Indian Journal of Science and Technology, *Vol 8(33), DOI: 10.17485/ijst/2015/v8i33/71204, December 2015*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Performance Evaluation of High Speed Radix 8 Tree Based Multiplier

Sharmila Hemanandh[1]* and A. Sivasubramanian[2]

[1]Department of Electronics and Communication Engineering, Sathyabama University, Chennai – 600119, Tamil Nadu, India; sharmilaahemanandh@gmail.com
[2]School of Electronics, Vellore Institute of Technology, Chennai - 600127, Tamil Nadu, India; shiva_31@yahoo.com

Abstract

Multipliers are the basic building blocks of signal processing and arithmetic based systems. The objective of this paper is to design a high speed multiplier that significantly improves the performance of many high performance DSP, multimedia and communications systems. This paper proposes a high speed radix 8 Booth multiplier employing signed digit representation for recoding and radix 8 modified Booth algorithm that reduces the number of partial products to n/3. The design of the multiplier is based on Wallace tree architecture with considerable improvement in performance. This paper compares the performance of conventional multiplier with radix 4 tree based Booth multiplier and radix 8 tree based Booth multiplier. The radix 8 tree based Booth multiplier is synthesized using Quartus II simulation tool. The proposed radix 8 tree based multiplier exhibits better performance with respect to area and operating frequency when compared to conventional multiplier.

Key Words: Higher Radix Booth Multiplier, Partial Product Reduction, Wallace Tree Multiplier

1. Introduction

Multiplication and Multiply and Accumulate (MAC) are the two important arithmetic operations used in most of the DSP algorithms. The design of the arithmetic unit plays a major role in the performance of any DSP system. The time consumed by the multiplier and the amount of hardware circuitry are more when compared with any other operation in an arithmetic unit. Designing a multiplier with high speed, low power and less area is a major subject of interest over a decade. Multiplication dominates the execution time of most computationally intensive DSP algorithms such as convolution, correlation, Fast Fourier Transform, Discrete Cosine Transform and Wavelet Transform. In general multiplication is carried out by repeated addition method. The number to be added is the multiplicand and the number of times the multiplicand is added is the multiplier resulting in the product of the two numbers. This repeated addition method is very slow. The two basic operations involved in the implementation of a multiplier are 1. Generation of partial product 2. Accumulating the shifted partial product. The number of partial products generated can be reduced using the booth algorithm. Improvement in speed is achieved using Wallace tree structure to accumulate the partial products. Over the years, many researchers have proposed high speed multipliers. Redundant binary adders are used in the design of radix 4 booth multiplier that increases the speed, whereas reduces the energy consumed by the design[1]. A neighborhood dependent approach is used in the design of high speed-low power radix 4 booth multiplier for video processing applications[2]. The dynamic power consumed by the design is reduced due to reduced switching activity. Bit serial interleaved multiplication algorithm is used to design radix 4 and radix 8 booth multiplier to reduce the number of clock cycles[3]. A high speed radix 4 multiplier for ALU's is designed in to generate minimal partial products thereby reducing the critical path delay[4]. The design of hybrid multiplier for mobile GPU applications using hybrid booth encoder and hybrid

Author for correspondence

truncation scheme that exhibits low power and delay is proposed[5]. The power dissipated by the modified radix 4 booth multiplier using ripple carry adder is less when compared with multiplier using carry-look-ahead adder[7]. High speed Booth multiplier design using reversible logic resulting in less logical complexity design is given by Amita Nandal[8]. The design results in less power dissipation and less delay with high speed of operation. An efficient radix 8, fixed width booth multiplier design is proposed with significant reduction in maximum absolute error and mean square error[9]. Partial products are generated using radix 10 recoding technique and the use of parallel multiplier architecture results in the optimization of area and power[10].

2. Array Multiplier

Add and shift method of multiplication is comparatively faster than repeated addition method. Array multiplier uses array of adders to shift and add at the same time. Consider two 4 bit numbers A (a0 a1 a2 a3) and B (b0 b1 b2 b3). Initially b0 is multiplied with a0 a1 a2 a3 to generate the first partial product a0 b0 a1 b0 a2 b0 a3 b0. Then b1 is multiplied to generate the second partial product a0 b1 a1 b1 a2 b1 a3 b1 and this is shifted one bit towards the left. Similarly the third and the fourth partial products are also generated by multiplying b2 and b3 respectively with A. Finally the partial products are summed up to obtain the final product as shown in Figure 1.

The carry generated in each column is passed on to the next column. All the product terms such as a3 b0, a2 b0, a1 b0 in the partial product are generated using AND gates. The product bit p0 is obtained using AND gate with inputs a0 and b0. The two product terms a1 b0 and a0 b1 are added using half adder. The sum output of the half adder is p1 and the carry output is passed on to the next column. In the third column a2 b0 is added with a1 b1 along with the carry received in the previous stage using a full adder. The sum is again added to a0 b2 to obtain the

				a3	a2	a1	a0
				b3	b2	b1	b0
				a3b0	a2b0	a1b0	a0b0
			a3b1	a2b1	a1b1	a0b1	
		a3b2	a2b2	a1b2	a0b2		
	a3b3	a2b3	a1b3	a0b3			
p7	p6	p5	p4	p3	p2	p1	p0

Figure 1. Array multiplier.

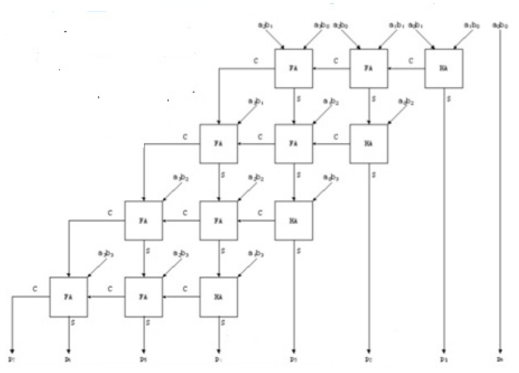


Figure 2. 4 X 4 array multiplier.

product bit p2 and the carry moves vertically to the next column. The same procedure is repeated to obtain the other product bits. In the next 7th column a3 b3 is added with the carry generated in the previous stage and may result in a carry which is p7. So for a 4 * 4 array multiplier in Figure 2 16 AND gates are required to generate the product terms and 12 adders (4 half adders and 8 full adders) are required to generate the product. In general for an x * y array multiplier xy number of AND gates, x number of half adders, (x-2) y number of full adders are required.

The size of the AND array depends upon the operand size. The number of rows and the columns of the array is given by the number of bits in the multiplier and the multiplicand respectively. The size of the array increases at a rate proportional to the size of the operand. So when the array size increases, the number of gates increases. Moreover, the delay depends upon the width of the multiplier. As the size of the array increases, both area and delay increase limiting the speed of operation. Array multipliers can be implemented to support high rate of pipelining. Owing to its regular structure array multipliers can be implemented as a rectangular shaped array without wasting chip area.

3. Wallace Tree Multiplier

A conventional Wallace tree multiplier is a tree based parallel multiplier that multiplies two numbers in three steps:

- Generation of partial products.
- Reduction of partial products.
- Addition of partial products to obtain the final product.

At first the partial products are generated by multiplying each bit of the multiplier with the multiplicand as done in the array multiplier. The partial products are arranged in a row one below the other with necessary shift based on the bit position of the multiplier multiplied with the multiplicand. In the next step the partial products are grouped as three rows. Reduction is performed in each column of the group using half adders and full adders depending upon the number of elements in the group. No reduction is done on a group with less than three rows. This process is repeated until the number of rows is reduced to two. Now the final product is obtained by adding the two rows using carry propagate adder. Wallace tree architecture exhibits improvement in speed because all the partial products in each column are added in parallel. The propagation delay is greatly reduced when compared with array multiplier. Figure 3 explains the algorithm of an 8x8 Wallace tree multiplier.

Waters and Swartzlander[11] proposed a reduced complexity Wallace tree multiplier design where the partial products are arranged in a reverse pyramid pattern. The number of stages in computing the product is the same as for the traditional Wallace tree multiplier. The reverse pyramid arrangement reduces the area of the reduction process. The reduced complexity Wallace tree algorithm is shown in Figure 4.

The Wallace tree multiplier uses a tree of carry save adders to reduce the partial products. The use of carry save adders avoids carry propagation. Using this algorithm the number of adders required and hence the critical path is reduced when compared with an array multiplier. The time required to calculate the product using the add-shift method linearly increases with increase in the size of the operand, whereas the Wallace tree multiplier computes the product much faster at the expense of additional hardware.

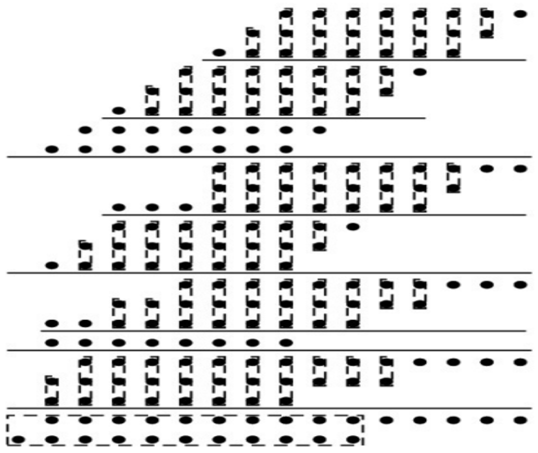


Figure 3. 8X8 Wallace tree algorithm.

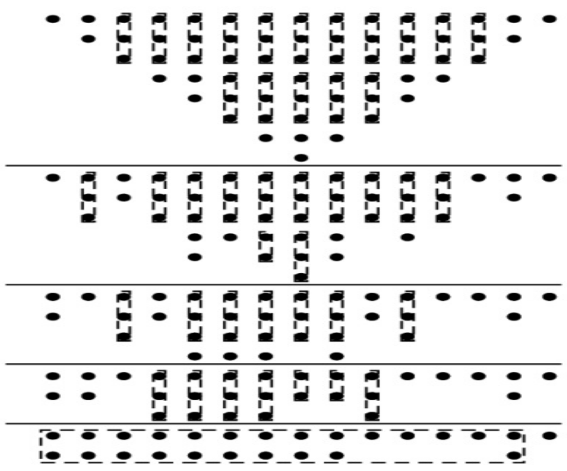


Figure 4. Reduced complexity Wallace tree algorithm.

4. Booth Multiplier

Booth multiplication algorithm is an efficient method to multiply two signed numbers using two's complement representation. Booth algorithm reduces the number of partial products resulting in better performance of the multiplier. This is achieved using Booth encoding technique.

4.1 Radix 2 Booth Algorithm

Using the Booth encoding technique the multiplier is recoded with values 0, +1 and −1 by examining a pair of multiplier bits starting from the LSB. Each pair is recoded with a value using Table 1.

As an example, consider the two numbers A = 12 and B =−13

The product P = −13 X 12 = − 156 (1101100100)

Binary representation of 13: 01101

Binary representation of −13: 10011

Binary representation of 12: 01100

Binary representation of −12:10100

Now operand A is the multiplicand and operand B is the multiplier. U and V are the two registers that store the product. M holds the value of the multiplier. Initially load a 0 in C.

Step 1: Initially the last two bits are 10 so subtract the multiplicand with U. Instead of subtracting, the two's complement of A is added with U. After addition the contents of U and V are shifted using right shift arithmetic. The contents of M are circularly shifted right by one position.

Table 1. Radix 2 booth recoding table

Multiplier bits	Recoded value	Action Performed
00	0	Shift
01	+1	Add
10	−1	Subtract
11	0	Shift

Table 2. Radix 2 booth multiplication algorithm

U	V	M	C	Action
00000	00000	10011	0	Sub and shift
10100	00000			
10100	00000	10011		
11010	00000	11001	1	Shift only
11101	00000	11100	1	Add and shift
01100	00000			
01001	00000	11100		
00100	10000	01110	0	Shift only
00010	01000	00111	0	Sub and shift
10100	01000			
10110	01000	00111		
11011	00100	10011	1	Shift only

Step 2: Now the last two bits are 11, the contents of U and V are shifted.

Step 3: Now the two bits are 01. So the multiplicand is added with Register U and then the contents of U and V are shifted.

Step 4: Since the last two bits are 00 the contents of register U and V are shifted.

Step 5: Now the last two bits are 10. The contents of u are subtracted from the multiplicand and the contents of U and V are shifted.

Step 6: The bits are 11. Hence the contents of U and V are shifted to get the final product.

The entire procedure is summarized in Table 2.

4.2 Radix 4 Booth Algorithm

In the radix 4 booth algorithm three bits are examined and recoded per cycle. Hence the number of cycles to obtain the final product is reduced when compared with radix 2 algorithm. The use of higher radix algorithm improves the speed of operation when the width of the operand increases. Consider the example: Multiplicand A = 12 and multiplier B = −13. At first a zero is added at the end of the multiplier. Then three bits are grouped per block starting from the LSB. Each block overlaps the previous block by one bit. Each block is assigned a scale factor (+1, 0, +2) from the recoding table as shown in Figure 5. Each scale factor performs a particular operation on the multiplier to generate the partial products.

Since the first block is 110 resulting in a recoded value of −1, the first partial product PP1 is obtained by taking the two's complement of the multiplicand. Next, the second block is 001 which is recoded to +1. The second partial product PP2 is the multiplicand itself. The third block is 110 resulting in −1. Hence PP3 is obtained by taking two's complement of the multiplicand. The final product P is obtained by adding the partial products using carry save adders. Since the product is negative, it is in the two's complement form. The entire procedure is summarized in Figure 6.

4.3 Radix 8 Booth algorithm

The delay of the multiplier mainly depends upon the number of adders used. Higher radix Booth algorithm reduces the number of partial products generated resulting in less area and less power consumption[12]. The number of partial products is reduced by n/3 where n denotes the number of bits in the multiplier when compared with radix 4 which is n/2. The radix 8 algorithm uses less number of transistors resulting in reduced power dissipation and area compared with radix 4 algorithm[13].

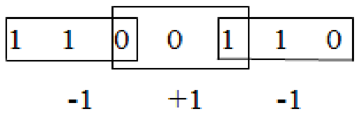


Figure 5. Radix 4 recodin.

						0	0	1	1	0	0	+12
						1	1	0	0	1	1	−13
1	1	1	1	1	1	1	1	0	1	0	0	PP1
0	0	0	0	0	0	1	1	0	0			PP2
1	1	1	1	0	1	0	0					PP3
1	1	1	1	0	1	1	0	0	1	0	0	P

Figure 6. Radix 4 booth algorithm.

Table 3. Radix 4 booth recoding table

Multiplier bits	Recoded value	Action Performed
000	0	0*Multiplicand
001	+1	+1*Multiplicand
010	+1	+1*Multiplicand
011	+2	+2*Multiplicand
100	–2	–2*Multiplicand
101	–1	–1*Multiplicand
110	–1	–1*Multiplicand
111	0	0*Multiplicand

Table 4. Radix 8 booth recoding table

Multiplier bits	Recoded value	Action Performed
0000	0	0*Multiplicand
0001	+1	+1*Multiplicand
0010	+1	+1*Multiplicand
0011	+2	+2*Multiplicand
0100	+2	+2*Multiplicand
0101	+3	+3*Multiplicand
0110	+3	+3*Multiplicand
0111	+4	+4*Multiplicand
1000	–4	–4*Multiplicand
1001	–3	–3*Multiplicand
1010	–3	–3*Multiplicand
1011	–2	–2*Multiplicand
1100	–2	–2*Multiplicand
1101	–1	–1*Multiplicand
1110	–1	–1*Multiplicand
1111	0	0*Multiplicand

In radix 8 algorithm 4 bits are grouped per block instead of 3 as in radix 4 algorithm. Each block is assigned a recoded value from Table 4. Each recoded value performs a particular operation on the multiplicand to generate the partial product. +1 – retains the multiplicand as such. –1 – computes the two's complement of the multiplicand, +2 – left shifts the multiplicand by one bit, –2 – left shifts the two's complement of the multiplicand by one bit, (if y is the multiplicand) +3 – calculates (y +3y), =3 – gives the complement of (y +3y), =4 – left shifts the multiplicand by two bit position, –4 – left shifts the complement of multiplicand by two bit position.

This paper compares the performance of four different types of multipliers, namely array multiplier, radix 4 booth multiplier, radix 4 tree based booth multiplier and radix 8 tree based booth multiplier.

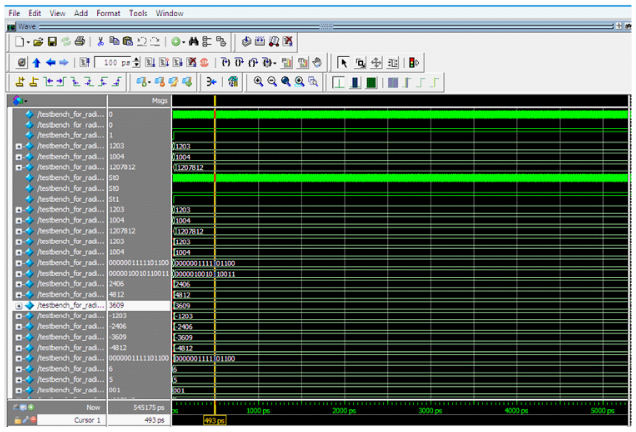
5. Results and Discussions

Booth multiplier is faster than array multiplier. As the number of bits increases, gate delay and area increase slowly when compared with other multipliers. The greatest advantage of Booth multiplier when compared with other multipliers is the regularity of its structure. Hence it can be easily realized on silicon to work at high speed without increasing the clock frequency. The proposed radix 8 Booth multiplier is based on Wallace tree architecture in which carry select adders are used for intermediate partial product accumulation and carry-look-ahead adder is used for final accumulation.

Figure 7 shows the simulation results of radix 8 tree based Booth multiplier for both signed and unsigned numbers.

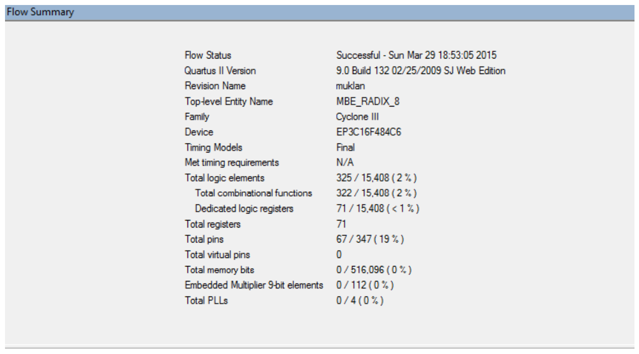
The design of the proposed architecture is targeted on to EP3C16F484C6 belonging to Cyclone III family. The multipliers are compiled and synthesized using Quartus II simulation tool. The compilation report and the timing analyzer report are shown in Figures 8 and 9. Figure 10 is the RTL schematic diagram of the proposed design.

Table 5 summarises the area and the operating frequency of the four multipliers. From the results it is



```
/testbench_for_radi...  0
/testbench_for_radi...  0
/testbench_for_radi...  1
/testbench_for_radi...  1203
/testbench_for_radi...  1004
/testbench_for_radi...  1207812
/testbench_for_radi...  St0
/testbench_for_radi...  St0
/testbench_for_radi...  St1
/testbench_for_radi...  1203
/testbench_for_radi...  1004
/testbench_for_radi...  1207812
/testbench_for_radi...  1203
/testbench_for_radi...  1004
/testbench_for_radi...  0000001111101100
/testbench_for_radi...  0000010010110011
/testbench_for_radi...  2406
/testbench_for_radi...  4812
/testbench_for_radi...  3609
/testbench_for_radi...  -1203
/testbench_for_radi...  -2406
/testbench_for_radi...  -3609
/testbench_for_radi...  -4812
/testbench_for_radi...  0000001111101100
/testbench_for_radi...  6
/testbench_for_radi...  5
/testbench_for_radi...  001
Now       545175 ps
Cursor 1  493 ps
```

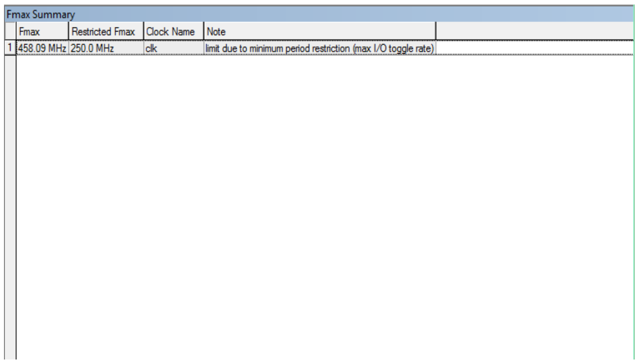
Figure 7. Simulation results of radix 8 tree based Booth multiplier.



Flow Summary

```
Flow Status                           Successful - Sun Mar 29 18:53:05 2015
Quartus II Version                    9.0 Build 132 02/25/2009 SJ Web Edition
Revision Name                         muklan
Top-level Entity Name                 MBE_RADIX_8
Family                                Cyclone III
Device                                EP3C16F484C6
Timing Models                         Final
Met timing requirements               N/A
Total logic elements                  325 / 15,408 ( 2 % )
   Total combinational functions      322 / 15,408 ( 2 % )
   Dedicated logic registers          71 / 15,408 ( < 1 % )
Total registers                       71
Total pins                            67 / 347 ( 19 % )
Total virtual pins                    0
Total memory bits                     0 / 516,096 ( 0 % )
Embedded Multiplier 9-bit elements    0 / 112 ( 0 % )
Total PLLs                            0 / 4 ( 0 % )
```

Figure 8. Compilation report of radix 8 tree based Booth multiplier.



Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	458.09 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

Figure 9. F-max summary report of radix 8 tree based Booth multiplier.

Table 5. Comparison of different multipliers

Multiplier Type	Area	F-max
Array Multiplier	327	129MHz
Radix 4 Booth multiplier	285	239.35MHz
Radix 4 tree based Booth multiplier	264	265.89MHz
Radix 8 tree based Booth multiplier	325	458.09MHz

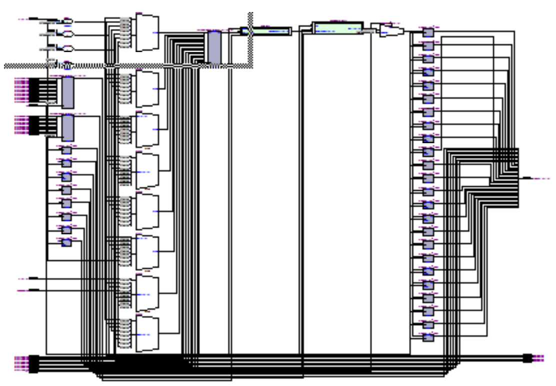


Figure 10. RTL schematic diagram.

observed that radix 8 tree based Booth multiplier is more efficient in the total number of logic elements used and the operating frequency when compared with array multiplier.

6. Conclusion

A unique high speed booth multiplier was proposed to improve throughput rate and to minimize the area complexity. The proposed multiplier is based on Wallace tree based architecture with reduced number of partial products and this played a major role in increasing the operating frequency. Hence the proposed multiplier achieves a maximum operating frequency and optimized synthesis results. The frequency with which the device can work is 458.09MHz, which is greater than the value with which the conventional array multiplier operates.

7. References

1. Surendran EKL, Antony PR. Implementation of fast multiplier using modified radix-4 Booth algorithm with redundant binary adder for low energy applications. Computational Systems and Communications (ICCSC). IEEE: Trivandrum; 2014 Dec 17-18. p. 266–71.
2. Ngo HT, Asari VK. Design of a radix-4 booth multiplier with neighbourhood dependent approach for video processing applications. Circuits and Systems, IEEE: Montreal QU; 2007 Aug 5-8. p. 53–6.
3. Javeed K, Wang X. Radix-4 and radix-8 booth encoded interleaved modular multipliers over general Fp. Field Programmable Logic and Applications. IEEE: Munich; 2014 Sep 2-4. p. 1–6.
4. Basha SS, Jahangir BS. 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. 2012 Jul; 4(1):314–25.
5. Choi S, Gyeonghoon K, Yoo HJ, Nam BG. Hybrid radix-4/-8 truncated multiplier for mobile GPU applications. IEEE. 2014 Jun; 50(23):1680–2.
6. Ramteke SK, Dubey A, Khandagare Y. VLSI designing of low power radix4 booths multiplier. International Journal of Electrical, Electronics and Computer Systems. 2014; 2(2):48–52.
7. Kumre L, Somkuwar A. Agnihotri G. Implementation of radix 4 booth multiplier using MGDI technique. International Conference on Microelectronics, Communication and Renewable Energy. IEEE: Kanjirapally; 2013. p. 1–5.

8. Nandal A, Vigneswaran T, Rana AK. Booth multiplier using reversible logic with low power and reduced logical complexity. Indian Journal of Science and Technology, 2014 Apr; 7(4):525–9.
9. Bhusare SS, Bhaskaran VSK. Design of a low-error fixed-width radix-8 booth multiplier. Fifth International Conference on Signal and Image Processing: IEEE; 2014. p. 2069.
10. Kumar HAA, Umadevi S. Implementation of fast radix-10 BCD multiplier in FPGA. Indian Journal of Science and Technology. 2015 Aug; 8(19):IPL0147.
11. Waters RS. Swartzlander EE. A reduced complexity wallace multiplier reduction. IEEE Transactions on Computers. 2010 Aug; 59(8):1134–7.
12. Raghavendra J, Vigneswaran T. Design of fused add-multiply operator using modified booth recorder for fast arithmetic circuits. Indian Journal of Science and Technology. 2015; 8(19): IPL0149.
13. Rubinfield LP. A proof of the modified booth's algorithm for multiplication. IEEE Transaction on computers. 2006: 39(10):1014–5.