IJEEE

International Journal of Ethics in Engineering & Management Education

Website: www.ijeee.in (ISSN: 2348-4748, Volume 2, Issue 9, September 2015)

Efficient Architecture of Multiplier for Digital Filters

Smitha N Mallya
Department of Electronics & Telecommunication
FCRIT, Vashi
Navi Mumbai, Maharashtra, India
[1]smithamallya@yahoo.com

Sneha Revankar
Department of Electronics & Telecommunication
FCRIT, Vashi
Navi Mumbai, Maharashtra, India
Sneha.revankar@gmail.com

***Abstract* - this paper presents a new architecture for multiplier unit. The new high speed and low area architecture of multiplier unit is then implemented in FIR Filter. The method used in this paper employs Encoded Booth Algorithm resulting in speedy operation. Architecture of Regular CSA is modified for Area optimization. Less area occupation makes the design consume less power. Multiplication plays a vital role in most of the high performance systems. By reducing the delay taken for calculations of results one can speed up the system performance. The methods include Modified Booth Multiplication along with carry select adder. Comparative study is done between normal multiplier-shift/add multiplier, encoded booth multipliers using RCSA and MCSA for Radix 8. The efficient among three is applied to filter structure and design is implemented on Spartan 6 FPGA kit. The proposed designs are designed using Verilog HDL and synthesized, implemented using Xilinx ISE.**

***Key Terms*— Shift/add Multiplier, Encoded Booth's Multiplier, RCSA, MCSA, Spartan 6 FPGA.**

I. INTRODUCTION

Filters are widely used in various DSP applications. Few applications, will be such that the filter circuit must be able to operate at high sample rates, while in other applications, the filter circuit must act as a low-power circuit that operates at moderate sample rates. Hence major component that affects a filter is the multiplier circuitry. It also affects the resultant power consumption and speed. Thus choosing a multiplier with more hardware breadth rather than depth would not only reduce the delay, but also the total power consumption. A lot of design methods of low power digital filter have been proposed. They use a modified common sub expression elimination algorithm to reduce the number of adders used in the multiplication operation.

Multiplication is a most commonly used operation in many computing systems. In fact multiplication is nothing but repetitive addition since, multiplicand adds to itself multiplier number of times gives the multiplication value between multiplier and multiplicand. But the facts that this kind of implementation shall take many hardware resources and make the circuit operate at utterly low speed. In order to address this issue so many ideas have been presented so far for the last three decades. Each one is aimed at a particular improvement according to the requirement. One may be aimed at high clock speeds and another may be aimed for low power consumption or less area occupation [1], [2]. Either way ultimate job is to come up with an efficient architecture which can address three constraints of VLSI speed, area, and power. Among these three, speed is the one

Which requires most important and special attention? On observing closely, multiplication operation involves two major steps: one is producing partial products and other is adding these partial products. Thus, the speed of a multiplier hardly depends on how fast the partial products are generated and how fast we can add them together. If the numbers of partial products to be generated are less then it is indirectly means that we have achieved the speed in generating partial products. Booth's algorithm is meant for achieving speed. To speed up the addition operation among the partial products, we need fast adder architectures. Since the multipliers have a significant impact on the performance of the entire system, one such high performance algorithms and architectures have been proposed by Renuka Narasimha, Rajasekhar and Sujana Rani [3].

The section II give a brief summary of FIR filter theory, section III presents FIR implementation which discusses the different multiplication architectures. Section IV discusses on simulation results. Section V is conclusion drawn between different multipliers and future work to be carried.

II. DIGITAL FILTER THEORY

Digital filters are main components that are usually used to modify or alter the attributes of a signal in the time or frequency domain. The linear time-invariant (LTI) filter is the most common digital filter. Interaction of an LTI system with input signal through a process called linear convolution, denoted by y = f * x where f is the filter's impulse response, x is the input signal, and y is the convolved output. The linear convolution process is formally defined by:

$$Y[n] = x[n] * f[n] = \sum_{k=0}^{L-1} x[n] f[n-k] \sum_{k=0}^{L-1} x[$$

$$\sum_{k=0}^{L-1} x[n] f[n-k] \sum_{k=0}^{L-1} x[\quad (1)$$

LTI digital filters are generally classified as being finite impulse response (i.e., FIR), or infinite impulse response (i.e., IIR). An FIR filter is a filter whose impulse response settles to zero in finite time. An IIR filters may have an internal feedback and continue to respond indefinitely. As the name implies, an FIR filter consists of a finite number of sample values, reducing the above convolution sum to a finite sum per output sample instant. An FIR with constant coefficients is an LTI digital filter. The output of an FIR of order or length L, to an input time-series x[n], is also given by a finite version of the convolution sum given in Eq (2), namely:

$$y[n] = x[n] * f[n] = \sum_{k=0}^{L-1} f[k] \qquad (2)$$

where *f*[0] ≠ 0 through *f*[*L*-1] ≠ 0 are the filter's *L* coefficients. They also correspond to the FIR's impulse response. For LTI systems it is sometimes more convenient to express in the z-domain with

$$Y(z) = F(z)\,X(z) \qquad (3)$$

Where F (z) is the FIR's transfer function defined in the z-domain by

$$F(Z \qquad (4)$$

The Lth-order LTI FIR filter is usually interpreted as shown in Figure.1. It comprises of large number of a "tapped delay line," adders, and multipliers. One of the operands given to each multiplier is an FIR filter coefficient, often named as a "tap weight".

The FIR filter with transposed structure Figure.1 has registers between the adders and can achieve high throughput without adding any extra pineline registers.

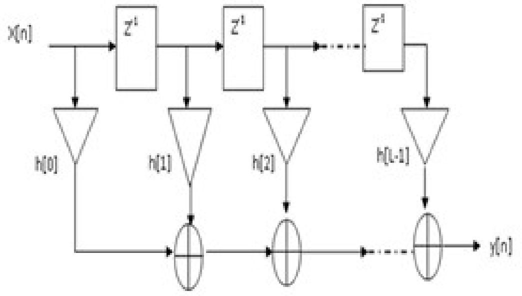


Figure 1: FIR filter in the transposed structure

The multiplier is one of the essential elements of the digital signal processing such as filtering, convolution, and inner products. Most digital signal processing methods use nonlinear functions such as discrete cosine transform (DCT) or discrete wavelet transform (DWT). Because they involve repetitive application of multiplication and addition, the speed of the multiplication and addition determines the execution speed and performance of the entire calculation. Because the multiplier requires the longest delay among the basic operational blocks in digital system, the critical path is determined by the multiplier, in general.

Fast multipliers are an integral part of digital signal processing systems. Initially multiplication was generally implemented by sequence of addition and shift operations. Multiplication can be considered as a series of repeated additions. The number to be added is the multiplicand, the number of times that it is added is the multiplier, and the result is the product. Each step of addition generates a partial product. In most computers, the operand usually contains the same number of bits. Final product is usually reserved to be of length two times than the length of input data bits so as to no information is lost . This repeated addition method that is suggested by the arithmetic definition is slow. It is almost always replaced by an algorithm that makes use of positional representation. It is possible to decompose multipliers into two parts. The first part is dedicated to the generation of partial products, and the second one collects and adds them. The basic multiplication principle is twofold i.e., evaluation of partial products and accumulation of the shifted partial products. It is performed by the successive additions of the columns of the shifted partial product matrix. The multiplier is successfully shifted and gates the appropriate bit of the multiplicand. The delayed, gated instance of the multiplicand must all be in the same column of the shifted partial product matrix. They are then added to form the final product. For high-speed multiplication, there are some of the methods discussed in this paper.

A. Shift and Add Multiplier

In this section we present a simple Shift and Add structure for multiplier used in filters [4]. Multiplication is performed by generating partial products and shifting the multiplicand left by one bit after every partial product calculation. The partial product of the current stage is set to the sum of the previous partial product and the shifted multiplicand of the current stage or 0, depending on whether the multiplier bit in the current stage is 1 or 0. Let MD be multiplicand and MR be he multiplier

Reference Model: Shift-and-Add for 3-bit operands
Stage 1:
Rule a: product = prod+ MD if MR [0] exists else
Rule b: product = product + 0 if MR [0] =0.
Stage 2:
Rule a: product = product + MD shifted left by 1 bit if MR [1] exits.
Rule b: product = product + 0 if MR [0] =0.
Stage 3:
Rule a: product = product + MD shifted left by 2 bit if MR [2] exits.

Rule b: product = product + 0 if MR [0] =0.

The same procedure is followed for n-bit multiplication.

B .Modified Booth Multiplier

In order to achieve high-speed, modified Booth algorithm has been presented in this section. Booth multiplication is a process that allows faster computation of results, by recoding the numbers that are to be multiplied [7]. It depends on radix we choose for recoding the multiplier bits. Radix 4 booth encoding leads to reduction in partial products by almost 50% .The method is, instead of shifting and adding multiplicand for every column of the multiplier bit and multiplying by 1 or 0, we group the multiplier bits, and multiply by ±1, ±2, or 0, to obtain the same results. This algorithm can be used for signed numbers as well by taking negative number in two's complement notation.

i. Radix-2 Multiplication

The simple multiplication generator can be used to reduce the number of partial products by grouping the bits of the multiplier into pairs, and selecting the partial products from the set 0, +-M, where M is the multiplicand. Here the multiplier is grouped into two bits. Each encoded digit performs some operation on the multiplicand generating the partial product with the help of the selection Table 1. Each partial product is shifted one bit position to the left with respect to its neighbors. These partial products are then added to obtain the final product.

Table 1: Partial product selection table for radix 2

Multiplier bits	Selection
00	0
01	+M
10	-M
11	0

ii. Radix -4 Multiplication

The Radix-4 Modified Booth's Algorithm reduces the number of partial products by about a factor of two. This selects the partial products from the set of 0, +-M, +-2M, where M is the multiplicand. The multiplier is appended by a '0' on LSB; we will call this bit as Z. The multiplier is partitioned into overlapping groups of 3 bits, and each group is decoded to select a single partial product as per the selection Table 2. Each partial product is shifted 2 bit positions with respect to its neighbors. The number of partial products will be reduced from 16 to 9 for a 16X16 multiplication. In general the there will be (n+2)/2 partial products, where n is the operand length.

Table 2: Partial product selection table for radix 4

Multiplier bits	Selection
000	0
001	+MD
010	+MD
011	+2MD
100	-2MD
101	-MD
110	-MD
111	0

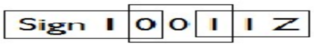


Figure 2: Recoding of multiplier

Each block is decoded to generate the correct partial product. The encoding of the multiplier Y, using the modified booth algorithm, generates the following five signed digits, -2, -1, 0, +1, +2. Each encoded digit in the multiplier performs a certain operation on the multiplicand as illustrated in the Table 2.

The modified Booth's algorithm (*radix-4 recoding*) starts by appending a zero to the right of x_0 (multiplier LSB). Triplets, group of three bits are taken beginning at position x_{-1} and continuing to the MSB with one bit overlapping between adjacent triplet groups. If the number of bits in the multiplier (excluding x_{-1}) is odd, the sign (MSB) is extended one position to ensure that the last triplet contains 3 bits. In every step we will get a signed digit that will multiply the multiplicand to generate a partial product entering the Carry select adder.

Radix -8 Multiplication Radix-8 Booth Multiplication

As a further enhancement we have implemented a next higher radix multiplier so that partial products are further reduced to {n+2)/3. It thus further reduces the area and contributes in reducing power consumption. The multiplier is partitioned into overlapping groups of 4 bits, and each group is decoded to select a single partial product as per the selection Table 3. Each partial product is shifted 3 bit positions with respect to its neighbors. The number of partial products will be reduced as compared to Radix-4.

Table 3: Partial product selection table for radix 8

Multiplier bits	Selection	Multiplie bits	Selection
0000	0	1000	-4 MD
M0001	+MD	1001	-3 MD
0010	+ MD	1010	-3 MD
0011	+2 MD	1011	-2 MD
0100	+2 MD	1100	-2 MD
0101	+3 MD	1101	- MD
0110	+3 MD	1110	- MD
0111	+4 MD	1111	0

Proposed Architecture for Multiplication

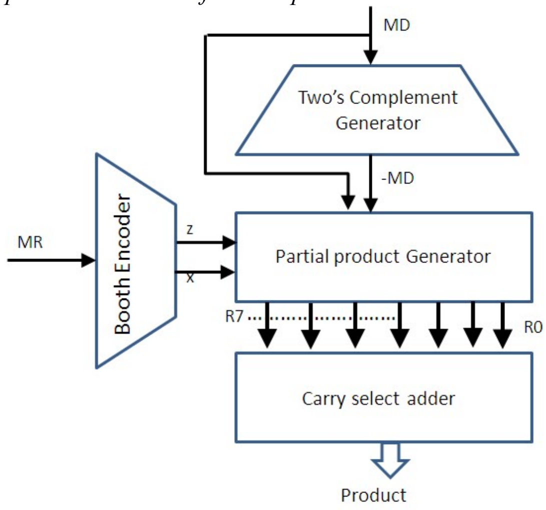


Figure 3: Architecture for Booth Multiplication

The multiplier is designed so that it can take two n-bit inputs: the multiplier (MR) and the multiplicand (MD), and results in producing the 2n-bit multiplication result of the two as its output. The architecture of the booth multiplier,

Primarily consists of four major modules as shown in Fig.3. They are: 2's Complement Generator, Booth Encoder, Partial Product Generator and Carry Select Adder. The first block gives the twos complement form of input multiplier data it takes the multiplicand (MD) as its input and produces -MD as its output. This is required when recoded multiplier bit signifies negative multiplication. 2's complement is generated by inverting all bits of the multiplicand and then adding 1 using adder. The Partial Product Generator uses two control signals x and z produced by the Booth Encoder and uses these signals to choose from and extend signs of '0', MD,-MD,2MD or -2MD for creating partial products.. The final intermediate results are added using a Carry Select Adder. Carry select Adder (CSA) adds two numbers with very lower latency. The carry Select adder will avoid the unwanted addition and thus minimize the switching power dissipation. Carry select adder is one of the fastest adders among parallel adders.CSA is made up of RCA arrays.

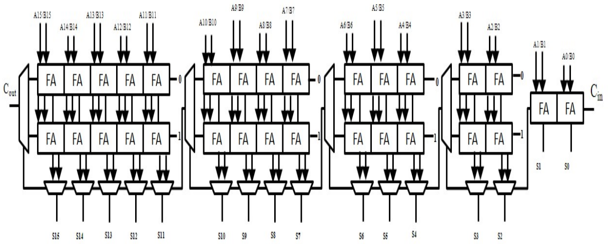


Figure 4: 16-Bit Regular Carry Select Adder Schematic

CSA is faster adder when compared to RCA .The only parameter to be taken care of now is to reduce the gate counts so as to reduce the area occupied. New architecture is proposed where in second row of RCA's are replaced by Binary to Excess one converter. The number of gate used in BEC is less compared to gates in RCA .The structure of Carry Select Adder using binary to excess 1 converter for RCA with C_{in}=1 to optimize the area and power is shown in Figure 3.13. BEC gets n inputs and generates n outputs for binary values without carry and n+1 output for binary values with carry. Large bit sized multipliers requires multiple BEC and each of them requires the selection input from the carry output of the preceding BEC. In this project expecting for maximum possible data and that carry will be generated we are replacing the RCAs having Cin =1 by BEC. Thus n bit RCA is replaced by n+1 bit BEC In our proposed method the carry Cin =1 RCA is replaced by the (n+1) bit BEC. The area is optimized by replacing regular CSA with new structure where in second array of RCAs are replaced with Binary to excess 1 converter which takes less gate counts. Figure below gives the new architecture of CSA.

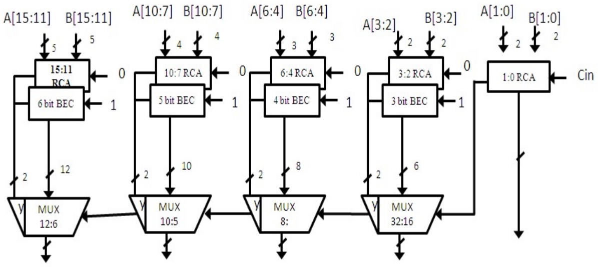
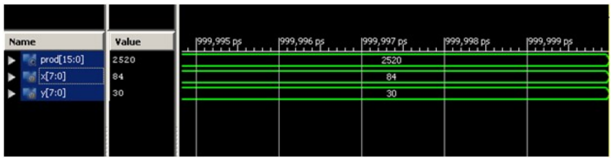


Figure 5: 16-Bit Modified Carry Select Adder Schematic

IV. RESULTS

The multiplier is designed using Verilog HDL and simulated using Xilinx ISE. The Figure 6 shows the simulation with MCSA.

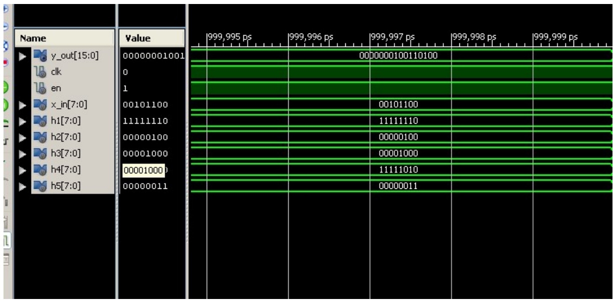


Name	Value
prod[15:0]	2520
x[7:0]	84
y[7:0]	30

Figure 6. Simulated output of Modified Booth Multiplier using MCSA

Efficient multiplier in then implemented into digital FIR Filter. Simulated result of filer using Radix 8 Encoded Booth multiplier for multiplication and Modified architecture of CSA to add the partial products is shown in figure 7.

.

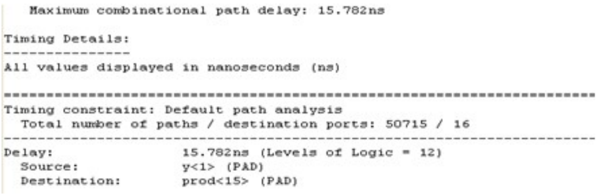


Name	Value
y_out[15:0]	00000001001
clk	0
en	1
x_in[7:0]	00101100
h1[7:0]	11111110
h2[7:0]	00000100
h3[7:0]	00001000
h4[7:0]	00001000
h5[7:0]	00000011

Figure 7: Simulation result of FIR Filter using Radix 8 Booth Multiplication with Modified CSA

Delay comparison

By comparing the time taken for computing the final product calculation by various methods, a clear picture is obtained that Radix 8 is faster. The number of partial products in Radix 8 is less as compared to other multiplier methodologies. The timing table displayed below is from the timing summary report that will be generated when running the simulation. As an example snapshot of timing report of efficiency multiplier taking 15.78ns for calculation is shown in figure 8. The delay mentioned is the time taken by various methods to calculate the final product that is it includes the time taken for generation of partial products and its addition by using the adder architecture.



```
   Maximum combinational path delay: 15.782ns

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 50715 / 16
-------------------------------------------------------------------------
Delay:                  15.782ns (Levels of Logic = 12)
  Source:               y<1> (PAD)
  Destination:          prod<15> (PAD)
```

Figure 8: Timing report of EBM with MCSA

Table 4: Comparative Table of time taken by various methods

Multiplication methods	**Delay (ns)**
Shift and Add	65.33
Radix 8 with Regular CSA	17.52
Radix 8 with Modified CSA	15.78

From the table above it is seen that delay of 8 bit MBM with RCSA and MBM with MCSA is reduced by 27 % and 24 % respectively.

The area calculation of regular CSA:

From the structure of RCSA, 8-bit, 16-bit adders' gate count is calculated.

8-bit RCSA:

Initial carry input $C_{in,}$

2-bit RCA = 1

Carry input C_{in} = 0,

2-bit RCA = 1

4-bit RCA = 1

Carry input C_{in} = 1,

2-bit RCA = 1

4-bit RCA = 1

The area count of RCA is tabulated below

Table 5. Area count of 8-bit RCSA

Word size & Adder	Number of gates
2-bit RCA	19
4-bit RCA	45
2:1 Mux	4

The total area of the 8-bit regular CSA is 179. The total area of the different adder is tabulated in Table 6

Table 6 Regular CSA Area

Word Size	Adder	Area (no. Of Gates)
8-bit	RCSA	179
16-bit	RCSA	427

Area calculation for MCSA:

The area calculation of modified CSLA is derived from the following steps. From the structure of MCSA, 8-bit, 16-bit, 32-bit and 64-bit area is calculated.

The Group 1 architecture calculation is,

Gate Count = 19 (FA+HA)
FA = 13 (1×13)
HA = 6 (1×6)

The Group 2 architecture calculation is,

Gate Count =43 (FA+HA+Mux+BEC)
FA = 13 (1×13)
HA = 6 (1×6)
Mux = 12 (3×4)
BEC: AND = 1
NOT = 1
XOR = 10 (2×5)

The Group 3 architecture calculation is,

Gate Count = 61 (FA+HA+Mux+BEC)
FA = 26(2×13)
HA = 6 (1×6)
Mux = 16 (4×4)
BEC: AND = 2
NOT = 1
XOR = 15 (3×5)

The Group 4 architecture calculation is,

Gate Count = 84 (FA+HA+Mux+BEC)
FA = 13 (3×13)
HA = 6 (1×6)
Mux = 20 (5×4)
BEC = 24

Similar Calculations are done for group 5 architecture and total number of gate counts is tabulated below for 8 bit and 16 bit MCSA .

Table 7. Modified MCSA Area

Word Size	Adder	Area (no. Of Gates)
8-bit	MCSA	145
16-bit	MCSA	329

Chart shown in figure 9 and 10 gves a clear picture of reduction in gate counts

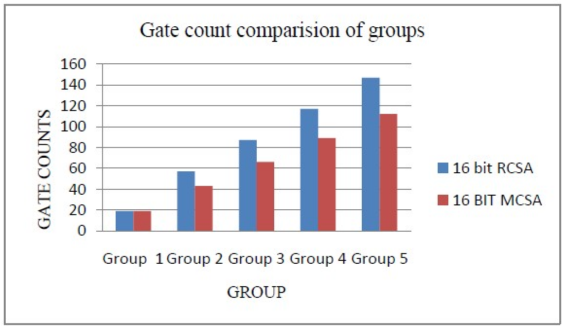


Figure 9.Gate count comparison of all 5 groups of RCSA and MCSA

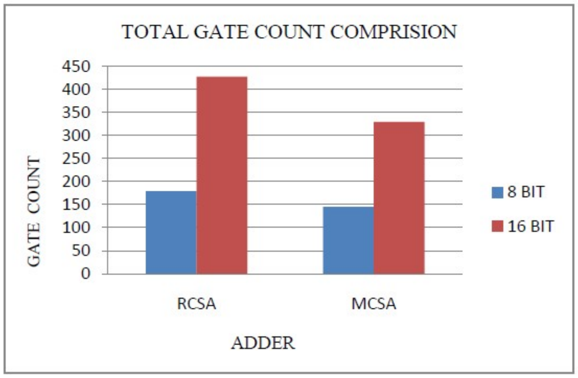


Figure 10. Total Gate counts of RCSA and MCSA

The graph comparison of total number of gates required for 16-bit regular and modified carry select adders. MCSA has a reduced the gate counts by 98 leading to reduction in area by 77%.

V.CONCLUSION

In this paper we are presenting an efficient architecture for multiplier. Multiplier is designed such that, delay time taken for final result calculation is reduced leading to increase in speed. It will also lead to faster performance of processors as most of the processors have Multipliers and/or MAC units in them. For making the architecture area efficient, we have modified the regular Carry select Adder by replacing RCA with Cin=1 by Binary to Excess 1 Convertor. The Radix-8 Modified Booth multiplication technique presented in this project will result in the reduction of the number of partial products, hence contributing in faster calculation of result. As the number of transistor switching will be less, the total power consumption will also be reduced leading to efficient implementation of multiplier .As a future advancement higher order radix can be implemented the proposed architecture has been designed and synthesized using Verilog in Xilinx ISE. The design is finally implemented using Spartan 6 FPGA.

.

REFERENCES

[1] H. S. Krishnaprasad Puttam, P. Sivadurga Rao & N. V. G. Prasad.”Implementation of Low Power and High Speed Multiplier-Accumulator Using SPST Adder and Verilog “, International Journal of Modern Engineering Research ,Sept-Oct 2012

[2] Yun-Nan Chang,Janardhan H Sathyanarayana,Keshab K. Parhi,”Design and implementation of low power digital Serial Multipliers,” University of Minnesota,Minneapolis

[3] Kousuke TARUMI, Akihiko HYODO, Masanori MUROYAMA and Hiroto YASUURA, “A design method for a low power digital FIR filter in digital wireless communication systems”, Graduate School of Information Science & Electrical Engineering, Kyushu University

[4] Rupali Madhukar Narsale, Dhanasri Gawali,”Design and implementation of low power FIR filter:A review” International Journal of VLSI and Embedded Systems-IJVES,March-April 2013

[5] A.Renuka Narasimha, K.Rajasekhar,A.Sujana Rani , “Implementation of low area and power efficient architectures for digital FIR filters”, IJARCSSE ,Volume 2, Issue 8, August 2012.

[6] Shahnam Mirzaei, Anup Hosangadi, Ryan Kastner,”FPGA implementation of high speed FIR filters using Add and Shift method”, IEEE, 2006

[7] Sarita Chouhan Kota, Yogesh Kumar, “Low Power Designing of FIR Filters,” IJATER ,May 2012

[8] Sukhmeet Kaur, Suman and Manpreet Singh Manna, “Implementation of Modified Booth Algorithm (Radix 4) and its Comparison with Booth Algorithm (Radix-2)”, Advance in Electronic and Electric Engineering. ISSN 2231-1297, Volume 3, Number 6 (2013)

[9] J. Umamaheshwari M. Veni Saranya,”Asic implementation of low power High Radix Booth Encoded Multiplier using Spst”, International Journal of Communications and Engineering ,March 2012

[10] B.N. Manjunatha Reddy, H. N. Sheshagiri, Dr. Shanthala S.,”Area Optimization of 8-bit multiplier using gate diffusion input logic,” International Journal of Advanced Trends in Computer Science and Engineering,2013

[11] Xilinx, “ISE Design Suite Software Manuals”, (v 13.1) March 1, 2011 www.xilinx.com

[12] Xilinx, “Xpower Estimator User Guide”, UG440 (v13.4) January 18, 2012

[13] K. Babulu, G.Parasuram,” FPGA Realization of Radix-4 Booth Multiplication Algorithm for High Speed Arithmetic Logics”, K. Babulu et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (5) , 2011