https://doi.org/10.29042/2020-10-5-08-14

Helix (2020) 10 (5): 08-14

Acceleration Techniques using Reconfigurable Hardware for Implementation of Floating Point Multiplier

[1] D S Bormane, [2] Sushma Wadar, [3] Avinash Patil, [4] S C Patil
[1] AISSMSCOE, Pune, MS-India
principal@aissmscoe.com
[2] E&TC Department, AISSMSIOIT, Pune, MS-India
sushma97in@yahoo.co.in
[3] E&TC Department, AIT, Pune, MS-India
avispatil@yahoo.co.uk
[4] Department of E&TC, RSCOE, Pune, MS-India
shailaja.patil11@gmail.com

Received: 09th July 2020, Accepted: 25th August 2020, Published: 31st October 2020

Abstract
A multiplier plays a vital part in multimedia and digital signal processors. The integer unit alone cannot achieve the desired computational speed required by modern applications. Unit specially designed to carry out operations on floating point numbers is required also commonly known as floating-point unit. The IEEE 754 standard is used for defining the format of floating point number which is widely accepted. Basically, two formats are used for representing the floating point numbers, single precision which works on 32- bit floating point numbers whereas double precision working on 64 bit. Though the number of bits on which double precision operates doubles as compared to single precision number, it is hardly used due to its requirement of very large memory and also the delay generated for the IEEE-754 standard floating point multiplication. This is the mojor reason for its rare implementation in designs requiinge high computing speed.[1] In this paper we are proposing three efficient algorithms for enhancing the speed and optimizing the area required for implementing single precision floating point multiplication. Also compared the results in terms of power dissipation, execution time and area requirement for the implementation with the conventional methods used. Here the algorithms are implemented and analyzed by using the most popular semi-custom design tool Vivado ISE 2015 and is synthesized by using Artix-7 FPGA and the same is reflected in the mathematical model purposed for each circuit.

Keywords
Floating-point Arithmetic, Double Precision, IEEE-754, Multiplication, Single Precision, Vivado ISE.

Introduction
The microprocessors / microcontroller technically designed to handle the arithmetic operations for integers and less attentation is paid on arithmetics for real numbers.[2]. There are different ways a CPU can use to calculate the values for any floating point operation. The first method is by calling a floating-point emulator, that is nothing but the library of series of simple floating-point functions a than CPUs integer arithmetic operations which runs on the fixed-point ALU.[3] This method saves hardware but very slow. Second method is to use separate FPUs. The speed up of these operations is quite important, because floating point numbers are used in a wide range of applications including CAD, games, graphics, multimedia, and scientific applications also. Also, it is very important for data analysis and manipulation of various signals within Digital Signal Processing (DSP) devices. Floating point numbers are used to represent very small to very large numbers. In floating point arithmetic operations, addition and subtraction are less complex and easy to implement in terms of area required and power dissipation. Multiplication of floating-point numbers requires complex algorithms and it uses more space and there is high power dissipation as well as complex circuits are required. Several algorithms are available for calculation of floating point multiplication.

IEEE754 Standard
A technical standard for representing floating-point numbers was established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE), known as IEEE 754 standard. The figure 1 shows the generalized format of floating point number for various precisions in IEEE standard. It is described by three integers:[1]

s = a sign (zero or one)
z = a significand
e = an exponent

The value is represented as shown in below equation for any floating point number,

$(-1)^s \times 2^{e-127} \times 1.z$

s	**E**	**z**
msb	exponent	Significand

Figure 1: Floating Point Number Format

The table given below summaries the number representation in various precision formats.

Table 1: Floating Point Format's Parameters[11][12]

Parameter	**Single Precision**	**Double Precision**	**Quad Precision**
Bytes	4	8	16
bias, E-e	127	1023	16383
sign bit	1	1	1
exponent bits	8	11	15
significand bits	23	52	112
Smallest Positive Normal Number	$1.175...\ 10^{-38}$	$2.225...\ 10^{-38}$	$3.362...\ 10^{-4932}$
Largest Positive Number	$2.225...\ 10^{+38}$	$1.797...\ 10^{+38}$	$1.189...\ 10^{+4932}$
Significant Digits (Decimal)	6 - 9	11 - 15	33

Table 2: Encoding of $(-1)^s \times 2^{e-127} \times 1.f$ into Binary Fields

Number Type	**Sign bit, s**	**Exponent bits, c**	**significand bits, q**	**significand bit, q+1th**
NaNs	?	111…111	Binary 1xx..x	1
+ Infinity	0	111…111	0000..…000	1
- Infinity	1	111…111	0000..…000	1
Subnormals	$(-1)^s$	0	$\neq 0$	0
Zeros	0	0	0	0

This paper is focusing on novel algorithms for performing multiplication operations on single precision floating point numbers. The format of single precision floating point number is shown in figure 1.[8] A Single precision result gives upto 9 significant decimal digits precision value. The sign of the number, s is the msb bit, that is reflecting whether significand is positive or negative. The unsigned integer e, is used to resemble the value of exponent from 0 to 255. It is the valid value of bias form in IEEE754 single precision number. The 23 bits at the right of the binary point indicates the signifand, z of the actual number. It do not constitute the leading bit 1, of the actual 24 bit precision.

The example below explains the procedure of multiplication of 2 floating-point numbers Y and Z.

Y = 23.5 and Z = 8.5

Binary equivalent of the decimal values Y and Z are:

Y = 10111.1 and Z = +1000.1

Normalized representation of the operands:

$Y = 1.0111x2^4$ and $Z = +1.0001x2^3$

IEEE representation of the operands:

Y = 1 10000011 01110000000000000000000

Z = 0 10000010 00010000000000000000000

Exponent of the result is calculated adding the operands exponents. A 1 can be added if needed by the normalization of the mantissas multiplication. The exponent fields (e_y and e_z) are biased, and are removed for carrying out the addition. Later on the bias should be added again to get the value to be entered into the exponent field of the result (e_r):

$e_r = (e_y - 127) + (e_z - 127) + 127$

$= e_y + e_z - 127$

in our example, 7 is the exponent of the result

$e_y = 10000011$, $e_z = 10000010$ and $-127 = 10000001$

The sign of the result (s_r) is calculated by the exclusive-OR of the operands sign bits (s_a and s_b). [9]

$s_r = s_a \oplus s_b$

in the example considered: $s_r = 1 \oplus 0 = 1$. It indicated a negative valu for the sign value generated.
After composing the result generated by setting above three intermediate results (sign, exponent and mantissa) final result of multiplication will be as follows:

$$Y \times Z = 23.5 \times 8.5 = 1.100011111 \times 2^{134-127}$$

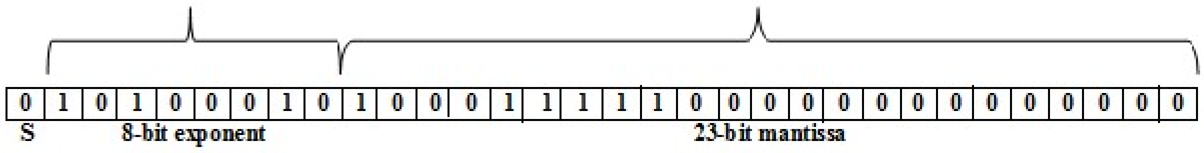


Figure 2: Single Precision Floating Point Format [10]

Real Number to IEEE754 Conversion

Algorithm for Conversion of Real Number to IEEE 754 Standard

1. Integer $(A)_{10}$ and fractional $(B)_{10}$ parts are extracted from the real number.
2. $(A)_{10}$ and $(B)_{10}$ of the real number will be converted to binary i.e. A_2 and B_2 using division methods respectively.
3. A_2 (n-bits) will be stored in 32-bit register.
4. A_2 will be shifted by 32 - n bits to its left by a barrel shifter.
5. Converted fractional part $(B)_2$ will be stored in a 32-bit register and will be shifted by 32- n - m bits to its left by the barrel shifter where m is number of bits in B_2.
6. The shifted value of A_2 and B_2 will be added by a 32-bit adder block having half adders.
7. Output of this will be normalized using a shifter and comparator circuit.
8. Normalized output will be a 24-bit mantissa which will be stored in the memory.

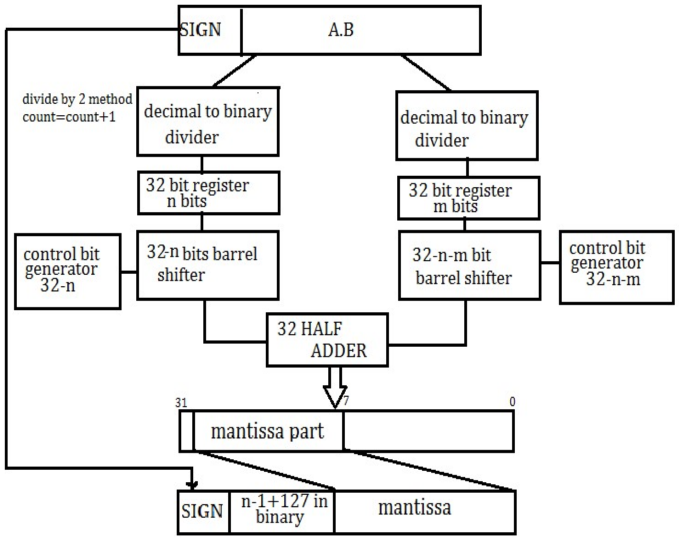


Figure 3: Block Diagram for Converting Real Number to IEEE 754 Standard

Floating Point Multiplication

A generalized diagram for carrying out the multiplication of two floating point numbers is shown in figure 4. The main focus of the proposed algorithms will be to optimise the 24-bit multiplier block in terms of area and speed of execution. . The main aim is to reduce the partial products count and also the size of the operands on which the efficient multiplication algorithms will be applied.

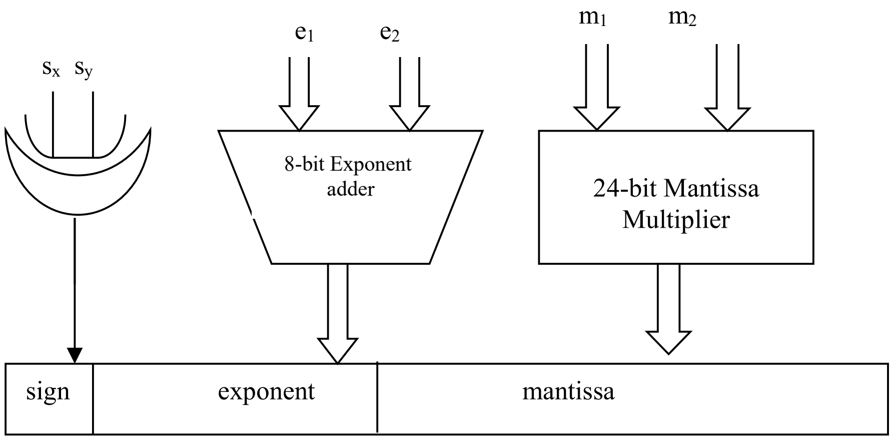


Figure 4: Procedure for Floating Point Multiplication

Proposed Work

The proposed multiplication algorithms can be applied on the 23-bit mantissa's of the two single precision floating point numbers. An Array multiplier generally requires $O(n^2)$ time for calculating the product of two numbers and also requires a large storage space hence not efficient if it is required to multiply long values. O indicates the complexity of the computation.[7] The conventional methods do not help in optimizing the area and speed of the multiplier block. Hence efficient algorithms are required to achieve the same.

1.1. Proposed Method 1

The multiplication methods based on Vedic mathematics have proved to be efficient where area and delay are of prime importance. Karatsuba algorithm splits the inputs into two halves.[5] The concept is to reduce the size of the multiplicand and multiplier of length n into three partial products of length n/2 alongwith some overhead.[6][13] Russian Peasant algorithm is helpful in case of multiplication of small numbers. It reduces the multiplicand to half on each step until it is reduced to one, parallelly doubling the multiplier on the other side at each step. Now note down the steps where the multiplicand is odd and add the corresponding multiplier which will give the final product. Russian Peasant Multiplication proves out to be faster than Urdhva Tiryagbhyam algorithm for the multiplication of two 'n' bit numbers. Taking the advantage of Karatsuba algorithm as best suited for operands of higher bit length and Russian Peasant Multiplication to perform the multiplication on generated partial products, an efficient multiplication algorithm can be implemented. Karatsuba algorithm is used to reduce the large data into smaller size as shown in figure 5 and Russian Peasant algorithm can be applied for multiplication on these reduced size data as shown in figure 7, to overcome the disadvantages of the algorithms makes the proposed multiplier efficient. The performance of Russian Peasant algorithm works very well for lower bit multiplication and Karatsuba algorithm is best for higher bits. By combining both the methods a new algorithm is proposed to reduce the drawbacks of both. .

Proposed Algorithm I:

Step 1. Divide the number of bits present in the operands in two equal halves.

Step 2. The equation of the multiplier becomes,

$= 2^n X_l Y_l + 2^{n/2}[(X_l.Y_l + X_r Y_l) - (X_l - X_r)(Y_l - Y_r)] + X_r Y_r$

Step 3. Compute the above variables individually.

Step 4. Apply Russian Peasant algorithm to calculate the partial products.

Step 5. Now add the individually acquired result to compute the final value .

Example: Multiply X =10101011 and Y=10100110 using proposed Algorithm I

$X = 2^{n/2}X_l + X_r$

$Y = 2^{n/2}Y_l + Y_r$

$X.Y = (2^{n/2}X_l + X_r).(2^{n/2}Y_l + Y_r)$

$= 2^n X_l Y_l + 2^{n/2}(X_l.Y_r + X_r Y_l) + X_r Y_r$

$= 2^n X_l Y_l + 2^{n/2}[(X_l - X_r)(Y_l - Y_r) - (X_l.Y_l + X_r Y_r)] + X_r Y_r$

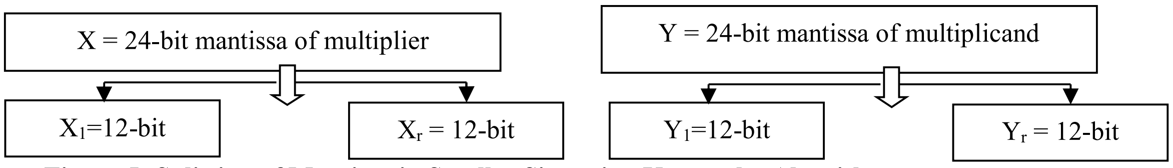
The above equation shows that using Karatsuba algorithm, three multiplications, three subtractions and three additions are the total number of operations required. To multiply two 2-digit numbers using $O(n^{\log_2 3})$ single digit multiplications are required[4].

$(X.Y) + Q = S$

Now if X is even then X=2a for some a.

So it can be written as:

$(2a.Y) + Q = S$
$(a.2Y) + Q = S$
If X is replaced with half its value, and Y is doubled in its value, and reflected as X' and Y' then the above equation can be written as:
$(X'.Y') + Q = S$
In other words, when X is even, it can halve the first number and double the second, and the condition still remains true.
When X is odd, then it can be written as X=2k+1 for some k.
$X.Y + Q = S$
$(2a+1). Y + Q = S$
$2a.Y + Y + Q = S$
$2a.Y + (Y + Q) = S$
$a.2Y + (Y + Q) = S$
$X'.Y' + (Y + Q) = S$
We can reflect X' for half value of X, and Y' for the double value of Y. this is valid if we add the number in the second column to the running total. It can be seen that after we follow the algorithm the ethis equation balanced. But when we finish, the number in the first column is 0. The final equation left is:
$0 \times Y' + Q = S$
Or simply Q = S. the sum is the previously unknown product that was required


X = 24-bit mantissa of multiplier
Y = 24-bit mantissa of multiplicand
X_l=12-bit
X_r = 12-bit
Y_l=12-bit
Y_r = 12-bit
Figure 5: Splitting of Mantissa in Smaller Size using Karatsuba Algorithm

The working of Russian Peasant algorithm is diagramically shown in figure 6. All the calculations are done simultaneously hence the time required for calculations will be comparatively less.

Table 3: Booths Multiplier Example[14][15]

A	Q	Q-1	M	Operation	Cycle
0000	0101	0	0110	Initial values	
1001 1100	0101 1010	0 1	0110 0110	A = A-M Shift	First cycle
1110	0101	0	0110	Shift	Second cycle
0101 0010	0101 1010	0 0	0110 0110	A = A - M Shift	Third cycle
0001	0101	0	0110	Shift	Fourth cycle

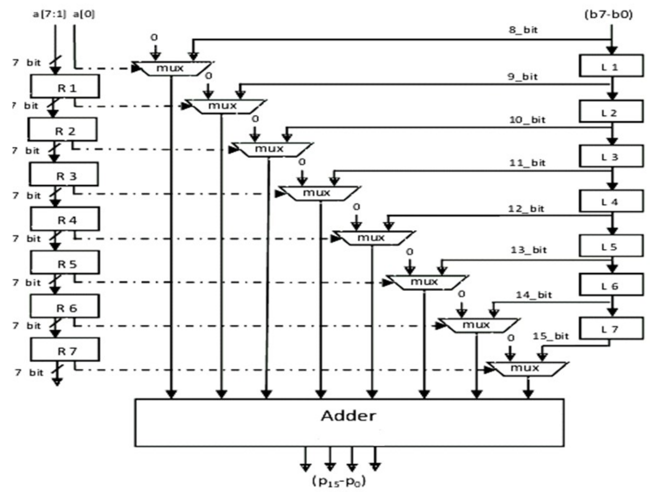


Figure 6: Example of multiplication using Russian Peasant

1.2. Proposed Method 2

Out of the various multipliers designed and developed, the Booth's multiplier is considered to be one of the standard technique used that enhances the operating speed with smaller circuits. This is achieved by using encoding techniques to the signed numbers of 2's complement. The proposed method is based on the same logic to reduce the size of the multiplier and multiplicand into half. This is done by using Karatsuba algorithm. The following example is demonstrated with the Booth algorithm in the above table 6_{10} x 5_{10} = 0110_2 x 0101_2

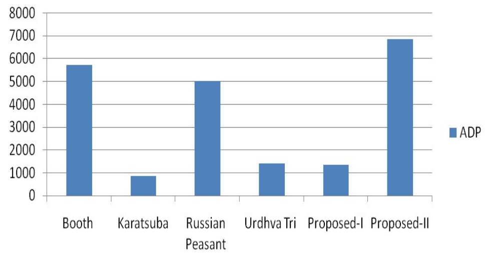
The multiplier and multiplicand can be denoted as M = $(6)_{10}$ and Q = $(5)_{10}$ in binary number system. Three registers, namely A, Q and M each of size 2n +1 bits is required, if data size is n-bits. The initial value stored in A is the most significant bits of M, Q will hold the most significant bits of -M and fill their remaining bits with zero. The two least significant bits of Q determine what operation will be performed on A and M. If these bits are "01" then, find the value of A+M else calculate A - M if they are "10". For "00" or "11" no operation is performed, the values of A are arithmetically shifted as shown in above example. The process is repeated n times and the product is obtained from Q by dropping its least significant bit.

Result

The proposed architectures of multiplier circuits are compared with the existing algorthms. It is found that the second algorithm proposed requires nearly 30% less power as compared to the first method proposed. Area rerequired is 50.75% less in the first proposed method as compared to second method whereas 14% more as compared to Karatsuba Algorithm. The computational delay i.e pin to pin delay on the device is 30.21% less as compared to the first proposed method and 12.8% less in comparision with Booth multiplier which is mostwidely used algorithm for multiplication.

Table 4: Comparison of different Algorithms on Device XC7A35T-CPG236

On-chip	Booth	Karatsuba	Russian Peasant	Urdhva Tri	Proposed-I	Proposed-II
SliceLUT	62	85	55	114	99	108
Slice Reg	20	--	60	--	--	93
Flip-flops	--	--	62	--	--	--
Bonded IOB	34	21	34	33	32	34
Power (W)	8.261	14.313	1.649	13.753	13.545	9.467
Delay (ns)	69.8	15.563	99.517	12.4	11.35	34.017
ADP	5723.6	1322.855	17614.509	1413.6	1340.955	6837.417
PDP (nJ)	401.629	222.753219	164.103533	170.537	153.736	322.039
APP	632.94	1216.605	291.873	1980.432	1340.955	1902.867



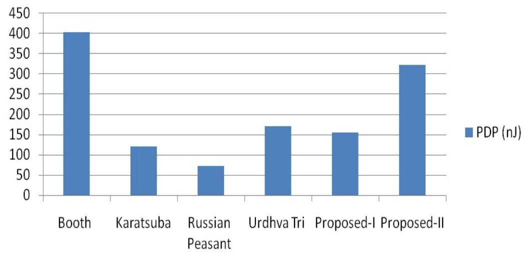
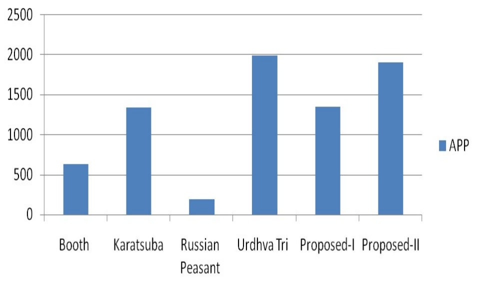


Figure 7: Graphical Comparision of Area Delay Product Performance Parameter and Graphical Comparision of Power Delay Product Performance Parameter



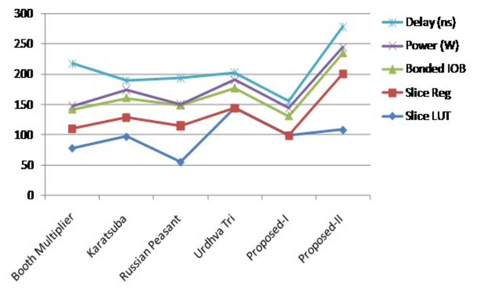


Figure 8: Graphical Comparision of Area Power Product Performance Parameter and Graphical Comparision of Different Performance Parameters Area, Delay and Power

Conclusion

Using Xilinx FPGA, two aalgorithms are proposed to perform a 24*24 significand multiplication for IEEE single-precision numbers. The first method proposed uses less area in terms of LUT's as compared to second method proposed. Also the computational time required to perform the multiplication is nearly one third as that required by the second proposed method. The proposed multiplier is faster than all the other algorithms and also the power delay parameter has proved out to be very good.

Refrences

[1] Shiann-Rong Kuang, Jiun-Ping Wang, and Hua-Yi Hong, "Variable-Latency Floating-Point Multipliers for Low-Power Applications", IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 18, NO. 10, pp. OCTOBER 2010.

[2] Jean-Pierre Deschamps, Gery Jean Antoine Bioul and Gustavo D. Sutter, "Floating Point Unit", in Synthesis of Arithmetic Circuit, Hoboken, New Jersey,A john Wiley & Sons, inc., publication, 2006, pp. 513-548.

[3] S. Sun and J. Zambreno, "A floating-point accumulator for fpga-based high performance computing applications," in Field-Programmable Technology, 2009. FPT 2009. International Conference on, 2009, pp. 493–499.

[4] B. Catanzaro and B. Nelson, "Higher radix floating-point representations for fpga-based arithmetic," in Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on, april 2005, pp. 161 – 170A.

[5] P.L.Montgomery, "Five, Six, and seven-Term Karatsuba-Like Formulae," IEEE Transactions on Computers, Vol. 54, No. 3, pp. 362-69

[6] A. A. Karatsuba: The Complexity of Computations. Proceedings of the Steklov Institute of Mathematics, Vol. 211, 1995, pages 169 - 183, available at http://www.ccas.ru/personal/karatsuba/divcen.pdf. A. A. Karatsuba reports about the history of his invention and describes it in his own words

[7] Julio Villalba, Javier Hormigo, Francisco Corbera, Mario Gonzalez and Emilio L. Zapata Dept. of Computer Architecture, University of Malaga, SPAIN "Efficient Floating-Point Representation for Balanced Codes for FPGA Devices"

[8] J. Villalba and J. Hormigo, "Apendix to paper efficient floatingpoint representation for balanced codes for fpga devices http://www.ac.uma.es/~julio/apendix iccd 2013.pdf."

[9] Xilinx Corporation. http://www.xilinx.com/tools/coregen.htm, 2012

[10] D. W. Bishop, "VHDL-2008 support library," 2011. [Online]. Available: http://www.eda.org/fphdl/

[11] IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, New York: IEEE, Inc., 2008.

[12] Manish Kumar Jaiswal.Ray C. C. Cheung "VLSI Implementation of Double-Precision floating Point Multiplier usingKaratsuba technique".

[13] Karatsuba & Yu. Ofman, Multiplication of Multidigit Numbers on Automata (in Russian), Doklady Akad. Nauk SSSR 145 (1962), pp. 293-294, Englis h translation in Soviet Physics Doklady 7 (1963), pp. 595-596.

[14] John and Earl E. Swartzlander, "Improved Architectures for a Fused Floating-Point Add-Subtract Unit Jongwook"

[15] Gustavo D. Sutter, Enrique canto and Jean-Pierre Deschamps,"Floating Point Arithmetic", in Guide to FPGA Implementation of Arithmetic Functions, springer