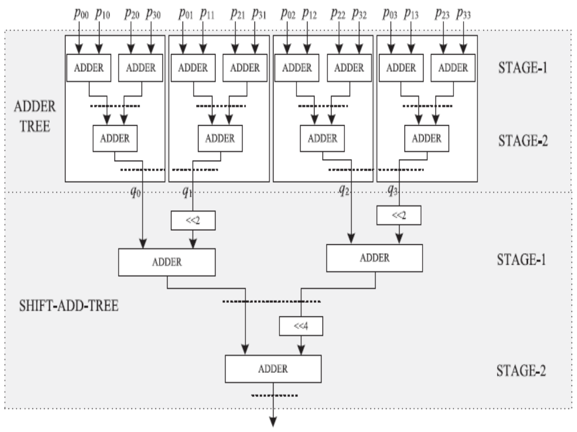

p_{00} p_{10} p_{20} p_{30} p_{01} p_{11} p_{21} p_{31} p_{02} p_{12} p_{22} p_{32} p_{03} p_{13} p_{23} p_{33}
ADDER TREE
ADDER
STAGE-1
STAGE-2
q_0 q_1 q_2 q_3
<<2
<<4
SHIFT-ADD-TREE
STAGE-1
STAGE-2

Fig.1.6. Adder structure of the filtering (N=4 & L=8)

1.4.ADAPTATION DELAY

The adaptation delay is decomposed into n1 and n2 shown in Fig. 2.. The error-computation block generates the delayed error by n1–1 cycles as shown , which is fed to the weight-update block shown in Fig. 8 after scaling by μ; then the input is delayed by 1 cycle before the PPG to make the total delay introduced by FIR filtering be n1. In Fig. 8, the weight-update block generates wn–1–n2, and the weights are delayed by n2+1 cycles. However, it should be noted that the delay by 1 cycle is due to the latch before the PPG, which is included in the delay of the error-computation block, i.e., n1. Therefore, the delay generated in the weight-update block becomes n2. If the locations of pipeline latches are decided as in Table I, n1 becomes 5, where three latches are in the error-computation block, one latch is after the subtraction in and the other latch

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

is before PPG . Also, n2 isset to 1 from a latch in the shift-add tree in the weight-update block.

1.5.FIXED POINT IMPLEMENTATION OPTIMIZATION SIMULATION ANALYSIS

In this section, we discuss the fixed-point implementation and optimization of the proposed DLMS adaptive filter. A bit level pruning of the adder tree is also proposed to reduce the hardwarecomplexitywithout noticeabledegradationofsteadystate MSE.

1.5.1.Fixed-Point Design Considerations

For fixed-point implementation, the choice of word lengths and radix points for input samples, weights, and internal signals need to be decided. shows the fixed-point representation of a binary number. Let (X, Xi) be a fixed-point representation of a binary number where X is the word length and Xi is the integer length. The word length and location of radix point of xn and wn need to be predetermined by the hardware designer taking the design constraints, such as desired accuracy and hardware complexity, into consideration. Assuming (L, Li) and (W ,Wi), respectively, as the representations of input signals and filter weights, all other signals can be decided as shown .

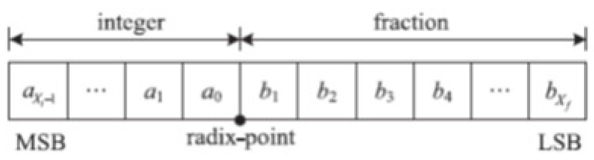


Fig. 1.8. Fixed-point representation of binary number

The signal pij, which is the output of PPG block, has at most three times the value of input coefficients. Thus, we can add two more bits to the word length and to the integer length of the coefficients to avoid overflow. The output of each stage in the adder tree is one bit more than the size of input signals, so that the fixed-point representation of the output of the adder tree with log2 N stages becomes (W+ log2 N +2,Wi +log2 N +2). Accordingly, the output of the shift–add tree would be of the form (W+L+log2 N,Wi+Li+ log2 N), assuming that no truncation of any least significant bits (LSB) is performed in the adder tree or the shift–add tree.

FIXED-POINT REPRESENTATION OF THE SIGNALS OF THE PROPOSED DLMS ADAPTIVE FILTER ($\mu = 2^{-(L_i+\log_2 N)}$)

Signal Name	Fixed-Point Representation
x	(L, L_i)
w	(W, W_i)
p	$(W+2, W_i+2)$
q	$(W+2+\log_2 N, W_i+2+\log_2 N)$
y, d, e	$(W, W_i+L_i+\log_2 N)$
μe	(W, W_i)
r	$(W+2, W_i+2)$
s	(W, W_i)

Table 1.9 : Fixed –point representation of signals.

However, the number of bits of the output of the shift–add tree is designed to have W bits. The most significant W bits need to be retained out of (W

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

IJARMATE
Your tab-MATE Research Paper !!!

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

+L+log2 N) bits, which results in the fixed-point representation (W ,Wi +Li +log2 N) for y, as shown. Let the representation of the desired signal d be the same way, even though its quantization is usually given as the input. For this purpose, the specific scaling/sign extension and truncation/zero padding are required. Since the LMS algorithm performs learning so that y has the same sign as d, the error signal e can also be set to have the same representation as y without overflow after the subtraction.

It is shown that the convergence of an N-tap DLMS adaptive filter with n1 adaptation delay will be ensured if

$$0 < \mu < 2/ (\sigma 2\ x\ (N - 2) + 2n1 - 2)\sigma 2\ x \qquad (5)$$

Where σ2 x is the average power of input samples.

Furthermore, if the value of μ is defined as (power of 2) 2−n

where n ≤ Wi+Li+log2 N

the multiplication with μ is equivalent to the change of location of the radix point. Since the multiplication with μ does not need any arithmetic operation, it does not introduce any truncation error. If we need to use a smaller step size, i.e., n >Wi+Li+log2 N, some of the LSBs of en need to be truncated. If we assume that n = Li +log2 N, i.e.= 2−(Li+log2 N), as in, the representation of μen should be (W,Wi) without any truncation. The weight increment term s (shown in Fig. 8), which is equivalent to μ* en* xn, is required to have fixed-point representation (W+L,Wi +Li). However, only Wi MSBs in the computation of the shift–add tree of the weight-update circuit are to be retained, while the rest of the more significant bits of MSBs need to be discarded.

This is in accordance with the assumptions that, as the weights convergetoward the optimal value, the weight increment terms become smaller, and the MSB end of error term contains more number of zeros. Also, in our design, L − Li LSBs of weight increment terms are truncated so that the terms have the same fixed-point representation as the weight values. We also assume that no overflow occurs during the addition for the weight update. Otherwise, the word length of the weights should be increased at every iteration, which is not desirable. The assumption is valid since the weight increment terms are small when the weights are converged. Also when overflowoccursduringthe trainingperiod,the weight updating is not appropriate and will lead to additional iterations to reach convergence. Accordingly, the updated weight can be computed in truncated form (W,Wi) and fed into the error computation block.

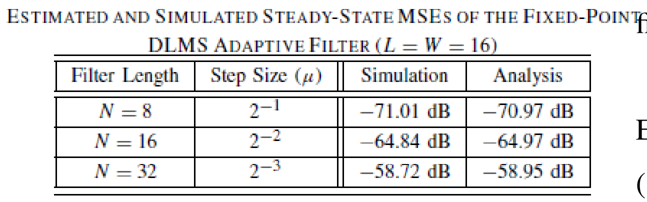
1.5.3. Steady-State Error Estimation

In this section, the MSE of output of the proposed DLMS adaptive filter due to the fixed-point

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

quantization is analysed. Based on the models introduced, the MSE of output in the steady state is derived in terms of parameters listed .


ESTIMATED AND SIMULATED STEADY-STATE MSES OF THE FIXED-POINT DLMS ADAPTIVE FILTER ($L = W = 16$)

Filter Length	Step Size (μ)	Simulation	Analysis
$N = 8$	2^{-1}	-71.01 dB	-70.97 dB
$N = 16$	2^{-2}	-64.84 dB	-64.97 dB
$N = 32$	2^{-3}	-58.72 dB	-58.95 dB

Table 3.2 : Steady-state MSE

Let us denote the primed symbols as the truncated quantities due to the fixed-point representation, so that the input and the desired signals can be written as

$x\square_n = x_n + \alpha_n$(6)

$d\square_n = d_n + \beta_n$(7)

where α_n and β_n are input quantization noise vector and quantization noise of desired signal, respectively. The weight vector can be written as

$w\square_n = w_n + \rho_n$(8)

whereρ_n is the error vector of current weights due to the finite precision. The output signal $y\square_n$ and weight-update equation can accordingly be modified, respectively, to the forms

$y\square_n = w\square^T_n x\square_n + \eta_n$(9)

$w\square_{n+1} = w_n + \mu e\square_n x\square_n + \gamma_n$(10)

whereη_n and γ_n are the errors due to the truncation of output from the shift–add tree in the error-computation block and weight-update block, respectively. The steady-state MSE in the fixed-point representation can be expressed as

$E|d_n - y\square_n|^2 = E|e_n|^2 + E|\alpha^T_n\ w_n|^2 + E|\eta_n|^2 + E|\rho^T_n\ x_n|^2$ (11)

where E|·|is the operator for mathematical expectation, and the terms e_n, $\alpha^T_n w_n$, η_n, and$\rho^T_n x_n$ are assumed to be uncorrelated. The first term

$E|e_n|^2$,where $e_n = d_n - y_n$,

is the excess MSE from infinite precision computation, whereas the other three terms are due to finite-precision arithmetic.

The second term can be calculated as

$E|\alpha^T_n\ w_n|^2 = |w^*_n|^2(m^2_{\alpha n} + \sigma^2_{\alpha n})$ (12)

where w^*_n is the optimal Wiener vector, and $m^2_{\alpha n}$ and$\sigma^2_{\alpha n}$ are defined as the mean and variance of αn when xn is truncated to the fixed-point type of (L, Li), as listed. αn can be modelled as a uniform distribution with following mean and variance:

$m^2_{\alpha n}$=2−(L−Li)/2 (13a)

$\sigma^2_{\alpha n}$=2−2(L−Li)/12. (13b)

For the calculation of the third term E|ηn|2 in (11), we have used the fact that the output from shift–add tree in the error computation block is of

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

IJARMATE

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

the type (W,Wi +Li +log2 N) after the final truncation. Therefore

$$E|\eta_n|^2 = m^2_{\square n} + \sigma^2_{\square n} (14)$$

where

$$m^2_{\square n} = 2^{-(W-(W_i + L_i + \log_2 N))/2} (15a)$$

$$\sigma^2_{\square n} = 2^{-2(W-(W_i + L_i + \log_2 N))/12}. (15b)$$

The last term $E|\rho^T_n x_n|^2$ in (11) can be obtained by using the derivation proposed in as

$$E|\rho^T_n x_n|^2 = m^2_{\gamma n} ((\textstyle\sum_i \sum_k (R_{ki}^{-1)})/\mu^2) + (N(\sigma_{\gamma n}^2 - m^2_{\gamma n})/2\mu) \qquad (16)$$

whereRki represents the (k,i)th entry of the matrix E(xnxT n). For the weight update in (10), the first operation is to multiply e n with μ, which is equivalent to moving only the location of the radix point and, therefore, does not introduce any truncation error. The truncation after multiplication of μe n with xn is only required to be considered in order to evaluate γn.

Then, we have

$$m^2_{\gamma n} = 2^{-(W-W_i)/2} (17a)$$

$$\sigma_{\gamma n}^2 = 2^{-2(W-W_i)/12} \qquad (17b)$$

For a large μ, the truncation error ηn from the errorcomputation block becomes the dominant error source, and (11) can be approximated as E|ηn|2. The MSE values are estimated from analytical expressions as well as from the simulation results by averaging over 50 experiments. It shows that the steady-state MSE computed from analytical expression matches with that of simulation of the proposed architecture for different values of N and μ.

2.PROPOSED SYSTEM

2.1. ADDER OPTIMIZATION

2.1.1 MEMORY ELEMENT BASED FULL ADDER

The memory element based full adder is a basic full adder which is combined witha flip flop to utilize the adder unit at different clock cycles in time-serialized ripple-carry manner and the number of clock cycles that it takes is equal to the number of

bits.

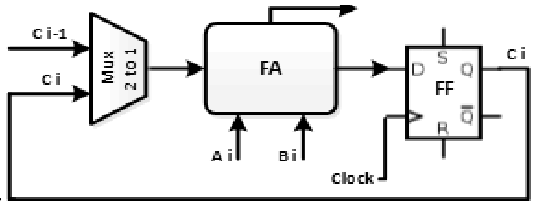


fig 2.1 memory element based full adder

2.1.2 MULTIPLIXER BASED FULL ADDER

2.1.3 NAND BASED FULL ADDER

Addition is one of the most commonly used arithmetic operations. And its operation leads to the application in digital circuits, like the adder/summer. The full adder/summer performsthe 0 addition of numbers. And its component is very essential especially in the arithmetic logic unit(s),digital signal processing, and in micro-processors. This circuit is my project for my course and I want it to share with you. I am tasked to design a 4-bit full adder (FA) using only two-input NAND gates.

In this paper, a schematic diagram and an IC layout of a 4-bit full adder were made and

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

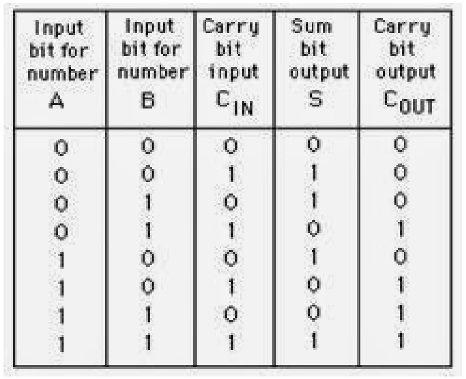
simulated in order to compare the acquired output waveform, both from the schematic simulation and IC layout aiming to grasp the concept behind lay-outing limiting/lessening any parasitic effect and having more compact circuit.

The binary adder circuit is an important building block of digital arithmetic circuits. Hence it becomes one of the most critical components of a processor, as it is used in the arithmetic logic unit(ALU), in the floating-point unit, and for address generation in case of cache or memory access. The 4-bit full adder's performance would affect the system as a whole since more transistors will be used which will add more time delay.

FA is a combinational circuit that forms the arithmetic sum of 3 bits. It consists of 3 inputs and 2 outputs. The design of a binary adder begins by considering the process of addition in base 2.

For example

$$\begin{array}{rcr} 1\ \ 1\ \ \ \ \ \ & & \\ 1\ 0\ 1\ 1 & = & 11 \\ +\ \underline{0\ 1\ 1\ 0} & = & \underline{6} \\ 1\ 0\ 0\ 0\ 1 & = & 17 \end{array}$$



Input bit for number A	Input bit for number B	Carry bit input C_{IN}	Sum bit output S	Carry bit output C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full-adder Truth Table

2.1.4 BEC ORIENTED ADDER

Basic architecture of conventional adders clears that clearly there is a chancethat we can minimize the requirement of area and power consumption of the conventional CSA. For this purpose here used Binary to Excess-1 Converter thateffectively replaces the ripple carry adder with carry input Cin=1 and for improving the working operations we have not used multiplexer. We can clearly understand that from the block diagram of proposed structure which is of the order of 16-bit is divided into five blocks of RCA and BEC with different order of bit size.

The first group in this structure i.e. group 1 contains only single RCAblock of 2- bit size, which gives there output sum and carry out. One is RCAblock and another is BEC block is used from the next coming groups that means it consist combination of 2 blocks.

In the designed circuit therefore there are two outputs one comes from the RCA block and other output from the BEC Sout and Cout is chosen from the upper RCA block if Cin=0 and if Cin=1 then the output is chosen from BEC block.

Final output is selected without the help of multiplexer. Using this concept and with the help of Xilinx calling function command we can design higher order bit like 32-bit and 64-bit adder.

2.2. MULTIPLIER ARCHITECTURE

2.2.1 VEDIC MULTIPLIER

The design of high speed Vedic Multiplier using the techniques of Ancient Indian Vedic Mathematics that have been modified to improve performance. Vedic Mathematics is the ancient system of mathematics which has a unique technique of calculations based on 16 Sutras. The work has proved the efficiency of UrdhvaTriyagbhyam– Vedic method for multiplication which strikes a difference in the actual process of multiplication itself. It enables parallel generation of intermediate products, eliminates unwanted multiplication steps with zeros and scaled to higher bit levels using Karatsuba algorithm with the compatibility to different data types. Urdhvatiryakbhyam Sutra is most efficient Sutra (Algorithm), giving minimum delay for multiplication of all types of numbers, either small or large. Further, the Verilog HDL coding of Urdhvatiryakbhyam Sutra for 32x32 bits multiplication and their FPGA implementation by Xilinx Synthesis Tool on Spartan 3E kit have been done and output has been displayed on LCD of Spartan 3E kit. The synthesis results show that the computation time for calculating the product of 32x32 bits is 31.526 ns

4.1.1. VEDIC ALGORITHIM

Multiplication is an important fundamental function in arithmetic operations. Multiplication-based operations such as Multiply and Accumulate(MAC) and inner product are among some of the frequently used Computation- Intensive Arithmetic Functions(CIAF) currently implemented in many Digital Signal Processing (DSP) applications such as convolution, Fast Fourier Transform(FFT), filtering and in microprocessors in its arithmetic and logic unit [1]. Since multiplication dominates the execution time of most DSP algorithms, so there is a need of high speed multiplier. Currently, multiplication time is still the dominant factor in determining the instruction cycle time of a DSP chip. The demand for high speed processing has been increasing as a result of expanding computer and signal processing applications. Higher throughput arithmetic operations are important to achieve the desired performance in many real-time signal and image processing applications [2]. One of the key arithmetic operations in such applications is multiplication and the development of fast multiplier circuit has been a subject of interest over decades. Reducing the time delay and power consumption are very essential requirements for many applications [2, 3]. This work presents

different multiplier architectures. Multiplier based on Vedic Mathematics is one of the fast and low power multiplier. Minimizing power consumption for digital systems involves optimization at all levels of the design. This optimization includes the technology used to

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

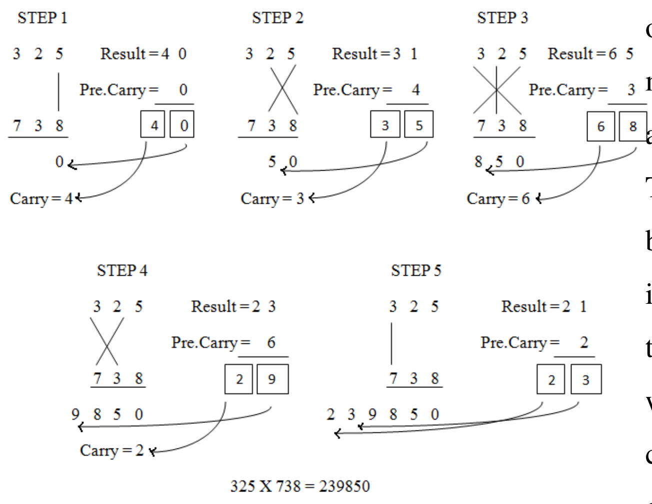
IJARMATE
Your ulti-MATE Research Paper !!!

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

implement the digital circuits, the circuit style and topology, the architecture for implementing the circuits and at the highest level the algorithms that are being implemented. Digital multipliers are the most commonly used components in any digital circuit design. They are fast, reliable and efficient components that are utilized to implement any operation. Depending upon the arrangement of the components, there are different types of multipliers available. Particular multiplier architecture is chosen based on the application. In many DSP algorithms, the multiplier lies in the critical delay path and ultimately determines the performance of algorithm. The speed of multiplication operation is of great importance in DSP as well as in general processor. In the past multiplication was implemented generally with a sequence of addition, subtraction and shift operations. There have been many algorithms proposals in literature to perform multiplication, each offering different advantages and having tradeoff in terms of speed, circuit complexity, area and power consumption. The multiplier is a fairly large block of a computing system. The amount of circuitry involved is directly proportional to the square of its resolution i.e. A multiplier of size n bits has n2 gates. For multiplication algorithms performed in DSP applications latency and throughput are the two major concerns from delay perspective. Latency is the real delay of computing a function, a measure of how long the inputs to a device are stable is the final result available on outputs. Throughput is the measure of how many multiplications can be performed in a given period of time; multiplier is not only a high delay block but also a major source of power dissipation. That□s why if one also aims to minimize power consumption, it is of great interest to reduce the delay by using various delay optimizations. Digital multipliers are the core components of all the digital signal processors (DSPs) and the speed of the DSP is largely determined by the speed of its multipliers[11]. Two most common multiplication algorithms followed in the digital hardware are array multiplication algorithm and Booth multiplication algorithm. The computation time taken by the array multiplier is comparatively less because the partial products are calculated independently in parallel. The delay associated with the array multiplier is the time taken by the signals to propagate through the gates that form the multiplication array. Booth multiplication is another important

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017



4.1.2. VEDIC MULTIPLICATION

The proposed multiplications were implemented using two different coding techniques viz., conventional shift & add and Vedic technique for 4, 8, 16, and 32 bit multipliers. It is evident that there is a considerable increase in speed of the Vedic architecture.

4.1.2.1. DESIGN OF VEDIC MULTIPLIER

Urdhvatiryakbhyam Sutra is a general multiplication formula applicable to all cases of multiplication. It literally means "Vertically and Crosswise". To illustrate this multiplication scheme, let us consider the multiplication of two decimal numbers (5498 × 2314). The conventional methods already know to us will require 16 multiplications and 15 additions. An alternative method of multiplication using Urdhvatiryakbhyam Sutra is shown in Fig. 1. The numbers to be multiplied are written on two consecutive sides of the square as shown in the figure. The square is divided into rows and columns where each row/column corresponds to one of the digit of either a multiplier or a multiplicand. Thus, each digit of the multiplier has a small box common to a digit of the multiplicand. These small boxes are partitioned into two halves by the crosswise lines. Each digit of the multiplier is then independently multiplied with every digit of the multiplicand and the two-digit product is written in the common box. All the digits lying on a crosswise dotted line are added to the previous carry. The least significant digit of the obtained number acts as the result digit and the rest as the carry for the next step. Carry for the first step (i.e., the dotted line on the extreme right side) is taken to be zero [9].

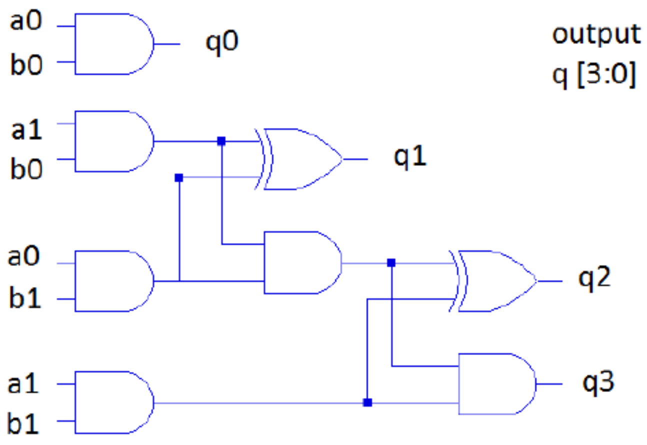


Figure 1: Alternative way of multiplication by Urdhvatiryakbhyam Sutra.

The design starts first with Multiplier design, that is 2x2 bit multiplier as shown in figure 2. Here, "UrdhvaTiryakbhyam Sutra" or "Vertically and Crosswise Algorithm"[4] for multiplication has been effectively used to develop digital multiplier architecture. This algorithm is quite different from

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 3, Special Issue 2, April 2017

the traditional method of multiplication, which is to add and shift the partial products.

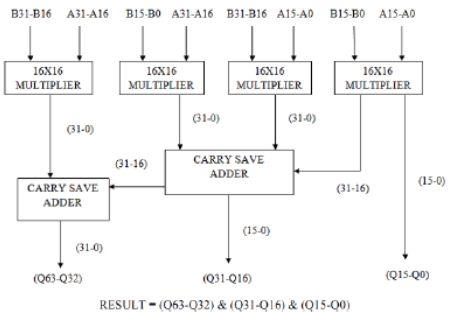
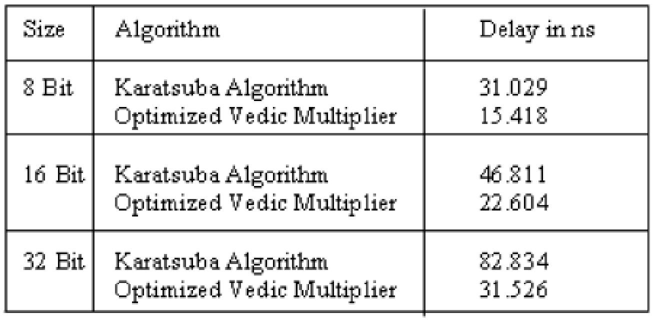


FIG 2. 32 BIT PROPOSED VEDIC MULTIPLIER

To scale the multiplier further, Karatsuba – Ofman algorithm can be employed. Karatsuba-Ofman algorithm is considered as one of the fastest ways to multiply long integers. It is based on the divide and conquer strategy . A multiplication of 2n digit integer is reduced to two n digit multiplications, one (n+1) digit multiplication, two n digit subtractions, two left shift operations, two n digit additions and two 2n digit additions.The Optimized Vedic multiplier case was found to be 31.526ns. To compare it with other implementations the design was synthesized on XILINX: SPARTAN: xc3s500e-5fg320 synthesis result for various implementations

TABEL1:



Size	Algorithm	Delay in ns
8 Bit	Karatsuba Algorithm Optimized Vedic Multiplier	31.029 15.418
16 Bit	Karatsuba Algorithm Optimized Vedic Multiplier	46.811 22.604
32 Bit	Karatsuba Algorithm Optimized Vedic Multiplier	82.834 31.526

.

CONCLUSION

The project presented here is an adaptive filter using bit serial adder and is functionally verified and simulated using Modelsim software and implemented on Spartan 3E FPGA kit using Xilinx software, parameter like area, speed and power will be compared to their implementation using conventional multiplier & adder architectures.

. In this design uses single half adder and full adder for design of adder with applying the input as a serious of clock pulse and also getting the output as a serious of clock pulse to reduce the area of the adder as well as multiplier and adaptive filter.