International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Volume 4, Issue 3 (May-June, 2016), PP. 370-375

COMPARATIVE ANALYSIS OF MULTIPLIER AND MULTIPLIER-LESS METHOD USED TO IMPLEMENT FIR FILTER ON FPGA

Mahesh Golconda[1], Prof. Maruti Zalte[2]
Dept. of Electronics and Telecommunication Engg.,
K. J. Somaiya College of Engineering, Mumbai
Affiliated to Mumbai University
[1]mahesh.golconda@somaiya.edu
[2]marutizalte@somaiya.edu

***Abstract*—Finite impulse response (FIR) filters are a type of digital filter that has a finite impulse response which is used in a communication system and signal processing. FIR filter structure consists of a multiplier, adder, and delay element. The multiplier is one of the key blocks in most digital systems which consume high power and more area. In this paper, FIR filter is implemented using both Multiplier and Multiplierless method. In multiplier method, Modified Booth and a Modified Booth with Wallace tree multiplier is designed while in the multiplier less method, distributed arithmetic and distributed arithmetic with partition is designed using Verilog. The code is simulated in Model Sim and synthesized in Xilinx 14.7. This paper summarizes the comparative study of the multiplier and multiplier-less method based on various parameters. There is a trade-off between area and delay. This paper will help to choose the best method according to the requirement.**

Keywords: Distributed Arithmetic, FIR, Modified Booth, Wallace tree, Multiplier, Multiplier-less, Xilinx.

I. INTRODUCTION

The filter is a circuit used to enhance some features of the input signal or to rejected the unwanted one. There are two types of filters used in DSP. One is FIR filter and another one is IIR filter. A FIR filter is a filter whose impulse response is of finite duration because it settles to zero in finite time. If you give input as an impulse, that is, a single "1" sample followed by many "0" samples, zeroes will come out after the "1" sample has made its way through the delay line of the filter**.**

The impulse response is finite because there is no feedback network in the FIR. Therefore, the term "finite impulse response" is nearly same with "no feedback". FIR filter offers many advantages as compared to IIR filters which are suited for many applications. FIR filter structures comprise of a multiplier, delay, and adder. In this paper, the emphasis is more on multiplier block. More efficient way of performing MAC operation is required to implement FIR filter on FPGA. In one method, modified version of traditional multipliers are used and in another method multiplier is replaced by multiplier-less structures.

II. MULTIPLIERS

Multipliers play a crucial role in today's DSP and various other applications. With advances in technology, many researchers have tried and are still trying to design multipliers which offer either of the following design targets – high speed, low power consumption, regularity of layout and less area or even combination of them in one multiplier.

The common multiplication method is "add and shift" algorithm. In multipliers, partial products are generated which has to be added. The addition of partial products is the important parameter that determines the performance of the multiplier. To reduce the number of partial products to be added, Modified Booth algorithm is one of the most popular algorithms. To achieve speed improvements Wallace Tree algorithm can be used to reduce the number of sequential adding stages. Further by combining both Modified Booth algorithm and Wallace Tree technique we can see the advantage of both algorithms in one multiplier.
The basic multiplication algorithm follows the steps shown below

- If the LSB of Multiplier is '1', then add the multiplicand into an accumulator.
- Shift the multiplier one bit to the right and multiplicand one bit to the left.
- Stop when all bits of the multiplier are zero.

From above it is clear that the multiplication has been changed to the addition of numbers [1]. If the Partial Products are added serially then a serial adder is used. The parallel multiplier is used to add all the partial products parallelly using one combinational circuit. However, compression technique can also be used to reduce the number of partial products and to perform addition with less latency.

A. Modified Booth Multiplier

The Original version of Booth's multiplier (Radix – 2) had two drawbacks.

- The number of operations became variable and inconvenient for parallel multiplication.
- When there are isolated 1s, the Algorithm becomes inefficient.

The above problems of Radix-2 are overcome by using Radix 4 Algorithm which can scan strings of three bits with the algorithm given below. The design of Booth's multiplier in this project consists of Modified Booth Encoded (MBE), sign extension corrector, partial product generators and finally an adder [2]. This method is used to increase speed by reducing the number of partial products by half. Since an 8-bit booth multiplier is used in this project, only four partial products need to be added instead of eight. The architecture of modified Booth is shown in figure 1.

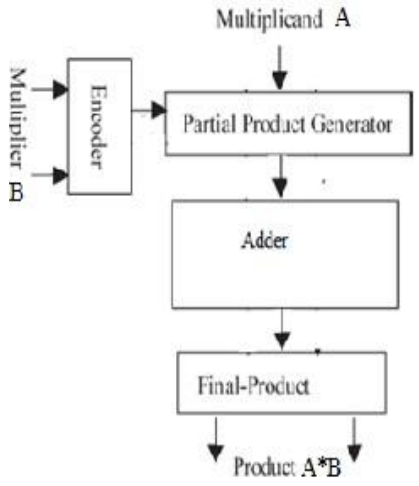

Fig.1 Modified Booth Architecture

Modified Booth Encoder (MBE)

Modified Booth encoding is used to avoid variable size partial product arrays. The multiplier B has to be converted into a Radix-4 number by dividing them into three digits respectively according to Booth Encoder Table given below. Before converting the multiplier, a zero is appended into the Least Significant Bit (LSB) of the multiplier. The multiplier has been fragmented into four partitions and hence four partial products will be generated using modified booth multiplier approach instead of eight partial products being generated using a conventional multiplier.

For eg. Convert an 8-bit number into a Radix-4 number. Let the number be -36 = 1 1 0 1 1 1 0 0. A '0' has to be appended to the LSB. Hence the new number is 1 1 0 1 1 1 0 0 0. Further, it is encoded into Radix-4 numbers according to the table I. Starting from right we have 0*Multiplicand, -1*Multiplicand, 2*Multiplicand, -1*Multiplicand.

Table 1 shows B_{n+1}, B_n, and B_{n-1} which are three bits wide binary numbers of the multiplier Bin which Bn+1 is the most significant bit (MSB) and Bn-1 is the least significant bit (LSB). Zn is representing the Radix-4 number of the three-bit binary multiplier number. For example, if the three-bit multiplier value is "111", so it means that multiplicand A will be 0. And it's the same for others either to multiply the multiplicand by -1, -2 and so on depending on 3-digit number.

TABLE I. MODIFIED BOOTH ENCODER'S TABLE TO GENERATE M, 2M, 3M CONTROL SIGNAL

B_{n+1}	B_n	B_{n-1}	Z_n	Partial Products	1M	2M	3M
0	0	0	0	0	1	1	0
0	0	1	1	1*Multiplicand	0	1	0
0	1	0	1	1*Multiplicand	0	1	0
0	1	1	2	2*Multiplicand	1	0	0
1	0	0	-2	-2*Multiplicand	1	0	1
1	0	1	-1	-1*Multiplicand	0	1	1
1	1	0	-1	-1*Multiplicand	0	1	1
1	1	1	0	0	1	1	0

Partial Product Generator (PPG)

Partial product generator is the combination circuit of the product generator and the 5 to 1 MUX circuit. By multiplying the multiplicand by 0, 1, -1, 2 or -2, partial products are produces using product generator. A 5 to 1 MUX is designed to determine which product is chosen depending on the M, 2M, 3M control signal which is generated from the MBE. For product generator, multiply by zero means the multiplicand is multiplied by "0". Multiply by "1" means the product remains the same as the multiplicand value. Multiply by "-1" means that the product is the two's complement form of the number. Multiply by "-2" is to shift left one bit the two's complement of the multiplicand value and multiply by "2" means just shift left the multiplicand by one place.

B. Modified Booth Multiplier with Wallace tree

In this section, Modified Booth algorithm is combined with Wallace tree adder to observed benefits from both the techniques in the single multiplier. Modified Booth algorithm provides better area performance and Wallace tree benefits with reduced delay [3]. This Multiplier architecture comprises of two architectures, i.e., Modified Booth and Wallace tree. Based on literature review, we find that Modified Booth increases the speed because it reduces partial products to half. The delay in multiplier can be reduced by using Wallace tree. The power consumption of Wallace tree multiplier is also less as compared to the booth. So the Advantages of both the multipliers can be combined to produce high speed and low power multiplier**.** The

Wallace tree uses both Full Adder and Half Adders because of which it is also called as compressor tree. The final results are added using a Carry Look-ahead Adder. The architecture of this combination is shown in figure 2.

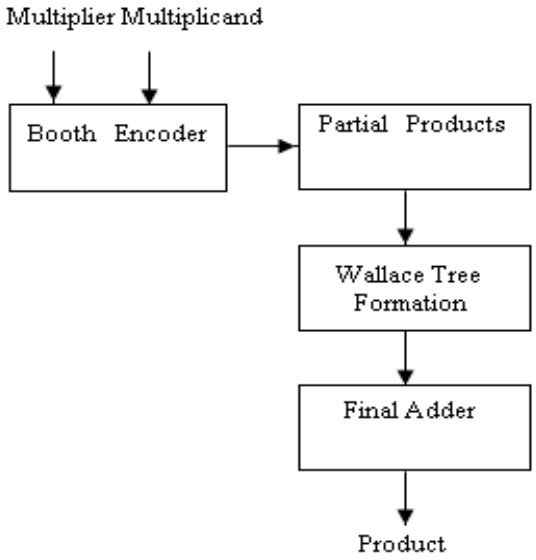

Fig. 2 Modified booth with Wallace tree

Wallace Tree Adder

The Wallace tree adder is used where speed factor is vital and used in order to produce two rows of partial products that can be added in the last stage. Here the Wallace tree is used to accelerate the accumulation of the partial products. Partial products are added by the carry save adder (CSA) and the final stage is added using carry look ahead (CLA) adder. CSA adder takes three inputs and produce sum and carry parallel. Three partial products are added by the CSA tree and finally when there are only two outputs left out, then finally CLA adder is used to produce the final result. In all multiplication operation product is obtained by adding partial products. Thus, the final speed of the multiplier circuit depends on the speed of the adder circuit and the number of partial products generated. In this regard, we can expect a significant reduction of time in computing multiplication operation using this method

III. MULTIPLIER LESS

In the multiplier-less technique, the MAC operation done by multipliers in multiplier methods is replaced by a technique which doesn't use multipliers but does the same calculation as multiplier technique does. Several multipliers-less techniques have been proposed. These methods can be classified into two types according to how they manipulate the filter coefficients for the MAC operation [4]. The first type of multiplier-less technique is the conversion-based approach, in which the coefficients are transformed to other numeric representations whose hardware implementation or manipulation is more efficient than the traditional binary representation. Example of such techniques is the Canonic Sign Digit (CSD) method, in which coefficients are represented by a combination of powers of two in such a way that multiplication can be simply implemented with adder/subtractors and shifters, and the Dempster-Mcleod method, which similarly involves the representation of filter coefficients with powers of two but in this case arranging partial results in cascade to introduce further savings in the usage of adders. The second type of multiplier-less technique involves the use of Look-Up Tables (LUTs) to store pre-computed values of coefficient operations. The very well-known multiplier -less technique used is Distributed Arithmetic.

Distributed arithmetic is used to implement MAC operation with a multiplier-less unit, where the MAC operations are replaced by a series of LUT access and summations. The basic idea in this method is to replace all multiplications and additions by a table and a shifter-accumulator. DA technique is bit-serial in nature. It is a bit-level rearrangement of the MAC operation. The basic DA is a computational algorithm that affords efficient implementation of the weighted sum of products, or dot product [5]. DA is a bit-serial operation used to compute the inner (dot) product of a constant coefficient vector and a variable input vector in a single direct step and is given by

$$Y = \sum_{k=1}^{k} A_k X_k \quad (1)$$

where,

y - Output response
Ak - Constant filter coefficients
Xk - Input data

Let Xk be an N-bits and can be expressed in scaled two's complement number as

$$X_k = -b_{ko} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \quad (2)$$

Substituting X_k into equation (1)

$$y = \sum_{k=1}^{k} Ak \; [-b_{ko} + \sum_{n=1}^{N-1} b_{kn} 2^{-n}] \quad (3)$$

$$y = -\sum_{k=1}^{k} b_{k0}.A_k + \sum_{k=1}^{K}.[\sum_{n=1}^{N-1}(b_{kn.} A_k) 2^{-n}] \quad (4)$$

Rearranging the summation based on power terms and then grouping the sum of the products,

$$y = -\sum_{k=1}^{k} b_{k0}.A_k + \sum_{n=1}^{N-1}.[b_{1n}.A_{1} + b_{2n}.A_2 + \ldots\ldots b_{kn}.A_k] 2^{-n} \quad (5)$$

The final formulation,

$$y = -\sum_{k=1}^{k} A_k b_{k0} + \sum_{n=1}^{N-1}.[\sum_{k=1}^{k} A_k b_{kn}] 2^{-n} \quad (6)$$

A. Basic Distributed Arithmetic method

The DA consists of Look Up Table (LUT), Shift registers and scaling accumulator. In DA method, the partial product outcomes are pre-computed and stored in a Look-Up Table (LUT) which is addressed by the multiplier bits. A

filter with N coefficients the LUT has 2^N values. The basic block diagram of DA is shown in figure 3.

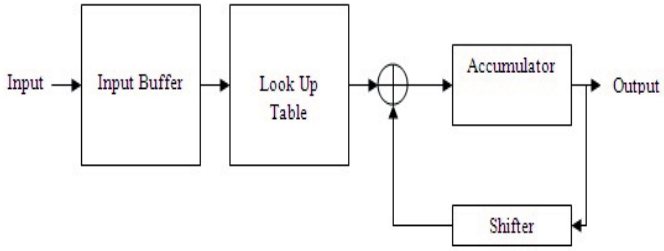

Fig.3 The basic Distributed Arithmetic structure

Distributed Arithmetic for 3rd order Filter

Coefficients = 4, No. of inputs = 4

LUT size = 2^4 = 16 memory location

In this method, possible outputs are pre-computed and stored in LUT.LUT is addressed through the input of the filter. For 4 tap filter, 4 tap represents the no. of the coefficient of the filter as well as it represents the no. of inputs to the filter and address bit for the LUT.

Table 2 shows the content of the LUT for 3rd order filter

TABLE 2.CONTENTS OF THE LUT

Address	Data
0000	0
0001	h3
0010	h2
0011	h2+h3
0100	h1
0101	h1+h3
0110	h1+h2
0111	h1+h2+h3
1000	h0
1001	h0+h3
1010	h0+h2
1011	h0+h2+h3
1100	h0+h1
1101	h0+h1+h3
1111	h0+h1+h2+h3

For corresponding inputs, each location of LUT is assigned to different outputs. The possible inputs for 3rd order filter are 0(0000) - 15(1111). The output is computed for each input using the technique shown below.

Input = 1011 means

Output = 1.h0 + 0. h1 + 1. h2 + 1.h3

= h0+h2+h3

Input = 1111 means

Output = h0+ h1+h2+h3

Input = 0101 means

Output = h1+h3

Input = X0, X1, X2, X3

X0 = 1011=11

X1 = 1101=13

X2 = 1010=10

X3 = 1001=9

h0 = h1 = h2 = h3 = 1

Step 1:

Store the values in input buffer.

X0[0] X1[0] X2[0] X3[0] =1101

X0[1] X1[1] X2[1] X3[1] =1010

X0[2] X1[2] X2[2] X3[2] =0100

X0[3] X1[3] X2[3] X3[3] =1111

Step 2:

Read the values from LUT for corresponding values in buffer.

Output of LUT:

O1 = 0011 = 3

O2 = 0010 = 2

O3 = 0001 = 1

O4 = 0100 = 4

Step 3:

If the value is multiplied by 2, it implies left shift.

Output = O1 + Shift the value of O2 one time + Shift the value of O3 2 times + Shift value of O4 3 times.

Output = 3 + 4 + 4 + 32 = 43.

Disadvantage

A filter with N coefficients the LUT has 2^N values. For higher order filter LUT size will increase, it required more memory space.

B. Distributed Arithmetic using Partition method

The above DA technique holds good only when filters are of low order. For higher order filters, the size of the LUT also increases exponentially with the order of the filter. For a filter with N coefficients, the LUT have 2^N values.

Therefore, for higher order filters, LUT size to be reduced to reasonable levels. To reduce the size, the LUT can be subdivided into a number of LUTs, called LUT partitions. Each LUT partition operates on a different set of filter taps. The results obtained from the partitions are summed.

Partition method for 3rd order filter

Number of partition = 2

So, the number of LUT'S used are 2. Each LUT has 2 inputs.

Memory location = no. of partition * $2^n = 2*2^2$ = 8 locations

n = number of inputs of LUT

LUT is divided into LUT 1 & LUT 2. Each LUT has 2 inputs and 4 memory location. It is shown in figure 4

Input = 1011 means
First two bits is address bit of LUT 1, output becomes 10 = h0
Remaining 2 bits are address bit of LUT 2, output becomes 11 = h2 + h3
Output = output of LUT1 + output of LUT 2 = h0+ h2 + h3

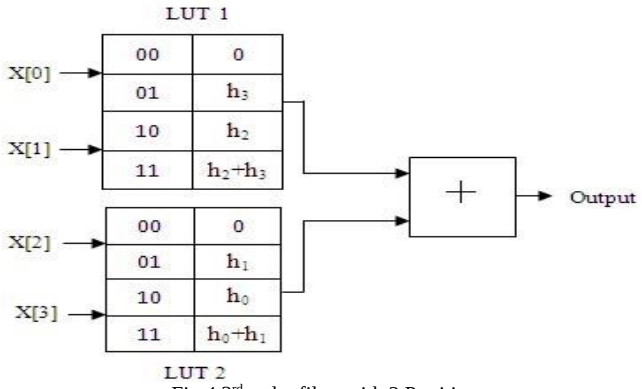

Fig 4 3rd order filter with 2 Partition

Partition method for 8-tap filter

FIR Filter with 8-tap is used in this project. If the partition is not used, then the number of memory locations that will be used is 2^8=256 locations. So, it will be a very time-consuming process to write code for such large number of locations. So, partition method is used in this project. The architecture of 8 tap filter with 4 partitions is shown in figure 5.

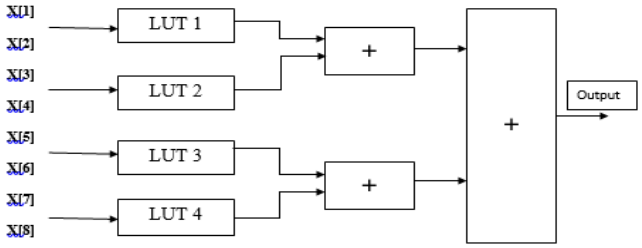

Fig. 5 8-tap filter with 4 partitions

IV. RESULTS

After generating coefficients from MATLAB FDA tool, the code is written in Verilog language and simulated using ModelSim-Altera 6.4a. The inputs given here are 10,44,128,64,6,51,9,2. Simulation results for all the methods are shown in figures below

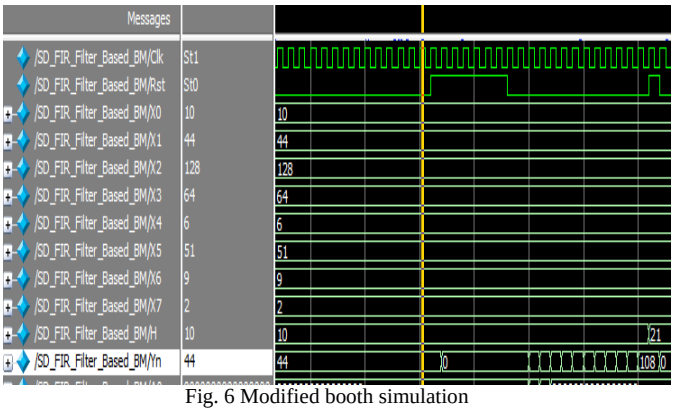

Fig. 6 Modified booth simulation

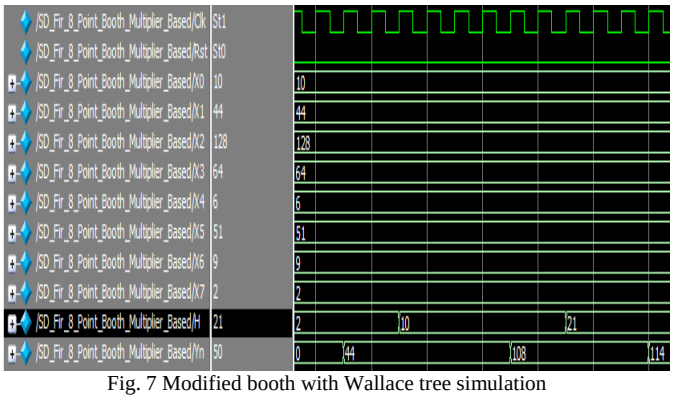

Fig. 7 Modified booth with Wallace tree simulation

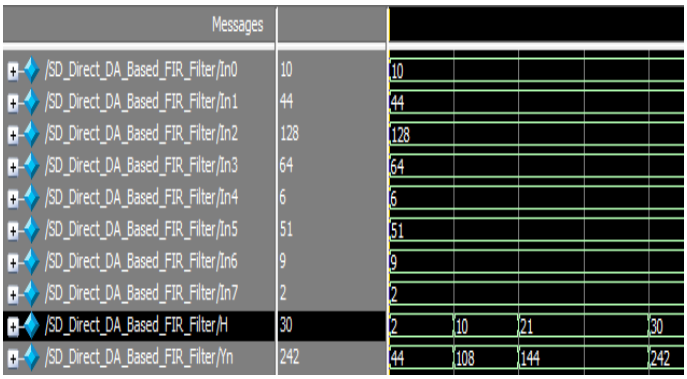

Fig. 8 Distributed Arithmetic simulation

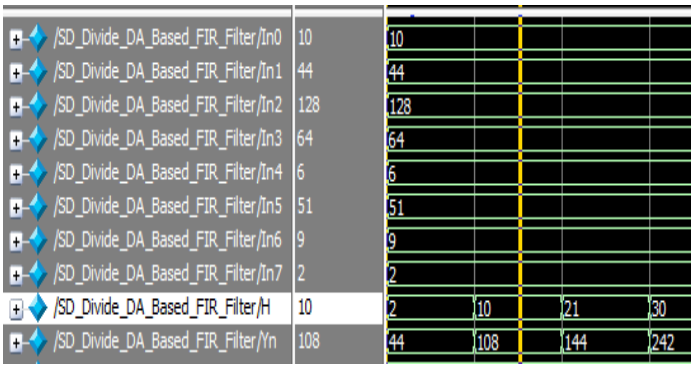

Fig.9 Distributed Arithmetic partition simulation

V. CONCLUSIONS

After analysing all the multiplier and multiplier-less methods, the comparison is made in terms of delay, slices, BELs, memory. Table 3 shows the comparison of different parameters of Modified Booth method, Modified Booth with Wallace tree adder method, distributed arithmetic and Distributed arithmetic with partition method.

TABLE 3 COMPARISON TABLE OF DIFFERENT METHODS

Methods	Modified Booth	Modified Booth with Wallace tree adder	Distributed Arithmetic	Distributed Arithmetic with partition
Delay (ns)	10.221	8.957	18.235	16.854
Slice LUTs	565	263	2532	165
IOs	91	91	84	84
BELs	1145	563	4060	220
Memory (Kbs)	301144	330208	409456	282656

This paper gives a clear concept of different methods used to implement FIR filter and their comparison based on delay, slices, BELS, and memory. Based on the comparison, we find that Modified booth with Wallace tree method has the least delay among all the other methods. So, it is the best method, if high speed is required. This method is fastest because of two reasons. One is a reduction of calculation of partial product by using modified booth algorithm and another reason is the use of Wallace tree adder to speed up the calculations. It reduces delay at the cost of a little increase in complexity.

Distributed arithmetic with a partition which is a multiplier-less method had a greater delay than multiplier methods but covers the least area. As the area is less, power dissipation is also less than others. The memory requirement for this method is also very least amongst all the methods.

As there is a trade-off between area and speed, this paper helps to make us choose a proper method according to the requirement.

REFERENCES

[1] Devika.S, S. Lokesh, H. Chandrasekhar," FPGA Implementation of Low Power FIR Filter using Modified Booth Algorithm," in *International Journal for Research in Applied Science & Engineering Technology (IJRASET), Volume 3 Issue VI*, June 2015 ISSN: 2321-9653

[2] Misra Susmita," Design and implementation of faster and low power Multipliers", in *International journal of science and Technology, Volume 3, issue* 4, oct 2012, ISSN:0976-8491

[3] Nair Savita,Saraf Ajit,Nair Swati," Analysis of multipliers based on various performance measures", in *International journal of Engineering Research & Technology, Volume 3,Issue 2*,February 2014,ISSN : 2278-0181.

[4] K. Tirupathiah, K.Rajasekhar,"The Design of FPGA Implementation of 16-Tap FIR filter using Improved DA Algorithm", in *International Journal of Computer Technology & Application, Volume 3, Issue 5*, ISSN: 2229-6093.

[5] M.Yazhini Ramesh," FIR filter Implementation using Modified Distributed Arithmetic Architecture", *in Indian Journal of science and Technology*, ISSN: 0974-5645