Volume 5, Issue 1, January 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Design & Implementation of High Performance Pipelined Single Precision Floating Point Multiplier Based on Vedic Mathematics in ASIC

[1]Rashmi Rajurkar, [2]P. R. Indurkar, [3]S. R. Vaidya
[1]Department of Communication Engineering, SDCOE, India
[2]Associate Professor, Department of Electronics and Telecommunication Engineering, BDCOE, India
[3]Assistant Professor, Department of Electronics Engineering, SDCOE, India

Abstract— High-speed multiplier is much desired to satisfy overall power budget of digital system. In this paper we describe an efficient implementation of an IEEE 754 pipelined single precision floating point multiplier targeted for Xilinx Virtex-4. VHDL is used to implement a technology-independent pipelined design. Design does not support rounding. Comparison between results we got and previous results shows that implemented work is improved in terms of power consumption and device utilization. ASIC verification is done using Synopsys Design Compiler tool.

Keywords— Floating Point Multiplier, Urdhva-Tiryakbhyam sutra, Vedic mathematics.

I. INTRODUCTION

A multiplier is one of the key hardware blocks in most digital signal processing (DSP) systems. Since multipliers are rather complex circuits and must typically operate at a high system clock rate, reducing delay of a multiplier is an essential part of satisfying the overall design. Multiplication operations require considerable amount of time and slows the overall operation and hence the performance of many computational problems are often dominated by the speed at which a multiplication operation can be executed. With an ever-increasing quest for greater computing power on battery-operated mobile devices, design emphasis has shifted from optimizing conventional delay time area size to minimizing power dissipation while still maintaining the high performance. The low power and high speed multipliers can be implemented with different logic styles and each logic style has its own advantages in terms of speed, power and device utilization [9].

II. VEDIC MATHEMATICS

The word "Vedic" is derived from the word "Veda" which means knowledge. The use of Vedic mathematics reduces the steps of calculation which are required in conventional mathematics. This is so because the Vedic formulae are proved to be based on the natural principles on which the human mind works. Vedic Mathematics is a methodology of arithmetic rules that allow more efficient fast implementation. Vedic mathematics is mainly based on 16 Sutras (Or aphorisms) dealing with various branches of mathematics. The ancient Vedic mathematics Sutra (formula) called Urdhva Tiryakbhyam (Vertically and Cross wise).Sutra which was traditionally used for decimal system in ancient India. The research has also shown that this sutra is very effective to reduce the N×N multiplier structure. Urdhva Tiryakbhyam Sutra is a general multiplication formula and thus applicable to all cases of multiplication. In this, the digits on the two ends of the line are multiplied and the result is added with the previous carry. When there are more lines in one step, all the results are added to the previous carry. The least significant digit of the number thus obtained acts as one of the result digits and the rest act as the carry for the next step and carry is taken to be as zero initially. Let's analyze 4x4 multiplications, say A3A2A1A0 and B3B2B1B0. Following are the output line for the multiplication result, S7S6S5S4S3S2S1S0. Let's divide A and B into two parts, say A3A2 & A1A0 for A and B3B2 & B1B0 for B. Using the Vedic multiplication technique, taking two bit at a time and using 2 bit multiplier block, we can have the following structure for multiplication.

A3A2 A1A0
X B3B2B1B0

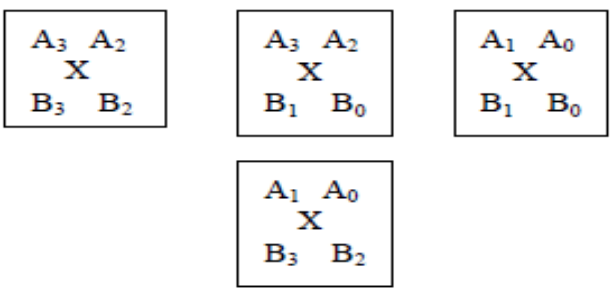

Fig 1. 4x4 Multiplications

So the final result of multiplication, which is of 8 bit, is S7S6S5S4S3S2S1S0.

III. FLOATING POINT NUMBERS

The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point and thus the decimal point can float. Floating point arithmetic is useful in applications where a large dynamic range is required maintaining the precision. Multiplying floating point numbers is a critical requirement for DSP applications involving large dynamic range. Here we will focus on single precision format. The Single precision format consist of 32 bits .The format is composed of 3 fields; Sign, Exponent and Mantissa. Mantissa is represented by 23 bits and 1 bit is added to the MSB for normalization, Exponent is of 8 bits which is biased to 127.The MSB is reserved for sign representation. When the sign bit is 1 that means the number is negative and when the sign bit is 0 that means the number is positive.[11]

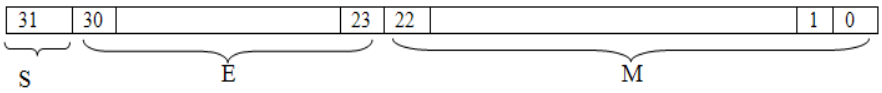

Fig 2 Single precision format

$$Z = (-1S) * 2 (E - Bias) * (1.M)$$

Where M = m22 2-1 + m21 2-2 + m20 2-3+…+ m1 2-22+ m0 2-23; Bias = 127.

IV. FLOATING POINT MULTIPLICATION ALGORITHM

To multiply two floating point numbers the following is done:

1. Multiplying the significand; i.e. (1.M1*1.M2)
2. Placing the decimal point in the result
3. Adding the exponents; i.e. (E1 + E2 – *Bias*)
4. Obtaining the sign; i.e. s1 xor s2
5. Normalizing the result; i.e. obtaining 1 at the MSB of the results' significand
6. Rounding the result to fit in the available bits
7. Checking for underflow/overflow occurrence

Consider the multiplication of two floating point numbers A and B, where A = -19.0 and B = 9.5. The normalized binary representation are A=-1.0011x24 and B = 1.0011x23.

IEEE representations of operands are:

	Sign	Exponent	Mantissa
A =	1	10000011	00110000000000000000000
B =	0	10000010	00110000000000000000000

Here, MSB of the 32 bit operand shows the sign bit, the exponents are expressed in excess 127 bit and the mantissa is represented in 23 bit. Sign of the result is calculated by XORing sign bits of both the operands A and B. In this case sign bit obtained after XORing is 1. Exponents of A and B are added to get the resultant exponent. Addition of exponent is done using 8 bit ripple carry adder. After addition the result is again biased to excess 127 bit Code. For this purpose 127 is subtracted from the result. Two's complement subtraction using addition is incorporated for this purpose. If ER is the final resultant exponent then,

ER = EA + EB - 127

Where EA and EB are the exponent parts of operands A and B respectively. In this case ER = 10000110. Mantissa multiplication is done using the 24 bit Vedic Multiplier. The mantissa is expressed in 23 bit which is normalized to 24 BIT by adding a 1 at MSB. The normalized 24 bit mantissas are

100110000000000000000000

Multiplication of two, 24 bit mantissa is done using the Vedic Multiplier. In this case 48 bit result obtained after the multiplication of mantissa is

101101001000000000000000000000000000000000000000

Now setting up three intermediate results the final result (normalizing the mantissa by eliminating most significant 1) we obtained is:

1 10000110 01101001000000000000000

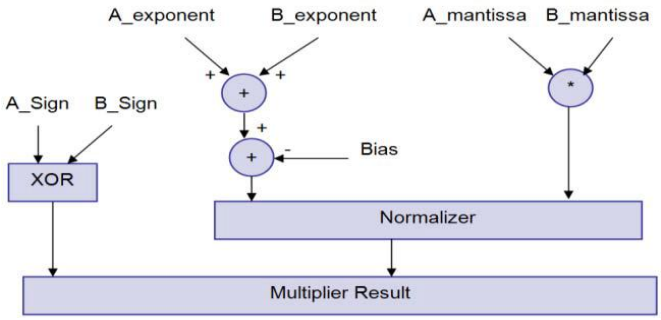

Fig 3. Block diagram of Floating Point Multiplier

V. PIPELINING THE MULTIPLIER

In order to enhance the performance of the multiplier, three pipelining stages are used to divide the critical path thus increasing the maximum operating frequency of the multiplier. The pipelining stages are imbedded at the following locations:

1. In the middle of the significand multiplier, and in the middle of the exponent adder (before the bias subtraction).
2. After the significand multiplier, and after the exponent adder.
3. At the floating point multiplier outputs (sign, exponent and mantissa bits).

Figure 4 shows the pipelining stages as dotted lines.

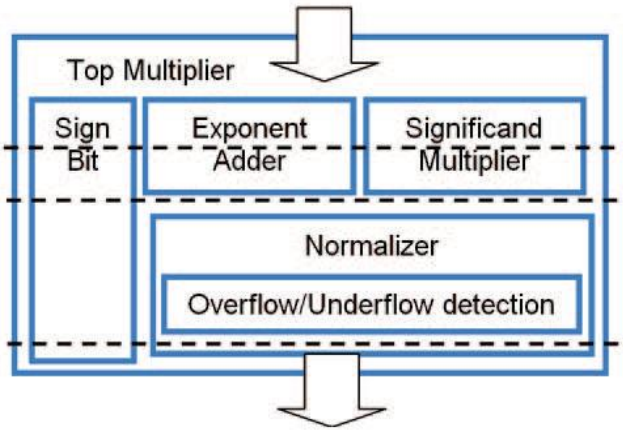

Fig 4. Floating point multiplier with pipelined stages

VI. DESIGN OF FLOATING POINT MULTIPLIER

Sign bit calculation

Multiplying two numbers results in a negative sign number if one of the multiplied numbers is of a negative value.

Unsigned Adder

This unsigned adder is responsible for adding the exponent of the first input to the exponent of the second input and subtracting the Bias (127) from the addition result (i.e. A_exponent + B_exponent - Bias). The result of this stage is called the intermediate exponent. The add operation is done on 8 bits. As there is no need for a fast result because most of the calculation time is spent in the significand multiplication process and so we need a moderate exponent adder and a fast significand multiplier. An 8-bit ripple carry adder is used to add the two input exponents. A ripple carry adder is a chain of cascaded full adders; each full adder has three inputs (A, B, Cin) and two outputs (S, Cout. The carry out (Co) of each adder is fed to the next full adder (i.e each carry bit "ripples" to the next full adder).

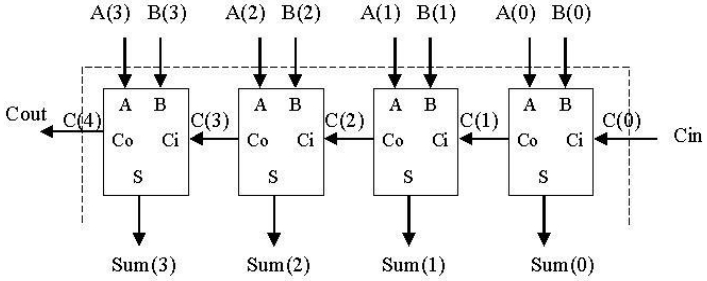

Fig. 5. Ripple Carry Adder

The addition process produces an 8 bit sum (S7 to S0) and a carry bit (Co,7). These bits, by concatenation, form a 9 bit addition result (S8 to S0). The Bias is then subtracted. The Bias is subtracted using an array of ripple borrow subtractors. The bias constant (127 = 001111111) is subtracted from unsigned exponent adder result,two zero subtractors (ZS) and seven one subtractors (OS) are used . S0…..S8 is the unsigned adder result (9 bit) .T=001111111 is the Bias constant. Bias subtractor result is R =S-T.

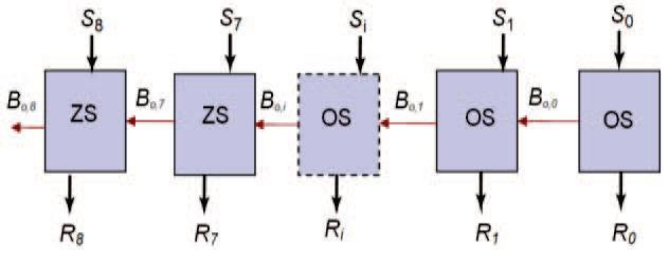

Fig 6. Ripple Borrow Subtractor

Unsigned Multiplier (for mantissa multiplication)

This unit is responsible for multiplying the unsigned significand and placing the decimal point in the multiplication product. The result of significand multiplication will be called the intermediate product (IP). The unsigned significand multiplication is done with vedic multiplier. A 24x24 bit carry save multiplier architecture is used since because of its simple architecture and moderate speed. In the carry save multiplier, the carry bits are passed diagonally downwards (i.e. the carry bit is propagated to the next stage).

Normalizer

The result of the significand multiplication i.e, mantissa multiplication (intermediate product) must be normalized to have a leading '1' just to the left of the decimal point (i.e. in the bit 46 in the intermediate product). Since the inputs are normalized numbers then the intermediate product has the leading one at bit 46 or 47. If the leading one is at bit 46 (i.e. to the left of the decimal point) then the intermediate product is already a normalized number and no shift is needed. If the leading one is at bit 47 then the intermediate product is shifted to the right and the exponent is incremented by 1.

Underflow/Overflow Detection

Overflow/underflow means that the result's exponent is too large/small to be represented in the exponent field. The exponent of the result must be 8 bits in size, and must be between 1 and 254 otherwise the value is not a normalized one. An overflow may occur while adding the two exponents or during normalization. Overflow due to exponent addition may be compensated during subtraction of the bias; resulting in a normal output value (normal operation). An underflow may occur while subtracting the bias to form the intermediate exponent. If the intermediate exponent < 0 then it's an underflow that can never be compensated; if the intermediate exponent = 0 then it's an underflow that may be compensated during normalization by adding 1 to it. When an overflow occurs an overflow flag signal goes high and the result turns to ±Infinity (sign determined according to the sign of the floating point multiplier inputs). When an underflow occurs an underflow flag signal goes high and the result turns to ±Zero (sign determined according to the sign of the floating point multiplier inputs). Denormalized numbers are signaled to Zero with the appropriate sign calculated from the inputs and an underflow flag is raised. Assume that E1 and E2 are the exponents of the two numbers A and B respectively; the result's exponent is calculated as

Eresult = E1 + E2 - 127 (6).

E1 and E2 can have the values from 1 to 254; resulting in Eresult having values from -125 (2-127) to 381 (508-127); but for normalized numbers, Eresult can only have the values from 1 to 254. Table I summarizes the Eresult, different values and the effect of normalization on it.

TABLE I NORMALIZATION EFFECT ON RESULT'S EXPONENT AND OVERFLOW/UNDERFLOW DETECTION

Eresult	Category	Comments
-125 ≤ Eresult < 0	Underflow	Can't be compensated during normalization
Eresult = 0	Zero	May turn to normalized number during normalization (by adding 1 to it)
1 < Eresult < 254	Normalized number	May result in overflow during
255 ≤ Eresult	Overflow	Can't be compensated

VII. IMPLEMENTATION AND TESTING

The whole design is implemented, synthesized for VHDL and simulated on Isim simulator integrated in ISE Xilinx 12.2. Simulation based verification is done for functional verification of a design. Simulation based verification ensures that the design is functionally correct when tested with a given set of inputs. Verification is done target devices XC4VSX25-12FF668.

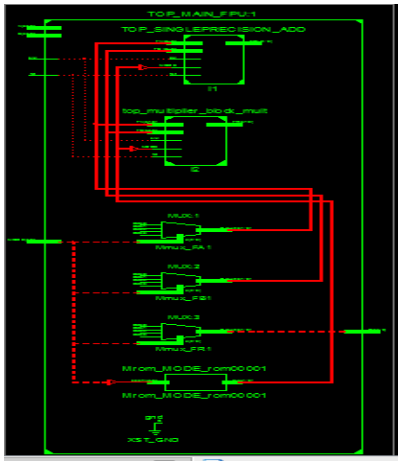

Fig 7. RTL schematic of top module

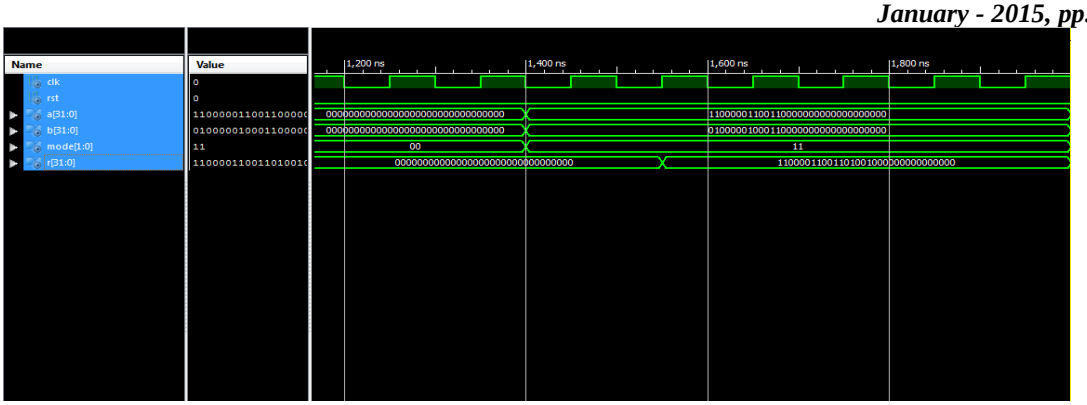

Fig 8. Simulation result of multiplier

VIII. CONCLUSION AND RESULT

The table II shows comparison between the implemented multiplier tested and reference no.[2]

TABLE III COMPARISON OF RESULTS

Parameter	[2]	Implemented work
Device	Vertex 4	Vertex 4
Number4 input LUT's	4,455	4,312
Number of occupied Slices	2,358	2,285
Number of bonded IOBs	100	100
Total Power	0.412 W	0.366W

From comparison it is clear that implemented multiplier is improved in terms of device utilization and power requirement. It offers 88.83% reduction in power consumption.

The same architecture is also verified using Synopsys Design Compiler tool for ASIC level verification. The table III summaries the results obtained from Synopsys Design Compiler tool.

TABLE IIIII SUMMARY OF RESULTS

Data arrival time	5.24 ns
Total Dynamic Power	38.2189 mW
Cell Internal Power	19.5239 mW
Net Switching Power	18.6950 mW
Total cell area	197046.762588μm^2

This paper presents an implementation of a pipelined single precision floating point multiplier based on vedic mathematics that supports the IEEE 754 binary interchange format. The design can be extended for Double Precision format. Multiplier doesn't implement rounding and can be used to significantly reduce power dissipation for applications that do not require correctly rounded results. It will give better precision if the whole 48 bits are utilized in another unit; i.e. a floating point adder to form a MAC unit.

ACKNOWLEDGMENT

I would like to thanks the anonymous reviewers for their insightful comments

REFERENCES

[1] Mohamed Al-Ashrafy, Ashraf Salem, Wagdy Anis, "*An Efficient Implementation of Floating Point Multiplier*", 978-1-4577-0069-9/11/$26.00 ©2011 IEEE .

[2] Korra Tulasi Bai, J. E. N. Abhilash, "*A New Novel Low Power Floating Point Multiplier Implementation Using Vedic Multiplication Techniques*" ,International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 Vol. 3, Issue 4, Jul-Aug 2013.

[3] Dinesh Kumar and Girish Chander Lall, "*Simulation And Synthesis Of 32-Bit Multiplier Using Configurable Devices*", International Journal of Advances in Engineering & Technology, Jan. 2013. ©IJAET ISSN: 2231-1963

[4] Syed Shahzad Hussain Shah, Muhammad Naseem Majoka, and Gulistan Raja, "*Design And Implementation Of 32-bit Vedic Multiplier On Fpga*", First International Conference on Modern Communication & Computing Technologies (MCCT'14).

[5] S. Kokila, Ramadhurai.R, L.Sarah, *" VHDL Implementation of Fast 32X32 Multiplier based on Vedic Mathematics"* , International Journal of Engineering Technology and Computer Applications Vol.2, No.1, Apr 2012.

[6] Addanki Purna Ramesh, Rajesh Pattimi, *"High Speed Double Precision Floating Point Multiplier*", International Journal of Advanced Research in Computer and Communication Engineering ,Vol.1, Issue 9, November 2012.

[7] Aniruddha Kanhe, Shishir Kumar Das shisdas ,Ankit Kumar Singh, *"Design and Implementation of Floating Point Multiplier based on Vedic Multiplication Technique"* , 2012 International Conference on Communication, Information & Computing Technology (ICCICT), Oct. 19-20, Mumbai, India .

[8] Sumit R. Vaidya, Deepak R. Dandekar , "*A Hierarchical Design Of High Performance 8×8 Bit Multiplier Based On Vedic Mathematics*", Icccs2011.

[9] P. Saha, A. Banerjee, A. Dandapat, P. Bhattacharyya , " *Vedic Mathematics Based 32-Bit Multiplier Design for High Speed Low Power Processors"*, International Journal On Smart Sensing And Intelligent Systems Vol. 4, No. 2, June 2011

[10] Kavita Khare, R.P. Singh and Nilay Khare, *"Comparison of pipelined IEEE 754 standard floating point multiplier with unpipelined multiplier"*, Journal of Scientific and Industrial Research Vol 65,November 26.

[11] IEEE 754-2008, *IEEE Standard for Floating-Point Arithmetic*, 2008

