IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 6, Issue 3, Ver. I (May. -Jun. 2016), PP 58-66
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Performance Analysis of Different Multipliers for Embedded and DSP Applications

K. Vijetha[1], S. Ch. Vijaya Bhaskar[2]
[1]*K. Vijetha, Assistant Professor, Department of ECE, Matrusri Engineering College, Hyderabad*
[2]*S. Ch. Vijaya Bhaskar, Assistant Professor, Department of IT, MVSR Engineering College, Hyderabad*

Abstract: *This Paper presents an efficient implementation of high speed multiplier using the shift and add method, Radix_2, Radix_4 modified Booth multiplier algorithm. In This paper we compare the working of the three multipliers by implementing each of them separately in FIR filter. The parallel multipliers like radix2 and radix4 modified booth multiplier does the computations using lesser adders and lesser iterative steps. As a result of which they occupy less space as compared to the serial multiplier. This a very important criteria because in the fabrication of chips and high performance systems requires components which are as small as possible. Low power consumption and smaller area are the most important criteria for the fabrication of DSP systems and high performance systems. Optimizing the speed and area of the multiplier is a major design issue. However, area and speed are usually conflicting constraints improving speed results mostly in larger areas. This is to determine the best solution to this problem by comparing a few multipliers. The paper gives the analysis of comparison of power consumption of all the multipliers & we find that serial multipliers consume more power. So where power is an important criteria parallel multipliers like booth multipliers are preferred to serial multipliers.*

Index Terms: *Multipliers, FPGA, DSP, Embedded and VHDL*

I. Introduction

Multipliers are key components of many high performance systems such as FIR filters, microprocessors, digital signal processors, etc. A system's performance is generally determined by the performance of the multiplier because the multiplier is generally the slowest clement in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a major design issue. However, area and speed are usually conflicting constraints so that improving speed results mostly in larger areas. As a result, a whole spectrum of multipliers with different area-speed constraints have been designed with fully parallel. Multipliers at one end of the spectrum and fully serial multipliers at the other end. In between are digit serial multipliers where single digits consisting of several bits are operated on. These multipliers have moderate performance in both speed and area. However, existing digit serial multipliers have been plagued by complicated switching systems and/or irregularities in design. Radix 2^n multipliers which operate on digits in a parallel fashion instead of bits bring the pipelining to the digit level and avoid most of the above problems. They were introduced by M. K. Ibrahim in 1993. These structures are iterative and modular. The pipelining done at the digit level brings the benefit of constant operation speed irrespective of the size of' the multiplier. The clock speed is only determined by the digit size which is already fixed before the design is implemented. Many DSP applications demand high throughput and real-time response, performance constraints that often dictate unique architectures with high levels of concurrency. DSP designers need the capability to manipulate and evaluate complex algorithms to extract the necessary level of concurrency. Performance constraints can also be addressed by applying alternative technologies. A change at the implementation level of design by the insertion of a new technology can often make viable an existing marginal algorithm or architecture.

The VHDL language supports these modeling needs at the algorithm or behavioral level, and at the implementation or structural level. It provides a versatile set of description facilities to model DSP circuits from the system level to the gate level. Recently, we have also noticed efforts to include circuit-level modeling in VHDL. At the system level we can build behavioral models to describe algorithms and architectures. We would use concurrent processes with constructs common to many high-level languages, such as if, case, loop, wait, and assert statements. VHDL also includes user-defined types, functions, procedures, and packages." In many respects VHDL is a very powerful, high-level, concurrent programming language. At the implementation level we can build structural models using component instantiation statements that connect and invoke subcomponents. The VHDL generate statement provides ease of block replication and control. A dataflow level of description offers a combination of the behavioral and structural levels of description. VHDL lets us use all three levels to describe a single component. Most importantly, the

standardization of VHDL has spurred the development of model libraries and design and development tools at every level of abstraction. VHDL, as a consensus description language and design environment, offers design tool portability, easy technical exchange, and technology insertion

II. Design of Filter

Digital filters are very important part of DSP. Infact their extraordinary performance is one of the key reasons that DSP has become so popular. Filters have two uses: signal separation and signal restoration. Signal separation is needed when the signal has been contaminated with interference, noise or other signals. For example imagine a device for measuring the electrical activity of a baby's heart (EKG) while in the womb. The raw signal will be likely to be corrupted by the breathing and the heartbeat of the mother. A filter must be used to separate these signals so that they can be individually analyzed.

Signal restoration is used when the signal has been distorted in some way. For example, an audio recording made with poor requirement may be filtered to better represent the sound as it actually occurred. Another example is of debluring of an image acquired with an improper focused lens, or a shaky camera.

These problems can be attacked with either digital or analog filters. Analog filters are cheap, fast and have a large dynamic range both in amplitude and frequency. Digital filters in comparison are vastly superior in the level of performance that can be achieved. Digital filters can achieve thousands of times better performance than an analog filter. This makes a dramatic difference in how filtering problems are approached. With analog filters, the emphasis is on handling limitations of the electronics such as the accuracy and stability of the resistors and capacitors. In comparison digital filters are so good that the performance of the filter is frequently ignored.

The emphasis shifts to the limitations of the signals and the theoretical issues regarding their processing. It is common in DSP to say that a filter input and output signals are in time domain. This is because signals are usually created by sampling at regular intervals of time. But this is not the only way sampling can take place. The second most common way of sampling is at equal intervals in space. For example imagine taking simultaneous readings from an array of strain sensors mounted at one centimeter increments along the length of an aircraft wing. Many other domains are possible; however, time and space are by far the most common. Then you see the term time domain in DSP, remember that it may actually refer to samples taken over time, or it may be a general reference to any domain that the samples are taken in.

Every linear filter has an impulse response, a step response and a frequency response. Each of these responses contains complete information about the filter, but in a different form. If one of three is specified, the other two are fixed and can be directly calculated. All three of these representations are important, because they describe how the filter will react under different circumstances. The most straightforward way to implement a digital filter is by convolving the input signal with the digital filter's impulse response. All possible linear filters can be made in this manner.

When the impulse response is used in this way, filters designers give it a special name: the filter kernel. There is also another way to make digital filters, called recursion. When a filter is implemented by a convolution, each sample in the output is calculated by weighting the samples in the input, and adding then together. Recursive filters are an extension of this, using previously calculated values from the output, besides points from the input. Instead of using a filter kernel, recursive filters are defined by a set of recursion coefficients. For now the important point is that all linear filters have an impulse response, even if you don't use it to implement the filter. To find the impulse response of a recursive filter, simply feed in the impulse and see what comes out. The impulse responses of recursive filters are composed of sinusoids that exponentially decay in amplitude. In principle, this makes their impulse responses infinitely long. However the amplitude eventually drops below the round off noise of the system, and the remaining samples can be ignored. Because of these characteristics, recursive filters are also called Infinite impulse response or IIR filters. In comparison, filters carried out by convolution are called Finite impulse response or FIR filters.

The impulse response is the output of a system when the input is an impulse. In this same manner, the step response is the output when the input is a step. Since the step is the integral multiple of the impulse response. This provides two ways to find the step response: (1) feed a step waveform into the filter and see what comes out. (2) Integrate the impulse response. The frequency response can be found out by taking the DFT of impulse response.

Time domain Parameters

It may not be obvious why the step response is of such concern in time domain filters. You may be wondering why the impulse response isn't the important parameter. The answer lies in the way that the human mind understands and processes information. Remember that the step, impulse and frequency responses all contain identical information, just in different arrangements. The step response is useful in time domain analysis because it matches the way humans view the information contained in the signals.

For example, suppose you are given a signal of some unknown origin and asked to analyze it. The first thing you will do is divide the signal into regions of similar characteristics. You can't stop from doing this; your mind will do that automatically.

Some of the regions may be smooth; others may have large amplitude peaks; others may be noisy. This segmentation is accomplished by identifying the points that separate the regions. This is where the step function comes in. the step function is the purest way of representing a division between two dissimilar regions. It can mark when an event starts or when an event ends. It tells you that whatever is on the right.

This is how the human mind views time domain information: a group of step functions dividing the information into region of similar characteristics. The step response, in turn, is important because it describes how the dividing lines are being modified by the filter.

Frequency domain parameters

The purpose of the filters is to allow some frequencies to pass unaltered, while completely blocking other frequencies. The pass band refers to those frequencies that are passed, while stop band contains those frequencies that are blocked. The transition band is between. A fast roll-off means that the transition band is very narrow. The division between the pass band and transition band is called the cut off frequency. In analog filter design ,the cut off frequency is usually defined are less standardized, and it is common to see 99%,90%,70.7% and 50% amplitude levels defined to be the cut off frequency.

Types of filters

High pass, band pass and band reject filters are designed by starting with a low pass filter, and then converting it into the desired response. For this reason, most discussions on filter design only give examples of low pass filters.

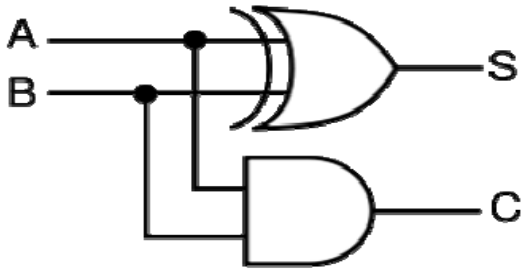
III. Adders Design

In electronics, an adder is a digital circuit that performs addition of numbers. In modern computers adders reside in the arithmetic logic unit (ALU) where other operations are performed. Although adders can be constructed for many numerical representations, such as Binary-coded decimal or excess-3, the most common adders operate on binary numbers. In case where two's complement is being used to represent negative numbers it is trivial to modify an adder into an adder-subtracter

Types of adders

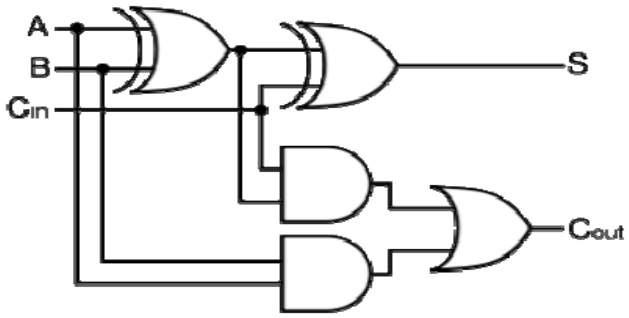
For single bit adders, there are two general types. A half adder has two inputs, generally labelled A and B, and two outputs, the sum S and carry C. S is the two-bit XOR of A and B, and C is the AND of A and B. Essentially the output of a half adder is the sum of two one-bit numbers, with C being the most significant of these two outputs. The second type of single bit adder is the full adder. The full adder takes into account a carry input such that multiple adders can be used to add larger numbers. To remove ambiguity between the input and output carry lines, the carry in is labelled Ci or Cin while the carry out is labelled Co or Cout.

A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits.



Half adder circuit diagram

$$S = A \oplus B$$
$$C = A \cdot B$$



Full adder circuit diagram

A full adder is a logical circuit that performs an addition operation on three binary digits. The full adder produces a sum and carry value, which are both binary digits.

$$S = (A \oplus B) \oplus C_i$$
$$C_o = (A \cdot B) + (C_i \cdot (A \oplus B)) = (A \cdot B) + (B \cdot C_i) + (C_i \cdot A)$$

A full adder can be constructed from two half adders by connecting A and B to the input of one half adder, connecting the sum from that to an input to the second adder, connecting Ci to the other input and or the two carry outputs. Equivalently, S could be made the three-bit xor of A, B, and Ci and Co could be made the three-bit majority function of A, B, and Ci. The output of the full adder is the two-bit arithmetic sum of three one-bit numbers.

IV. Design of Multipliers

Binary Multiplier

A Binary multiplier is an electronic hardware device used in digital electronics or a computer or other electronic device to perform rapid multiplication of two numbers in binary representation. It is built using binary adders.

The rules for binary multiplication can be stated as follows

1. If the multiplier digit is a 1, the multiplicand is simply copied down and represents the product.
2. If the multiplier digit is a 0 the product is also 0. For designing a multiplier circuit we should have circuitry to provide or do the following four things:

1. It should be capable of identifying whether a bit 0 or 1.
2. It should be capable of shifting left partial products.
3. It should be able to add all the partial products to give the products as sum of partial products.
4. It should examine the sign bits. If they are alike, the sign of the product will be a positive, if the sign bits are opposite product will be negative. The sign bit of the product stored with above criteria should be displayed along with the product.

From the above discussion we observe that it is not necessary to wait until all the partial products have been formed before summing them. In fact the addition of partial product can be carried out as soon as the partial product is formed.

Notations:
ı a – multiplicand
ı b – multiplier
ı p – product
Binary multiplication (eg n=4)
p=a×b
an−1 an−2…..a1a0
bn−1bn−2…...b1b0
p2 n−1 p2 n−2…..p1 p0
x x x x a
x x x x b
---
x x x x b0a20
x x x x b1a21

x x x x b2a22
x x x x b3a23
---
x x x x x x x x p

Basic Hardware Multiplier

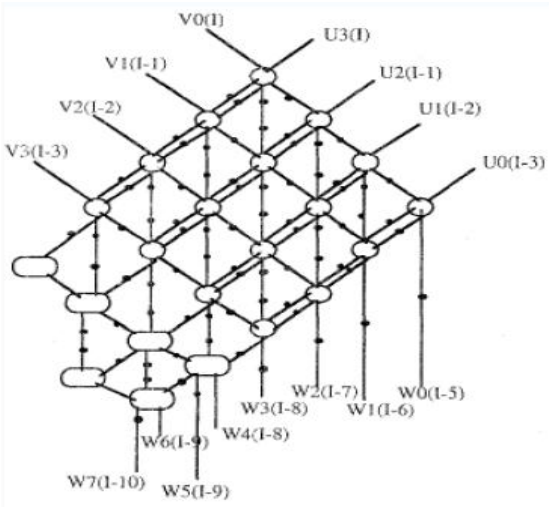
Multiply Accumulate Circuits
Multiplication followed by accumulation is a operation in many digital systems, particularly those highly interconnected like digital filters, neural networks, data quantisers, etc.

One typical MAC (multiply-accumulate) architecture is illustrated in figure. It consists of multiplying 2 values, then adding the result to the previously accumulated value, which must then be restored in the registers for future accumulations. Another feature of MAC circuit is that it must check for overflow, which might happen when the number of MAC operation is large.

This design can be done using component because we have already design each of the units shown in figure. However since it is relatively simple circuit, it can also be designed directly. In any case the MAC circuit, as a whole, can be used as a component in application like digital filters and neural networks

Architecture of A Radix 2^n Multiplier

The architecture of a radix 2^n multiplier is given in the Figure. This block diagram shows the multiplication of two numbers with four digits each. These numbers are denoted as V and U while the digit size was chosen as four bits. The reason for this will become apparent in the following sections. Each circle in the figure corresponds to a radix cell which is the heart of the design. Every radix cell has four digit inputs and two digit outputs. The input digits are also fed through the corresponding cells. The dots in the figure represent latches for pipelining. Every dot consists of four latches. The ellipses represent adders which are included to calculate the higher order bits. They do not fit the regularity of the design as they are used to "terminate" the design at the boundary. The outputs are again in terms of four bit digits and are shown by W's. The 1's denote the clock period at which the data appear.



Architecture of Radix 2^n Multiplier

Booth Multiplier

The decision to use a Radix-4 modified Booth algorithm rather than Radix-2 Booth algorithm is that in Radix-4, the number of partial products is reduced to n/2. Though Wallace Tree structure multipliers could be used but in this format, the multiplier array becomes very large and requires large numbers of logic gates and interconnecting wires which makes the chip design large and slows down the operating speed.

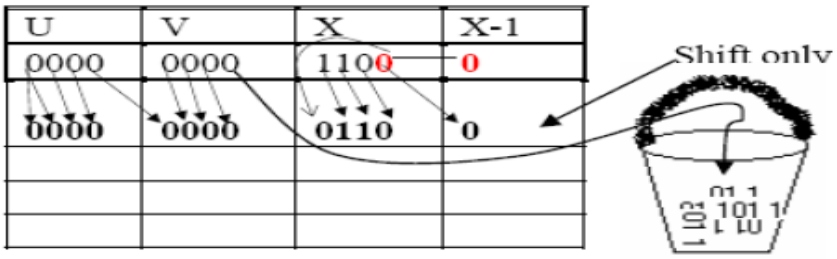
Booth Multiplication Algorithm
Booth Multiplication Algorithm for radix 2
Booth algorithm gives a procedure for multiplying binary integers in signed –2's complement representation.
I will illustrate the booth algorithm with the following example:
Example, $2_{ten} x (- 4)_{ten}$
$0010_{two} * 1100_{two}$

Step 1: Making the Booth table
I. From the two numbers, pick the number with the smallest difference between a series of consecutive numbers, and make it a multiplier.
i.e., 0010 -- From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change, so there are two changes on this one
1100 -- From 1 to 1 no change, 1 to 0 one change, 0 to 0 no change, so there is only one change on this one.
Therefore, multiplication of 2 x (– 4), where $2_{ten}(0010_{two})$ is the multiplicand and $(-4)_{ten}(1100_{two})$ is the multiplier.
II. Let X = 1100 (multiplier)
Let Y = 0010 (multiplicand)
Take the 2's complement of Y and call it –Y
–Y = 1110
III. Load the X value in the table.
IV. Load 0 for X-1 value it should be the previous first least significant bit of X
V. Load 0 in U and V rows which will have the product of X and Y at the end of operation.
VI. Make four rows for each cycle; this is because we are multiplying four bits numbers.

U	V	X	X-1
0000	0000	1100	0

Load the value
1st cycle
2nd cycle
3rd Cycle
4th Cycle

Step 2: Booth Algorithm
Booth algorithm requires examination of the multiplier bits, and shifting of the partial product. Prior to the shifting, the multiplicand may be added to partial product, subtracted from the partial product, or left unchanged according to the following rules:
Look at the first least significant bits of the multiplier "X", and the previous least significant bits of the multiplier "X - 1".
I 0 0 Shift only
1 1 Shift only.
0 1 Add Y to U, and shift
1 0 Subtract Y from U, and shift or add (-Y) to U and shift
II Take U & V together and shift arithmetic right shift which preserves the sign bit of 2's complement number. Thus a positive number remains positive, and a negative number remains negative.
III Shift X circular right shift because this will prevent us from using two registers for the X value.



Repeat the same steps until the four cycles are completed

U	V	X	X-1
0000	0000	1100	0
0000	0000	0110	0
0000	**0000**	**0011**	**0**

← Shift only

U	V	X	X-1	
0000	0000	1100	0	
0000	0000	0110	0	
0000	0000	0011	0	
1110	**0000**	**0011**	**0**	← Add –Y (0000 + 1110 = 1110)
1111	**0000**	**1001**	**1**	← Shift

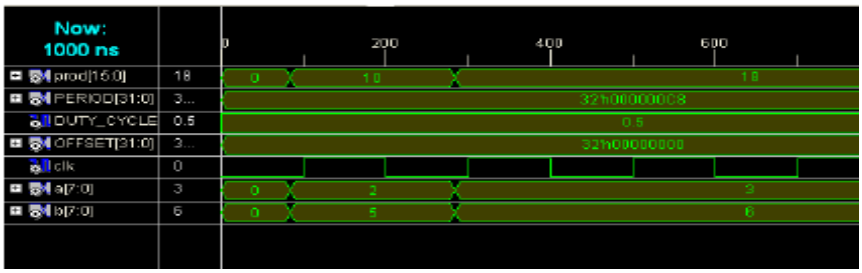
U	V	X	X-1	
0000	0000	*1100*	0	
0000	0000	0110	0	
0000	0000	0011	0	
1110	0000	0011	0	
1111	0000	1001	1	
1111	**1000**	***1100***	**1**	← Shift only

We have finished four cycles, so the answer is shown, in the last row of U and V which is 11111000_{two} Booth multiplication algorithm for radix 4 one of the solutions of realizing high speed multipliers is to enhance parallelism which helps to decrease the number of subsequent calculation stages. The original version of the Booth algorithm (Radix-2) had two drawbacks. They are: (i) the number of add subtract operations and the number of shift operations becomes variable and becomes inconvenient in designing parallel multipliers. (ii) The algorithm becomes inefficient when there are isolated 1's. These problems are overcome by using modified Radix4 Booth algorithm which scan strings of three bits with the algorithm given below:

1) Extend the sign bit 1 position if necessary to ensure that n is even.
2) Append a 0 to the right of the LSB of the multiplier.
3) According to the value of each vector, each Partial Product will be 0, +y, -y, +2y or -2y.
The negative values of y are made by taking the 2's complement and in this paper Carry-look-ahead (CLA) fast adders are used. The multiplication of y is done by shifting y by one bit to the left. Thus, in any case, in designing an n-bit parallel multipliers, only n/2 partial products are generated.

X(i)	X(i-1)	X(i-2)	y
0	0	0	+0
0	0	1	+y
0	1	0	+y
0	1	1	+2y
1	0	0	-2y
1	0	1	-y
1	1	0	-y
1	1	1	+0

V. Results & Discusions



Array Multiplier

Number of Slices	229
Number of 4 input LUTs	302
Number of bonded INPUT	16
Number of bonded OUTPUT	16
CLB Logic Power	104mW

Radix 2 Booth Multiplier

Number of Slices	130
Number of 4 input LUTs	249
Number of bonded INPUT	16
Number of bonded OUTPUT	17
CLB Logic Power	79mW

Radix 4 Booth Multiplier

Number of Slices	229
Number of 4 input LUTs	302
Number of bonded INPUT	16
Number of bonded OUTPUT	16
CLB Logic Power	47mW

This paper gives a clear concept of different multiplier and their implementation in tap delay FIR filter. We find that the parallel multipliers are much better than the serial multiplier. We concluded this from the result of power consumption and the total area. In case of parallel multipliers, the total area is much less than that of serial multipliers. Hence the power consumption is also less. This is clearly depicted in our results. This speeds up the calculation and makes the system faster.

While comparing the radix 2 and the radix 4 booth multipliers we found that radix 4 consumes lesser power than that of radix 2. This is because it uses almost half number of iteration and adders when compared to radix 2.

When all the three multipliers were compared we found that array multipliers are most power consuming and have the maximum area. This is because it uses a large number of adders. As a result it slows down the system because now the system has to do a lot of calculation.

Multipliers are one the most important component of many systems. So we always need to find a better solution in case of multipliers. Our multipliers should always consume less power and cover less area. So through our project we try to determine which of the three algorithms works the best. In the end we determine that radix 4 modified booth algorithm works the best.

VI. Conclusion

In this paper, designed three different type of multipliers using shift and add method, radix2 and radix4 modified booth multiplier algorithm. We used different type of adders like sixteen bit full adder in designing those multiplier. Then we designed a 4 tap delay FIR filter and in place of the multiplication and additions we implemented the components of different multipliers and adders. Then we compared the working of different multipliers by comparing the power consumption by each of them. The result of our project helps us to choose a better option between serial and parallel multiplier in fabricating different systems. Multipliers form one of the most important component of many systems. So by analysing the working of different multipliers helps to frame a better system with less power consumption and lesser area.

References

[1]. Naveen Kumar, Manu Bansal, Navnish Kumar" VLSI Architecture of Pipelined Booth Wallace MAC unit" International Journal of Computer Application(0975-8887)

[2]. Fayed, Ayman A., Bayoumi, Magdy A., "A Merged Multiplier-Accumulator for high speed signal processing applications", IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp 3212 -3215, 2002.

[3]. Abenet Getahun "Booth Multiplication Algorithm" Fall 2003 CSCI 401. [4] Sarita Singh, Sachin Mittal"VHDL Design and implementation for optimum delay and area for Multiplier and Accumulator unit by 32 bit Sequential Multiplier" International Journal of engineering Trends and Technology Volume3 issue 5-2012

[4]. N. Honarmand, M.R.Javaheri, N.SedaghatiMokhtari and A. Afzali-Kusha "PowerEfficient Sequential Multiplication Using Pre-computation" ISCAS 2006.

[5]. S.Shafiulla, Syed Jahangir Badashah "Design and Implementation of Radix 4 based high speed Multiplier for ALU's using minimal partial products" International Journal of Advances in Engineering and Technology.

[6]. S. Jagadeesh, S. Venkata Chary"Design of Parallel Multiplier-Accumulator based on Radix-4 Modified Booth Algorithm with SPST" International Journal of Engineering Research and Application.

[7]. Priya stalin, anuradha, k ranjithkumar, n vaishnav, d vigneswara, s t santhosh "high speed multiplier with pipelining" International Journal of VLSI and Embedded Systems-IJVES .

[8]. Navdeep Kaur, Rajeev Kumar Patial "Implementation of Modified Booth Multiplier using Pipeline Technique on FPGA" International Journal of Computer Applications (0975 – 8887).

[9]. K. Srishylam, Prof. Syed Amjad Ali, M.Praveena " Implementation of Hybrid CSA, Modified Booth Algorithm and Transient power Minimization techniques in DSP/Multimedia Applications" International Journal of Engineering Research and Applications (IJERA) .

[10]. Shanthala S, S. Y. Kulkarni "VLSI Design and Implementation of Low Power MAC Unit with Block Enabling Technique" European Journal of Scientific Research

[11]. Iffat Fatima "Analysis of Multipliers in VLSI" Journal of Global Research in Computer Science.