Power and Area Efficient Multiplier Unit Design for Implementation of Adaptive Filter

A Dissertation Submitted in Partial Fulfillment of the Requirement for the Award of the Degree
of

MASTER OF TECHNOLOGY
IN
VLSI DESIGN

Submitted By
Sharanjit Kaur
601562022

Under the Supervision of
Dr. Ravi Kumar
Associate Professor
(ECED)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT
THAPAR UNIVERSITY, PATIALA, PUNJAB
JUNE, 2017

DECLARATION

I, Sharanjit Kaur hereby declare that the work presented in this thesis entitled "Power and area efficient Multiplier Unit Design for implementation of Adaptive Filter" in partial fulfillment of the requirement for the award of degree of Master of Technology in VLSI Design submitted at Electronics and Communication Engineering Department, Thapar University, Patiala is an authentic record of work carried out under supervision of Dr. Ravi Kumar (Associate Professor, ECED, Thapar University) from 2015 to 2017. The matter presented in this thesis has not been submitted either in part or full to any other university or institute for the award of any other degree.

Sharanjit Kaur

Date: 21/08/2017

Sharanjit Kaur
601562022

It is certified that the above statement made by the candidate is correct to the best of my knowledge and belief.

Ravi

Date: 21/08/2017

Dr. Ravi Kumar
Associate Professor
Thapar University, Patiala

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to **Dr. Ravi Kumar, Associate Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala for his patient guidance and support throughout this report. I am truly very fortunate to have the opportunity to work with him. I found this guidance to be extremely valuable.

I am also thankful to **Head of the Department, Dr. Alpana Agarwal** as well as our P.G. coordinator **Dr. Anil Arora, Assistant Professor.** Further, I would like to thank entire faculty member, staff of Electronics and Communication Engineering Department, and then friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

Lastly, I would like to thank my parents for their years of unyielding love and for constant support and encouragement. They have always wanted the best for me and I admire their determination and sacrifice.

Sharanjit Kaur

ABSTRACT

Adaptive filters are used in various digital signal processing (DSP) applications such as noise cancellation, echo cancellation, channel equalization and system identification. Area-delay and power efficient realization of the adaptive filters has great practical importance for resource constrained DSP applications. The least mean square (LMS) algorithm, which is derived from the minimization of the mean squared error between the output of the adaptive filter and some desired signal, has the advantage of low complexity as well as simplicity of implementation and offers low throughput rate and is suitable for signals having smaller sampling frequencies. In this thesis, main concern is to develop efficient implementation techniques of this algorithm to provide fast convergence of the adaptive filter coefficients and in the same time good filtering performance.

As DSP systems can be realized using programmable processors or custom designed hardware circuit fabrication using VLSI technology, low-complexity design is necessary for efficient hardware implementation which requires low area-delay product, high speed of operation and low power consumption. In order to develop area-delay efficient computing structures for LMS-based adaptive filters, Distributed arithmetic (DA) approach using Block LMS (BLMS) algorithm has been proposed which resulted in reduction of area and delay as well as increased throughput as both convolution operation, to compute filter output, and correlation operation, to compute weight-increment term, is performed by using the same LUT.

Further, multipliers for an important part in filtering process where high speed calculations are required and time taken by multiplication operation is the dominant factor in determining the instruction cycle time of a DSP. Moreover, multiplier lies in the critical delay path and determines the performance of processor, speed of multiplication operation is highly important in DSP operations as it is generally the slowest element in the system. Hence, efficient design and implementation of dedicated squaring and cubing unit is proposed which has reported significant reduction in the power and delay while minimizing silicon area at the same time.

The proposed design of DA-BLMS algorithm is implemented using Verilog and Dedicated Squaring unit using VHDL in XILINX ISE14.5. The Power Efficient cubing Unit is implemented at device level using Cadence-Virtuoso ADE.

TABLE OF CONTENTS

LIST OF FIGURES

LISTS OF TABLES

CHAPTER 1
INTRODUCTION TO ADAPTIVE FILTERS

1.1 DEFINITION OF ADAPTIVE FILTERS

An adaptive filter, as the name suggests is a self-adjusting filter that changes its transfer function according to an optimization algorithm [1]. Because of the complexity of the optimizing algorithms, most adaptive filters are digital filters that perform digital signal processing and adapt their performance based on the input signal. For some applications, the desired operating parameters are not known prior to the filtering process, which generates the need for adaptive coefficients [2]. In these situations it is common to employ an adaptive filter, which uses feedback to refine the values of the filter coefficients and hence its frequency response.

The adapting process involves the use of a cost function [3], [4]. For example, minimizing the noise component of the input, is a criterion for optimum performance of the filter, which determines how to modify the filter coefficients to minimize the cost on the next iteration. The weight updating process of an adaptive filter can be performed by using different algorithms like Wiener filter, which is considered as the optimum linear filter with respect to mean squared error, and many other algorithms that tries to approximate the mean squared error, such as the method of steepest descent. The Windrow and Hoff's Least-Mean-Square algorithm, which was actually developed for use in artificial neural networks [5], is also a preferred method. The selection of an algorithm is extremely reliant on the nature of required signal and the operating background, as well as the convergence time needed and obtainable computation power. The significance of adaptive filters lies in their high performance and robustness to noisy environment [6].

An adaptive digital filter can be designed using an IIR (Infinite impulse response) filter implementation or FIR (Finite impulse response) filter. FIR filters are innately secure because of its forward path constituency. The presence of feed back to the input may lead the filter to be unstable and oscillation may occur. Because of that our filter design is accomplished through FIR realization.

Figure 1.1 depicts the basic structure of adaptive filter. Here x(n) is the input signal, y(n) is the output signal, d(n) is desired signal, e(n) is the error signal and w_n is the filter weights or coefficients. The first block is a digital filter which can be an FIR or IIR filter. However, in general we use FIR filters because they are less complicated and highly stable. The second block is a weight adaptation block which updates the filter weights or coefficients according to the adaptive algorithm to reduce the error signal. The coefficients of the adaptive filter are determined during a training sequence where a known data pattern is transmitted. The adaptive algorithm first matches the received data to the training sequence by adjusting its filter coefficients then in order to receive the actual data, the switches are repositioned. During this time, the error signal is generated by subtracting the output signal from the reference signal. The input signal is filtered to produce an output that is typically passed on for subsequent processing. This, signal is then passed on to a circuit to modify the parameters of the filter unit output is acceptable.

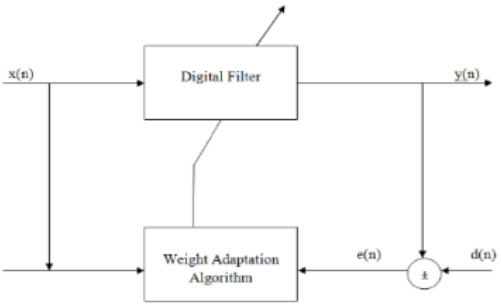


Figure 1.1 Basic Structure of Adaptive Filter

Then the input signal is given as the sum of a desired signal d(n) and interfering noise v(n) i.e.

$$x(n) = d(n)+v(n) \tag{1.1}$$

The coefficients for a *mth* order filter are defined as

$$w_n = [w_n(0), w_n(1) \dots (p)]^T \tag{1.2}$$

The error signal, also known as cost function is taken as the difference between the desired and the estimated signal

$$e(n)=d(n)-y(n) \tag{1.3}$$

The coefficient variable filter estimates the desired signal by finding the convolution between the input and its impulse response. In vector notation this can be expressed as

$$y(n) = w_n \cdot x(n) \tag{1.4}$$

where,

$$x(n) = [x(n), x(n-1) \ldots (n-p)]^T \tag{1.5}$$

is an input signal vector. Moreover, the updated filter coefficients during every iteration is given by

$$w_{n+1} = w_n + \Delta w_n \tag{1.6}$$

Where Δwn is a correction factor for the filter coefficients [7].

The output of adaptive filter is mainly determined by the step size (δ), type of adaptive algorithm applied, the chosen filter length and the structure or design used for the execution of the adaptive operation [8]. Out of all these parameters step size (δ) is the most imperative one as it determines the performance of adaptive filter with highest impact degree than all other attributes, when they are set to their best values. It mainly finds the amount of correction applied for which filter adapts from one iteration to the next. Though, selecting the appropriate step size (δ) [9] is not always easy task, which needs much experience in adaptive filter design. The step size (δ) of the adaptive filter should range from neither too small nor too large, as it measures the performance rate of the adaptive filtering algorithm in terms of the varying convergence rate, satiability and mean square error (MSE) generated [10]. If the step size (δ) is large then coefficients of the adaptive filter causes divergence instead of convergence and hence the filter will not be able to eliminate noise and hence lessen error. This causes a serious issue with stability, as the resulting filter might not remain stable. On the other hand when the step size (δ) is too little then the convergence time of the adaptive filter coefficients increases which makes the adaptive filter slow in eliminating noise and error. This causes an issue of speed and accuracy [11]. As a rule of thumb, smaller step size tends to improve the accuracy of the filter to converge to the characteristics of the unknown system but at the cost of the increased time. This problem of choosing a step size (δ) can be solved by using a variable step size (δ). In case of a variable step size adaptive filter algorithm we have choice between two or more step sizes. A larger step size (δ) is utilized when a signal is changing slowly and a smaller step size (δ) is used when the signal is changing at fast rate [12].

1.2 NEED OF ADAPTIVE FILTER

Adaptive filters are required when some parameters of the desired processing operation are not known in advance. In other words it can be said that adaptive filters are used in the situations where the statistics of the signal are unknown or changes with time. This happens mostly in the case of random or non-deterministic signals. Thus, it can be said that adaptive filters are used mostly for de-noising of random or non-deterministic signals [13], [14]. As most of the random or non-deterministic signals occur in real time situations such as in case of an airplane cruise where the noise characteristics continuously changes with planes speed, height and environmental conditions. In such a case it is imperative to remove noise from the pilot's mic, so that there is a proper communication between the pilot and air traffic control (ATC) tower. Adaptive filters come handy in these situations. Thus, an adaptive filter may be understood as a self-modifying digital filter that adjusts its coefficients in order to minimize an error function.

1.3 APPLICATIONS OF ADAPTIVE FILTER

There are four main applications of adaptive filters which are explained below:

System Identification[15]: The adaptive filter's system identification process is achieved by determining a distinct estimation of the unknown transfer function of an unknown digital or analog system. The same input x(n) is applied to the adaptive filter as well as the unknown system and then both the outputs are compared as in Figure 1.2. Then the output of the adaptive filter y(n) is deducted from the output of the unknown system which is a desired signal d(n). The resulting difference is an error signal e(n) which changes the filter coefficients in order to attain error signal of zero magnitude. After a number of iterations are being perfumed and if the system's is correctly designed, the adaptive filter's transfer function will tend to converge to that of the unknown system's transfer function.

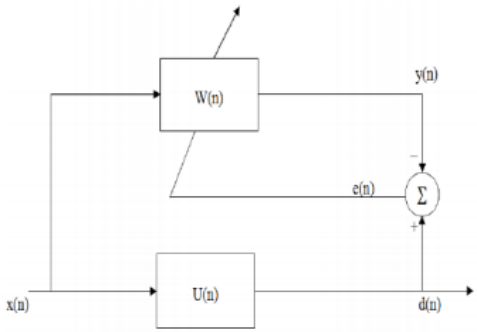


Figure 1.2 System identification configuration

Adaptive Noise Cancellation[16]: In the configuration shown in Figure 1.3, the input x(n) and noise source N_1(n) are compared with a desired signal d(n), which consists of a signal s(n) altered by another noise N_0 (n). The adaptive filter coefficients converges the error signal to be a noiseless version of the signal s(n). For this configuration, both noise signals are uncorrelated to the signal s(n). Additionally, the noise sources must be correlated to each other in some way to get the best results.

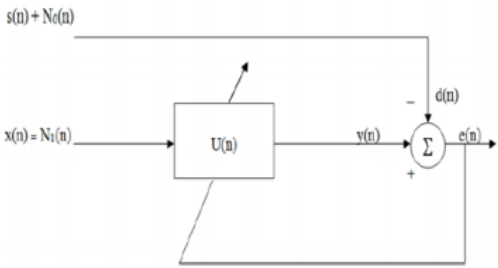


Figure 1.3 Adaptive Noise Cancellation System

Adaptive Linear Prediction[17]: This is a configuration of adaptive filter that is used to predict future values of a signal from its past values. It is shown in Figure 1.4. To achieve this goal the same input signal is applied to both the input port and reference or desired port i.e. d(n) = x(n). Also, the input to the adaptive filter is the delayed version of the system input. The adaptive filter

coefficients are being trained to minimize the error and hence predict the next input signal from the statistics of the input signal x(n).

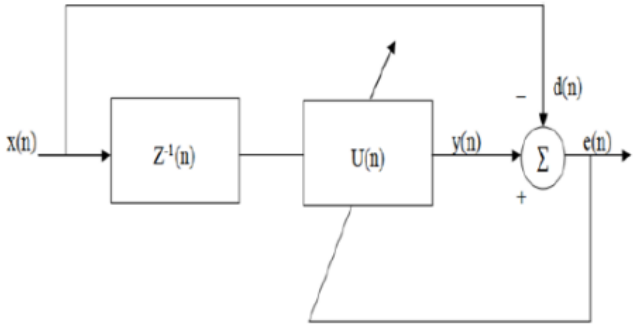


Figure 1.4 Adaptive Linear Prediction

Adaptive Inverse System Configuration[18]: This configuration is shown in Figure 1.5 .It models the inverse of any unknown system say u(n). First, the input x(n) is passed through the unknown filter u(n) and the output of this filter is passed through the adaptive filter generating an output y(n). Secondly, the input signal is delayed by passed through a specified delay to form d(n). The error signal e(n) is considered as the difference between signals d(n) and y(n). As the error signal tends toward zero, the adaptive filter coefficients w(n) starts converging to the inverse of the unknown system u(n). In this configuration, the error can theoretically reduce to zero.

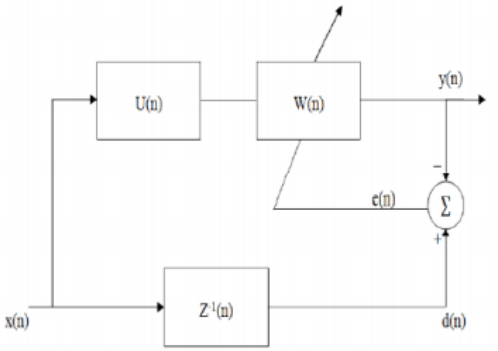


Figure 1.5 Adaptive inverse system

1.4 THE LMS ADAPTIVE ALGORITHM

Widrow and Hoff had proposed the Least Mean Square (LMS) algorithm in 1960 [1] based on steepest decent method [2], [5]. LMS algorithm mechanism is based on the estimation of the gradient of the input vector. It utilizes an iterative procedure to update the filter coefficients or to update the weight vectors in the direction of negative of the gradient vector as it leads to decrease in the mean square error (MSE) which is the main motive. This algorithm is more preferred as compared to other adaptive algorithms because of its computational simplicity, stability and suitable convergence property. For each applied input, the LMS algorithm computes an output, and determines the error which is the taken as the difference between the calculated output and the desired response, and finally this error is used to update the weights and filter coefficients in every iteration cycle.

Consider an N order LMS adaptive filter having input x(n) and weights or coefficients w(n). At the end of the nth iteration the input x(n) and weights w(n) can be represented as

$$x(n) = [x(n), x(n-1) \dots x(n-N+1)]^T \quad (1.7)$$

$$w_n = [w_n(0), w_n(1) \dots w_n(N-1)]^T \quad (1.8)$$

By using the property of convolution the output y(n) of the adaptive filter at some discrete time n is given by following equation

$$y(n) = \sum_{k=o}^{N-1} w(n)x(n-k) \quad (1.9)$$

The output of the adaptive filter can also be represented in vector form as

$$y(n) = w^t(n)x(n) \quad (1.10)$$

The error signal e(n) which is the difference between the desired or reference signal d(n) and filter output is given as

$$e(n)=d(n)-y(n) \quad (1.11)$$

On squaring error signal e(n) we get the following equation

$$e^2(n) = d^2(n) - 2w^t(n).x(n) + w^t(n).x(n)x^t(n)w(n) \quad (1.12)$$

In order to optimize the adaptive filter, the mean square error (MSE) must be minimized. For this the MSE must be represented in terms of cost function J.

$$J = E[e(n).e^*(n)] = E[|e(n)^2|] \quad (1.13)$$

Taking gradient Δ of the cost function, a gradient vector Δ is obtained as

$$\Delta J(n) = -2P + 2Rw(n) \quad (1.14)$$

or

$$\Delta J(n) = -2x(n)d(n) + 2x(n)\, x^t(n)w(n) \quad (1.15)$$

Where, R is the autocorrelation matrix of x(n) and P is the cross correlation matrix of d(n) and x(n).

Considering a cost function J(w) i.e. a continuously differentiable function of some unknown weight vector w. To find an optimal solution suppose a initial weight w(0) that satisfies the condition

$$J(w(0)) \leq J(w) \text{ for all } w \quad (1.16)$$

Equation (1.10) is a mathematical statement of unconstrained optimization. As, w(0) is the initial weight (assumed) and after each iteration new weights w(1), w(2).....w(n-1) are generated in such a way that the cost function J(w) is reduced at each iteration. Thus,

$$J(w(n+1)) \leq J(w(n)) \quad (1.17)$$

where, w(n) is the value of weight at previous iteration and w(n+1) is the present value of weight, by substituting equation (1.9) in steepest decent algorithm equation a new recursive equation is obtained for updating weights which is given below

$$w_{n+1} = w_n + \mu.e(n).x(n) \quad (1.18)$$

Equation (1.12) is the governing equation for the LMS algorithm which is used for updating the weights. Here a new parameter also known as step size is introduced in the equation, the function of which is to control the step width of the iteration and thus determines the stability and convergence speed of the adaptive algorithm [15],[17].

In case of LMS algorithm MSE is a quadratic function of filter weights which means that there is only one possible (that is optimum) weight which can minimizes MSE to optimum desired value.

It approaches towards these optimum weights by increasing or decreasing the MSE vs. filter weight curve [17-21].

1.5 CHALLENGES IN LMS IMPLEMENTATION ON VLSI TECHNOGOLY

Digital signal processing (DSP) applications have numerous advantages over analog signal processing functions, as word length of the digital signal can be changed which enhances the provision to control its property of convergence and accuracy. Further-more, DSP techniques can eliminate the noise and external interference while amplifying the signal in contrast to analog where both original signal and unwanted noise are amplified. Digital signals can be stored and recovered, transmitted and received, processed and manipulated, all without any error. While analog signal processing is indispensable for systems that needs extreme high frequencies such as the radio frequency trans-receiver in wireless communications, or extremely low area and low power such as micro machine sensors used to detect cracks and other stress-related material defects, that's why all the complex systems are digitally realized keeping high precision, high signal to noise ratio (SNR), high repeatability and more flexibility.

DSP systems can be implemented in programmable processors or in traditional designed hardware circuitry fabricated in very-large-scale-integrated (VLSI) technology. Low-complexity designs have more importance in efficient hardware implementations, to maximize the performance while making the designing cost small. In case of general digital designs, performance is determined by the amount of hardware circuitry and resources required (i.e., space or area); the speed of execution, depending on the throughput and clock rate; and the amount of power or total energy required to perform a specific task.

Digital filters with fixed-coefficient are considered as ideal for frequency shaping in adaptive digital filters which predict one random process from observations of another random process using linear models such as digital filters .Hence, major challenges in LMS implementation are:

I. **More Area utilization**: The adaptive filter is generally constructed by employing the multiplier units, adders, large memories, and so on, which causes more area utilization. Moreover, multipliers have long latency and large power requirements.

II. **Lower Speed**: Since multiplier lies in the critical delay path and determines the performance of processor, speed of multiplication operation speed is main parameter in

DSP operations as multiplier is normally the slowest element in the system. Though, they are necessary part of modern electronic systems where high speed computations are required such as in FIR filters, digital signal processors and microprocessors, but generation of square and cube of a number typically require a large number of multipliers and the time taken by multiplication operation is still the major factor in determining the instruction cycle time of a DSP chip and hence the operating speed.

III. **High Power Consumption**: Power being a valuable resource in modern VLSI design, even more than area, plays a vital role in performance measures of filters. With large number of applications implemented in functional units like squares, cubes and other higher order units, it becomes crucial to implement such functions on hardware. Implementing these functions, using existing general-purpose multipliers, may seems economical in terms of area but needs extremely high power than is necessary.

1.6 OBJECTIVES AND NOVEL ASPECT OF THIS DISSERTATION

Concisely this dissertation attempts to achieve the following objectives:

I. Distributed Arithmetic based Block LMS adaptive filter implementation which utilizes LUT's to obtain filter output and weight increment computation in order to minimize area utilization and delay at the same time.
II. As these LUT's have to be updated in each iteration, they consume considerable amount of energy, which generates need for fast and computationally efficient Block Least Mean Square Adaptive Filter (BLMS-ADF) in which convolution and correlation are performed by same LUT, making the design area as well as energy efficient.
III. To make an efficient and dedicated squaring and cubing unit which can significantly reduce the delay and increase the speed of execution.
IV. In order to control power dissipation, a full adder unit is to be implemented in NMOS technology which will result in significant reduction in transistor count as well as power.

The novel aspect of this dissertation is that all the performance parameters are improved by implementing the new proposed designs which have not implemented yet to the best of author's knowledge. First, DA based BLMS design is proposed in which an considerable weight updating

scheme, by making use of LUTs repeatedly for reducing LUT access per output (LAPO), is recommended to manage the overall power consumption of the structure. Further, dedicated squaring and cubing units are designed using partial product reduction techniques in order to reduce the height of partial product matrix. Then, NMOS pass transistors are employed for device level design of full adder unit which is the major part of fast cubing unit. Introduction of NMOS pass transistor significantly reduced the total number of transistors required to implement the adder unit which in turn has resulted in reduced power dissipation and area optimization.

1.7 ORGANIZATION OF THE DISSERTATION

The main aim of this dissertation is to design competent and simplified Multiplier architecture which is area, speed and power efficient. In order to achieve this, the dissertation is divided into following five chapters:

Chapter 2 presents the literature review which summarizes the research work done till date related to this topic.

Chapter 3 gives the overview of implementation and importance of Distributed Arithmetic based Block LMS implementation for area and delay optimized design.

Chapter 4 elaborates the need and significance of fast multiplier units and explains the proposed dedicated multiplier units with reduced partial products and more throughputs having low latency. Also, the full adder unit is implemented using NMOS technology which has significantly reduced the transistor count and power.

Chapter 5 deals with final experimental results and discussions implemented on XILINX ISE 14.5 using Verilog and VHDL languages as well as Cadence-virtuoso ADE which will show the actual reduction in area, speed and power.

Chapter 6 sums up the conclusions of the work and suggests some ideas for future endeavor.

CHAPTER 2
LITERATURE REVIEW

Widrow *et. al.* [1-3] had explored the use of learning algorithms both linear and nonlinear adaptive filters, to be used in significant number of applications. It had studied and found that the adaptive filters mainly comprises of the adaptive linear combiner with a set of applied input signals, which are later multiplied by variable coefficients or some weights to get the output signal. This received signal is then compared with a required signal which is applied during the updation process. It had defined the error signal as the difference between the desired output and the actual output. When the input and the equivalent desired-response input are applied to the linear combiner during updation in the iteration process, the weights are so adjusted that it minimizes the mean-square-error. Thus, the linear combiner attempts to produce the best possible least squares estimate of the linear desired response. These functions form a fundamental in learning for adaptive filters and neural networks.

Soria *et. al.* [4] had given a new adaptive algorithm which is strong against the impulse noise and requires low computational load. The algorithm is based on two factors: the cost function used in component analysis and hyperbolic tangent modeling function. The conservation method is used for detection of steady-state error and tracking capability of the algorithm. It had measured the performance of the algorithm by making the use of cost function using the mean square error as its preferred for the LMS algorithm. Using this cost function, the author had made complete assumption about the error produced as Gaussian and then justifies it by the central limit theorem. However, due to the noise, present at either the input signal or at the desired signal which may make the adaptive algorithm to become unstable, an alternative based this convex combination of cost function is utilized.

Lee *et. al.* [5-6] had proposed a new low-pass notch filter PLL (LPN-PLL) control strategy to lessen the power consumption of the grid core of distributed energy resources (DERs) to optimize the grid code and grid-connected inverters based on DER's a phase-locked loop (PLL), which is used for measuring the accurate grid phase angle whenever the grid voltage constitutes harmonics and causes imbalance. This novel filter matches with the true phase angle of the grid instead of using a predictable reference frame of PLL (SRF-PLL. This proposed LPN-PLL is an extended

version of the PLL model, hard to the harmonics and imbalance caused in the grid voltage, using the fast Fourier transform concept (FFT-PLL) .This proposed algorithm is then compared with conventional SRF-PLL and FFT-PLL and is executed digitally using a digital signal processor and a verification experiment which reveals the high performance and robustness under applied low-voltage operation.

Jimaa *et. al.* [7] had measured the accuracy of mean square error (MSE) for different adaptive filtering algorithms in the specific process of system identification which is primarily done on two defined channels. The MSE results of the proposed algorithms are then compared with that of the traditional Normalized Least Mean Square (NLMS) algorithm. To extend this performance, non-mean-square algorithms have been used, where Least Mean Fourth (LMF) algorithm serves as switching algorithm. In this paper the author had made use of random step sizes and the NLMS algorithm made switching between NLMS, LMF and LMS+F mixed norm over two specific Finite Impulse Response (FIR) channels then results are compared to standard NLMS.

Allred [22-23] had presented Adaptive filtering as a important class of DSP algorithms used in many mobile devices for wide applications such as echo cancellation, signal de-noising and channel identification and equalization. This paper had presented a special pipelined architecture for low-power accomplishment of Adaptive filter based on distributed arithmetic (DA) technique. For Distributed Arithmetic based computation of inner-product, the traditional adder-based shift accumulation is replaced by conditional signed carry-save accumulation. A fast bit clock is employed in this technique for carry-save accumulation only, which resulted in reduction of power in the proposed design, while much slower bit clock is used for rest of the operation. It contained smaller Look-Up Table (LUT) but with same quantity of multiplexers and almost half the number of adders in comparison to the traditional Distributed Arithmetic-based design. Thus, by changing the inner block, it had reported reduction in power consumption by implementing DA-based adaptive filter for different filter lengths.

Mohanty [24-27] had suggested an competent distributed-arithmetic (DA) based implementation of block least mean square (BLMS) algorithm by employing look-up table (LUT)-sharing aspect which is a novel technique for calculation of filter outputs and weight-increment terms in corresponding BLMS algorithm. Apart from this, it has also offered significant saving of in number of adders being the major components of DA structures. In this projected new aspect of LUT-based

weight increment technique for BLMS algorithm, only one set out of the N LUTs set need to be altered during every iteration, where N=ML , M and L being the filter length and input block-size, respectively. Based on this proposed formulation, the author has found a resultant parallel architecture for the efficient implementation of BLMS which is then compared with existing DA-based LMS structures, showing huge depreciation in number of adders and LUT words to nearly half times, and offers nearly twice the throughput of the existing ones. It does not engage variable shifters like those in traditional structures, but needed nearly 25% more flip-flops. It had involved less LUT access per output (LAPO) than the present structure for block-size greater than 4.

Dauglas *et. al.*[28] had shown an analysis on calculation complication in block least mean square (BLMS) for finite impulse response (FIR) filters and has divided the filter calculations into further M sub-filters, where M = N/L, N being filter length and L the block-size. Each sub-filter is now acting as a short-length BLMS FIR filter of each size L. The proposed decomposition scheme also supports time multiplexing for the filtering process and weight-updation of each short-length filter. Using this proposed scheme, the author has derived an efficient architecture for BLMS FIR filter which is reconfigured for diverse filter lengths but with slight overhead complexity and variable convergence factor μ. Besides, the structure had 100% hardware utilization efficiency (HUE) and register complexity independent of block-size, but required L times more multipliers, few adders an same number of registers, offering L times more throughput. Because of register and adder reduction, this anticipated structure had notably less area-delay product (ADP) and energy-per sample (EPS) than the existing structure. ASIC synthesis results had shown that 22.4% less ADP and 25.6% less EPS than those of the existing structure for block-size 4 and filter length 64 and had offered 3.8 times higher throughput.

Allred *et. al.* [29] had explained the role of adaptive filters in signal processing application and algorithms for implementation of such filters like Least mean square (LMS), Recursive least square (RLS), etc. The author has proposed the LMS algorithm as the most competent algorithm for implementing adaptive filters whereas RLS algorithm gives faster convergence on the head of increased computational complexity. Thus, an effective distributed arithmetic based architecture is suggested to implement the block least mean square algorithm (BLMS) using LUT sharing method to compute the filter output and weight increment terms in BLMS algorithm which had saved

significant count of adders. This paper has also presented a literature survey on the different possible algorithms for FIR adaptive filters using distributed arithmetic and block LMS algorithm.

Baghel *et. al.* **[30-31]** had designed a wireless transmission based system for justifying the presence Inter Symbol Interference (ISI) which primarily occurs because of propagation of the transmitted signal on multipath by designing Adaptive decision feedback equalizers (ADFEs). These rapidly converging and simple ADFEs have been employed in high speed communication having rapidly varying channels. Frequency based block processing of signals is utilized as an efficient mean for managing complexities of high speed systems. However, as block ADFEs are non-causal, the feedback filter (FBF) had to suffer from unknown decisions during every block's calculation. In this study, the author had proposed a competent approach for the calculation unknown decisions by minimizing the cost function based on two criteria's namely mean-absolute difference (MAD) and mean-square difference (MSD) between the ADFE output and feed forward filter (FFF) and FBF summation result. A depository of registers to save all the symbols is also employed to rephrase the block ADFE equations in the frequency field by means of distributed arithmetic (DA). The algorithm had shown a excellent convergence performance by using only few computations as of existing schemes.

Woolley *et. al.* **[32]** had implemented Decomposition logic in digital multiplier to improve their speed and to decrease the critical path delay using Baugh-Woolley algorithm. In this paper the author has proposed a high speed multiplier design to compare it with booth multiplier, implemented using Xilinx 12.3 device. As, in booth signed multiplication the length and count of the partial products was very high, that's why the author has chosen this algorithm. The Baugh-Woolley multiplication is considered amongst the cost-effective ways to handle the sign bits. This method also aimed at designing regular multipliers which are suitable for 2's compliment numbers also.

DADDA **[33]** had focused on power efficient multiplier architecture as the Multipliers being fundamental components in all digital processing systems form heart of many applications. As in previous literature, many research efforts have been dedicated to power dissipation reduction in different multipliers, the major contribution to the power consumption in made during generation and reduction of partial products. Among many multipliers, tree multipliers are preferred in high

speed applications such as in filtering, but these utilize large area. The carry-select-adder (CSA)-based radix multipliers though have low area requirement but occupies large active transistors for the multiplication operation and hence eats more power. Hence in this paper, a new power competent VLSI architecture had been proposed as a DADDA multiplier using schematic editor in tanner tool, T-spice and then compared with traditional multipliers.

Dauglas *et. al.* [34] had proposed an hardware-efficient pipelined structure for the LMS adaptive FIR filter similar to standard LMS adaptive filter architecture but with the only difference that it will produce the output and error signal without adaptation delay. In past implementations of least-mean-square (LMS) adaptive filter for parallel and pipelined architectures had either caused delay in the coefficient updation process or had huge hardware requirements. So, different from existing delay less LMS adaptation architecture, the new proposed architecture's output is independent of the filter length. There are two commonly used algorithms for realization of adaptive filters; namely the Recursive Least Square (RLS) and Least Mean Square (LMS) algorithm. Though, RLS algorithm gives better convergence performance than LMS, but on the expenses of increased computational complexity which causes obstruction towards its real time application. Alternatively, the LMS algorithm is a simple and robust way to achieve good performance at the cost of lower convergence rates, for which Block LMS algorithm is provided as an improvement. In addition to this, distributed arithmetic (DA) approach is used to implement the Block Least Mean Square (BLMS) algorithm for better computationally efficiency using bit-serial operations and look-up tables (LUTs) sharing for weight updation of BLMS algorithm to get high throughput filters. This paper also focuses on different variant algorithms for implementing FIR adaptive filters using Distributed Arithmetic and Block LMS algorithm.

Das *et. al.* [35] had given a better block formulation technique for an active noise control (ANC) system by making use of only convolution operation. This proposed approach is different from conventional block least-mean-square (LMS) algorithms which utilizes both convolution and cross-correlation operations. This block execution causes filtering of any reference signal by the means of secondary-path estimate by using Hartley transform (FHT) to develop transform-domain ANC structures for reducing computational complexity, instead of fast Fourier transform. Later, some of the FFT and FHT blocks were removed resulting in the reduced-structure of FFT-based block

filtered-X LMS (FBFXLMS) and FHT-based block filtered-X LMS (HBFXLMS) algorithms to get additional decline of the computational overhead.

Meher [36] had found that in the coefficient updation process, the step size had a key role in determining the convergence and stability of the algorithm, so an upper bound for the step size is derived by using Delayed LMS (DLMS) algorithm. Here, relationship between the step size and the convergence speed, and its effect on the delay in the convergence speed is also studied Any novel partial product generator (PPG) works to improve adaptation delay and area-delay-power so this proposed architecture comes into the picture where the authors have used a novel PPG for implementing general multiplication process and inner-product computation by using common sub expression sharing. Thus, power efficient, low adaptation-delay and low area-delay architecture for implementing fixed-point LMS adaptive filter has been intended.

Thapliyal *et. al* [57] made use of Ancient Indian Vedic Mathematics algorithms for the hardware implementation of RSA encryption or decryption algorithm with proposed hierarchical overlay multiplier architecture. The important aspect of this paper lies in expansion of this architecture to division unit based on Straight Division algorithm of Vedic Mathematics embedded in RSA encryption/decryption circuitry for better efficiency. The Verilog HDL and FPGA synthesis have shown the area and speed accuracy of RSA circuitry compared to its implementation on traditional multiplier and division architectures. Due to the parallel and regular structure of this proposed architecture, it can be efficiently implemented on silicon chip which worked at improved speed without any change in the clock frequency, which improves its gate delay and area utilization compared to RSA traditional multipliers and division algorithm circuitry.

Kunchigi *et. al.* [58] had implemented a Multiply and Accumulate (MAC) unit using Vedic Multiplier, based on Urdhva Tiryagbhyam Vedic Sutra. The paper has focused on an efficient designing of 32-bit MAC architecture with 8-bit and 16-bit design and comparison results are presented. The reduced area, high speed, low critical delay and low hardware complexity had made this as an acceptable design. This MAC unit has shown significant reduction the area by sinking the quantity of multiplication and additions performed in the multiplier unit. Boost in the operational speed is due to hierarchical nature of the Vedic multiplier unit implemented on a field programmable gate array (FPGA). In literature, Vedic mathematics methods had been implemented

for multiplication but this proposed design of 32-bit MAC unit had significant improvements in the delay and area prospective.

Stine [59] had presented a high-speed and memory efficient method for calculating elementary functions by employing parallel lookup tables and multi-operand additions which had reported significant decline in the memory requirement by using symmetric and leading zeros in the coefficients of the table. As, this method had shown a general form solution for reducing the table entries, it is applicable to any differentiable function. For instance, this method required two to three times less memory than conventional table methods for a 24-bit operand. This paper is based on Symmetric Table Addition Method (STAM) for elementary function approximation and comparison with standard table lookups, and had reported dramatic reduction in the memory utilization. The STAM is able to overcome memory requirements of standard table method because of its symmetric and leading zeros quality. This architecture is implemented with simple hardware that could operate over wide range of functions with low to moderate precisions, with accurate initial approximations, needed for improved converging rates of divide and square root algorithms.

Ercegovac [60] deals with the computation of general elementary functions like reciprocals, square roots, inverse square root functions using lesser tables, smaller multipliers and full length multiplication operation. This method aimed at computation reduction and series expansion to achieve fast evaluation with high precision. The strength of this method is that all the above mentioned functions can be implemented with same proposed scheme. For estimating delay, the constituents' size or count is compared with traditional methods. The author had proposed a single scheme for calculating reciprocals, square-roots, inverse square-roots, logarithms and exponentials. Though, for reciprocal operation in double precision, computational delay has increased, for square root and for inverse square root it had offered faithful rounding.

Schulte *et. al.* [61] had proposed an efficient designs for multiplication and squaring unit which performs its operation depending on an input control signal, because multiplication and squaring are frequently used operations in various digital signal processing and multimedia applications. In contrast to conventional parallel multipliers, these units permit either multiplication or sum-of-squares computations to be performed with modest area and delay increase. In this paper, the author had suggested collective multiplication and sum-of-squares units for both unsigned and two's complement numbers, along with the incorporated designs which are operational in both

unsigned and two's complement operands. The design is further extended by introducing third accumulator operand in the design to perform additions along with other reported operations. Synthesis results had clearly shown its efficiency and operational accuracy in comparison to old methods by occupying roughly 15% more area and almost the same worst case delay but with significant saving in the computation time, which was the main motive of the author.

Stine [62] had shown that the required number of partial product bit generators required for squaring a number can be significantly reduced, by about ½ as compared to the conventional squaring circuit, by using different partial product reduction techniques, explained in this paper, to make the associated Wallace tree structure simple and small and hence fast execution can be implemented because of the reduction in partial product bits. The technique behind this methods explains that a circuit for squaring an n-bit value needs a partial product bit generator arrays which will perform logical AND operation on a bit of weight 2^k (k is an integer) with the same bit 2^k to generate a partial product bit of weight 2^{2k}. Any other partial product bit generator array will simply receive n-bit value of weight 2^k and logically AND it with another bit of weight 2^m (m is an integers) to produce bit of weight 2^{k+m+1}. This second partial product bit generator array is the only generator array in the squaring circuit which perform logical AND of two different bits having weight 2^n and 2^k, respectively.

Kunchigi *et. al.* [63] had proposed a high speed 3 staged pipelined multiplier architecture where 1st stage comprises a 4 -bit Vedic Multiplier, 2nd stage constitutes of generated partial products and corresponding generated carry and final 3rd stage is of adders to form result of the multiplication. This paper had utilized the competency of Urdhva Triyagbhyam Vedic sutra for performing multiplication which is distinct than actual process of traditional method of multiplication as it discards the unwanted multiplication steps to produce parallel partial products. The proposed method is encoded using Verilog, a hardware description language, for computerized synthesis on Xilinx FPGA device, Sparten-3E. The synthesis had clearly shown that only 2 logic cells are necessary to build nibble multiplier so it will direct affect the propagation time of the proposed architecture, which is later found to be 4.585ns also efficiency with respect to area and speed is shown compared to traditional methods using Array and Booth multiplier circuits.

Liddicoat [64] had suggested that a dedicated squaring circuit is mandatory for fast and efficient multiplication then a traditional multiplier can do, so the author had proposed different dedicated

multiplier unit designs, which can be employed to calculate the square and cube of an operand. This paper specifically had proposed a parallel cubing unit which had shown that it can compute the cube of any operand up to 25 to 30% faster than can be done by traditional circuit using multipliers. Furthermore, the improved and modified squaring and cubing units are mathematically modeled for achieving better performance in terms of area requirements based on the operand length. The operational capability of the proposed dedicated squaring and cubing circuit is implied with relevance to the Newton-Raphson and Taylor series function evaluation units.

Blank [65] had discussed the modifications in the above proposed parallel cubing unit by suggesting distinct architecture for a variety of operand sizes ranging from 8 to 32 bits. This method, proposed by the author, decomposes the partial product matrix of cubing operation into few small elements and further organizes these partial products into manageable and repeated groups. As a result, the overall partial product matrix becomes considerably small in contrast to previous method. This algorithm also resulted in area and delay reduction, after analyzing its computational efficiency on Xilinx FPGA Virtex 5, for different operand lengths.

Wires [66] had mentioned that when squaring is performed using special squaring circuit rather than traditional multiplier it will cause significant reduction in area, delay and power consumption, is squaring is a significant process of various digital signal processing applications. Although, many researchers have successfully implemented parallel squaring circuits, but here the author is more concerned about design of unsigned squaring method, as squaring of two's complement numbers is frequently visited in literature. So this paper had implemented the parallel squaring multiplier that can compute either unsigned or two's complement squaring, based on applied input control signal. Moreover, the requirement of area and associated delay has been reduced to great extent as compared to unsigned parallel squarer. This combination supporting computation of both unsigned and two's complement square in a single architecture has wide utilization in ubiquitous digital signal processing applications.

CHAPTER 3

IMPLEMENTATION OF BLMS ALGORITHM BASED ON DISTRIBUTED ARITHMETIC

3.1 INTRODUCTION TO DISTRIBUTED ARITHMETIC

In recent years, adaptive filters have been widely used in many applications, for example in echo cancellers, as noise canceller, as adaptive equalizers and so on, and the need to implement the same is growing in many fields. Adaptive filters require performance in terms of high speed, lower power dissipation, good convergence properties and small output latency etc. The echo-canceller can be used in the video conferencing needs a fast convergence property and the capability to follow the corresponding time varying impulse response. Therefore, very high order adaptive filter implementation is the need of the hour. In order to satisfy these requirements, highly-efficient algorithms and architectures are desired. The adaptive filter is generally consist of the multipliers, adders, memories whereas the LMS adaptive filter based on the distributed arithmetic can be realized by employing adders and memories without use of multipliers, that is, it can be achieved with a compact and small hardware. To implement this High Speed Digital VLSI Signal Processor systems on field programmable gate array (FPGA) is preferred hardware platform whereas Application Specific Integrated Circuits (ASIC) are also used tremendously these days. FPGA has the major reward over traditional DSPs and ASICs in terms of project cost, flexibility, reconfigures ability and reliability [22].Therefore, we have chosen the distributed arithmetic algorithm instead of multiply and Accumulative Unit (MAC) to enhance the performance of FIR Filter. Also, because of the complexity of the optimizing algorithm for the adaptive filter designing, majority of the adaptive filters are digital filters that perform digital signal processing and adapt their performance based on the input signal. Adaptive filter are required when either the fixed specifications are unknown or time invariant filters are not able to satisfy the specifications.

3.2 DISTRIBUTED ARITHMETIC TECHNIQUE

Distributed arithmetic (DA) based design approach measures, to develop low complexity hardware structures, inner-product calculation involving a constant vector. DA was studied by Croisier (1973). Subsequently, White (1989) developed mathematical formulations to execute inner product

computation using DA and developed structures to perform efficient computations of DA algorithm using logic and memory resources [23-24]. The inner product computation of a constant vector A with an input vector X of each length P can be expressed as:

$$Y = \sum_{i=0}^{P-1} A_i X_i \tag{3.1}$$

Where A_i and X_i are the i^{th} component of vector A and X respectively, and X_i is a W- bit 2's compliment binary fraction expressed as:

$$X_i = -x_{i,0} + \sum_{j=1}^{W-1} x_{i,j} 2^{-j} \tag{3.2}$$

Where $x_{i,0}$ is the sign bit and $x_{i,W-1}$ is the LSB. Substituting equation [3.2] in [3.1] we get:

$$Y = \sum_{i=0}^{P-1} A_i(-x_{i,0} + \sum_{j=1}^{W-1} x_{i,j} 2^{-j}) \tag{3.3}$$

Rearranging the summation order, we have:

$$Y = -\sum_{i=0}^{P-1} A_i x_{i,0} + \sum_{j=0}^{W-1}[\sum_{i=0}^{P-1} A_i x_{i,j}]2^{-j} \tag{3.4}$$

$$Y = \sum_{j=0}^{W-1} sign_j C_j 2^{-j} \tag{3.5}$$

Where $sign_j = 1$ for $1 \leq j \leq W-1$

$sign_j = -1$ for $j=0$

C_j For $0 \leq j \leq W-1$ is defined as:

$$C_j = \sum_{i=0}^{P-1} A_i x_{i,j} \tag{3.6}$$

Equation 3.5 can be expressed as memory read operation as:

$$Y = \sum_{j=0)}^{W-1} sign_j \, F(x_j) \, 2^{-j} \tag{3.7}$$

Where F (.) is the memory read operation function and $x_j = [x_{0,j}, x_{1,j}, \ldots x_{P-1,j}]$ is the P-point bit-vector (bit-slice).

In this applied LMS algorithm, both the input-vector value and the weight-vector value will changes during each iteration. Therefore, the DA based scheme, shown in Figure 3.1, cannot be applied directly in the LMS algorithm. Allred et al. (2005) had studied that LMS algorithm and all alternate schemes implementing LMS adaptive filter on DA scheme [25]. Based on this proposal, two LUTs are required, DA filtering LUT (DA-F-LUT) and DA auxiliary LUT (DA-A-LUT) to perform DA computations of LMS algorithm as shown in Figure 3.2.

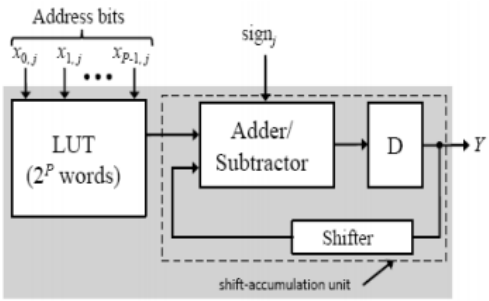


Figure 3.1 The block diagram of DA computation of inner-product of two P-point vectors.

DA- F-LUT is known to stores the limited results of the weight-vector to perform filtering operation while the DA-A-LUT stores the partial results of input-vector to perform calculation of the weight updating operation. Both DA-F-LUT and DA-A-LUT are implemented in random access memory (RAM), as the intermediate results stored in both the LUTs change their values during each iteration. The complexity of this RAM-based LUT is considerably higher than the complexity of ROM-based LUT because area, delay and power consumption of RAM based LUT is more than ROM based LUT requirements. Also the RAM LUT method introduces large amount of delays for updating the content. Therefore, the LUT resources and LUT updating time constitute the main design constraint for the DA-based structure of LMS adaptive filter. Several attempts have been made in past to reduce this LUT complexity of DA-based structure of LMS adaptive filter [25].

0
$w(n)(0)$
$w(n)(1)$
$w(n)(1) + w(n)(0)$
$w(n)(2)$
$w(n)(2) + w(n)(0)$
$w(n)(2) + w(n)(1)$
$w(n)(2) + w(n)(1) + w(n)(0)$

DA-F-LUT

0
$x(n\text{-}2)$
$x(n\text{-}1)$
$x(n\text{-}1) + x(n\text{-}2)$
$x(n)$
$x(n) + x(n\text{-}2)$
$x(n) + x(n\text{-}1)$
$x(n) + x(n\text{-}1) + x(n\text{-}2)$

DA-A-LUT

Figure 3.2 LUT contents in the DA-F-LUT and DA-A-LUT for filter length-3

3.3 BLOCK LMS ALGORITHM

An Adaptive Filter is a linear system with controlled transfer function by variable parameters according to applied optimization algorithm. Due to its complexity, most adaptive filters are digital filters. They find their usage in the Digital Signal Processing applications like noise cancellation, echo cancellation, system identification and channel equalization [26].Least Mean Square (LMS) based Finite Impulse Response (FIR) Filter is the most preferred one due to its simplicity and acceptable convergence behavior [27-30]. Traditional LMS based adaptive filter utilizes direct form structure which does not support pipeline structure due to its recursive behavior and thus produces fewer throughputs [31]. To support pipelined implementation, an adaptation delay needs to be introduced in the error value of LMS (which then becomes DLMS) lowering the error performance. Modified DLMS has been introduced by Poltmann [32] to eliminate performance barriers and cause area-delay efficient implementation of Distributed Arithmetic based LMS adaptive filter design for obtaining filter output and weight increment updating [33-38]. But as these LUT's have to be updated in each iteration, considerable amount of energy is being consumed. For fast and computation proficient ADF's, Block Least Mean Square Adaptive Filter (BLMS-ADF) is one of the most preferred structure where convolution and correlation are performed using FFT/IFFT [39]. This approach of designing efficient VLSI architecture for BLMS adaptive filter is multiplier based [40], where it is found to take the advantage of analogous area-delay efficient multiplier-less design. This multiplier based ADF structural design is directly formed from traditional LMS by introducing L-fold parallelism, which supports adaptive filter

implementation regarding various lengths at low cost, alternative to fixed length ADF's as implementation of these separate ADF's are expensive and time consuming. It also supports the following features:

i) Lesser Area

ii) Maximum throughput

iii) 100% Hardware Utilization Efficiency (HUE)

iv) Reduced Hardware complexity.

3.4 BLMS TECHNIQUE

The BLMS ADF compute one block of output (y_k) and one block of error (e_k) from the available one block of inputs and one block of desired response (d_k) during every iteration. This BLMS algorithm for updating the weights for the $k+1^{th}$ iteration is mathematically represented by:

$$w_{k+1} = w_k + \Delta w_k \tag{3.8}$$

Where $$\Delta w_k = \mu X_k^T e_k \tag{3.9}$$

Here, w_k *and* e_k are the weight-vector and error-vector, respectively defined as:

$$w_k = [w_k(0)\ w_k(1) \dots w_k(N-1)]^T$$
$$e_k = [e(kL)\ e(kL-1) \dots e(kL-L+1)]^T$$

The convergence factor is denoted by μ and X_k is derived as :

$$\mathbf{X}_k = \begin{bmatrix} x(kL) & x(kL-1) & .. & x(kL-N+1) \\ x(kL-1) & x(kL-2) & .. & x(kL-N) \\ .. & .. & .. & .. \\ x(kL-L+1) & x(kL-L) & .. & x(kL-N-L+2) \end{bmatrix}$$

The error vector is computed as $e_k = d_k - y_k$ (3.10)

and the desired response and output is defined as:

$$d_k = [d(kL)\ d(kL-1) \dots d(kL-L+1)]^T \tag{3.11}$$

$$y_k = X_k . w_k \tag{3.12}$$

As block convolution is performed between L rows of x_k and w_k to compute a block of filter output, according to (3.9), similar block correlation is performed between N columns of x_k and L-point error vector e_k to compute the N-point weight increment vector Δw_k. This block convolution operation uses L number of N-point inner-product computation, where the block correlation uses N number of L point inner-product computations. Both these block convolution and block correlation have the same amount of computational complexity and thus they form a common input matrix .

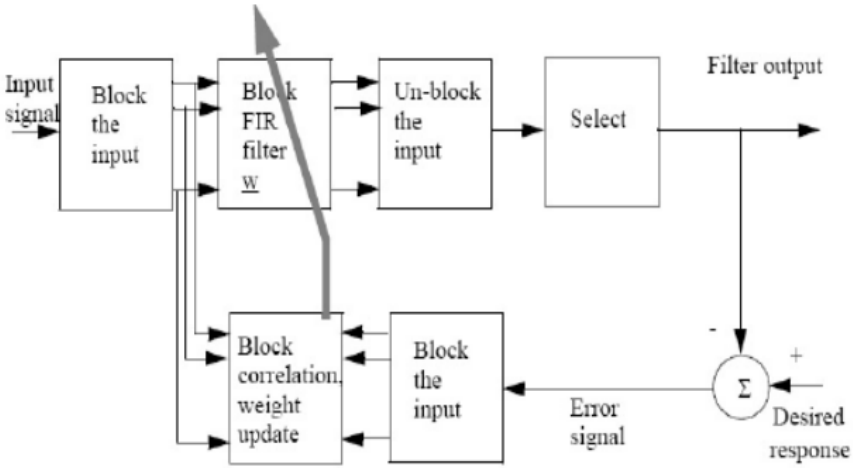


Figure 3.3 BLMS Adaptive Filter

The HUE can be approximated by the following formula.

$$HUE = \left(\frac{A_s}{A_P}\right)\left(\frac{T_s}{T_P}\right) \; X\; 100\,\%$$

Where, As =Arithmetic complexity of serial design, Ap =Arithmetic complexity of parallel design, Ts = Iteration period of serial design, and Tp = Iteration period of concurrent design.

Since, the convolution operation involves large size inner-products than correlation operation; the convolution operation needs to be expressed in small sized inner-product computations such that both the operations can be performed using same sized inner-product computations.

3.5 THE PROPOSED ARCHITECTURE

In this section, an efficient design for the implementing finite impulse response (FIR) adaptive filter using block least mean square (BLMS) algorithm based on distributed arithmetic (DA) formulation has been proposed. Generally DA–based architectures are bit–serial in nature and uses lookup tables (LUT) for both filter outputs and weight vectors of BLMS algorithm. The memory size of a LUT depends on the block–size, thus the number of words that can be stored in a LUT can take 2^L possible values of the inner products. The proposed structure uses the coefficient as an address to access the corresponding value of stored input signal from the LUTs. During the LUT update process, only the weight vectors are shifted left hand side instead of shifting the whole LUT contents to the right hand side circularly, it minimizes the time and power requirements.

This proposed formation consists of four processing elements of the block of size L= 4. Here, each processing element have 4 LUTs and in total it contains 16 LUTs according to N=LM where (L= 4 AMD M= 4). These LUTs are updated with the values of present input samples and the current past input samples according the filter output given by:

$$Y = \sum_{k=0}^{K} x_k w_k \tag{3.15}$$

where, $x_k \; and \; w_k$ and are the input matrix and the filter coefficient respectively. The input samples are updated in the LUTs to the corresponding addresses as shown in Table 3.1.

The weights of FIR filter are first updated by the BLMS algorithm and later the weights are matched from the weight computation block based on the optimization algorithm which depends on the error signal generated from the error computation block .The weights of the filter or an error signal is chosen from the MUX unit by select line which acts as an address to read the LUT contents. The control unit makes control on all the blocks by determining in which clock cycle, a specific task should take place as shown below.

A_0	A_1	A_2	A_3	Contents
0	0	0	0	0
0	0	0	1	x_3
0	0	1	0	x_2
0	0	1	1	$x_2 + x_3$
0	1	0	0	x_1
0	1	0	1	$x_1 + x_3$
0	1	1	0	$x_1 + x_2$
0	1	1	1	$x_1 + x_2 + x_3$
1	0	0	0	x_0
1	0	0	1	$x_0 + x_3$
1	0	1	0	$x_0 + x_2$
1	0	1	1	$x_0 + x_2 + x_3$
1	1	0	0	$x_0 + x_1$
1	1	0	1	$x_0 + x_1 + x_3$
1	1	1	0	$x_0 + x_1 + x_2$
1	1	1	1	$x_0 + x_1 + x_2 + x_3$

Table 3.1 Content of LUT for the Block Size L= 4 [14]

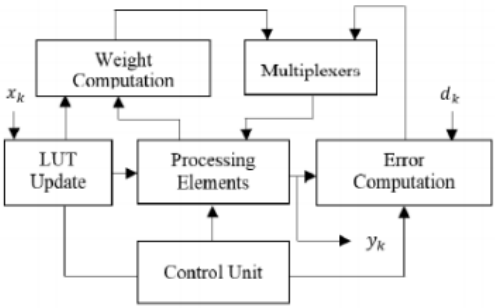


Figure 3.4 The proposed structure of BLMS – FIR adaptive filter based on DA technique.

A. LUT UPDATE

Each processing element comprises LUTs which store the partial products. The LUTs of one processing element is updated during every iteration with the values of the present input sample and current past input samples. These values are stored at corresponding addresses generated from the address generation block. These addresses can have 2^L possible combinations where L is the block– size and then update the values $Z_i(r)$ to the corresponding address as shown in figure 3.5 , where $0 \leq i \leq 3$. This LUTs update process has taken place for 16 clock cycles.

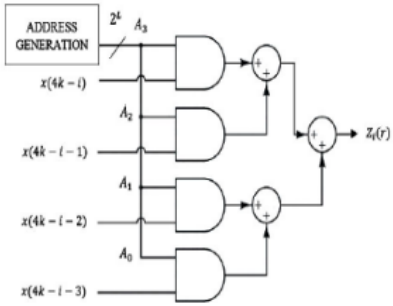


Figure 3.5 Structure of LUT update block for block of size L= 4, where $0 \leq i \leq 3$

B. PROCESSING ELEMENTS

The total applied processing elements for the block size L= 4 and the filter length N=16 is shown in Figure 3.6. These LUTs are arranged in a matrix form of L X M matrix where each column denotes the processing element. These processing elements computes the block of filter output y_k and the block of weight increment terms $w_{k_l}^0$ where $0 \leq l \leq B' - 1$ where B' is the word–length.

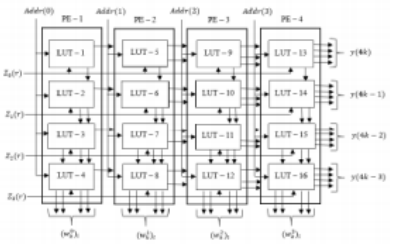


Figure 3.6. Structure of processing elements for the block of size L= 4 and weight vector of size M= 4.

C. LOOK–UP TABLE

The control line RW is used for read/write operation of the LUT. In figure 3.7, when RW=1 it enables the LUT update block to allow it to write the possible partial products to the corresponding address. When RW=0 it enables the weight vectors and error signals to be updated at the addresses *Addr(i)* for reading the partial products to further calculate the filter outputs and the filter weights respectively. The partial products are summed and shifted right to produce the final output. The control line CTR1 is enabled at the MSB bit to subtract this accumulated value from the MSB bit value since 2's complement representation place the sign bit at the MSB. The de-multiplexer with the select line CTR6 is take the filter output in one line and place it on another line for arithmetic shift of scaled filter weight.

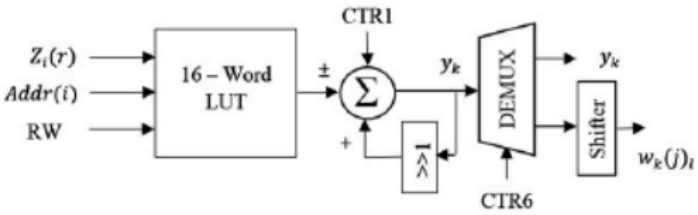


Figure 3.7 Internal structure of LUT for the block of size L= 4.

D. MULTIPLEXERS

The multiplexers with select line CTR5 choose the weight vectors or the error signals. These chosen weight vectors and error vectors acts as addresses to read the corresponding values from the LUTs as shown in figure 3.8.

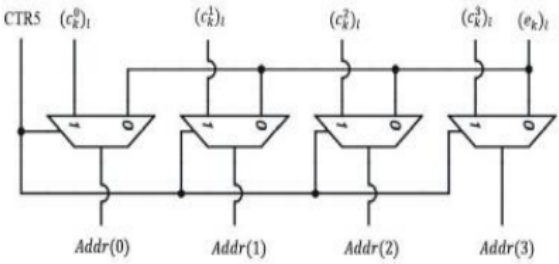


Figure 3.8 The structure of multiplexers for the block of size L= 4.

E. WEIGHT COMPUTATION

The new weight vector is generated by the weight computation block to adjust the filter coefficients determined by error signals. This new weight vectors updation are shifted left circularly in order to read the filter output from the LUTs, instead of making right shift. The figure 3.9 shows computation of one bit address out of four bit address. To compute a 4 – bit address, the four structures of figure 3.9 will be required. Thus the structure will form a matrix form of L X M dimension. The new weight vectors $c_k(j)$ are used as addresses to compute the filter output.

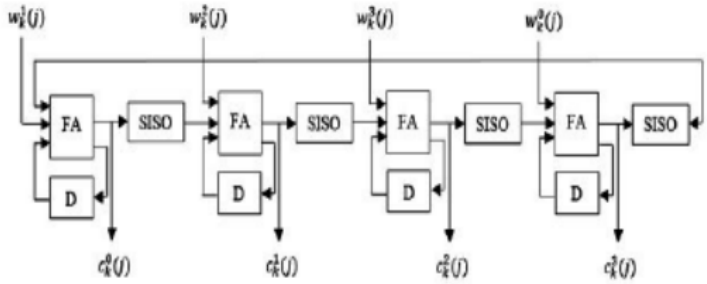


Figure 3.9 Structure of weight computation block for L= 4.

F. ERROR COMPUTATION

The structure for error computation as shown in figure 3.10, computes the error signal $e_{k_l}^0$ where $0 \leq l \leq B' - 1$, by comparing the filter output with the desired signals. The weights of the FIR filter to be updated by the BLMS algorithm is driven by the error signal. CTR4 and CTR2 are used in Parallel In Serial Out (PISO) shift register for load and shift operation. The LUT updation based on DA technique updates only one set of LUTs in a processing element during every iteration for saving the time and power. In future work, the number of words to be stored in a LUT can be further decreased by OBC (Offset Binary Coding) technique for memory reduction. Comparatively this proposed structure has major advantages for higher block size and large filter length.

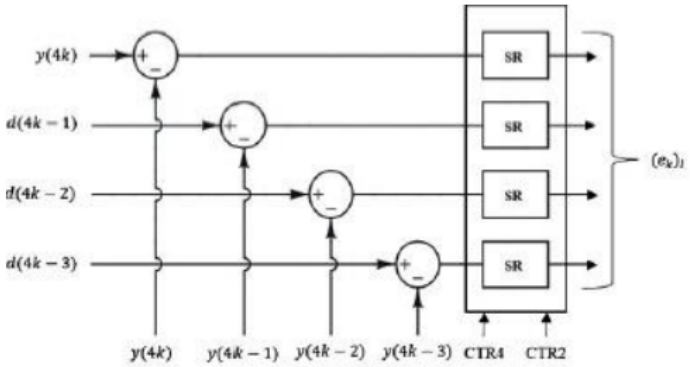


Figure 3.10 Structure of error computation block for block of size L= 4.

3.6 SUMMARY OF THE CHAPTER

In this chapter, efficient hardware architecture design and implementation of DA based BLMS adaptive filter is presented giving for purpose of obtaining high throughput with reduced latency for higher order filter design. The concept of distributed arithmetic (DA) is used for reduction of hardware complexity which is best suitable for efficient implementation of higher order BLMS filters with minimum area requirement, low power dissipation and reduced delay. The DA based FIR filter is implemented in XILINX ISE Design Suite 14.5 and the synthesis report shows that the proposed structure has less area and power requirements as compared to the existing structure of FIR filters. The detailed explanation of design and synthesis report is shown in Chapter 5.

CHAPTER 4

EFFICIENT DESIGN AND IMPLEMENTATION OF DEDICATED MULTIPLIER UNITS

4.1 INTRODUCTION TO MULTIPLIER

Multipliers are necessary part of modern electronic systems where high speed computations are required such as in FIR filters, digital signal processors and microprocessors etc. Multiplication-based operations like Multiply and Accumulate (MAC) and partial or inner product are some of the commonly used Computation-Intensive Arithmetic Functions (CIAF) implemented in many Digital Signal Processing (DSP) applications such as convolution, filtering etc [45]. Squaring and cubing are the special cases of multiplication and are frequently performed operations in ubiquitous DSP functions like Decoding, Image Compression, Demodulation, Adaptive Filtering, Least Mean Squaring, cryptography, computation of Euclidean distance among pixels for a graphics processor or in rectangular to polar conversions in several signal processing circuits where full precision results are not required [46] .Generation of square and cube of a number typically require a large number of multipliers. Presently, computation time of multiplication operation is considered as the leading factor in determining the instruction cycle time of a DSP chip and also high speed processing is in demand because of growing computing and signal processing applications. Since multiplier lies in the critical delay path and determines the performance of processor, speed of multiplication operation is highly important in DSP operations as multiplier is generally the slowest element in the system [47]. Hence, efficient design and implementation of dedicated squaring and cubing units can significantly reduce the delay while minimizing silicon area at the same time.

With expansion in VLSI technology, digital designs are integrated with more functional complexity in order to support applications which target floating point arithmetic or complex arithmetic like multipliers and powering units. With technology scaling, the goal has been to function designs at the fastest frequency ever possible to achieve high performance. But the problem with the complex arithmetic blocks such as multipliers unit is the requirement of long cycle time for calculations and in order to meet this frequency requirement, these designs always conclude at being pipelined, which not only results in increased area but also incurs a power penalty for operating at higher

clock speeds. In several applications this high power penalty is not acceptable for designers, for which they have to budget the power linked with individual resources. Often, designers prefer to use general-purpose multipliers for computing square of a number, as they are available as part of design packages which reduces their design time, but it causes increased area and power requirements for the design. A dedicated multiplier unit is both area and energy efficient and also performs much better as compared to general purpose multipliers [48]. Thus, in this chapter we have proposed dedicated multiplier unit design for computation of square and cube of a number with less power and area utilization.

4.2 PARALLEL MULTIPLIERS

The most commonly used parallel multiplication techniques are:

- Booth Encoding
- Modified Booth Encoding (MBE)

A. **Booth Encoding**:

In 1951, Andrew Donald Booth had proposed the Booth's Algorithm which is a multiplication algorithm for two signed binary numbers in two's complement notation [50]. It make use of repetitive addition of two predetermined values called A and S to form a product P after which it performs arithmetic shift on P in right hand side. Let x and y be the multiplicand and multiplier, respectively:

- First two predetermined variables A and S are computed as shown below to obtain the product P. Length of all these numbers should be equal to (m+n+1) :
    1. A: obtained by substituting m (in binary form) in MSB and append remaining bit (n+1) zeros.
    2. S: by substituting –m (in binary form) in MSB and append remaining bit (n+1) zeros.
    3. P: Substitute m bit of zeros in MSB. Then right insert value of n and append value LSB bit for zeros.
- Now considering last two significant bit of P following operations will follow.
- After determining the two LSB bits (rightmost) of *P*.

1. If they are 00, do nothing. Use *P* unaltered in the next step.
2. If they are 01, compute P + A while ignoring any over-flow.
3. If they are 10, compute P + S while ignoring any over-flow.
4. If they are 11, do nothing. Use *P* unaltered in the next step.

- Perform arithmetic right and get new result of P.
- Repeat the process n times.
- The final product is summarize in Table 4.1.

Block (multiplier bits) B	Re-coded Digit	Operation on A(multiplicand)
00	0	0*A
01	+1	+1*A
10	-1	-1*A
11	0	0*A

Table 4.1 Booth Recoding algorithm Table

For an example:

Let x=2, y = 3 than in binary x=0010, -x=1110, y=0011, m=4, n=4

A: 0010 0000 0

S: 1110 0000 0

P: 0000 0011 0

Applying 2nd and 3rd step y times:

P= 0000 0011 0 the LSB is 10 then P= P+S = 1110 0011 0 after shifting right

P= 1111 0001 1 the LSB is 11 then no change and no shifting operating is done

P= 1111 1000 1 the LSB is 01 then P=P+A after shifting right

P= 0000 0110 0 the LSB is 00 then no change and the final output after four loops is:

P= 0000 0110 0 that is P=2*3=6

The two drawbacks in original version of booth algorithm are:

1. Because of large number of add subtract and shift operation, there are many variables which are not able to be used in parallel multipliers

2. Presence of isolated 1's in the algorithm make it inefficient.

These drawbacks are overcome in modified booth algorithm.

B. **Modified Booth Encoding**

The modified booth multiplier is one of the popular multiplication algorithms proposed by D. L. Macsorley in 1961. MBA (Modified Booth Algorithm) is a high speed multiplier which utilizes parallelism to reduce count of partial product row and hence reduces the partial product matrix height. By employing modified booth algorithm, the number of partial product reduces from N to N/2 where N is multiplicand [51], by a digital bit recoding method done in two step i.e. encoding and selection as shown in figure 4.2. It is overcome in modified booth multiplier Radix-4 or higher Radix which cut down the number of partial product by N/2 or half instead of repeated adding and shifting for every column. This forms a significant enhancement for VLSI circuit design as it is directly associated to propagation delay while executing the circuit [54] and utilizes less hardware than long multiplication methods by using three bit pairing or triplet bit for performing booth recoding.

Table 4.2 pictures the implementation rules of modified booth multiplier. There are two operand A and B which represents multiplicand and multiplier bits respectively, the main task of decoder here is to convert the given input to equivalent booth value.

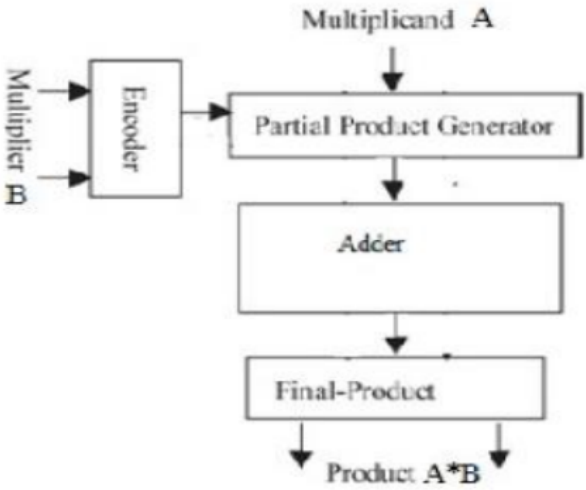


Figure 4.1: The basic Architecture of Modified Booth Multiplier

Block(Multiplier bits)B	Re-coded Digit	Operation on A(Multiplicand)
000	0	0*A
001	+1	+1*A
010	+1	+1*A
011	+2	+2*A
100	-2	-2*A
101	-1	-1*A
110	-1	-1A
111	0	0A

Table 4.2 Booth Recoding Table Radix - 4

Suppose A is multiplicand and B is multiplier, in order to form the final product z the partial product generation is done by using AND operation and then adding to get the final product of A*B. In Modified Booth algorithm Radix-2 we add bit "0" in LSB from rightmost position to fill two bits overlap over one bit previous of adjacent bit.

For an example:

Let's take A and B as :

Multiplier B=010011

Multiplicand A=01011

Now by using recoding technique first three bit pairing is done for multiplier, after which we get

$$\underset{+1}{\overline{0\ 1\ 0}}\ \underset{}{\overline{0\ 0\ 1}}\ \underset{-1}{\overline{1\ 1\ 0}}$$

Three bits pairing is done by extending LSB by putting bit '0' to complete the pair of triplet bit and then to calculate partial product as shown below

```
                0 0 1 0 1 1   multiplicand
                0 1 0 0 1 1   multiplier
                  1   1  -1   booth encode multiplier
-------------------------------------------------
1 1 1 1 1 1 0 1 0 0
0 0 0 0 1 0 1 1
0 0 1 0 1 1
              0 0 0 0 1       error correct negation
-------------------------------------------------
0 0 1 1 0 1 0 0 0 1           discard the carried bit  = P
```

which is the final result as expected.

4.3 TREE MULTIPLIERS

The most commonly used tree multiplication techniques are:

- DADDA Multiplier
- Wallace Tree Multiplier

A. **DADDA Multiplier**

DADDA has proposed a structure of matrix height that predetermines the height of matrix to provide the minimum number of reduction stages. To reduce the NxN partial product matrix, DADDA multiplier provides a sequence of column heights that works back from the final two-row matrix. In order to get the least number of reduction stages, the height of next matrix column is limited to 1.5 times the height of its successor. This process of reduction of DADDA multiplier [55] is implemented using the following recursive algorithm:

1. Let us assume that d1=2 and dj+1 = [1.5*dj], where dj represents the matrix height for the jth stage from the end. Then we've to find the smallest possible value of j such that at least one column in original partial product matrix has more bits then that in dj .

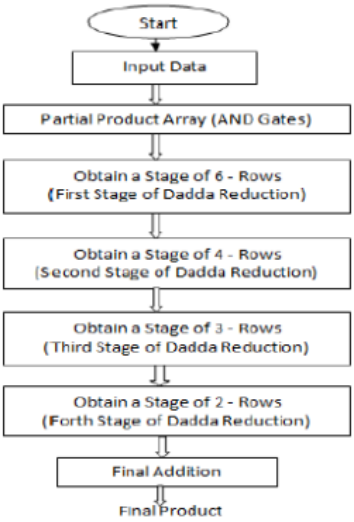


Figure 4.2 Flow Diagram of 8x8 DADDA Multiplier

2. In the jth stage falling from the end, we'll employ (3, 2) and (2, 2) counters to get a reduced matrix with not more than dj bits present in any column.

3. Then we'll put j = j-1 and repeat step 2 until a two rows matrix is generated.

This method of reduction, as it aims to compress each column, is also known as a column compression technique. Another benefit of using DADDA multipliers is that it utilizes the minimum possible number of (3, 2) counters. Therefore, the number of middle stages is set in lower limits: 2, 3, 4, 6, 9 . . . . as for DADDA multipliers there are N^2 bits in the original partial product matrix and 4.N-3 bits in the final two row matrix. As each (3, 2) counter will take three inputs and produces two outputs, with every (3, 2) counter the number of bits in the matrix is reduced by one, for which, the total number of (3, 2) counters required will be given by relation: N2 – 4.N+3 and the length of the carry propagation adder is given as CPA length = 2.N–2.

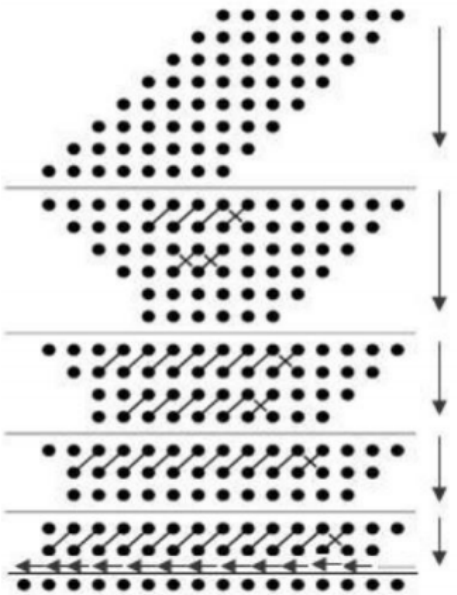


Figure 4.3 Dot diagram for 8 by 8 DADDA Multiplier

The (2, 2) counters to be used in DADDA's technique is given by N-1. The calculation diagram for number of bits for an 8X8 DADDA implementation is shown in figure 4.3. Dot diagram is considered as a useful tool for predicting the assignment of (3, 2) and (2, 2) counter in parallel multipliers. Each bit is represented by a dot. For each (3, 2) counter, the output is represented by two dots connected by a plain diagonal line whereas the output of each (2, 2) counter is represented by two dots connected by a crossed diagonal line. The 8 by 8 multiplier will take 4 reduction stages, with column height 6, 4, 3 and 2. This reduction utilizes 35 (3, 2) counters, 7 (2, 2) counters and a 14-bit carry propagate adder. The total generation delay for the final product is given by the sum of one AND gate delay, delay for each (3, 2) counter in every reduction stage and finally the delay through the 14-bit carry propagate adder which arrives in later stage but effectively reduces the worst case delay. As, critical path is used to determine the time taken by the DADDA multiplier, it begins at the AND gate of the first partial product then goes through the full adder of

every stage and then finally passes through all the vector merging adder stages. This multiplier has fewer stages in comparison to the carry save multiplier and hence it's fast and efficient.

B. **Wallace Multiplier**

Wallace has introduced a very significant iterative implementation of parallel multipliers. This advantage becomes more evident for 16-bit and higher order multipliers.

Considering an 8 bit multiplication where input is X7X6X5X4X3X2X1X0 and multiplier is Y7Y6Y5Y4Y3Y2Y1Y0.This multiplication needs 64 AND logic gates. First Y0 is multiplied with X7X6X5X4X3X2X1X0 and results X0Y0, XIY0, X2Y0, X3Y0, X4Y0, X5Y0, X6Y0 and X7Y0. Then Y1 is multiplied with X7X6X5X4X3X2X1X0 and results X0Y1, XIY1, X2Y1, X3Y1, X4Y1, X5Y1, X6Y1 and X7Y1. Similarly all multiplications are done in this way. In every emulation, there is one binary shift in the resultant logic. All AND logics can be represented as one bit starting from K0 to K63 sequentially as shown in figure 4.4. After the completion of 64 AND logic tree there follows an addition process which is done by using a tree formed. This is implemented further by using 3:2 compressors, 4:2 compressor and 5:2 compressor instead of using 3:2 compressors only. This additive process can be done by using

3:2 compressors only but the use of 4:2 and 5:2 compressors will reduce the latency and will increase the speed. The Wallace tree addition is shown in figure 4.5

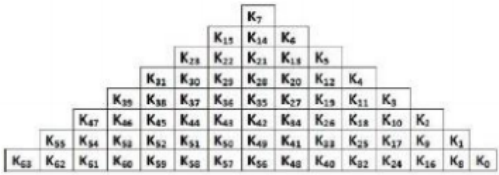


Figure 4.4 Multiplication Wallace Logic tree

In this process of addition, the sum output of intermediate stages become the input for the next compressor in the same column and the carry out is propagated to next column adders [56]. The result is represented by 16 bit number [P15….P0]. In Wallace tree architecture, all the bits of

partial products in each column are added in parallel by a set of counters without propagating any carry. Another set of counters is employed to this new matrix and so on, until a two-row matrix is formed. The most commonly used counter used is the Full Adder 3:2 counter. The final result is obtained using carry propagate adder. The primary advantage of Wallace tree is high speed as the addition of partial products is now O (log N).

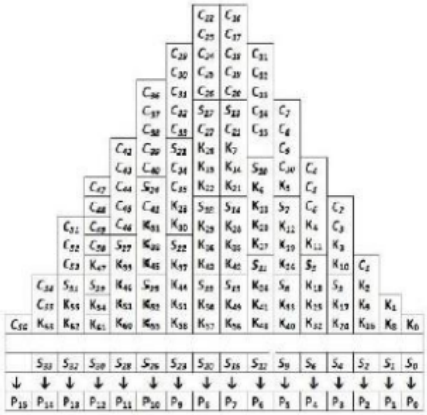


Figure 4.5 Wallace Tree Multiplier Using Optimized Compressor

As the block diagram shows that partial products are added in Wallace tree block, the result of these additions is the final product bits forming sum and carry bits. The 3:2 compressors utilizes carry save adder which gives two output numbers of the same dimension as the input where one number forms sequence of partial sum bits and other is a sequence of carry bits. Here, the carry digit is taken from the right and passed to the left, like conventional addition, which is the result of the previous calculation. So in every clock cycle, carry bit moves only one step along the clock. The carry-save adder produces parallel output values and has the same delay as a single full-adder. The 4:2 compressors have been applied because of its lower latency rate of the partial product accumulation stage which can be built using two 3:2 compressors. Due to its regular structure, the 4:2 compressors is considered as ideal for the construction of regular Wallace Tree with low complexity.

4.4 DEDICATED SQUARING UNIT DESIGN

4.4.1 Binary squaring

Squares are special case of multiplication where both inputs are identical. Since the two inputs are identical, many optimizations can be made in the implementation of a dedicated squaring unit. Such a squaring unit requires less area compared to multipliers as nearly half of the partial products can be combined using the equivalence $a_i a_j + a_i a_j = 2.a_i a_j$ which can be represented by adding aiaj to the next column to the left. This reduces the depth, which can be defined as the number of partial products to be added together in a column. With a reduction in depth, the design can operate faster as the number of terms on the critical path reduces. Figure 4.7 shows a 4- bit unsigned squaring unit. We can observe from figure 4.6 that two a1a0 terms in column 2 are reduced to having only one a1a0 term in column 3. Similarly other partial products can be reduced. Also the property that a0a0 = a0 allows reducing terms in the final partial products. The square of a 4-bit number can be computed by adding the rows at the bottom part of figure 4.6. From figure 4.6 we have noted that the depth has also reduced; an initial depth of four for a multiplier configuration was reduced to three for squaring. The depth requirement for booth multiplier and squaring unit is compared in Table 4.3 which implies that squaring unit is faster than a parallel multiplier.

				×	A_3 A_3	A_2 A_2	A_1 A_1	A_0 A_0
					A_3A_0	A_2A_0	A_1A_0	A_0A_0
			A_3A_1		A_2A_1	A_1A_1	A_1A_0	
		A_3A_2	A_2A_2		A_2A_1	A_2A_0		
	A_3A_3	A_3A_2	A_3A_1		A_3A_0			
–	A_3	A_3A_1	A_3A_0		A_2A_0	A_1	–	A_0
–	A_3A_2	–	A_2A_1		–	A_1A_0	–	–
–	–	–	A_2		–	–	–	–

Figure 4.6 Partial Product Reduction in Squaring Computation

Bits	Booth's Multiplier		Squaring Units	
	Max	*Average*	*Max*	*Average*
8	5	4	7	2.9
16	9	7	11	4.83
32	17	13	19	8.79

Table 4.3 Depth Requirement

So we prefer dedicated squaring units to perform binary squaring rather than parallel multipliers. Therefore, in this chapter, different algorithms are explained and compared to design a fast squaring unit and a dedicated architecture is proposed which is efficient and fast in comparison to other methods.

4.4.2 The Proposed Architecture for Fast Squaring

To overcome the disadvantage of Vedic Sutra method, a new architecture is proposed for fast and efficient squaring which primarily aims in reduction of partial product number such that associated Wallace tree structure can be modified to make it smaller in order to get a faster and compact squaring circuit.

Conventional Method:

A squaring circuit for an n-bit value constitutes a partial product bit generator which logically AND's a bit with weight 2^k (k is an integer) with the same bit 2^k to generate a partial product bit of weight 2^{2k}. Another partial product bit generator unit receives and logically AND's a 2^k bit with that of 2^m (m is an integers) to generate a partial product bit of weight 2^{k+m+1}. The second partial product bit generator is the only bit generator in the squaring circuit which logically AND the bits 2^m and 2^k. The circuit may also involve other partial product bit generators. Here, the k^{th} bit in the multiplicand and the m^{th} bit in the multiplier both from the right represented as "multiplicand bit k" and "multiplier bit m" respectively. Additionally, the k^{th} bit in the m^{th} partial product, from the right, is referred to as "bit mk". A partial product bit generator unit mk make use of AND gate in which one input is multiplicand bit k while the other input is multiplier bit m and the computation is shown in figure 4.7 below.

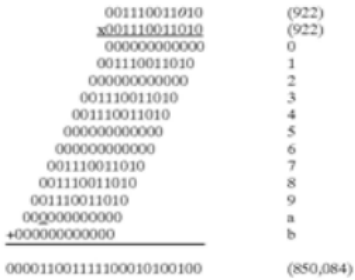


Figure 4.7 Conventional Multiplication Method

A circuit which implement this method will need minimum of n^2 (suppose here we are taking n=12 so 144) AND gates to square an n-bit value. Additionally, in a Wallace tree of 3:2 configuration carry save adders, each column utilizes up to n-2 (e.g., 10) carry save adders. If each column has similar carry save adder units, a total of 2n (e.g., 24) columns may require up to (2n) (n-2) = $2n^2$-4n (e.g., 240) carry save adders. These AND gates and carry save adders will take up significant space on a chip. Therefore, it is mandatory to lessen the partial product bit generator units and carry save adders required to perform squaring. By doing so, the associated partial product bit generator array and accompanying Wallace tree are made compact and fast as compared to conventional squaring circuit.

Proposed Method

In the conventional squaring method, Partial product bits are "mirrored". For illustration, in the following multiplication, the italicized partial product bits are vertically downward mirrored about the partial product bits which are bolded. As, each partial product bit generator unit gets multiplicand bit k for the non-mirror bits and a bit of weight 2^k and 2^m to generate a bit of weight 2^{k+m+1}, all of the lower bits can be deleted, after which the number of product bits required will reduced from n^2 (e.g., 144) as in the conventional method to n(n+1)/2 (e.g., 78) in proposed method, achieving reduction of almost 50%.

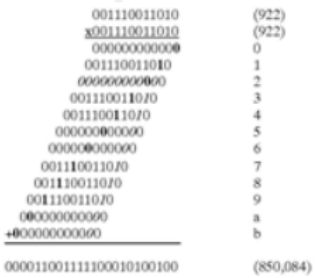


Figure 4.8 Multiplication with Mirrored Bits

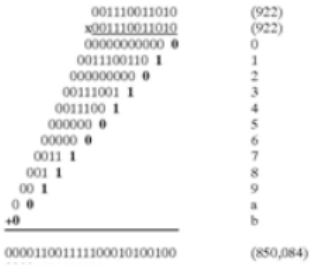


Figure 4.9 Partial Products after removing the Mirrored part

Moreover, the maximum number of partial product bits per column is [(n/2) +1] (e.g., 7). Therefore, the maximum number of carry save adders essential for a column is condensed from n-2 (e.g., 10) to [(n/2)-1] (e.g., 5). Figure 4.10 shows a block diagram of a circuit that implements the above squaring method. This method constitutes two 12-bit registers, each having 12-bit value [b_{16}:0] to be squared. Each bit z[q] of the 12-bit value z [b_{16}:0] has a weight2^x, where for x forms a set of integers from 0 to b_{16}. In first process, only one register is used to provide bits z[b_{16}:0]. In second process, bits z [b_{16}:0] are provided by a circuit rather than a register. In reaction to a signal SQUARE, signals equivalent to each bit of z[b_{16}:0] are provided to a partial product bit generator array which produces partial product bits and make them available to respective column adders CA0 to CA23 according to the weight of the partial

product bits. The column adders CAO to CA23 provide the result in redundant form i.e., both carry and sum which is provided to the carry propagate adder for final output.

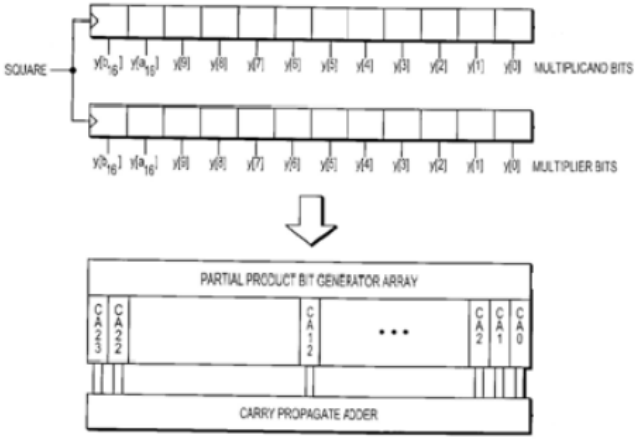


Figure 4.10 Proposed Block diagram of squaring unit

The column adders of figure 4.15 receives and sum up the partial product bits mk according to the Table 4.5.

Along with the received partial product bits, the column adders CAO to CA23 also receives carry bits from the right column, and generates a sum and carry bit to be further added by carry propagate adder which generates carry out bits to the left column. As shown in Table 4.4, the maximum number of partial product bits as received by column adder CA12 is 7 and the required 3:2 carry save adders desired to reduce the 7 bits of products part to a final sum and carry out value is only 5. Therefore, the above described circuit and method of squaring significantly reduces the requirement of partial product bit generator units by almost 50% compared to the prior method. This not only simplifies the adder tree structure but also reduces the area needed by the adder tree to aggregate the reduced number of partial product bits. Therefore, the above described squaring circuit is fast and compact as compared to conventional method.

Column Adder	Partial product bits received	# of partial products
CA0	00	1
CA1	none	0
CA2	01, 11	2
CA3	02	1
CA4	03, 12, 22	3
CA5	04, 13	2
CA6	05, 14, 23, 33	4
CA7	06, 15, 24	3
CA8	07, 16, 25, 34, 44	5
CA9	08, 17, 26, 35	4
CA10	09, 18, 27, 36, 45, 55	6
CA11	$0a_{16}$, 19, 28, 37, 46	5
CA12	$0b_{16}$, $1a_{16}$, 29, 38, 47, 56, 66	7
CA13	$1b_{16}$, $2a_{16}$, 39, 48, 57	5
CA14	$2b_{16}$, $3a_{16}$, 49, 58, 67, 77	6
CA15	$3b_{16}$, $4a_{16}$, 59, 68	4
CA16	$4b_{16}$, $5a_{16}$, 69, 78, 88	5
CA17	$5b_{16}$, $6a_{16}$, 79	3
CA18	$6b_{16}$, $7a_{16}$, 89, 99	4
CA19	$7b_{16}$, $8a_{16}$	2
CA20	$8b_{16}$, $9a_{16}$, $a_{16}a_{16}$	3
CA21	$9b_{16}$	1
CA22	$a_{16}b_{16}$, $b_{16}b_{16}$	2
CA23	none	0

Table 4.4 Partial Product Bits after reduction

It is worth noticing that in one process, AND gates 00, 11, 22, 33, 44, 55, 66, 77, 88, 99, $a_{16}a_{16}$ and $b_{16}b_{16}$ are not being used in generation of their respective partial product bits. Instead of which, bits z[0], z[1], z[2], z[3], z[4], z[5], z[6], z[7], z[8], z[9], z[a_{16}] and y[b_{16}] are provided as it is to the respective partial product bit, for which, the number of AND gates needed to square is further reduced by n. Taking an example of squaring an n-bit value, the number of AND gates required is n(n-1)/2 i.e. 66 for a 12-bit value which shows reduction of over 50% compared to the conventional circuit.

The maximum number of partial product bits per column may be further reduced from [(n/2)+1] (e.g., 7) to [n/2] (e.g., 6) as described. This reduction is accompanied by shifting one partial product bit from the column having maximum partial product bits (e.g., column 12) to its most significant neighbor (e.g., column 13). This reduction is shown with reference in figure 4.11 and 4.12.

Figure 4.11 shows that the AND gates receiving 56 and 66 inputs in column 12 generates two partial product bits 56 and 66, while column 13 stays ideal. In Figure 4.17, now column 12 generates only one partial product bit say p', while column 13 also generates a partial product bit say p". Although the total count of partial product bits does not alter, but the number of maximum

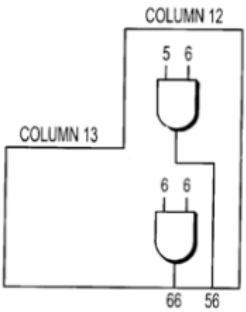


Figure 4.11 Portion of Partial Product Bit generator array

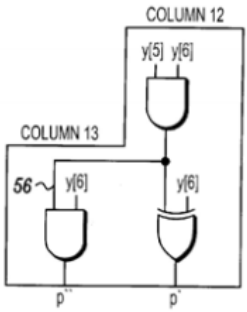


Figure 4.12 Extended version of Partial Product Bit generator array

partial product bits generated by column 12 is reduced from 7 to 6, whereas, maximum number of partial product bits generated by column 13 has increased from 5 to 6. As, the maximum number of partial product bits generated by any one column is reduced from 6 to 1, the number of 3:2 carry save adders required per column also reduced to 4 for squaring a 12-bit value. The truth table 4.5 shows the relationship between input bits z[5] and z[6] and output partial product bits p' and p". Here, "X" means that the output bits p' and p" are not dependent on bit Z[5] if bit Z[6] is 0. Bit p' has a 1 value only if bit Z[5] has a 0 value and bit Z[6] has a 1 value.

Input Bits		bit	Output Bits	
y[5]	y[6]	56	p"	p'
x	0	0	0	0
0	1	0	0	1
1	1	1	1	0

Table 4.5 Relationship between input and output bits

Bit p" contains 1 value only if both of bits z[5] and z[6] have a 1 value. Figure 4.12 implements this truth Table 4.6 as a circuit. One AND gate logically AND's bits Z[5] and Z[6] to generate bit 56. Another AND gate logically AND's bits 56 and z[6] to generate bit p". An XOR gate logically XOR's bits 56 and z[6] to generate partial product bit p'.

4.5 DEDICATED CUBING UNIT DESIGN

4.5.1 Binary Cubing

The cubing operation is one of the most vital operations in arithmetic process but gets more complex when we take higher radix numbers. Cubing operation can be performed using regular multipliers, being scalable with larger delay, or Structure based array implementation, which is faster but scalability increases complexity and expenses of the design. Moreover, multipliers take up large area, have long latency and dissipate considerable power. Therefore, multipliers which offer design targets such as scalability, re-configurability, high speed, low power consumption, regularity of layout and less area are preferred [58].

Power is gradually becoming a valuable resource in modern VLSI design systems, even more than area. As huge number of applications requires involvement of functional units like squares, cubes and other higher order units, it have become imperative that such functions be implemented in hardware. Implementing these functions using traditional general-purpose multipliers in a design may be economical in terms of area but have more power requirements than actually required. As the scalability of designs keeps escalating, most of the signal processing system is being implemented on a VLSI chip. These applications not only necessitate superior computation capacity but also consume substantial amount of energy. As, power dissipation in a device increases, extra circuitry is compulsory to cool the device and to guard it from thermal breakdown which will upshot the total area of the device thus power consumption causes severe issues in its implementation. The need for low power VLSI systems primarily derived from factors. First, the operation frequency and processing capacity per chip, which causes large current and generates heat due to large power consumption that should be avoided by proper cooling techniques. Second, the short battery life in modern portable electronic devices, which has made low power designs crucial for its prolonged function [59]. Therefore, the goal here is to design a cubing architecture which is better in terms of speed, power and area than a design using standard multiplier. The inspiration behind this work is to implement the design and for low power requirements.

4.5.2 Methods to compute cube of an operand

Let's assume the input operand is an n-bit unsigned integer *a and P* is the output of the cubing unit. Applying simple mathematics to compute the cube of an unsigned integer *a* can be expressed as:

$$a = \sum_{i=0}^{n-1} a_i . 2^i \qquad (4.1)$$

$$P = a^3 = \sum_{i=0}^{n-1} a_i . 2^{3i} + 3. \sum_{i=0}^{n-2} . \sum_{j=i+1}^{n-1} a_i a_j . \left(2^{2i+j} + 2^{i+2j}\right) + 6. \sum_{i=0}^{n-2} . \sum_{j=i+1}^{n-2} . \sum_{k=j+1}^{n-1} a_i a_j a_k . 2^{i+j+k}$$

Similar to that in multiplication, the problem in expressing this equation as a Partial Product Matrix (PPM) is the size of matrix which grows exponentially with the size of operand. As, the output expression P is calculated by summing each row from the right to the left hand side, it can be simplified to derive relation for the number of bits that must be added as well as for the height of the PPM.

Therefore, the intricacy in reducing the PPM height results in the following relations:

$$\textit{Number of bits in PPM} \quad = n^3$$

$$\textit{PPM height} \quad = 3.n^2$$

To overcome this constraint on the PPM size, competent computational techniques other than conventional computation are required for cubing operation. Some of the commonly used methods are given below.

- One normal method to find cube of an operand is by making use of lookup tables (LUTs) to store many values for a given range [60]. If the operand doesn't lies within the given range, it can be modified to fit within an allowable region, then evaluated and reconstructed in later processing stage. But, because of size of the operand, the storage size can be impractical. One possible up gradation is to do small multiplications along with LUT computations to achieve a table-driven method [61].

- Another convenient method of cubing is to employ traditional multiplier in serial fashion. For this method, operand is multiplied by itself and stored in a temporary register. After the intermediate result is obtained, the final result is achieved by multiplying the original operand by the stored result. Though, the resulting latency will be twice the amount of the multiplier, this latency will grow exponentially as the operand length increases. Improvements can be made by

using an efficient squaring unit [62] for the intermediate result followed by a nonrectangular multiplier.

- The technique to enhance application specific processors and embedded processors is by using higher levels of parallelism through SIMD (single instruction multiple data) processing [63]. These enhancements are implemented along with an extended instruction set that takes extra hardware to allow more operations to be performed in a single instruction. With this increased parallelism and the shrinking feature sizes, the upcoming generation of application-specific processor can give more throughput. Consequently, new techniques for computing the cube of an operand in parallel without the use of basic multipliers or tables become an eye-catching substitute.

Though, the above methods are theoretically feasible but they cannot be implemented practically on hardware because of their limited scope. Therefore, we need a fast algorithm which can overcome all the problems that occurred in the above methods.

4.5.3 The Proposed Architecture for Fast Cubing

One noteworthy method to improve cubing by utilizing parallel unit is presented in [65]. This method had proposed a parallel cubing technique which computed the cube of an operand in parallel by employing three reduction method techniques that primarily reduces the height of the partial product generation matrix. By doing so, the latency of the employed carry-save adder (CSA) unit has significantly condensed.

The proposed architecture is an extension to this technique, using the three partial product reduction techniques previously proposed, but with different computation orders and a optimized design of counter [66]. These extensions further decrease the number of CSA stages required to reach a final two operand result.

Considering an n-bit unsigned integer $A = a_{n-1}.a_{n-2} \dots \dots \dots \dots \dots a_1.a_0$ which is equivalent to:

$$A = \sum_{i=0}^{n-1} a_i.2^i \qquad (4.2)$$

with the partial product matrix for a^3 with n =4 is shown in Figure 4.13. This case is taken for simplicity as large operand length results exponential rise in the partial product matrix height. In this figure, each partial product is simplified so that it can achieve an efficient reduction. As in squaring units [65-66], the original matrix can be replaced by an equivalent matrix which constitute reduced partial product bits. In squaring, the partial products are symmetric to the anti diagonal. However, in cubing the symmetries are not along the anti diagonal, but lies within smaller subsets of the partial product. Therefore, using the identities: $a_i a_j a_i + a_j a_i a_i + a_i a_i a_j = 3\ a_i . a_j$ and $a_i a_i a_i = a_i$ the partial product matrix can be redesigned, as shown in Figure 4.14.

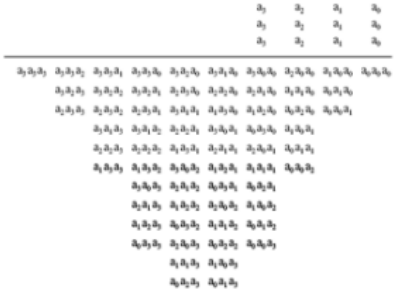


Figure 4.13. Reorganization of Partial Products

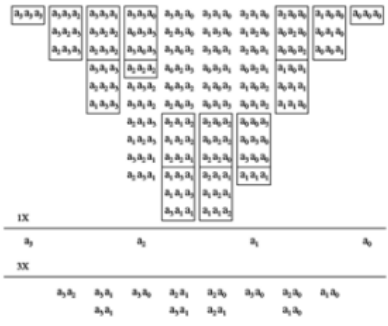


Figure 4.14. Partial Product Array Reduction

Similar to a reduction technique in [65], the partial products formed by these identities are shown at the bottom of Figure 4.14. From this new reorganized partial product matrix, another pattern can be framed which can offer major saving in matrix height. Since cubing produces 3! = 6 terms for several different sub matrices, the matrix can be reframed as shown in Figure 4.15. Similar to reduction technique in [65], the 6*X* term is combined with the 3*X* term in the succeeding column as shown in Figure 4.15. Using the three reduction techniques described before, the partial product matrix is reduced considerably resulting in a design which is more organized to use with higher-order counters. In [65], the 3*X* term is computed by exploiting (15, 4) counters to reduce the 1*X* partial product column height to that of two in one stage. In contrast, this paper utilizes only (3, 2)

counters to normalize and abridge the analysis. Once the combined and organized partial product matrix with reduced partial products is formed, many carry-save adder stages are applied to further reduce the height of matrix to two [66].

The proposed order for forming the PPA (partial product array) is that instead of forming 3*X* partial product arrays and then reducing those arrays, the bits are weighted by 3 and then combined with 1*X* array. Therefore, as shown in Figure 4.16, the partial product matrix is reorganized such that $3aiaj = 2aiaj + aiaj$.

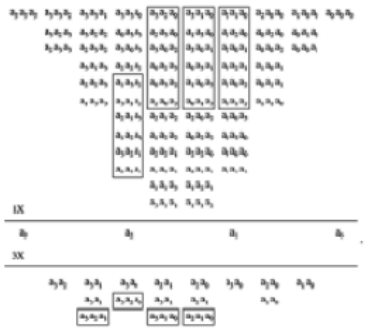


Figure 4.15. Reorganization of Partial Products

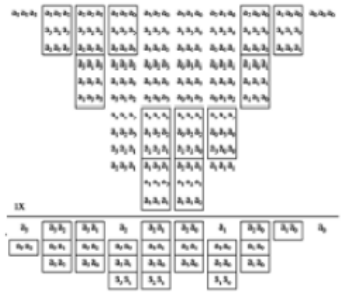


Figure 4.16. New Approach to Group of Three Reduction

Here, the 2*aiaj* term is shifted one position to the left. Similar to the 3*aiaj* reduction, the 6*aiaj* term is reorganized such that it is further shifted by one position to the left. This new organization is shown in Figure 4.17. To improve the regularity of the matrix and to further reduce its maximum height, many 6*X* terms have the same weight as the 3*X* terms, so the 6*aiajak* combines with certain indices of 3*aiaj* when $k = j$ or $k = i$. This new formation is illustrated in Figure 4.18. The primary benefit of this new method of parallel cubing over previously proposed methods is the reduction in the number of carry-save adder stages required to reach a two-operand result as it is one of the primary sources of latency in the parallel cubing operation. Minimum number of stages is achieved by performing more parallel computations, but this advantage is achieved at the cost of additional hardware due to more bits to be reduced in the partial product matrix.

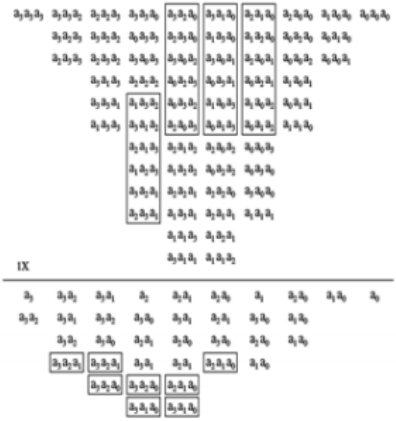


Figure 4.17 New Approach to Group of Six Reductions

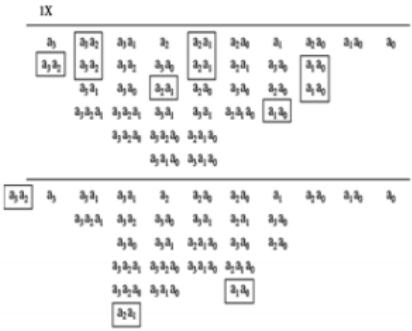


Figure 4.18. New Approach to Group of Six Reductions

To further improve the proposed reduction method, implementing Full Adder or (3, 2) counter for low power and low area is the main motivation behind this work. The proposed design is implemented at device level in Cadence-Virtuoso using NMOS pass transistor instead of CMOS technology in order to cut down the transistor count and area and also to reduce its power consumption as elaborated below.

I. CMOS Implementation of Full Adder Unit

After reducing the number of partial products, the goal is to make area and power efficient design which is achieved by reducing the transistor count. The full adder is implemented in two different technologies and compared in terms of area utilization. First, the full adder implementation in CMOS technology is shown in figure 4.19 which further comprises of sub units like XOR gates, inverters, PMOS and NMOS transistors.

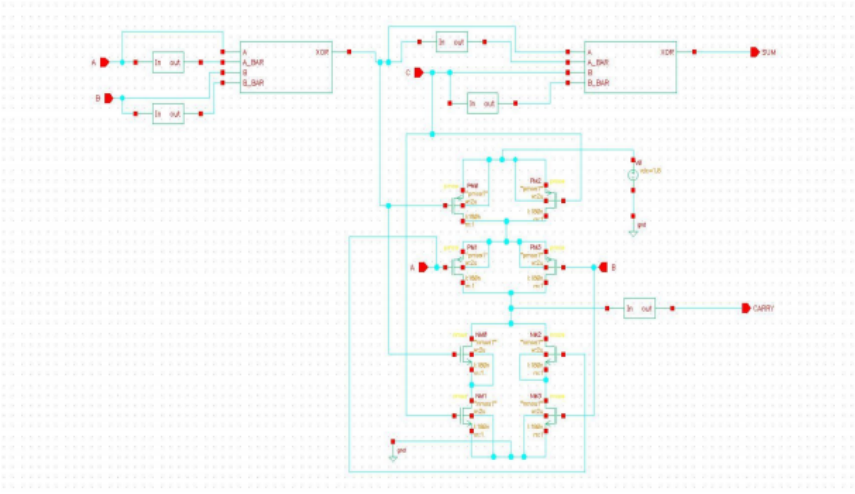


Figure 4.19 The CMOS based Full Adder Schematic

The sub units are also designed in Cadence Virtuoso 180nm Technology as shown below

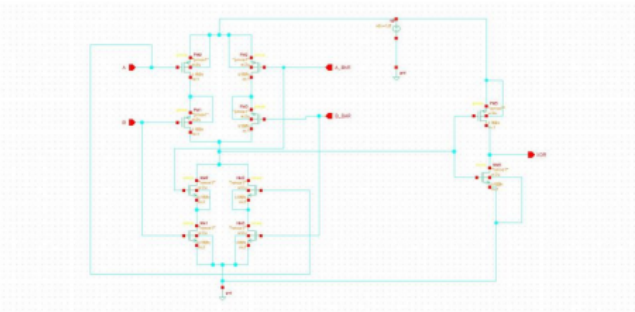


Figure 4.20 XOR Schematic in CMOS technology

The total number of transistors used is 38 which will occupy large area so in order to cut down this number we have used NMOS Pass transistor to implement Full adder with only 29 transistors which is the main advantage in area optimization as shown in next section.

II. NMOS Pass Transistor Implementation of Full Adder Unit

In VLSI system design, pass transistor logic (PTL) represents a logic family used in the design of several integrated circuits. It is used in different logic gates design, with the advantage of reduced transistor count, by eliminating redundant transistors. Pass Transistors are used to pass logic levels between nodes of a circuit, unlike switches which are directly connected to supply voltages. This causes reduction in the number of active devices connected to the design, but with the disadvantage of decreased voltage between high and low logic levels at each stage. As transistor in series is less saturated at its output than at its input, a traditionally constructed gate is necessary in design to restore the signal to full rail voltage. By contrast, conventional CMOS logic switches transistors so the output connects to one of the power supply rails, so logic voltage levels in a sequential chain do not decrease, but this effect is more pronounced in higher order computations. Here, our main motive to cut down transistor number is achieved with NMOS Pass transistor as shown below.

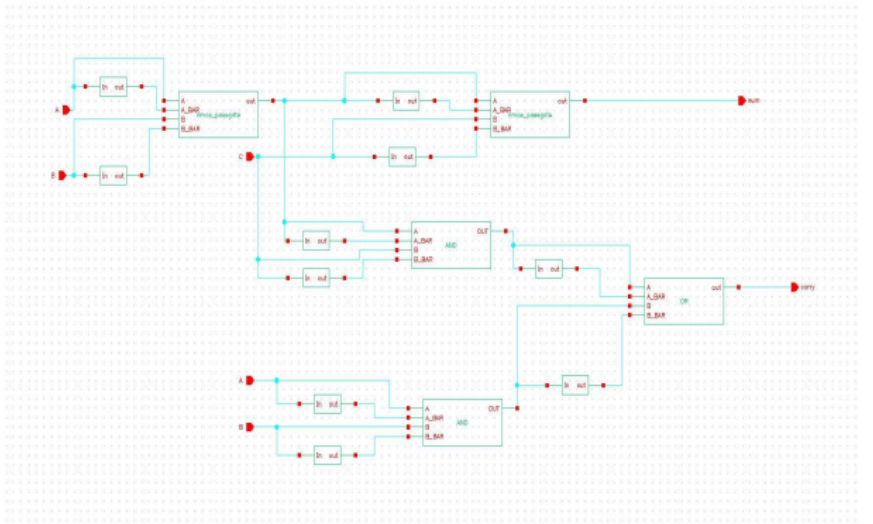


Figure 4.21 NMOS Pass Transistor based Full Adder Schematic

Here, the sub units are also designed using NMOS Pass Transistor so that maximum reduction in transistors can be achieved. The various sub units implementation is also shown below.

A.XOR using NMOS Pass Transistor

The number of transistors required is 4 which are very less as compared to CMOS where 10 transistors are required for one unit.

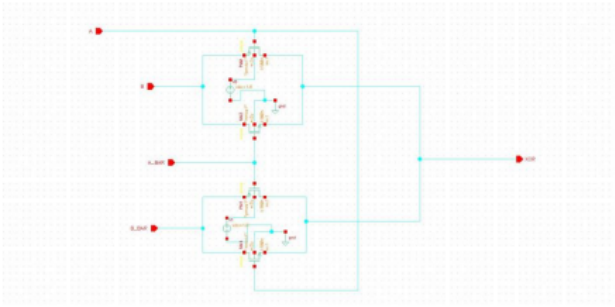


Figure 4.22 XOR Schematic in NMOS technology

B. AND and OR gate using NMOS Pass Transistor

Each gate requires only 3 transistors per unit as compared to CMOS where 8 transistors are required per design.

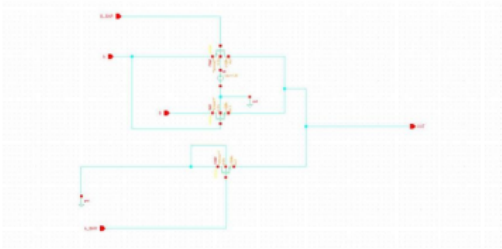
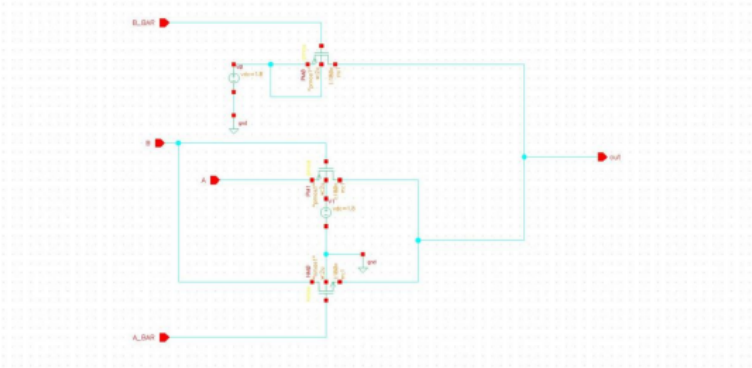


Figure 4.23 AND Schematic in NMOS technology



Figur 4.24 OR Schematic in NMOS technology

Therefore, the difference in number of transistors is summed up in table 4.6 below

COMPONENTS	CMOS DESIGN		NMOS PASS TRANSISTOR DESIGN	
	UNITS	NO. OF TRANSISTOR PER UNIT	UNITS	NO. OF TRANSISTOR PER UNIT
XOR GATE	2	10	2	4
INVERTER	5	2	6	2
MOS TRANSISTOR	8	1	-	-
AND GATE	-	-	2	3
OR GATE	-	-	1	3
TOTAL		38		29

Table 4.6 Comparison Between CMOS And NMOS Implementation

This method has resulted in huge reduction in transistor count which will optimize the area requirement. For power concern, same designs are implemented in ADE(analog design environment) Simluation window and psf files are generated for corresponding designs which are then compared and resulted in power reduction as discussed in chapter 5.

CHAPTER 5
EXPERIMENTAL RESULTS AND DISCUSSIONS

This chapter contains detailed explanation of the simulation results of all the proposed architecture divided into the following three sections:

- Simulation results of DA based BLMS adaptive filter using Xilinx ISE Design Suite 14.5 and Power Estimation Using XPE (Xilinx Power Estimator)
- Simulation results of dedicated squaring unit using Xilinx ISE Design Suite 14.5
- Power analysis of NMOS Pass Transistor based Full Adder Design using Cadence-Virtuoso 180nm technology.

5.1 IMPLEMENTATION OF DA BASED BLMS ADAPTIVE FILTER FOR AREA, DELAY AND POWER REDUCTION

A. Description of tools used :

The tools used for area, delay and power reductions are:

- *Xilinx ISE Design Suite 14.5* **:** It is a software tool designed by Xilinx for synthesis and analysis of HDL designs, which enable the developer to compile their designs, perform timing analysis, produce elaborated RTL diagrams, simulate their design using different inputs, and to configure the target device with the high level language.

- *XPE (Xilinx Power Estimator)*: It is a power estimation tool commonly used in the pre-design and pre-implementation phases of a design. XPE provides assistance in architecture evaluation, device selection and to select the appropriate power supply as well as the thermal manager component for the efficient application. As a pre-implementation tool, XPE can be used in the early phase of a design cycle when the RTL description of the design is not yet completed. After implementation, the Xilinx Power Analyzer (XPA) tool (available in the ISE Design Suite software) can be used for more accurate estimation and power analysis.

B. Verilog Code And Simulation Results For DA Based BLMS Adaptive Filter

i) Verilog code:

The verilog code (Appendix A) explains the block updating process by using DA technique in which same LUT is used for coefficient and weight updating by using two different blocks and some temporary registers.

ii) Simulation Results:

The proposed design is simulated using I-SIM simulator as shown in Figure 5.1

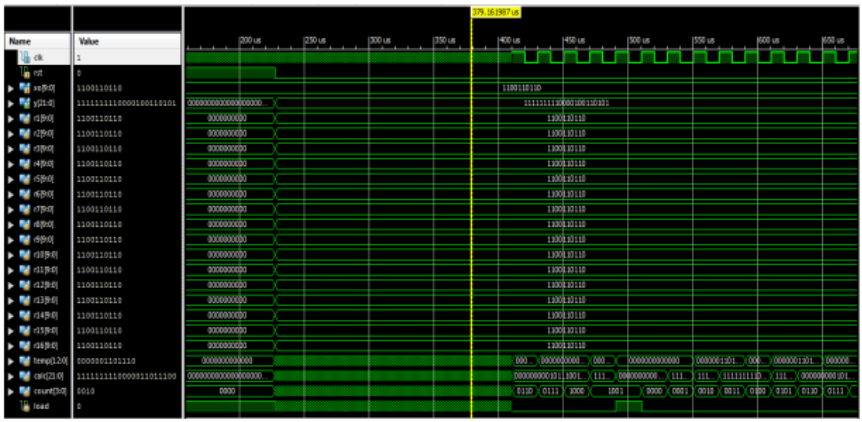


Figure 5.1 Simulation of proposed architecture

In programmable logic system design, the size is typically measured in terms of number of logic elements used and memory utilization. The metrics for measuring number of logic element and memory usage are obtained from the synthesis report generated by Xilinx ISE 14.5 software. Figure 5.2 shows the number of logic elements used in proposed structure and Figure 5.3 shows component constituents in existing architecture.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	29	126800	0%
Number of Slice LUTs	24	63400	0%
Number of fully used LUT-FF pairs	6	47	12%
Number of bonded IOBs	19	210	9%
Number of BUFG/BUFGCTRL/BUFHCEs	1	128	0%

Figure 5.2 Area utilization of proposed architecture

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	193	126800	0%
Number of Slice LUTs	196	63400	0%
Number of fully used LUT-FF pairs	62	327	18%
Number of bonded IOBs	34	210	16%
Number of BUFG/BUFGCTRL/BUFHCEs	1	128	0%

Figure 5.3 Area utilization of existing architecture

The power consumption estimates are obtained from the simulation report generated by Xilinx X Power Analyzer software as shown in Figure 5.4.

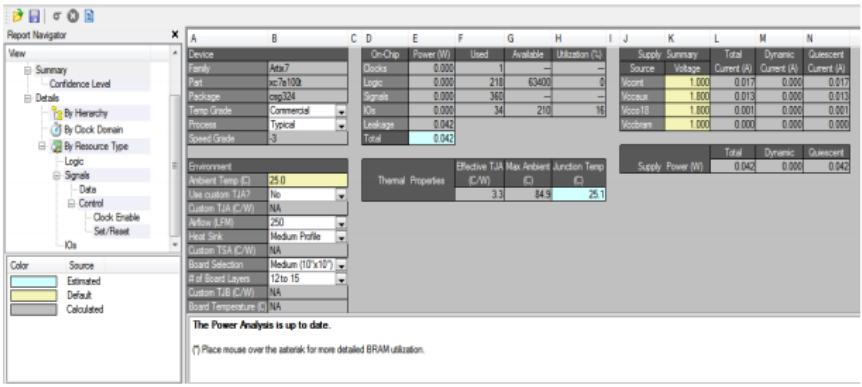


Figure 5.4 power consumption of proposed architecture

C. Comparison between Existing and Proposed Design

The two designs are compared in terms of area, power and delay as shown in Table 5.1 below. This comparison proves that DA is more area, delay and power efficient.

Design	Delay (ns)	Power (mW)	LUT's	No. of IOB's
Existing	8.75	85	196	34
Proposed	5.828	42	24	19

Table 5.1 Comparison between DA based proposed structure and existing structure

5.2 IMPLEMENTATION OF DEDICATED SQUARING UNIT FOR AREA UTILIZATION USING XILINX ISE DESIGN SUITE

This squaring unit is area-delay efficient as compared to normal squaring unit as shown in synthesis report.

A. VHDL Code For Dedicated Squaring Unit

The VHDL code using structural architecture (Appendix B) explains the squaring operation by using different case statements and variables.

B. Simulation Results

The simulation is done using I-SIM simulator as shown in Figure 5.5.

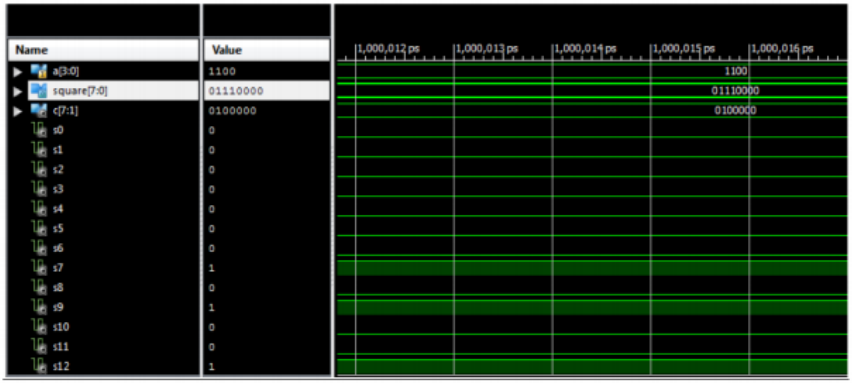


Figure 5.5 Simulation result of proposed squaring unit

Area utilization in synthesis report for two structures is shown in Figure 5.6 and 5.7,respectively.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	6	126800	0%
Number of Slice LUTs	19	63400	0%
Number of fully used LUT-FF pairs	4	21	19%
Number of bonded IOBs	12	210	5%

Figure 5.6 Area Utilization in Proposed Structure

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	127	63400	0%
Number of fully used LUT-FF pairs	0	127	0%
Number of bonded IOBs	16	210	7%

Figure 5.7 Area Utilization in Existence Structure

C. Comparison between Existing and Proposed Design

The comparison between two designs is done on the basis of area and delay as shown in Table 5.2 which shows that dedicated unit is more efficient.

Design	Delay (ns)	No. of LUT's
Existing	12.68	127
Proposed	4.72	19

Table 5.2 Comparison between two structures

5.3 IMPLEMENTATION OF FULL ADDER DESIGN IN CADENCE-VIRTUOSO 180nm TECHNOLOGY

The Full Adder unit is designed using NMOS Pass transistors, in order to scale down the required no. of transistors, using AND,OR and XOR gates based on NMOS Pass Transistors which further reduces device utilization to make area efficient design.

A. Design and Analysis of Gates Based on NMOS Pass Transistor

- AND GATE Design and simulation

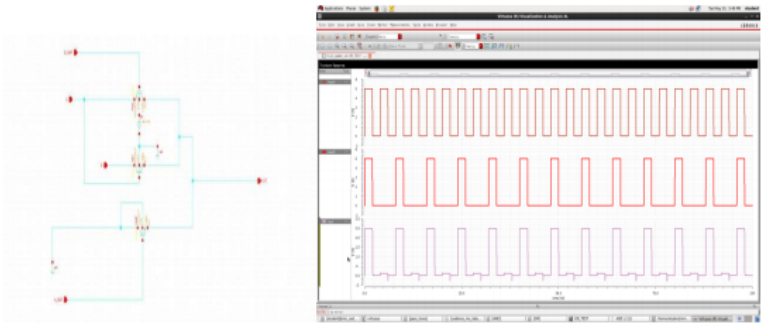


Figure 5.8 AND Design using NMOS Technology

- OR GATE Design and Simulation

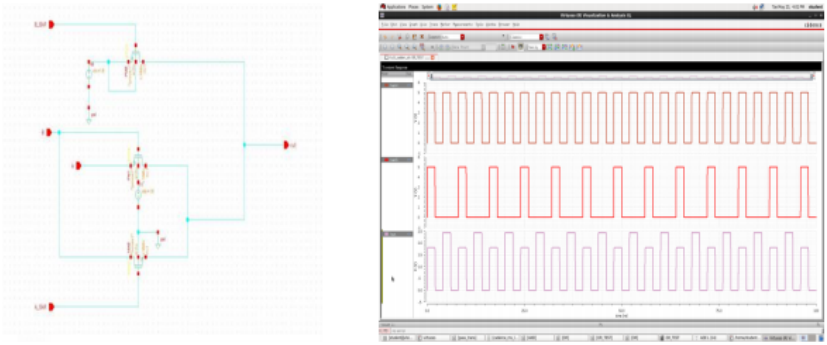


Figure 5.9 OR Design Using NMOS Technology

- XOR GATE Design and Simulation

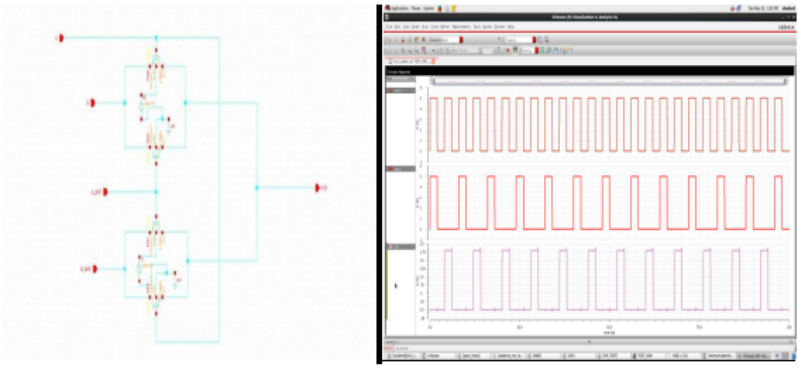


Figure 5.10 XOR Design Using NMOS Technology

B. Simulation Result and Power Analysis of NMOS Full Adder Unit

The simulation is done in Analog Design Environment (ADE) Using Cadence180nm technology and power is calculated using psf file as shown in figure 5.11 below:

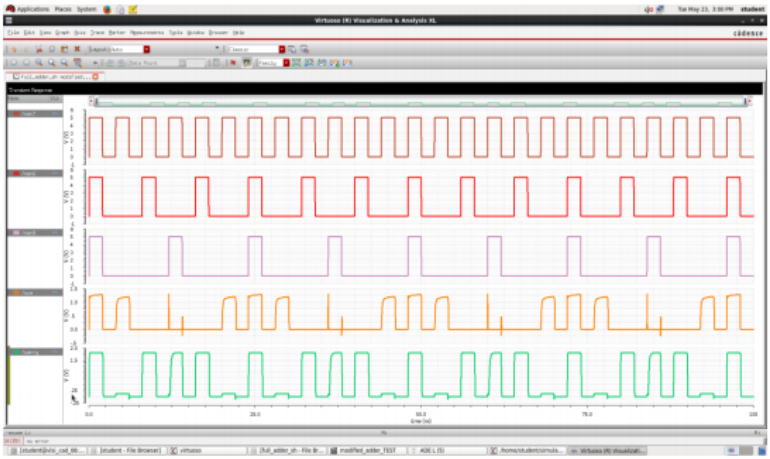


Figure 5.11 Simulation Result of NMOS Full Adder in ADE

Figure 5.12 Power Estimation of NMOS Full Adder using psf file

C. Simulation Result and Power Analysis of CMOS Full Adder Unit

The simulation is done in Analog Design Environment (ADE) Using Cadence180nm technology and power is calculated using psf file as shown in figure 5.13 below:

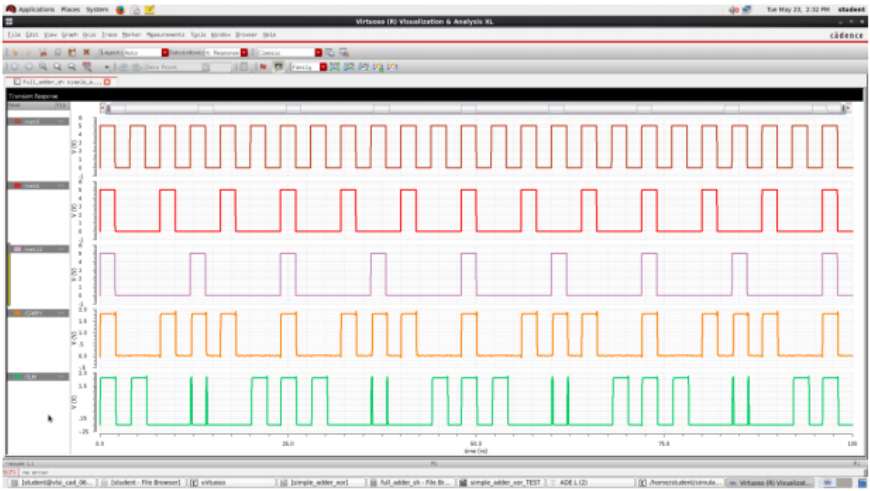


Figure 5.13 Simulation Result of CMOS Full Adder in ADE

Figure 5.14 Power Estimation of CMOS Full Adder using psf file

D. Comparison between Two Designs

The proposed design is area as well as power efficient than CMOS unit as shown in Table 5.3 below:

Components	CMOS F.A	NMOS Pass Transistor F.A	Reduction (%)
Power(W)	126.6X 10^{-6}	89.93X10^{-6}	28.74
Transistor count	38	29	23.68

Table 5.3 Comparison between CMOS and NMOS Adder Design

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

In this work, adaptive filter is explored due to its huge applications in digital processing field, implemented using LMS algorithm because of its simplicity and computational efficiency. In order to achieve efficient hardware implementation on VLSI technology, a new programmable architecture of distributed arithmetic based BLMS adaptive filter has been proposed and implemented successfully for area delay and power optimization. By using block least mean square method, we could decrease the inner products which have further reduced the block size and hence the area and delay. Furthermore, the number of look up table words to be updated per output has been minimized. This has saved external logic and power consumption. Compared to the available traditional design, this new approach has reduced the area delay product (ADP) up to 33 %. The simulation results have shown that the proposed method gives enhanced performance than the existing methods in terms of area and power consumption. This performance is achieved at the cost of a slightly increased computational complexity, having major advantage for large block size and filter length. As filtering data in real-time demands dedicated hardware to meet time requirements, dedicated multiplier units are designed as multipliers are the major block of adaptive filters and causes high latency and computational complexity.

Different multiplier structures for binary multiplication e.g. parallel and tree multipliers have been studied in detail. As squaring and cubing are the most frequent perform operation in ubiquitous DSP functions and generation of square and cube requires high computational complexity, dedicated units are designed and implemented. The proposed squaring unit is implemented to reduce the height of partial product matrix and simulation results shows that proposed unit requires less area and time as compared to conventional method. Further, as cubing unit become more complex for higher radix number, it is implemented for low power design with reduced partial products using NMOS Pass Transistor based adder design which also reduces transistor count and hence the area utilization.

6.2 FUTURE SCOPE

In future this work can be extended to address the following issues:

1.The number of words to be stored in a LUT can be reduced further, by using OBC (Offset Binary Coding) technique which reduces memory requirement.

2.The proposed adaptive filter structure can be modified by implementing pipelining technique using relaxed look-ahead algorithm to reduce its power dissipation.

3.The proposed adaptive filter structure can be further improved by exploiting parallelism in the LMS algorithm architecture to meet the desired computational throughput and power requirements but as it results in adaptation delay, a tradeoff between algorithm convergence speed and the power reduction factor is mandatory so that ultra high speed adaptive filters can be designed.

4.The dedicated multiplier units can be further implemented for higher radix number, for high speed and reduced complexity, by using VLSI system designed low power carry save adder architecture. Instead of CMOS and NMOS Pass transistor logics, other technologies can be utilized.

References

[1] S. Haykin, *Adaptive Filter Theory*. Upper Saddle River, NJ: Prentice-Hall, 2004.

[2] B. Widrow and S. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[3] A. Sayed, *Fundamentals of Adaptive Filtering*. New York: Wiley, 2003.

[4] E. Soria, J. D. Martín, A. J. Serrano, J. Calpe, and J. Chambers (2007). Steady-state and tracking analysis of a robust adaptive filter with low computational cost, *Signal Process*ing, 87(1), 210–215.

[5] Jih-Gau Juang, Li-Hsiang Chien and F. Lin (2011). Automatic Landing Control System Design Using Adaptive Neural Network and its Hardware Realization, *IEEE Systems Journal*, 5(2), 266-277.

[6] K. J. Lee, J.P.Lee, D. Shin and D. W. Yoo (2014). A Novel Grid Synchronization PLL Method Based on Adaptive Low-Pass Notch Filter for Grid-Connected PCS, *IEEE Trans. Ind. Electron.*, 61(1), 292–301.

[7] Jimaa S and Al-Ali S (2011). Convergence Performances of various Adaptive Filter Algorithms with Application to System Identification," *Proceedings of IEEE GCG conference and Exhibition*, [4th: Dubai: 2011], pp. 19-22.

[8] B. F. Boroujeny, *Adaptive Filters: Theory and Applications*. New York: Wiley, 1998.

[9] R. H. Kwong and E. W. Johnsto (1992). A variable step size LMS algorithm, *IEEE Trans. Signal Process.*, 40(7) , 1633-1642.

[10] B. K. Mohanty and P. K. Meher (2013). A High-Performance Energy-Efficient Architecture for FIR Adaptive Filter Based on New Distributed Arithmetic Formulation of Block LMS Algorithm, *IEEE Transactions on Signal Processing*, 61(4), 921-932.

[11] M. D. Meyer and D. P. Agrawal (1992). An Arbitrarily High Sampling Rate DLMS Adaptive Filter, *IEEE Transactions On Signal Processing*, 40(11), 2176-2179.

[12] K. K. Parhi, *VLSI Digital Signal Processing Systems—Design and Implementation*. New York: Wiley, 1999.

[13] E. S. Nejevenko and A. A. Sotnikov (2006). Adaptive modeling for Hydroacoustic signal processing, *Pattern Recognition Image Analysis*, 16(1) , 5-8.

[14] S. Ikeda and A. Sugiyama (1999). An adaptive noise canceller with low signal distortion for speech codec, *IEEE Transactions on Signal Processing*, 47(3), 665-674.

[15] P. Prandoni and M. Vetterli (1998). An FIR Cascade Structure for Adaptive Linear Prediction, *IEEE Transactions on Signal Processing*, 46(9), 2566-2571.

[16] M. S. Burt (2007). Inverse Identification Adaptive IIR Filtering: Convergence Speed Analysis and Successive Approximations Algorithm, in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, 2007, pp. 1309 - 1312.

[17] Monson Hayes H, *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons Inc: Kundli , 2002.

[18] B.Widrow, J. M. McCool, M. G. Larimore and C. R. Johnson Jr.(1976). Stationary and non stationary learning characteristics of the LMS Adaptive Filters, *Proceedings of the IEEE*, 64(8), 1151-1162.

[19] D. L. Jones (1992). Learning Characteristics of Transpose-Form LMS Adaptive Filters, *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 39(10),745-749.

[20] D. T. M. Slock (1993). On the Convergence Behavior of the LMS and the Normalized LMS Algorithms, *IEEE Transactions on Signal Processing*, 41(9), 2811-2825.

[21] I Tudosa and Adochiei N(2012). LMS Algorithm Derivatives used in Real-Time Filtering of ECG Signals: A Study Case on Performance Evaluation, *Proceedings of IEEE International Conference and Exposition on Electrical and Power Engineering*, [5th: Romania: 2012], pp.565-570.

[22] D.J.Allred, H.Yoo, V. Krishnan, W. Huang, and D.V. Anderson (2005). LMS adaptive filters using distributed arithmetic for high throughput, *IEEE Trans. Circuits Syst.*, 52(7), 1327–1337.

[23] D.J.Allred, H.Yoo, V.Krishnan, W.Huang, and D.V.Anderson (2004). An FPGA implementation for a high throughput adaptive filter using distributed arithmetic, in *Proc. IEEE Symp. Field-Programmable Custom Computing*, [12th: 2004], pp. 324–325.

[24] K.Mohanty and Pramod.k.Meher (2013). High-Performance Energy- Efficient Architecture for FIR Adaptive Filter Based on New Distributed Arithmetic Formulation of Block LMS Algorithm, *IEEE transactions on signal processing*, 61(4), 5-8.

[25] Basant.K.Mohanty (2002). Digit-Serial Architecture for VLSI implementation of Delayed LMS FIR Adaptive Filters, *IEEE transactions on signal processing*, 45(4), 1-8.

[26] Basant. K. Mohanty, Pramod Kumar Meher. (2008). Delayed Block LMS algorithm and concurrent architecture for high-speed implementation of Adaptive Filters, *IEEE Region Conference*, [10th : 2008], pp. 1-5 .

[27] Mohanty, B.K., Meher, P.K. (2013) 'A High-Performance Energy-Efficient Architecture for FIR Adaptive Filter based on new Distributed Arithmetic formulation of Block LMS algorithm', *IEEE Transactions on Signal Processing*, 45(6), 921-932 .

[28] S.C.Douglas, Q.Zhu, and K.F.Smith (1998). A pipelined LMS adaptive FIR filter architecture without adaptive delay, *IEEE Transactions on Signal Processing*, 54(8), 11-54.

[29] Allred, D.J., Yoo and H.Krishnan (2004). A Novel High Performance Distributed Arithmetic Adaptive Filter implementation on an FPGA, *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP),* [4th: Atlanta, US: 2004], pp. 161-164.

[30] S. Baghel and R. Shaik (2011). FPGA implementation of fast block LMS adaptive filter using distributed arithmetic for high-throughput, in *Proceeding International Conference on Communication Signal Processing (ICCSP)*, [11th : 2011], pp 443-447.

[31] S. Baghel and R. Shaik (2011). Low power and less complex implementation of fast block LMS adaptive filter using distributed arithmetic, in *Proceeding IEEE Students Technology Symposium* , [Kharagpur,India:2011], pp 15-20.

[32] C.R, Wooley and B.A (1973). A Two's complement parallel array multiplication algorithm, *IEEE Transactions on Computers*, 22(7), 1045-1047.

[33] Dadda, L (1965). Some schemes for parallel multipliers, *Alta Frequenza*, 34(7), 349-356.

[34] Douglas S.C, Zhu Q and Smith K.F (1998). A Pipelined LMS Adaptive FIR Filter Architecture without adaptive delay,*IEEE Transactions on Signal Processing*,39(1),19-31.

[35] D.P. Das, G. Panda, and S.M. Kuo (2007). New block filtered X LMS algorithms for active noise control systems, *IEEE Transactions on Signal Processing.*, 1(2), 73–81.

[36] Meher P.K, Park S.Y (2014). Area-Delay-Power Efficient Fixed-Point LMS Adaptive Filter with low adaptation-delay, *IEEE Transactions on Very Large Scale Integrated Systems*, 34(9), 362-371.

[37] Wallace, C.S (1964). A suggestion for a fast multiplier, *IEEE Transactions on Computers*, 13(2), 14-17.

[38] Guo R and Debrunner L.S (2011). Two high performance adaptive filter implementation schemes using distributed arithmetic, *IEEE Transactions on Circuits Systems II, Exp. Briefs*, 58(9), 600–604.

[39] Jayashri R, Chitra H and Kusuma H (2011). Memory based architecture to implement simplified block LMS algorithm on FPGA, in *Proceedings of International Conference on Commununication Signal Process. (ICCSP)*, [10th: Calicut, India: 2011], pp. 179-183.

[40] Huang W and Anderson D.V (2009). Adaptive filters using modified sliding-block distributed arithmetic with offset binary coding," in *Proceedings on IEEE International Conference on Acoustic, Speech, Signal Process. (ICASSP)*, [15th: Taipei, Taiwan: 2009], pp 545-548.

[41] Shen Q and Spanias A.S (1996). Time and frequency domain block LMS algorithm for single channel active noise control, *Control Engineering,* Arizona State University: US.

[42] Van L.D and Feng W.S (2006). An efficient architecture for the DLMS adaptive filters and its applications, *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.* 48(4), 359–366.

[43] Van L.D and Feng W.S (2000). Efficient systolic Architectures for 1-D and 2-D DLMS adaptive digital filters," in *Proc. IEEE Asia Pacific Conference Circuit*, [5th: Tianjin, China: 2000], pp 535-539.

[44] Visvnathan V and Ramanathan R (1995). A modular systolic architecture for delayed least mean square adaptive filtering, in *Proceedings on IEEE International Conference on VLSI Design*, [4th : India : 1995], pp. 332–337.

[45] Tiwari H.D, Gankhuyag G andKim C.M (2008). Multiplier design based on ancient Indian Vedic Mathematics, *in Proceedings IEEE International SOC Design Conference*, [Busan:2008], pp. 65-68.

[46] Fengqi Y.Yu and Willson A. N.(2001). Multirate digital squarer architectures, in Proceedings on *IEEE International conference on Electronics, Circuits and Systems (ICECS),* [8th : Malta:2001], pp 25-29.

[47] Arunachalain T and Kirubaveni S (2013). Analysis of High Speed Multipliers, *in proceedings of IEEE International Conference on Communication and Signal Processing*, [8th: India:2013], pp. 211-214.

[48] Kwon Taek-Jun, Sondeen Jeff and Draper Jeffrey (2007). Floating-Point Division and Square Root using a Taylor-Series Expansion Algorithm, *in Proceedings of IEEE International Midwest Symposium on Circuits and Systems*, [50th :India: 2007] , pp. 305 – 308.

[49] Booth A (1951). A signed binary multiplication technique, *Quarterly J. Mechanics & Applied Mathematics*, 4(2), 236- 240.

[50] Macsorley O.L (1961). High-Speed Arithmetic in Binary Computers, *in Proceedings of the IRE* , 49(1), 67-91.

[51] Yen W. C and Jen C. W (2000). High Speed Booth encoded Parallel Multiplier Design, *IEEE transactions on Computers*, 49(7), 692-701.

[52] Chidgupkar P.D and Karad M.T (2004). The Implementation of Vedic Algorithms in Digital Signal Processing, *Global Journal of Engineering Education*, 8(11), 153-158.

[53] Dhillon H.S and Mitra A(2008). A Reduced-Bit Multiplication Algorithm for Digital Arithmetic, *International Journal of Computational and Mathematical Sciences*, pp.64-69.

[54] Mehta P and Gawali D (2009). Conventional versus Vedic Mathematical Method for Hardware Implementation of a Multiplier, *in Proceedings of International Conference on Advances in Computing, Control, and Telecommunication Technologies*, [5th : Kerala, India: 2009], pp.640-642.

[55] Nair P, Paranji D, and Rathod S.S (2008). VLSI Implementation of Matrix- Diagonal Method of Binary Multiplication, in *Proc. of SPIT-IEEE Colloquium and International Conf.*, [11th : Mumbai, India:2008], pp.55-58.

[56] Ramalatha M, Dayalan K.D, Dharani P and Priya S.D (2009). High Speed Energy Efficient ALU Design using Vedic Multiplication Technique, *Lebanon* , pp. 600-603,

[57] Thapliyal H and Srinivas M.B (2005). VLSI Implementation of RSA Encryption System Using Ancient Indian Vedic Mathematics, *in Proceedings of SPIE VLSI Circuits and Systems II*, 48(4), pp.888-892.

[58] Kunchigi Vaijyanath and Linganagouda Kulkarni (2013). 32-Bit MAC Unit Design Using Vedic Multiplier, *International Journal of Scientific and Research Publications (IJSRP)*, 3(2), 15-25.

[59] Stine J. E. and Schulte M. J (1999). The symmetric table addition method for accurate function approximation. *Journal of VLSI Signal Processing*, 21(4), 167–177.

[60] Ercegovac M. D, Lang T and Muller J.M (2000). Reciprocal, square root, inverse square root and some elementary functions using small multipliers, *IEEE Transactions on Computers*, 49(7), 628–637.

[61] Schulte M. J, Marquette L. P and Krithivasan S (2003). Combined multiplication and sum-of squares units, i*n Proceedings of Application-Specific Systems, Architectures, and Processors*, 34(9), 204–214.

[62] Stine J.E, Grad J and Castellanos I (2005). A framework for high-level synthesis of system on chip designs, *In International Conference on Microelectronic Systems Education*, [5th : China: 2005], pp 67–68.

[63] Kunchigi V ,Kulkarni L and Kulkarni S (2012). High speed and area efficient Vedic multiplier, *International Conference on Devices, Circuits and Systems (ICDCS)*, [4th : India : 2012],pp.360,364.

[64] Liddicoat A. A. and Flynn M. J (2000). Parallel square and cube computations, *In Proceedings of the Thirty Fourth Asilomar Conference on Signals, Circuits and Systems,* [34th: China : 2000], pp 1325–1329.

[65] Blank J.M, *Optimized parallel cubing units*, Illinois Institute of Technology Chicago, December 2004.

[66] Wires K. E, Schulte M. J and Balzola P. I (1999). Combined unsigned and two's complement squarers, *In Proceedings of the Thirty Third Asilomar Conference on Signals, Circuits and Systems,* [33rd: US: 1999], pp 1215–1219.

APPENDIX

APPENDIX A: Verilog Code for Distributed Arithmetic Based Block LMS Algorithm

```
module fir_filter (clk,rst,y,xo);
input clk,rst;
input [9:0] xo;
output [21:0] y;
reg [21:0] y;
reg [9:0] r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15,r16;
reg [12:0] temp;
reg [21:0] calc;
reg [3:0] count;
reg load;

always@( posedge clk or posedge rst )
begin

if (rst)
begin
r1<=0;r2<=0;r3<=0;r4<=0;r5<=0;r6<=0;r7<=0;
r8<=0;r9<=0;r10<=0;r11<=0;r12<=0;r13<=0;r14<=0;r15<=0;r16<=0;
temp=0;
calc=0;
count=0;
load=1;
y=0;
end

else
begin
if(load)
```

```
begin
r1<=xo;r2<=r1;r3<=r2;r4<=r3;r5<=r4;r6<=r5;r7<=r6;
r8<=r7;r9<=r8;r10<=r9;r11<=r10;r12<=r11;r13<=r12;r14<=r13; r15<=r14;r16<=r15;

load=0;
count=0;
end

else
begin

temp = block1value (r1[count],r2[count], r3[count], r4[count] ) +
        block2value (r5[count],r6[count], r7[count], r9[count] ) +
        block2value ( r12[count],r11[count],r10[count],r9[count] ) +
        block1value ( r16[count],r15[count],r14[count],r13[count] )  ;

if (count == 4'b1001)
calc = calc+ {(temp << count)+1 } ;

else
calc = calc+ {
(temp << count)
} ;

if (calc[12]-1'b1)
calc[21:13]=9'b111111111;

else
calc[21:13]=9'b000000000;
```

```
if(count==4'b1001)
begin
y=calc;
calc=0;
temp=0;
load=1;
end

else
count=count+1;
end
end
end

function [12:0]block1value;
input a1,a2,a3,a4;
begin
case ({
a1,a2,a3,a4
})

4'd 0 : block1value=13'd 0;
4'd 1 : block1value=13'd 2048;
4'd 2 : block1value=13'd 1111;
4'd 3 : block1value=13'd 0981;
4'd 4 : block1value=13'd 7;
4'd 5 : block1value=13'd 6;
4'd 6 : block1value=13'd 9;
4'd 7 : block1value=13'd 172;
4'd 8 : block1value=13'd 197;
4'd 9 : block1value=13'd 941;
```

```
4'd 10 : block1value=13'd 111;
4'd 11 : block1value=13'd 1;
4'd 12 : block1value=13'd 3;
4'd 13 : block1value=13'd 90;
4'd 14 : block1value=13'd 40;
4'd 15 : block1value=13'd 50;
    default block1value=13'd 0;
endcase
end
endfunction

function [12:0]block2value;
input b1,b2,b3,b4;
begin
case ({
b1,b2,b3,b4
})

4'd 0 : block2value=13'd 0;
4'd 1 : block2value=13'd 2;
4'd 2 : block2value=13'd 11;
4'd 3 : block2value=13'd 09;
4'd 4 : block2value=13'd 7;
4'd 5 : block2value=13'd 6;
4'd 6 : block2value=13'd 9;
4'd 7 : block2value=13'd 13;
4'd 8 : block2value=13'd 1;
4'd 9 : block2value=13'd 9;
4'd 10 : block2value=13'd 11;
4'd 11 : block2value=13'd 1;
4'd 12 : block2value=13'd 3;
```

```
4'd 13 : block2value=13'd 9;
4'd 14 : block2value=13'd 4;
4'd 15 : block2value=13'd 5;
  default block2value=13'd 0;
endcase
end
endfunction

endmodule
```

APPENDIX B: VHDL Code For Fast Squaring Unit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity squaring_fast is
port( a:in std_logic_vector(3 downto 0);
square:out std_logic_vector(7 downto 0));
end squaring_fast;

architecture structural of squaring_fast is

component and21 is
Port ( n,b : in STD_LOGIC;
m : out STD_LOGIC);
end component;
component fulladder12 is
Port ( u,v,w : in STD_LOGIC ;
x,y : out STD_LOGIC );
end component;
```

```
signal c: std_logic_vector (7 downto 1);
signal s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14:std_logic ;
begin
A1 :and21 port map (a(0),a(1),square(1));
A2: and21 port map (a(0),a(2),s0);
A3: and21 port map (a(1),a(2),s2);
A4: and21 port map (a(0),a(3),s3);
A5: and21 port map (a(1),a(3),s6);
A6: and21 port map (a(2),a(3),s9);
FA8:fulladder12 port map (s4,s1,s13,square(3),s5);
FA1:fulladder12 port map (a(1),s0,c(1),square(2),s1);
FA2:fulladder12 port map (s3,s2,c(2),s4,s13);
FA3:fulladder12 port map (a(2),s6,c(3),s7,s14 );
FA4:fulladder12 port map (s7, s14,s5,square(4),s8);
FA5:fulladder12 port map (c(4),s8,s9,square(5),s10);
FA6:fulladder12 port map (a(3),c(6),s10,s11,s12);
FA7:fulladder12 port map (s11,s12,c(7),square(6),square(7));
P:process(a)
begin
if ( a(0) and a(1) ) = '1' then
c(1) <= '1';
elsif ( a(0)and a(2)) = '1' then
c(2)<= '1';
elsif ( a(0) and a(3)) = '1' then
c(3) <= '1';
elsif ( a(2) and a(1)) = '1' then
c(4) <= '1';
elsif ( a(3) and a(1)) = '1' then
c(5) <= '1';
elsif ( a(2) and a(3) )= '1' then
```

```
c(6) <= '1';
elsif a(3)='1' then
c(7)<='1';
else c <= "0000000";
end if;

end process;
square(0) <= a(0);
end structural;
```

PUBLICATIONS

1. Sharanjit Kaur and Ravi Kumar, "Implementation of a Low Power Area Efficient Cubing Unit with Reduced Partial Product using NMOS Pass Transistor," in *Proc. of IEEE 4th International Conference on Signal Processing, Computing and Control (ISPCC),* 2017. (Communicated).

2. Sharanjit Kaur and Ravi Kumar, "Low Power and Area Efficient Cubing Unit with Reduced Partial Product using NMOS Pass Transistor," *International Journal of Engineering Research and Technology (IJERT)*, 6(7), 2017, (Communicated).

3. Sharanjit Kaur and Ravi Kumar, "Power and Area Efficient Cubing Unit with Reduced Partial Product using NMOS Pass Transistor," *Canadian Journal of Electrical and Computer Engineering* , 2017, (Communicated).

Sharanjit_18-08-2017

by Sharanjit Kaur

Submission date: 18-Aug-2017 11:01AM (UTC+0530)
Submission ID: 837958218
File name: modified_thesis_plagcheck.docx (2.93M)
Word count: 14542
Character count: 78092

Sharanjit_18-08-2017

ORIGINALITY REPORT

35%	21%	29%	13%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Gowtham, N., and P. Babu. "An efficient architecture for BLMS adaptive filter based on distributed arithmetic technique", 2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE), 2014. Publication	5%
2	cmc.rice.edu Internet Source	3%
3	ijese.org Internet Source	2%
4	Submitted to Visvesvaraya Technological University Student Paper	1%
5	Salah, Mohamed, Abdel-Halim Zekry, and Mohammed Kamel. "FPGA implementation of LMS adaptive filter", 2011 28th National Radio Science Conference (NRSC), 2011. Publication	1%

Jeff M. Blank. "Partial Product Reduction for