ISSN: 2456-043X

International Journal of Engineering Science and Generic Research (IJESAR)
Available Online at www.ijesar.in
Volume 3; Issue 6; November-December; 2017; Page No. 20-25

Design of 128 bit Q-format multiplier by using Higher Radix Algorithm

K.Mounika [1], A. Jaya Lakshmi [2]

[1]P. G. Student, [2]Associate Professor

[1]Student at Vidya Jyothi Institute of Technology, Aziz Nagar, Hyderabad, India.

[2]Assistant Professor at Vidya Jyothi Institute of Technology, Aziz Nagar, Hyderabad, India.

Abstract

There has always been a quest going on for improving the performance of the multiplier as it is the key component in determining the performance of the digital signal processor. The Q-format multiplier implemented with Radix algorithm multiplier to be faster and area efficient and for increasing the performance of the Q-format multiplier resulted in the outcome of this paper. This paper presents a novel method using Booth encoding towards generation of reduced number of partial products and redundant binary adder for adding these partial products for implementation of 128 bit Q-format signed multiplier which substantially improved the performance, area. This method has also been implemented for 64 bit multipliers along with 128 bit Q-format signed multiplier using Booth encoding and RB addition in Verilog targeted towards Xilinx FPGA Spartan3E and results compared with those obtained by using tool Xilinx 14.7

Keywords: Q-format: Booth encoding; Redundant binary adder; Xilinx; Verilog.

INTRODUCTION

Most of the latest digital signal processors in use are of 64- bit. They require a faster 64-bit Q-format multiplier for performing digital signal processing applications. At higher bit lengths, the multiplier occupies larger area and also the performance decreases because of carry propagation while adding partial products. In an effort to decrease the area and increase the performance of Q-format multiplier, authors of implemented Q-format multiplier with Vedic Urdhva Triyagbhyam Sutra and found a significant decrease in the combinational delay and area. Several authors implemented higher radix Booth multiplier in combination with redundant binary adder for the design of FIR filters to enhance the speed of FIR filter.

Higher radix algorithms reduce the number of partial products and the redundant binary adder, being carry propagate free adder, reduces the combinational delay in adding the partial products to a large extent, thus enhancing the speed of multiplier. In this paper, the Q-format multiplier which is the key component of a digital signal processor is implemented with Radix-256 Booth multiplier and redundant binary adder and found that there is a significant reduction of delay and area.Radix-256 Booth encoding reduces the partial products by 8 fold.

Redundant binary addition is a carry propagation free addition. Both these techniques combined to implement the multiplier considerably reduced the combinational path delay to a considerable extent. The proposed method is implemented in Xilinx 14.7 targeted towards Spartan3E FPGA.

I. Q-FORMA REPRESENTATION

Fractional numbers are represented in either floating point or fixed point format in computers. The fixed point numbers are represented in the computer in Q-format. In the Q-format, a signed number of N-bit length is represented as $Q_{m,n}$ where m denotes the number of bits in the integer part and n, the number of bits in the fractional part, where N=m+n+1. The extra one bit indicates the sign bit in the MSB position.

Thus N-bit binary number in $Q_{m,n}$ format is represented as given below

$$a_{n+m}\, a_{n+m-1}\, a_{n+m-2}\ldots a_n . a_{n-1}\ldots a_1\, a_0 \qquad (1)$$

In the above representation '.' between the a_n and a_{n-1} is the fixed point. Its value is given by

$$(a_{n+m}\ 2^{N-1} + a_{n+m-1}2^{N-2} + \ldots\ \ldots + a_2 2^2 + a_1 2 + a0)2^{-n} \qquad (2)$$

When such two fractional numbers are multiplied, the integer digits in the result shifts to left and the

Corresponding author: K.Mounika

fraction digits shift to right. Hence there is the problem of overflowing of the integer digits when they exceed the length assigned to them. Loss of MSB gives quite an inaccurate value of the result. If the LSB bits are lost, some precision is lost but the value will be close to the correct value. To avoid the problem of overflow in MSB position, the signed fractional numbers of $Q_{m,n}$ format are first multiplied by 2n for conversion to $Q_{0,n}$ (Q_n) format. Then Q-format multiplication is implemented. After multiplication where to keep the point in the result to get the correct result is within the hands of the programmer, which is an easy task. This can be processed separately in parallel.

II. Q-FORMAT MULTIPLIER

Implementation of Q-format signed multiplier is shown in the Fig.1. The MSB in the number represents the sign bit. On multiplication of two 64 bit numbers, 128 bit long result is generated. The MSB bit in the result will be the redundant sign bit and hence left off. The next bit will be the sign bit of the result.

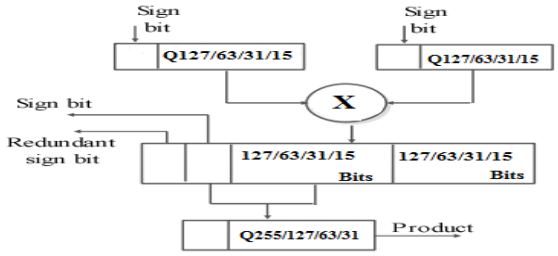

Fig.1. Q-format 128/64/32/16signed multiplier

The remaining 126 bits indicate the value of the multiplied product. These 126 bits are truncated to 63 bits leaving off the least significant 63 bits. This process is called rounding off or truncating to nearest value

III. PROPOSED METHOD

In the proposed method the multiplication of the two Qformat numbers is carried out using the Radix-256 Booth encoding [5] for generation of partial products and redundant binary addition for adding these partial products. Radix-256 Booth encoding reduces the partial products by 8 fold. If N is the number of bits, and if Radix = 2K, then the number of partial products will be (N/K) in Booth encoding.

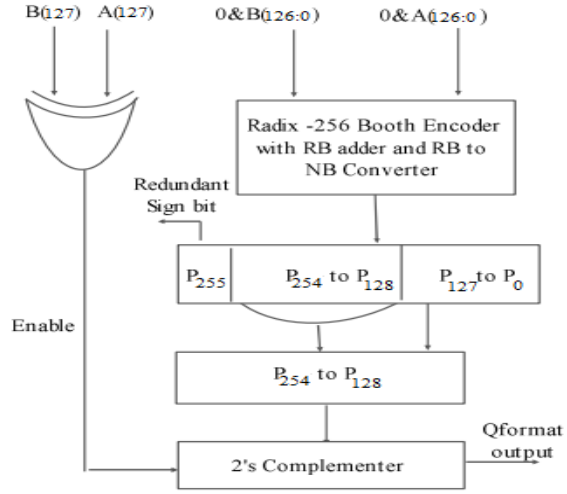

Fig.3 shows the overall implementation of Radix-256 Booth multiplier with RB addition.

Thus in 128 bit multiplier, 16 partial products are generated. Accordingly in 64 bit multiplier 8, 32 bit multiplier 4 and in 16 bit multiplier, only 2 partial products are generated. Redundant binary addition is used for adding the partial products. The partial products generated above are first converted to RB form and then added and finally converted to natural binary form. The following steps illustrate the implementation of proposed method in detail. The basic implementation is shown in fig.2.

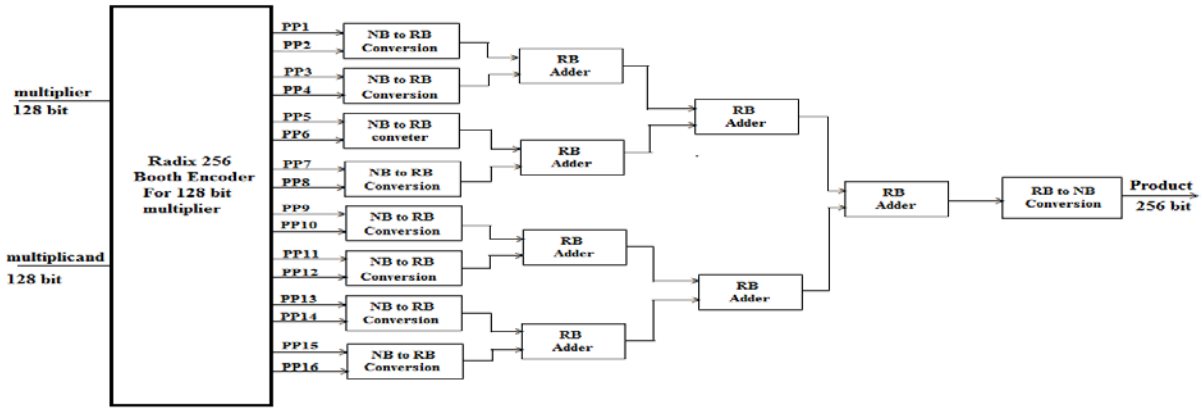

Fig.3 Implementation diagram of Radix-256 Booth multiplier with RB addition

A. Processing the input numbers

The sign bits of both numbers are separated for separate processing and replaced with '0's. For +ve numbers, no processing is required. Negative numbers are 2's complemented excluding the sign digit. Now the two numbers are fed to the multiplier.

B. Radix-256 Booth encoding

The generation of the 8 partial products from the 64 bit multiplier is as follows. The multiplier is concatenated with a '0' in the LSB position and divided into 8 groups (D_i ' s) each of 9 bits with one overlapping bit in MSB position. The D_i ' s multiplied with the multiplicand forms the partial products.

The number of Di's is equal to (N/K). Thus for 64 bit multiplier 8 D_i ' s and hence 8 partial products are generated. If A is the multiplier, Di can be represented as follows, where the last bit is the overlapping bit.

$$D_i = A_{(i.k)+k-1} \ldots\ldots A_{(i.k)+1} A_{(i.k)} A_{(i.k)-1} \quad (3)$$

The decimal value of Di can be computed from (4) below.

$$D_i = -A_{(i.k)+k-1}2^{k-1} + \sum^{k-2}_{J=0} A_j 2^j + A_{(i.k)-1} \quad (4)$$

The decimal equivalent of the D_i, as given in (4) may assume one of the values (-2^{k-1}, $-2^{k-1}+1$ ….-1, 0, 1, $2^{k-1}-1$, 2^{k-1}). If D0= 100101011, its decimal value is

$= -1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 + 1*2^0$.

= -128+ 0+ 0 + 16+ 0 + 4+ 0 + 1 + 1

= -106.

If B is the multiplicand then, $A.B=_{i=0} \sum^{(N/K)-1} 2^{k.i} D_i B$ ..

(5)

$A.B = 2^{56}.(D7.B) + 2^{48}.(D6.B) + 2^{40}.(D5.B) + 2^{32}.(D4.B) + 2^{24}.(D3.B) + 2^{16}.(D2.B) + 2^8.(D1.B) + 2^0.(D0.B)$ ..

(6)

C. Pre computation

For implementation of this multiplier, any 8 from 255 possible partial products are required at a time. These are generated using a precomputer by suitably shifting and complementing the multiplicand as shown in fig.4.

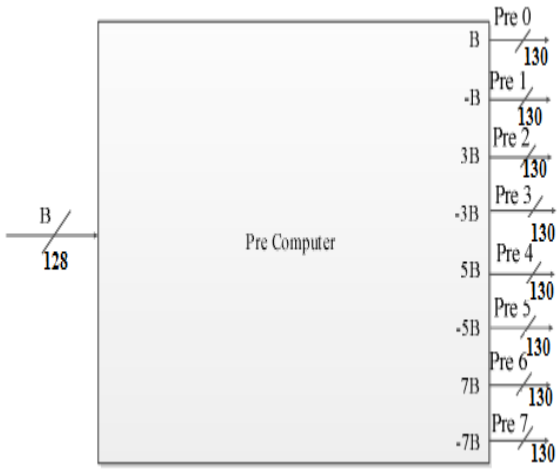

Fig.4 Precomputation block

The precomputer values B, -1B, 3B, -3B, 5B, -5B, 7B and -7B are called FDMPPS (Fundamental Group Digit Multiplied partial Products). From these FGDMPPs, the secondary Group Digit Multiplied Partial Products (SGDMPPs) and Tertiary Group Digit Multiplied Partial Products (TGDMPPs) are generated by suitable shifting operations and/or by combining suitable values. The suitable partial products are selected with control signals which are generated from the digit sets as follows.

D. Control signal generator

The control signal generator, shown in fig.5, taking the multiplier as input, generates 16 sets of control signals of two categories, the sdigit and the tdigit.

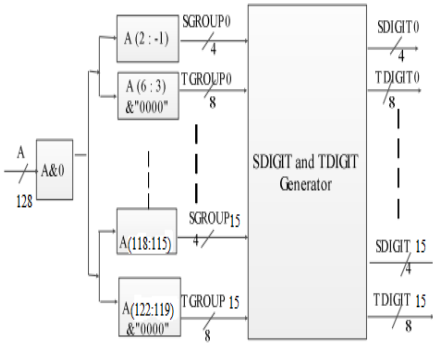

Fig.5. Control signal generator.

The flow chart for generation of these control signals in the SDIGIT and TDIGIT generator block of fig.5 is shown in fig.6.The algorithm for generation of these control signals is given below. Sixteen temporary group variables, two from each Di are generated as given in (7…22). Sgroup0 and Tgroup0 correspond to D0, Sgoup1 and

Tgroup1 correspond to D1 and so on.

$$Sgroup0 = 4 * A_2 + 2 *A_1 + 1 *A_0 + 1 *A_{-1} \quad (7)$$
$$Tgroup0 = 16 * (4 * A_6 + 2 *A_5 + 1 *A_4 + 1 * A_3) \quad (8)$$
$$Sgroup1 = 4 * A_{10} + 2 *A_9 + 1 * A_8 + 1 *A_7 \quad (9)$$
$$Tgroup1=16*(4 * A_{14}+2 *A_{13} + 1 *A_{12} + 1 *A_{11}) \quad (10)$$
$$Sgroup2 = 4 * A_{18} + 2 *A_{17} + 1 *A_{16} + 1 *A_{15} \quad (11)$$
$$Tgroup2 =16 * (4 * A_{22} + 2*A_{21}+1*A_{20} +1 * A_{19}) \quad (12)$$
$$Sgroup3 = 4 * A_{26} + 2 *A_{25} + 1 * A_{24} + 1 *A_{23} \quad (13)$$
$$Tgroup3=16 * (4 *A_{30} + 2*A_{29}+1*A_{28} + 1 *A27) \quad (14)$$
$$Sgroup4 = 4 * A_{34} + 2 *A_{33} + 1 *A_{32} + 1 *A_{31} \quad (15)$$
$$Tgroup4= 16 * (4 * A_{38} +2 *A_{37}+1*A_{36}+ 1*A_{35}) \quad (16)$$
$$Sgroup5 = 4 * A_{42} + 2 *A_{41} + 1 * A_{40} + 1 *A_{39} \quad (17)$$
$$Tgroup5= 16 * (4 * A_{46} +2 *A_{45} +1*A_{44} +1*A_{43}) \quad (18)$$
$$Sgroup6 = 4 * A_{50} + 2 *A_{49} + 1 *A_{48} + 1 *A_{47} \quad (19)$$
$$Tgroup6 = 16 * (4*A_{54} +2*A_{53} +1*A_{52} + 1 * A_{51}) \quad (20)$$
$$Sgroup7 = 4 * A_{58} + 2 *A_{57} + 1 * A_{56} + 1 *A_{55} \quad (21)$$
$$Tgroup7 = 16 * (4 * A_{62} + 2 *A_{61} + 1 *A_{60} + 1 *A_{59}) \quad (22)$$
$$Sgroup8 = (4 * A_{66} + 2 *A_{65} + 1 *A_{64} + 1 * A_{63}) \quad (8)$$
$$Tgroup8 = 16 * (4 * A_{66} + 2 *A_{65} + 1 *A_{64} + 1 * A_{63}) \quad (8)$$
$$Sgroup9 = 4 * A_{70} + 2 *A_{69} + 1 * A_{68} + 1 *A_{67} \quad (9)$$
$$Tgroup9 = 16 * (4 * A_{74} + 2 *A_{73} + 1 *A_{72} + 1 *A_{71}) \quad (10)$$
$$Sgroup10= 4 * A_{78} + 2 *A_{77} + 1 *A_{76} + 1 *A_{75} \quad (11)$$
$$Tgroup10 = 16 * (4 * A_{82} + 2 *A_{81} + 1 *A_{80} + 1 * A_{79}) \quad (12)$$
$$Sgroup11 = 4 * A_{86} + 2 *A_{85} + 1 * A_{84} + 1 *A_{83} \quad (13)$$
$$Tgroup11 = 16 * (4 * A_{90} + 2 *A_{89} + 1 *A_{88} + 1 *A_{87}) \quad (14)$$
$$Sgroup12 = 4 * A_{94} + 2 *A_{93} + 1 *A_{92} + 1 *A_{91} \quad (15)$$
$$Tgroup12 = 16 * (4 * A_{98} + 2 *A_{97} + 1 *A_{96} + 1 * A_{95}) \quad (16)$$
$$Sgroup13 = 4 * A_{102} + 2 *A_{101} + 1 * A_{100} + 1 *A_{99} \quad (17)$$
$$Tgroup13 = 16 * (4 * A_{106} + 2 *A_{105} + 1 *A_{104} + 1 *A_{103}) \quad (18)$$
$$Sgroup14 = 4 * A_{110} + 2 *A_{109} + 1 *A_{108} + 1 *A_{107} \quad (19)$$
$$Tgroup14 = 16 * (4 * A_{114} + 2 *A_{113} + 1 *A_{112} + 1 * A_{111}) \quad (20)$$
$$Sgroup15 = 4 * A_{118} + 2 *A_{117} + 1 * A_{116} + 1 *A_{115} \quad (21)$$
$$Tgroup15 = 16 * (4 * A_{122} + 2 *A_{121} + 1 *A_{120} + 1 *A_{119}) \quad (22)$$

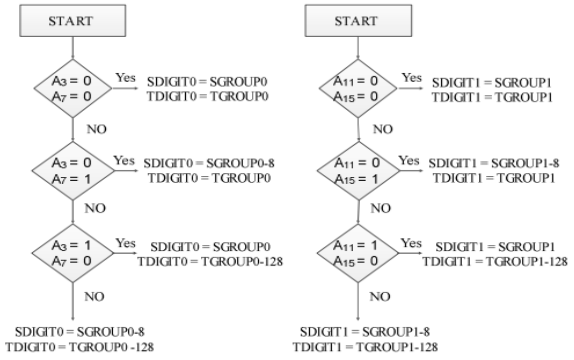

Fig.6. Flow chart for Sdigit and Tdigit Signal Generation

From these temporary variable groups, the sdigit and tdigit control signals are generated as given in the flow charts presented in Fig.6. The flow chart shows the generation of Sdigit0, tdigit0, sdigit1 and tdigit1. The other 12 control signals are generated accordingly. The Sdigit0 and Tdigit0 select the corresponding SGDMPP0 and TGDMPP0, the addition of which gives D0.B. Accordingly the other control signals are used to generate other partial products D1.B … D7.B.These control signals select the proper partial products, Di .Bs. These partial products are added using RB addition to avoid carry propagation.

E.Redundant binary addition of partial products.

• Each pair of the 8 partial products generated above are added separately using redundant binary addition.

• Redundant binary system was proposed by Avizienis [4] in the year 1960. Redundant number system uses more digits than the radix to represent the numbers in that system. In redundant binary number system the digit set {1, 0, 1} is used to represent {0,1}. Thus the binary numbers are encoded using the table-1 for conversion to RB number.

TABLE-1 REDUNDANT BINARY ENCODING

X^+	X^-	RB Digit
0	0	0
0	1	$\bar{1}$
1	0	1
1	1	0

• The sum of two NB numbers in RB can be found by the following formula 2015 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia) 139 x+y = x-(-y) = x-(y^- +1) = (x- y)-1 = (x, y)-1..

(23)

$$x = \quad 1 \quad 0 \quad 1 \quad 1$$
$$\bar{y} = \quad 0 \quad 1 \quad 0 \quad 1$$
$$(x, \bar{y}) = \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1$$
$$(x, \bar{y})-1 = 10011011 - 1 = 10011010$$

• Thus x+y in RB form can be obtained [8] by concatenating the corresponding digits of x and complement of y and subtracting 1 from the same. Thus if x=1011 and y=1010 then x+y in RB form will be 10011011-1=10011010 as follows.

Thus from the 8 partial products 4 intermediate partial products are obtained in RB form. • If any of the RB digit pairs (x+ ,x-) is (1,1), then that pair may be converted to (0,0) to avoid any carry during addition as both (1,1) and (0,0) represent the NB number '0'.

C. Addition of RB partial products

• Each of the two above RB partial products are added to get another two RB partial products using redundant binary addition.

• Redundant binary addition is a carry propagate free addition.

• Two numbers can be added without the need of propagating the carry in the following steps.

TABLE-II RULES FOR COMPUTATION OF RB ADDITION

(x_i)	(y_i)	(x_{i-1}, y_{i-1})	(S_i)	(C_i)
1	1	Any value	0	1
1 0	0 1	Both non-negative	$\bar{1}$	1
		Other wise	1	0
0	0	Any value	0	0
1	$\bar{1}$			
$\bar{1}$	1			
0	$\bar{1}$	Both non-negative	$\bar{1}$	0
$\bar{1}$	0	Other wise	1	$\bar{1}$
$\bar{1}$	$\bar{1}$	Any value	0	$\bar{1}$

• If Xi and Yi are two RB digits, then Xi+Yi = 2Ci+Si where Ci €{1 ,0,1} is the intermediate carry digit and Si €{ 1,0,1} is the intermediate sum digit. 2Ci means 'Ci generated in this process' is shifted to the left by one position.

• The intermediate sum Si and carry Ci are computed by following the rules given in table II. This addition must be performed considering the values added in the adjacent lower order positions.

D. RB to NB conversion

RB to NB conversion is carried using the following formula after separating the X+ and X- parts of the RB number. V= X+ -X- = X+ + (X -) +1 …(28) If the RB number is X = 1 110 = 01011000 after encoding = 0010+0011+1 = 0111 = 6 This operation is performed with a carry look ahead adder. The output so obtained is truncated to 64 MSB bits after leaving off the redundant sign bit.

E. 2's Complementer

The sign of the output obtained above depends on the sign bits of the input numbers. The sign bits separated in the beginning are XORed and the output of it is used as an enable signal to a 2's complementer stage. This enable signal if '1' complements the output of the complementer and gives the output in 2's complement and if '0', no change takes place and the output is obtained as it is.

Simulation and Synthesize Results

The simulation results for 128 bit Q-format multipliers are given in fig.8 The delay in ns and the area in terms of LUTs for128 bit and 64 bit Q-format multipliers for Radix-256 Booth encoding with RB addition along with comparison are given in table.III and IV

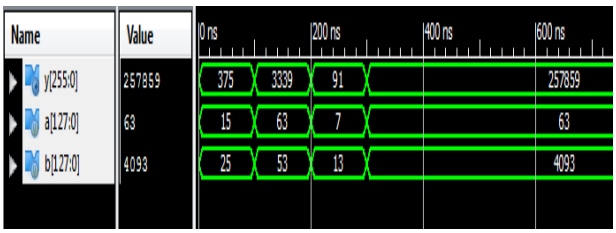

Fig .8 simulation wave form for 128 bit multiplier

Area comparisons:

	64 BIT	128 BIT
No of slices	7497	29542
No of 4 input LUTS	13911	54709

DELAY COMPARISIONS:

	DELAY
64 BIT	51.729 ns
128 BIT	63.179 ns

CONCLUSION

The Q-format multiplier for128/ 64/32/16 bits is implemented with Radix-256 Booth multiplier with RB adder and the results are compared with those obtained with Vedic multiplier and found that there is a significant decrease in the combinational delay with the implementation with the former. It is also established that as the number of bits is increasing the combinational delay is decreasing with Radix-256 Booth multiplier with RB addition. Hence it is advantageous to use Radix-256 Booth multiplier with RB adder at higher bit lengths.

References

1. Saokar, S.S., Banakar, R.M.; Siddamal, S., "High speed signed multiplier for Digital Signal Processing applications," in Signal Processing, Computing and Control (ISPCC), 2012 IEEE International Conference on , vol., no., pp.1-6, 15-17 March 2012
2. Jagadguru Swami Sri Bharati Krisna Tirthaji Maharaja, "Vedic Mathematics: Sixteen Simple Mathematical Formulae from the Veda," Motilal Banarasidas Publishers, Delhi, 2009, pp. 5-45.
3. Sahoo, S.K.; Reddy, K.S., "A High Speed FIR Filter Architecture Based on Novel Higher Radix Algorithm," in VLSI Design (VLSID), 2012 25th International Conference on , vol., no., pp.68- 73, 7-11 Jan. 2012.
4. Avizienis, "Signed-digit number representation for fast parallel arithmetic," IRE Trans. Electron. Computer., vol.EC-10, pp. 389- 400, Sept.1961.
5. A.D. Booth, "A signed binary multiplication technique," Quarterly J.Mech. Appl. Math, Vol. 4, Part 2, pp. 236-240, 1951
6. N. Takagi, H. Yasura and S. Yajima., "High-speed VLSI multiplication algorithm with a redundant binary addition tree," IEEE Transactions on Computers, C-34(9): 217-220, Sept.1985.
7. Hiroshi Makino,Yasunobu Nakase, Hiroaki Suzuki, Hiroyuki Morinaka, Hirofumi Shinohara, and Koichiro Mashiko, "An 8.8-ns 54 x 54-Bit multiplier with high speed redundant binary architecture," IEEE J. Of Solid State Circuits, Vol.31, No.6, pp. 773-783, June 1996.
8. Yum Kim, Bang-Sup Song, John Grosspietsch, and Steven F. Gilling , "A carry-free 54b x 54b multiplier using equivalent bit conversion algorithm, " IEEE J. Of Solid State Circuits, Vol. 36, No.10, pp. 1538-1544, Oct. 2001.
9. Y. Harata, Y. Nakamura, H. Nagese, M. Takigawa, and N. Takagi, "A high-speed multiplier using a redundant binary adder tree," IEEE J.Solid-State Circuits, Vol. 22, pp. 28-34, Feb. 1987.
10. Filix AsaJyothi, V. koteswara Rao, "An efficient architecture of the radix based FIR filter design", International journal of reviews on recent electronics and computer science (IJRRECS), August 2013, pp. 300-305.
11. K.K.Parhi, "VLSI Digital Signal Processing Systems", John Wiley and Sons, UK, 2007 pp, 529-551

Authors Details:

K.Mounika has completed his B.Tech in Electronics and Communication Engineering from Jagruti Institute of Engineering College, J.N.T.U.H affiliated College in 2015. He is pursuing his M.Tech in VLSI System Design from Vidya Jyothi Institute Of Technology, (Autonomous) J.N.T.U.H affiliated college.

A. Jaya Lakshmi is an Associate Professor at Vidya Jyothi Institute of Technology from (ECE), Hyderabad. She received her Bachelor degree in Electronics and Communication Engineering from Narsaraopeta Engineering College, M.Tech from Sana Engineering College. Her interest in ES and VLSI design and Signal processing.