ISSN: 2456-043X

International Journal of Engineering Science and Generic Research (IJESAR)
Available Online at www.ijesar.in
Volume 3; Issue 5; September-October; 2017; Page No. 25-30

A Floating Point Multiplier Design Using Karatsuba and Urdhva-Tiryagbhyam Algorithms for High Speed Applications

A. Raviteja[1], [2]A. Jaya Lakshmi

[1] PG Scholar, Vidya Jyothi Institute of Technology, Aziz Nagar, Hyderabad, India.

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

ravitejaamanaganti@gmail.com

Abstract

Floating Point Multiplication is mainly used for high speed operations such as digital image processing and digital signal processing etc. But multiplication is a time taking and power consuming process. In this paper an IEEE754 floating point multiplier is designed and implemented which is efficient in terms of area and delay. For implementing unsigned binary multiplier for mantissa multiplication both Karatsuba and Urdhva- Tiryagbyam which is a sutra of Vedic mathematics is used. The multiplier is implemented using verilog, targeted on Spartan-3E

Keywords: Floating Point Multiplier, Urdhva-Tiryagbyam algorithm and karatsuba algorithm.

1. INTRODUCTION

Floating point multiplication is an essential IP for modern multimedia and high performance operations such as graphics acceleration, signal processing, image processing ect. There are lot of effort is made over the past few decades to improve the performance of floating point computations. Floating point units are not only complex but it requires large area and the accuracy becomes a major issue. IEEE754 support different floating point formats such as Single Precision format, Double Precision format, Quadruple Precision format etc. The combination not only reduces delay but also reduces the percentage of hardware as compared to fixed point multipliers.

IEEE754 format specifies two different formats namely Single Precision and Double Precision formats [1,2]. Fig.1 shows the different IEEE754 floating point formats used commonly. The Single Precision format is 32-bit wide and Double Precision format is of 64-bit wide. The most significant Bit is the Sign bit. The exponent is a singed integer. It is often represented as an unsigned value by adding a bias. In Single Precision format, the exponent is of 8-bit wide and the bias value is 127, i.e.the exponent has a range of (-127to128).In Double Precision format, the exponent is of 11-bit wide and the bais is 1023,i.e. the exponent has the range of (-1023to1024).The mantissa or signified of Single Precision format is of 23-bit and of Double Precision format is 52 bit wide. The maximum value that can be represented using floating point format is show in the figure above.

*Largest significant*base (largest exponent)*

And the minimum value that can be represented is

*Smallest significant*base (smallest exponent)*

Single Precision

1	8	23
Sign	Exponent	Mantissa

Double Precision

1	11	52
Sign	Exponent	Mantissa

Figure 1: Floating Point formats

2. FLOATING POINT MULTIPLIER

A floating point number has four parts

1. Sign
2. Exponent
3. Mantissa and
4 exponent base

A floating point number is represented in IEEE754 format as + .s*b^e .The exponent base for binary format is2.Perform multiplication of two floating point numbers +-s1*$b^{e1}$and +-s2*b^e, the significant or mantissa parts are multiplied to get the product mantissa and exponents are added to get the product exponent i.e.; the product is +-(s1*s2)*b(e^1+e^2).The

Corresponding author: A. Raviteja

hard ware block diagram of floating point multiplier is shown in fig.2.

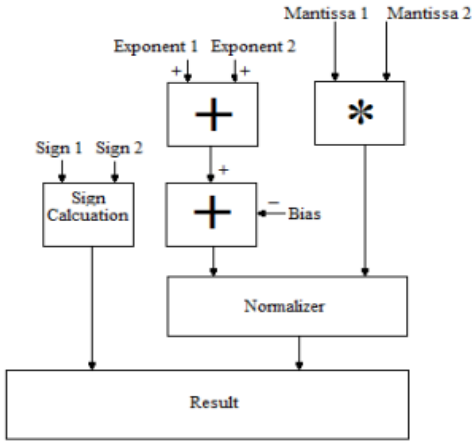

Figure 2: Floating point multiplier

A. Sign calculation

The MSB of floating point number represents the sign bit. The sign of product will be positive if both the numbers are same sign and will be negative if numbers are of opposite sign. So ,to obtain the sign of product, we can use a simple XOR gate as the sign calculation .

B. Addition of Exponents

To get the product exponents, the input exponents are added together. Since we use a bias in floating point format exponent, we need to subtract the bias from the sum of exponents to get the actual exponent. The value of bias is 12710(01111111_2) for single precision format and 1023_{10}(0111111111_2) for double precision format.

The computational time of mantissa multiplication operation is much more that exponent addition. So a simple simple ripple borrow subtract is optimal exponent addition .

C. Mantissa multiplication

In floating point multiplication, most important and complex part is the mantissa multiplication .Multiplication operations require more time compared to addition. And as the number of bits increase, it consumes ore area and time. In the double precision there is a need of 54*54 bit multiplier and in the single precision need of 24*24 bit multiplier .It requires much time to perform the operations and it is the major contributor to delay of floating point multiplier. To make the multiplication efficient in area and delay several method are implemented one of the algorithm is Urdhva – Tiryagbyam algorithm is used but it takes more time and more area for higher level bits and to implement the multiplication faster the proposed method is used i.e. combination of Urdhva-Tiryagbyam and karatsuba algorithm is used.

D. Normalization of the result

Floating point representations have a hidden bit in the mantissa, which always have the value 1and hence it is the not stored in the memory to save one bit. A leading 1 in the mantissa is considered to be the hidden bit, i.e. the 1 just immediate to the left of decimal point. Usually normalization is done by shifting, so that the MSB of mantissa becomes non zero and in radix2, nonzero means1.The decimal point in the mantissa multiplication is shifted left if the leading 1 is not at the immediate left of decimal point. And for each left shift operation of the result, the exponent value is incremented by one. This is called normalization of result. Since the value of hidden bit is always 1, it is called 'hidden 1'.

E. Representation of exceptions

Some of the numbers cannot be represented with a normalized significand. To represent those numbers a special code is assigned to it. In the proposed model, we use four output signals namely zero, infinity, an(not-a- number)and denormal to represent these exceptions. If the product has exponent bias=0 and significand=0,then the result is taken to zero. If the product has exponent bias=255and significand =0,then the result is taken as infinity. If the product has exponent bias=255 and significant is not equal to zero then the result is taken as Nan. De normalization values or Denormals are the numbers without a hidden 1 and with the smallest possible exponent. Denormals are used to represent certain small numbers that cannot be represented as normalized numbers. If the product has exponent +bias =0 and significand is not equal to zero Then the result is represented as Denormal. Denormal represented as $0.s*2^{-126}$, where s is the significant.

3. EXISTING METHOD

4. *Urdhva- Tiryagbhyam algorithm for multiplication*

Urdhva-Tiryagbhyam sutra is an ancient Vedic mathematics method for multiplication [4, 5, 6, and 7]. It is a general formula applicable to all cases of multiplication. The formula is very short and consists of only one compound word and means "Vertically and crosswise". In Urdhva Triyagbhyam algorithm, the number of steps required for multiplication can

be reduced and hence the speed of multiplication is increased.

An illustration of steps for computing the product of two 4-bit numbers is shown below [8, 9]. The two input are $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ and the $p_7p_6p_5p_4p_3p_2p_1p_0$ is the product. And the temporary partial products are t 0t 1t 2t....., t6. The partial products are obtained from the steps given below. The line notation of the steps is shown in Fig.3.

Step1: t_0 (1bit) = a_7b_7.

Step 2: t_1(2bit) = $a_1b_0 + a_0b_1$.

Step3: t_2(2bit) = $a_2b_0 + a_1b_1 + a_0b_2$

Step 4: t_3 (3bit) = $a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3$.

Step 5: t_4(2bit) = $a_3b_1 + a_2b_2 + a_1b_3$.

Step 6: t_5(2bit) = $a_3b_2 + a_2b_3$.

Step 7: t_6 (1bit) = a_3b_3

The product is obtained by adding s_1, s_2 and s_3 as shown below, where s_1 s_2 and s_3 are the partial sum obtained.

$S_1 = t_6t_5$ [0]t_3[0]t_2[0]t_1[0]t0

$S_2 = t_5$[1]t_4]1]t_3[2]t_3[1]t_1[1]

$S_3 = t_3$ [2]

Product = $t_6\, t_5$ [0] t_4[0] t_3[0] t_2[0] t_1[0] t_0 +

t_5 [1] t_4[1] t_3[1] t_2[1] t_1[1] 0 +

t_3 [2] 0 0 0 0 0

P_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0

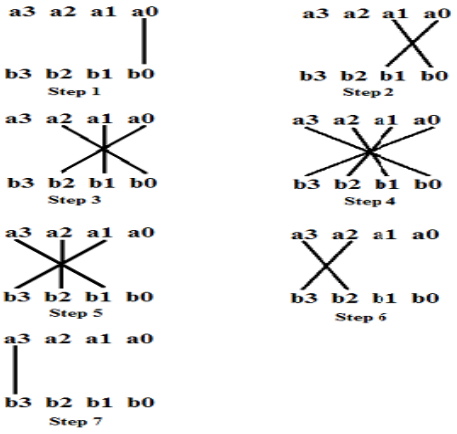

Figure 3: LINE NOTATION OF URDHVA-TIRYAGBYAM

This method can be further optimized to reduce the number of hardware. More optimized hardware architecture [9, 10] is shown in Fig. 3 This model actually helps to eliminate the need for three operand 7-bit adder and hence reduces hardware and delay. The adders are connected in ripple manner. But it requires more time and hard ware structure to implement for higher bit operations.

The expressions for product bits are as shown above

$P_0 = a_0b_0$

P_1 = LSB of (Sum ADDER 1))

=LSB of ($a_1b_0+a_0b_1$)

P_2= LSB of (sum (ADDER 2))

= LSB of (MSB (ADDER1) +$a_2b_0+a_1b_1+a_0b_2$)

P_3= LSB of (sum (ADDER3))

= LSB of (MSB (ADDER2) +$a_3b_0+a_2b_1+a_1b_2+a_1b_3$)

P_4= LSB of (sum (ADDER3))

= LSB of (MSB (ADDER1) +$a_3b_1+a_2b_2+a_1b_3$)

P_5= LSB of (MSB (ADDER1) +$a_3b_1+a_2b_2+a_1b_2$)

= LSB of (MSB (ADDER1) +$a_3b_2+a_2b_3$)

P_6= LSB of (sum (ADDER6))

= LSB of (MSB (ADDER1) +a_3b_3)

P_7=carry

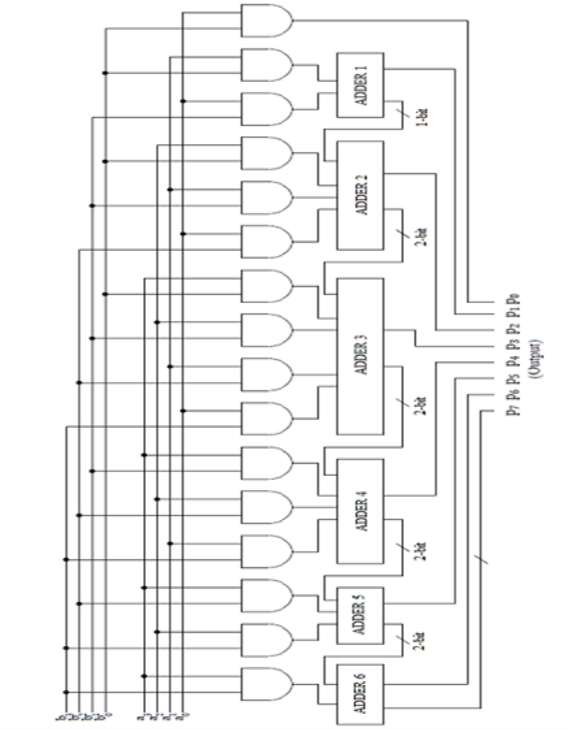

Figure 4: URDHVA-TIRYAGBYAM

Since there are more than two operands in adder 2to5,we can use carry save addition to implement adders 2to5.This technique reduces the delay to a great extent compared to the ripple carry adder

4. PROPOSED METHOD

Karatsuba algorithm for multiplication

Karatsuba multiplication algorithm is best suited for multiplying very large numbers. This method is discovered by multiplying very large numbers. This method is discovered by Anatoli Karatsuba in1962.it divide and conquers method half and least significant half and then multiplication is performed.

Karatsuba algorithm reduces the number of multipliers required replacing multiplication operations by additions operations. Additions operations are faster than multiplications and hence the speed of multiplier is increased. As the number of bits of inputs increase, Karatsuba algorithm becomes more efficient. This algorithm is optimal if the width of inputs is more than 16 bits. The hardware architecture of Karatsuba algorithm is shown in figure.4. Karatsuba algorithm for two inputs

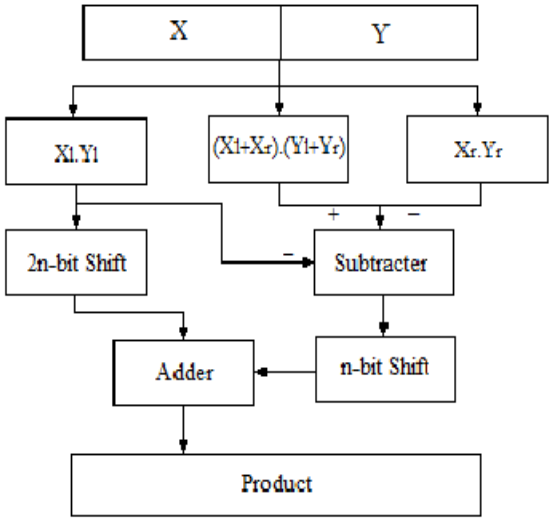

Figure 5: Karatsuba Multiplier

Product =X.Y

X and Y can be written as,

$X=2^{n/2}.X_1 + X_r$ ---------- (1)

$Y=2^{n/2}.Y_1+Y_r$---------- (2)

Where X1,Y1 and X r and Yr are the most significant half and least significant half of X and Y respectively, and n is the number of bits.

Then, $X.Y=(2^{n/2}.X1+Xr).(2^{n/2}.Y1+Yr)$

$=2^n.X1\ Y1+2^{n/2}\ (X1Yr+Xr+Y1)+Xr\ Yr$---------(3)

The second term in equation(3)can be optimized to reduce the number of multiplication operations.

i.e; X1Yr+XrY1=(X1+Xr)(Y1+Yr)-X1Y1-XrYr

the equation (3)can be written re written as,

$X.Y=2^n.X1Y1+XrYr+2^{n/2}(X1+Xr)(Y1+Yr)-X1Y1-XrYr)$

The recurrence of Karatsuba algorithm is

$T(n) =3T(n/2) +0(n^{1.585})$

Karatsuba-Urdhva Tiryagbyam multiplier

The proposed model uses the combination of Urdhva Tiryagbyam and Karatsuba algorithm. Karatsuba algorithm uses a divide and conquers approach where it breaks down the inputs into most significant half and least significant half and the process continues until the operands are 8-bit wide. Karatsuba algorithm is best suited for higher bit operands of higher bit length. But at lower bit lengths it is not efficient as it is in higher bit lengths. To eliminate this problem Urdhva-Tiryagbyam algorithm is used at lower stages. The model of Urdhva-Tiryagbyam algorithm is shown in fig.5

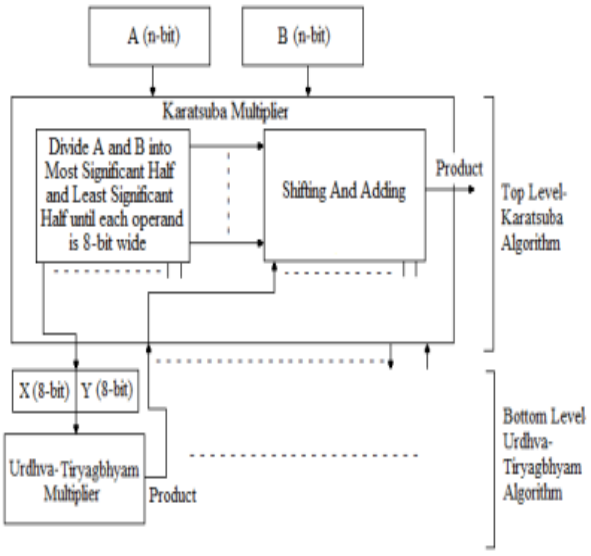

Figure 6: Karatsuba- Urdhva multiplier

Urdhva-Tiryagbyam algorithm is the best algorithm for binary multiplication in terms of area and delay. But as the `number of bits increases, delay also increases as the partial product are added in the ripple manner. For example for 4bit multiplication, it requires 6adders connected in ripple manner. And 8-

bit multiplication requires 14 adders and so on. Compensating the delay will cause increase the area. So Urdhva-Tiryagbyam algorithm is not optimal if the number of bits is much more. If we use Karatsuba algorithm at higher stages and Urdhva-Tiryagbyam algorithm at lower stages, it can somewhat compensate the limitations in both the algorithms and hence the multiplier becomes more efficient. The circuit is future optimised by using carry look ahead adder instead of ripple carry adder. This reduces the delay to a great extent with minimal increase in hardware.

5. KOGGE-STONE ADDER

Kogge-stone adder is a form of carry look ahead adder. Other parallel prefix adders include the Brent-Kung adder, the Han Carlson adder, and the fastest known variation, the Lynch-Swartzlander Spanning Tree adder.[1]

The Kogge–Stone adder takes more area to implement than the Brent–Kung adder, but has a lower fan-out at each stage, which increases performance for typical CMOS process nodes. However, wiring congestion is often a problem for Kogge–Stone adders. The Lynch-Swartzlander design is smaller, has lower fan-out, and does not suffer from wiring congestion; however to be used the process node must support Manchester Carry Chain implementations. The general problem of optimizing parallel prefix adders is identical to the variable block size, multi level, carry-skip adder optimization problem, a solution of which is found in.[2]

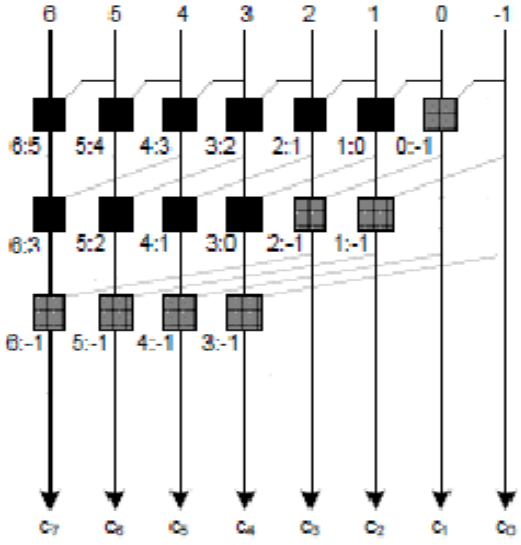

Figure 7: KOGGE_STONE ADDER

An example of a 4-bit Kogge–Stone adder is shown in the diagram. Each vertical stage produces a "propagate" and a "generate" bit, as shown. The culminating generate bits (the carries) are produced in the last stage (vertically), and these bits are XOR'd with the initial propagate after the input (the red boxes) to produce the sum bits. E.g., the first (least-significant) sum bit is calculated by XORing the propagate in the farthest-right red box (a "1") with the carry-in (a "0"), producing a "1". The second bit is calculated by XORing the propagate in second box from the right (a "0") with C0 (a "0"), producing a "0".

6. IMPLEMENTATION AND RESULTS

The main objective of this paper is to design and implement a floating point multiplier which must be efficient in its operation both in terms of delay and area. Since mantissa multiplication is the most complex part of floating point multiplier, so here is a new design multiplier which operate at high speed and increase the delay and area significantly less with increase in number of bits.IEEE-754standard format is implemented using verilog HDL and tested. The multiplier units are future optimized by replacing ripple carry adders by carry look adder(kogge-stone adder).k The model is synthesised and simulated using XILINX synthesis Tools (ISE 14.7) targeted on saprtan-3E and virtex-4. The summary of results on virtex-4 is shown below

TABLE 1: PERFORMANCE OF URDHVA TIRYAGBYAM KARATSUBA ALGORITHM

	Single pression ripple carry adder	Single pression kogge-stone adder
Delay	**81.84ns**	**68.311ns**

TABLE 2: PERFORMANCE OF SINGLE PRESSION RIPPLE CARRY ADDER AND KOGGE-STONE ADDER

	double pression ripple carry adder	double pession kogge-stone adder
Delay	**166.50ns**	**128.312ns**

TABLE 3: PERFORMANCE OF DOUBLE PRESSION RIPPLE CARRY ADDER AND KOGGE-STONE ADDER

	Urdhva algorithm	Karatsuba algorithm
No of slices	1785	1716
No of 4 input LUTS	3018	3058

7. CONCLUSION AND FUTURE SCOPE

Floating multipliers are used for high speed operations like digital image processing and digital signal processing the existing method is not an efferent for high level bits so in order to overcome this problem the proposed method is used. This paper shows how to effectively reduce the percentage increase in delay and area of a floating point multiplier by using a very efficient combination of Karatsuba and Urdhva-Tiryagbyam algorithms. This model can be further optimized in terms of delay by using pipeline methods and precision of the result can be increases by adding efficient truncation and rounding methods

REFFERENCES

1. IEEE 754-2008, IEEE standard for floating point arithmetic, 2008.
2. Computer Arithmetic, Behorroz Parhami, Oxford University press, 2000.
3. B.Jeevan, S.Narender, C.V.KrishnaReddy, K.Sivani,"A High speed Binary Floating point multiplier using Dadda Algorithm", International multi-conference and compressed sensing, pp.455-460, 2013
4. "Vedic mathematics", Swami Sri Bharati krsna Thirthaji maharaja, Motilal Banarasidass Indological publishers and Book sellers, 1965
5. R.Sridevi Anirudh Palakurthi,Akhila Sadhula,Hafsa Mahreen,"Design of Highspeed multiplier(Ancient Vedic mathematics Approach)",international journal of Engineering Research(ISSN:2319-6890),volume No,Issue No3,pp:183-186,july 2013
6. Nivedita A.Pande,Vaishali Niranjane,Anagha V.Choudhari,"Vedic Mathematics for Fast Multiplication in DSP",International Journal of Engineering and Innovative Technology (IJEIT), Volume 2,Issue 8,pp.245-247,February 2013

Author Details

	**A.Raviteja** has completed his B.Tech in Electronics and Communication Engineering from Brilliant 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, 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.