IETE 46th Mid Term Symposium "Impact of Technology on Skill Development" MTS- 2015
Special Issue of International Journal of Electronics, Communication & Soft Computing Science and Engineering, ISSN: 2277-9477

Floating-Point Multiplier for DSP Using Vertically and Crosswise Algorithm

Ajay A. Raut **Dr. Pravin K. Dakhole**

***Abstract*— This paper presents a fast IEEE-754 single precision floating point multiplier. Multiplication of two floating point numbers which involves large dynamic range is critical requirement of various DSP applications. Vertically and crosswise algorithm is used for the multiplication of mantissa. The inputs to this multiplier are provided in IEEE-754 single precision binary floating point format and it gives output in double precision format to obtain error free result. All basic modules of this multiplier are written in Verilg HDL and targeted to Vertex-5 Field Programmable Gate Array in Xilinx 13.1 ISE software.**

***Index Terms*— Vertically and Crosswise algorithm, IEEE-754, Floating point multiplier, FPGA.**

I. INTRODUCTION

A considerable amount of processing time is devoted by CPU to perform arithmetic operations. In arithmetic operation, multiplication is an important function because; compared to other arithmetic operations it requires more hardware resources and processing time. Today accuracy is also an important factor along with the speed. Floating point number system is widely used in various scientific areas, since it provides better precision and higher dynamic range compared to any other number system. Many applications such as DSP, Image processing, 3D technology and arithmetic unit in microprocessors/controllers use floating point number system to obtain error free results, and multiplying two floating point numbers which involves large dynamic range is critical requirement of these applications.

Now computers are the essential part of life and as the computers and signal processing applications are expanding, demand of high speed processing is also increasing. Currently the multiplication time is still dominant factor in DSP, and fast multiplier circuit has been the subject of interest over decades [1]. IEEE 754 standard provides two types of format, binary floating point interchange format and decimal floating point interchange format. Here we deal with the binary floating point interchange format. The binary floating point interchange format is represented in three formats, single precision, double precision and extended-double precision with encoding in 32-bit, 64-bit and 128-bits respectively [2]. Each format composed of three fields: sign, exponent and fractional part. Following fig.1 and fig.2 shows the structure of single and double precision format respectively.

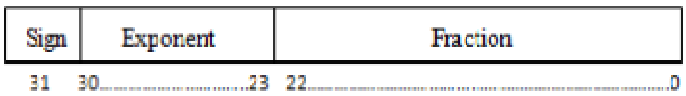

Fig.1: Single precision floating point representation (32-bit).

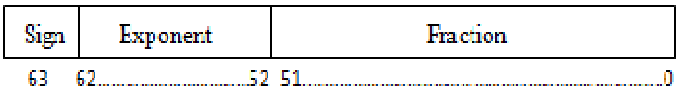

Fig.2: Double precision floating point representation (64-bit).

In case of single precision the exponent comprises of 8-bit, which is represented in excess 127 bit format. The fractional part consists of 23 bits and 1 bit is added to MSB for normalization which is also known as mantissa or significand. In Double precision the exponent is represented by 11-bit which is bias to excess 1023 bit format. The fractional part consists of 52-bit. In both formats the MSB is reserved for sign bit.

As stated earlier sign bit, Exponent and significant are the main constitutional fields of binary floating point number. The multiplication of binary floating point number can be determined as:

a. Sign of result is obtained by XORing the sign bits of two input floating point numbers.
b. Exponent of the result is calculated by simply adding the exponent of inputs. As long as we keep in mind that the exponents are biased, addition of exponents is a trivial solution.
c. The multiplication of significant is nothing but an unsigned, integer multiplication. In this paper, we propose a technique to carry out the multiplication of two floating point numbers using Vertically and Crosswise algorithm which is popular as Urdhva-triyakbhyam.

Vedic mathematics has proved to be the most robust technique for arithmetic capable of solving almost all mathematical problems. Vedic Mathematics is the ancient methodology of Indian mathematics which has a unique technique of calculations based on 16 *Sutras* (Formulae). This system of computation covers all forms of mathematics, be it geometry, trigonometry or algebra [1]. They can even be applicable to complex problems which involves large number of mathematical operations .These formulae form the backbone of Vedic mathematics. Vedic *sutras* can reduce the significant amount of delay in hardware implementation which occurs in conventional multiplication techniques. Urdhva Tiryakbhyam *Sutra* is one of the formulae applicable to all cases of multiplication which means "Vertically and Crosswise" [3]. Vertically and crosswise algorithm is one of the most efficient sutra giving minimum delay and simpler means for multiplication of all types of problems encountered in Engineering environment. On account of this formula, the partial products and sums are generated in one step which reduces the carry propagation from LSB to MSB [4]. Thus the multiplier design based on Vedic algorithm is efficient to use in various DSP applications [5].

II. FLOATING POINT MULTIPLICATION

The floating point number multiplier, represented in IEEE-754 format can be divided in four parts:

1. Sign calculation unit,
2. Exponent calculation unit,
3. Mantissa calculation unit,
4. Control unit.

The standard form of floating point number representation is:

Floating point number = $(-1)^s$ X b^e X m

Where,

a. „s" comprises of single bit representing sign of the floating point number, if „s" is 1 the number is negative and positive if sign bit is 0.

b. „e" represents the exponent of floating point number which comprises of exponent of given number and Bias (E + bias), which varies as encoding format, for e.g. for single precision bias is 127.

c. „m" is a number which is represented by a string of digit of the form d0 • d1 d2…dp−1 where di is an integer digit $0 \leq di < b$ of the floating point number, which is generally known as mantissa or significant.

In this paper we propose the multiplication of Binary floating point numbers hence b = 2.

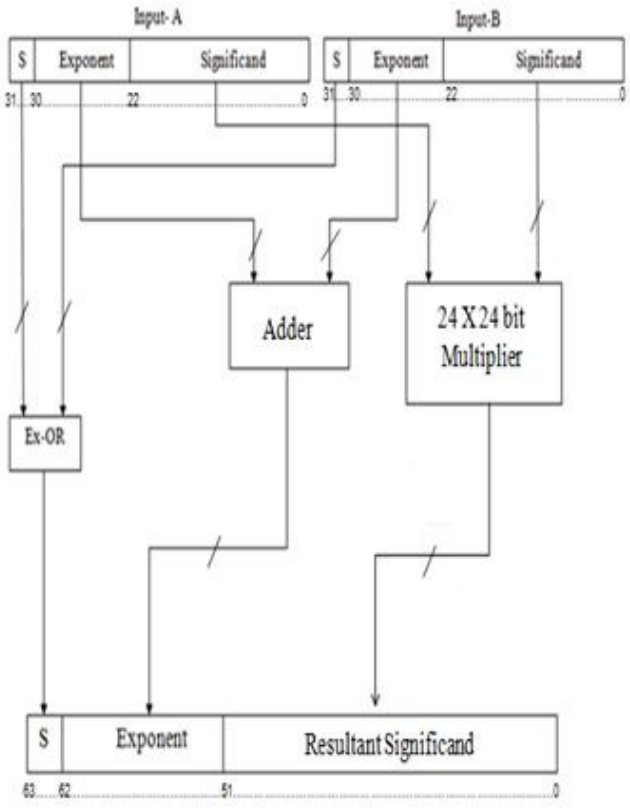

Fig.3 Proposed Architecture of Floating-point multiplier

Fig.3 shows the proposed architecture of floating point multiplier. As explained, the result of two single precision floating point number multiplications is carried out by:

XORing the sign bit of two input numbers. This operation is carried out using sign calculation unit. In this paper the exponent calculation is carried out using Ripple Carry Adder [8]. As the resultant output is in the double precision format, the exponent must be bias to 1023 which is normalized by exponent calculation unit. Ease of implementation, simple layout and low area requirement are the advantages of Ripple Carry Adder for the addition [9], hence it is one of the popular adder. Mantissa calculation unit requires 24X24 bit multiplier for the multiplication of two mantissa fields. Here we propose the efficient use of Vertically and crosswise algorithm of Vedic mathematics for the calculation of resultant multiplication.

The infinity, zero, Not a Number (NaN), underflow, overflow cases are handled by control unit. According to the occurrence of various cases stated, appropriate flag is raised by the control unit. The constituent flags of various cases are as follows:

Zero if e = 0 and zero fraction.

Infinity if e = 2047 and zero fraction.

NaN if e=2047 and non-zero fraction.

If $0 < e < 2047$, then Number is $(-1)^s$ 2^{e-1023} (1·fraction).

If e = 0 and f ≠ 0, then $(-1)^s$ 2^{-1022}(0· fraction) (demoralized numbers).

Consider the multiplication of two numbers X, Y where X= -39 and Y=19.5. The normalized binary floating point representation of these two numbers can be given as X= -

$1.00111X2^5$ and Y= $1.00111X2^4$. IEEE 754 representation of these numbers will be:

X 1 10000100 00111000000000000000000
Y 0 10000011 00111000000000000000000

Here, the MSB represents the sign of respective operand; exponent is of 8-bit and fraction is represented by 23-bits. The exponent is represented in excess 127 bit for ease of calculations. Sign of the result is obtained by XORing the sign bit of operand, which is 1 in this case. Exponent of result is calculated by adding the two exponents of operand, let"s say E_X and E_Y respectively. Since the exponent has to be bias to 1023, addition of 769 is done. If E_Z is the result of addition then

$$E_Z = E_X + E_Y + 769$$

Ripple Carry Adder is used for the addition of exponent as shown in fig.4.

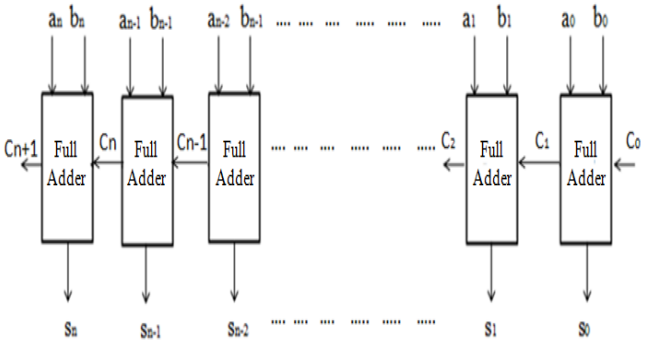

Fig.4 n-bit Ripple Carry Adder

In this case E_Z=10000001000. The 23-bit fraction is converted into mantissa of 24-bit simply by appending 1 at the MSB of fraction. The normalized 24-bit mantissa of operand X and Y can be given as:

100111000000000000000000
100111000000000000000000

The multiplication of above two, 24-bit mantissa is obtained using Vertically and Crosswise algorithm. In this case output of 48-bit in hexadecimal form is obtained by multiplication of mantissa which is

16"hbe20000000000000

Now combining all three intermediate results to get the final result in hexadecimal (Eliminating Most significant bit i.e. 1 for normalizing the mantissa)

c087c40000000000

This result is decoded as
X X Y= -39 X 19.5 = -760.5

$-1.0111110001 \text{ X } 2^{1032-1023} = (-1011111000.1) = (-760.5)$

MULTIPLIER DESIGN

The overall performance of Floating-point multiplier is highly affected by the mantissa calculation unit because it is the only unit which dominates the execution time to obtain result. Vertically and Crosswise algorithm is used to implement this unit which results in minimum delay [1], and simpler means for multiplication of all types of problems encountered in Engineering environment [6]. Implementation of mantissa calculation unit using Vertically and Crosswise algorithm results in simultaneous generation of partial products and sums which itself makes the unit faster and reduces the delay. Following fig. 5 shows the two, 3-bit multiplication based on vertically and crosswise algorithm.

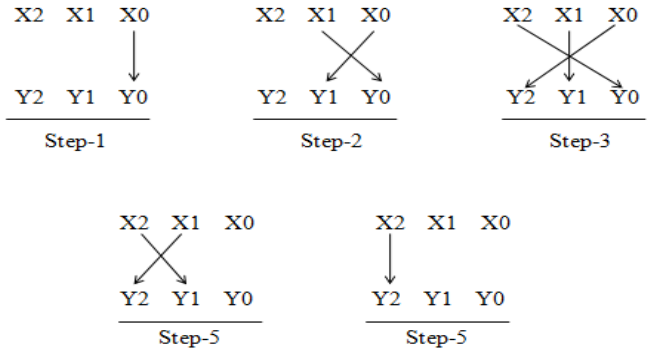

Fig. 5 Implementation of Vertically and Crosswise algorithm

Consider two operands X and Y such that X= $X_2X_1X_0$ and Y=$Y_2Y_1Y_0$.
In step-1 LSB of X is multiplied with LSB of Y which results in $Z_0= X_0 Y_0$.
In next step-2 X_0 is multiplied with Y_1 and X_1 is multiplied with Y_0 and result in added together as;
$C_0Z_1= X_0 Y_1+ X_1 Y_0$;
Where, C_0 is a carry and Z_1 is sum.
In step-3 as shown in fig. 5 multiplication results in;
$C_1Z_2= C_0+X_0 Y_2+ X_1 Y_1+ X_2 Y_0$.
The carry of precious result is added to the next step. Similarly step-4 and step-5 results in equations as follows;
$C_2Z_3= C_1+ X_1 Y_2+ X_2 Y_1$;
$C_3Z_4= C_2+ X_2 Y_2$;
Now, final result of 3X3 bit multiplication is
$C_3Z_4 Z_3 Z_2 Z_1 Z_0$
Above 3X3 bit multiplier is the basic module for construction of required 24X24 bit mantissa multiplication unit. Following fig. 6 shows the hardware realization of 3X3 bit multiplier unit.

For multiplication of two 6-bit operands with the help of 3-bit multiplier, let us consider 6-bit operands denoted as A_HA_L and B_HB_L where, A_H, B_H are 3 MSB bits and A_L,B_L are 3 bits of operand A & B respectively. When the operands are multiplied using vertically and crosswise algorithm, we get,

$$\begin{array}{r} A_H\ A_L \\ B_H\ B_L \\ \hline \end{array}$$

$$(A_HXB_H) + (A_HXB_L + B_HXA_L) + (A_LXB_L)$$

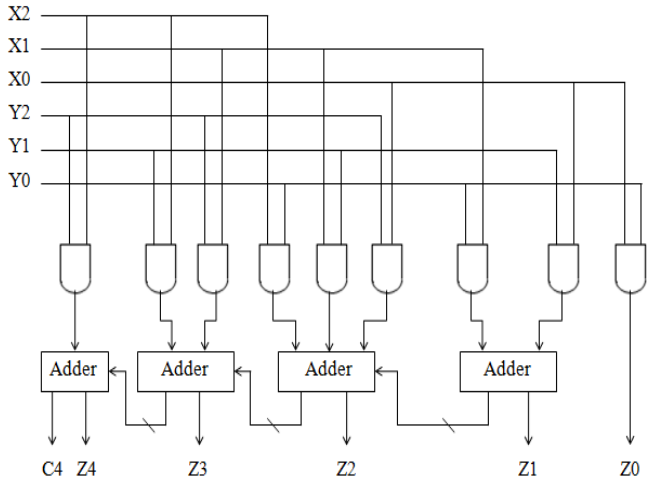

Fig.6 Hardware realization of 3X3 bit Multiplier

Thus to multiply two 6-bit operands there is requirement of four 3X3-bit multipliers and three adder units for addition of partial products. Since the output of 3X3-bit multiplier is 6 bits in length, in every step 3 LSB bits correspond to the product and remaining 3 MSB bits are carried to next step. In this case, above procedure is carried out up to 3 steps. Hence 6X6 multiplier is designed using four 3X3 bit multipliers. Similarly, using four 6X6 multiplier units a 12X12 bit multiplier is designed. Finally required 24X24 bit multiplier is implemented by four 12X12 bit multiplier units and three Carry Ripple Adders. Fig.7 shows the block diagram of 24X24 bit multiplier.

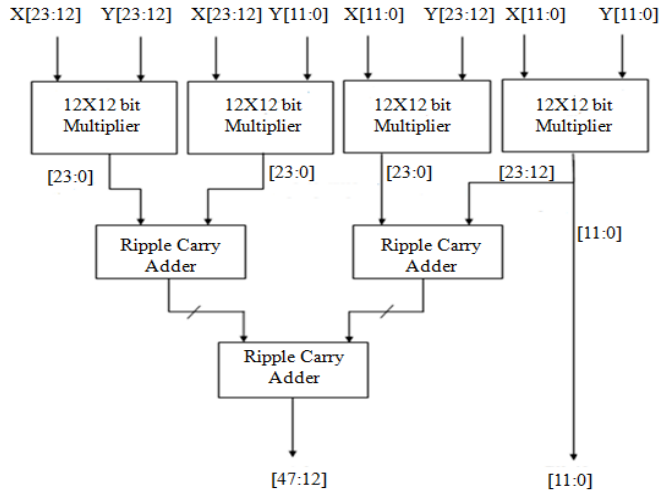

Fig.7 24X24 bit multiplier unit

III. EXPERIMENTAL RESULT

The main objective of this paper is to design the Floating point multiplier using vertically and crosswise algorithm. Initially a basic module of 3X3 bit multiplier is designed based on same algorithm. Following fig. 8 shows the simulation result of the 3X3 bit multiplication module. With the help of above multiplier module, mantissa calculation unit is designed.

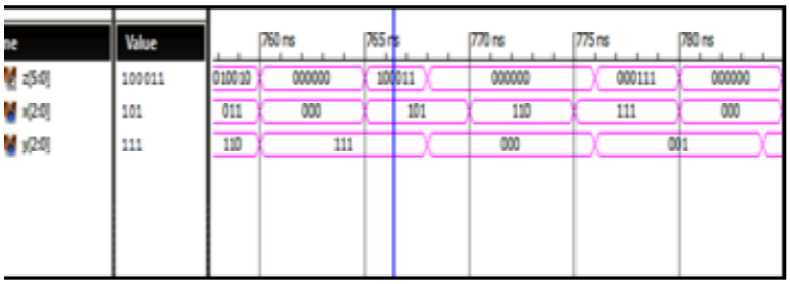

Fig. 8 Simulation result of 3X3 bit multiplier

The multiplier has two operands x and y each of 32-bit. The result is of 64 bit in order to optimize the error. Fig.9 shows the simulation result of single precision floating point multiplier. We take randomly two 32-bit floating point numbers as input and it gives output in 64-bit format.
For example consider two inputs as:

x= 8"hc2890000
y= 8"h429f0000
then output is

z= 16"hc0b545c000000000

Different basic modules such as half adder, full adder, XOR gate etc., are separately written in Verilog HDL. With the help of these basic modules the proposed multiplier is designed. Table 1 shows the resource utilization summary like number of slice LUTs and bonded IOBs, along with percentage utilization. Table 2 shows the timing constrains of the floating point multiplier.

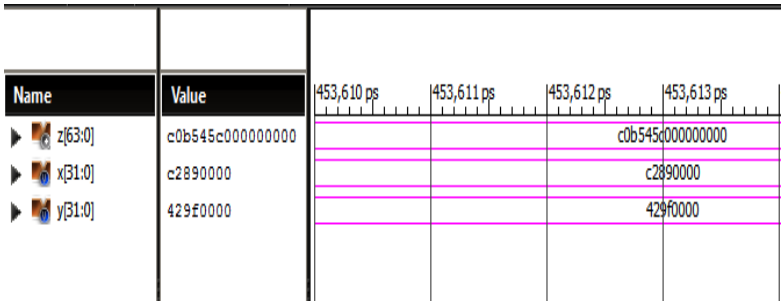

Fig. 9 Simulation result of Floating-point Multiplier

Table 1.Resource Utilization Summary

Logic Utilization	Proposed Multiplier on vertex5	Available resources	Utilization (in %)
Number of Slice LUTS	1083	19200	5%
Number of bonded IOBs	128	220	58%

Table 2.Timing Constrains

Delay Specification (In ns)	Proposed Multiplier on vertex5	Percentage (%)
Logic	5.24	25.3
Route	15.45	74.7

IV. Conclusion And Further Scope

The multiplier is designed in Verilog hardware description language and simulated using Modelsim Simulator. The design was synthesized using Xilinx ISE 13.1 tool targeting the Xilinx Virtex-5 xc5vlx30-3-ff324 FPGA and find out delay and slices utilization. Multiplier operation is verified by generating stimulus with the help of test bench. In order to show underflow and overflow cases the respective flags are incorporated in the design.

The paper presents very fast floating point multiplier unit supporting to IEEE-754 single precision binary floating point format. For reducing delay in calculation the sum and partial products are generated in one step. The speed of multiplication operation can be substantially increase by multiplier unit based of vertically and crosswise algorithm because as already mentioned, multiplication dominates the execution time. The further scope of this particular work is can be extended in design of MAC units and RISC processors.

References

[1] Ashwath M. and Premanand B. S., "Signed Fixed- point Multiplication for DSP using Vertically and Crosswise algorithm", 4th ICCCNT, July2013, DOI:10.1109/ICCCNT.2013.6726694, pp.1-6.

[2] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, Sponsored by the microprocessor standard committee, 2008.

[3] A. Radhika and pavan kumar U.C.S., Saiprasad Goud A., "FPGA implementation of high speed 8-bit vedic multiplier using barrel shifter", International conference on Energy Efficient technologies for sustainability-2013, DOI: 10.1109/ICEETS.2013.6533349, pp.13-17.

[4] Prabir Saha, Arindam Banerjee, Partha Bhattacharyya, Anup Dandapat, "High speed ASIC design of complex multiplier using vedic mathematics" , Proceeding of the 2011 IEEE Students' Technology Symposium 14-16 January, 2011, IIT Kharagpur, pp. 237-241.

[5] Honey Durga Tiwari, Ganzorig Gankhuyag, Chan Mo Kim and Yong Beom Cho, "Multiplier Design based on Ancient Indian Vedic Mathematics", International SoC Design Conference, Nov. 2008, Volume-2, DOI - 10.1109/SOCDC.2008.4815685, pp. 65-68.

[6] P. Mehta, and D. Gawali, "Conventional versus Vedic mathematical method for Hardware implementation of a multiplier," in Proceedings IEEE International Conference on Advances in Computing, Control, and Telecommunication Technologies, Trivandrum, Kerala, Dec. 28-29, 2009, pp. 640-642. [7] M. Pradhan and R. Panda, "Design and Implementation of Vedic Multiplier," A.M.S.E Journal, Series D, Computer Science and Statistics, France, July 2010, Volume-15, Issue 2, pp. 1-19.

[8] M. Al-Ashrafy, A. Salem, W. Anis,"An Efficient Implementation of Floating point multiplier", Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International ,DOI:10.1109/SIECPC.2011.5876905,pp.1-5.

[9] Chetana Nagendra, Robert Michael Owens, and mary Jane Irwin,"Power-Delay Characteristics of CMOSAdders", IEEE Transactions onVery Large Scale Integration (VLSI) Systems, Vol. 2, No. 3, September1994.

[10] Aniruddha Kanhe, Shishir Kumar Das and Ankit Kumar Singh, "Design And Implementation Of Low Power Multiplier Using Vedic Multiplication Technique," International Journal of Computer Science and Communication (IJCSC) Vol. 3, No. 1, January-June 2012, pp. 131-132.

[11] Xilinx 13.1, ISE Design Suite 13: Release Notes Guide, UG631 (v 13.1), March 1, 2011.