IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 4 Issue 5, May 2017
ISSN (Online) 2348 – 7968 | Impact Factor (2016) – 5.264
www.ijiset.com

Implementation of a High Speed Binary Floating point Multiplier Using Dadda Algorithm in FPGA

Ms.Komal N.Batra[1], Prof. Ashish B . Kharate [2]

[1]PG Student, ENTC Department, HVPM'S College of Engineering and Technology, Amravati, India

[2] Asso. Professor, ENTC Department, HVPM'S College of Engineering and Technology, Amravati ,India

Abstract

In Digital Signal Processing, Floating Point (FP) Multiplication is widely used in large set of scientific and signal processing computation. Multiplication is one of the common arithmetic operations in these computations. Most of the DSP applications need floating point numbers multiplication. The possible ways is to represent real numbers in binary floating point numbers format. The IEEE 754 standard represents two floating point formats, Binary interchange format and Decimal interchange format resp. The main object of this paper is to increase the speed of execution for multiplying two floating point number using FPGA.

Floating Point (FP) Multiplication is widely used in large set of scientific and signal processing computation. This paper describes an implementation of a Floating point multiplier using Dadda Multiplier that supports the IEEE 754-2008 format. To improve speed multiplication of mantissa is done using Dadda Multiplier replacing Carry save multiplier. The multiplier is more precise because it doesn't implement rounding and just presents the significand multiplication result as is (48 bits).The Significand multiplication time is reduced by using Dadda Algorithm. The Floating point multiplier is developed to handle the underflow and rounding cases. The binary floating point multiplier is to be implementing using VHDL and it is simulated and synthesized by using Modelism and Xilinx ISE software respectively.

.

The aim of this paper is to develop an efficient Floating point multiplier algorithm based on computational approaches, with more efficiency.

.

Keywords: *Single Precision and double precision, Dadda Multiplier, Floating point, VHDL, FPGA, Modelsim, Xilinx ISE, Digital signal processing, IEEE Standard 754.*

Introduction

Most of the DSP applications need floating point numbers multiplication. The real numbers represented in binary format are known as floating point numbers. Based on IEEE-754 Standard, floating point formats are classified into binary and decimal interchange formats. Floating point multipliers are very important in DSP applications.

Floating point numbers are one possible way of representing real numbers in binary format; the IEEE 754 Standard presents two different floating point formats, Binary interchange format and Decimal interchange format. Multiplying floating point numbers is a critical requirement for DSP applications involving large dynamic range. This paper focuses only on single precision normalized binary interchange format. It consists of a one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction (M or Mantissa). An extra bit is added to the fraction to form what is called the significant. If the exponent is greater than 0 and smaller than 255, and there is 1 in the MSB of the significant then the number is said to be a normalized number. Multiplying two numbers in floating point format is done by adding the exponent of the two numbers then subtracting the bias from their result, and multiplying the significant of the two numbers, and calculating the sign by XOR'ing the sign of the two numbers.

Floating Point Representation

There are several mechanisms by which strings of digits can represent numbers. In common mathematical notation, the digit string can be of any length, and the location of the radix point is indicated by placing an explicit "point" character (dot or comma) there. If the radix point is not specified, then it is implicitly assumed to lie at the right (at the least significand) end of the string (that is, the number is an integer). In fixed point systems, some specific assumption is made about where the radix point is located in the string; For example, the convention could be that the string consists of 8 decimal digits with the decimal point in the middle, so that "00012345" represents the value 1.2345.

In scientific notation, the given number is scaled by a power of 10, so that it lies within a certain range—typically between 1 and 10, with the radix point appearing immediately after the first digit. The scaling factor, as a power of ten, is then indicated separately at the end of the number. For example, the revolution period of Jupiter's moon Io is 152853.5047 seconds, a value that would be represented in standard form scientific notation as

$$1.528535047 \times 10^5 \text{ seconds.}$$

IEEE 754 floating point standard is the most common representation today for real numbers on computers. The IEEE (Institute Of Electrical And Electronics Engineers) has produced a standard to define floating –point representation and arithmetic. Although there are other representation used for floating point numbers. The standard brought out by the IEEE come to be known as IEEE 754.It is interesting to note that the string of significant digits is technically termed the mantissa of the number, while the scale factor is appropriately called the exponent of the number. The general form of the representation is

$$(-1)^S * M * 2^E$$

Where S represents the sign bit, M represents the mantissa and E represents the exponent. When it comes to their precision and width in bits, the standard defines two groups: base and extended format. The basic format is further divided into Single –Precision format with 32-bits wide, and double-precision format with 64-bits wide. The three basic components are the sign, exponent, and mantissa.

IEEE 754 Floating Point Formats:

Most of the DSP applications need floating point multiplication. The possible ways to represent real numbers in binary format floating point numbers are; the IEEE 754 standard represents two floating point formats, Binary interchange format and Decimal interchange format.

IEEE 754 specifies following formats for representing floating-point values:
1. Single precision (32-bit)
2. Double precision (64-bit)

Single Precision floating point Numbers:

Single precision normalized binary interchange format is implemented in this design. Representation of single precision binary format is shown in Figure 1; starting from MSB it has a one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction (M or Mantissa). Adding an extra bit to the fraction to form and is defined as significand1. If the exponent is greater than 0 and smaller than 255, and there is 1 in the MSB of the significant then the number is said to be a normalized number; in this case the real number is represented by-

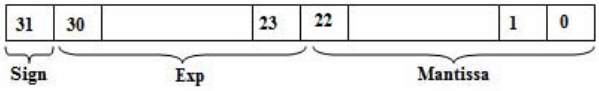


Figure 1 IEEE single precision floating point format

Z = (-1S) * 2 (E - Bias) * (1.M) (1)
Where M = n22 2-1 + n21 2-2 + n20 2-3+…+ n1 2-22+ n0 2-23;
Bias = 127.
1Significant is the extra MSB bit appended to mantissa

Double Precision floating point Numbers:

The double precision number is 64-bit wide. The double precision number has three main fields that are sign, exponent, and mantissa. The 52-bit mantissa, while an 11-bit exponent to an implied base of 2 provides a scale factor with reasonable range. Thus a total of 64 bits is needed for single-precision number representation. To achieve this, a bias equal to 2n-1-1 is added to the actual exponent in order to obtain the stored exponent. This is equal 1023 for an 11-bit exponent of the double-precision format. The addition of bias allows the use of an exponent in the range from -1023 to +1024, corresponding to a range of 0-2047 for double precision. . The double precision format offers the range from 2-1023 to 2+1023. This is equivalent to 10-308 to 10+308

Sign: 1-bit wide and used to denote t he sign of the number i.e., 0 indicate positive number, 1 represent negative number.

Exponent: 11-bit wide and signed exponent in excess -1023representations.

Mantissa: 52-bit wide and fractional component**.**

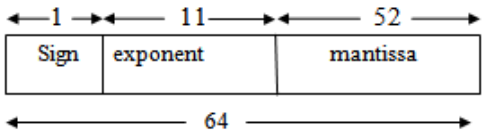


Figure 2 Double-precision floating point representation

The multiplier was verified against Xilinx floating point multiplier core. In this project representation of floating point multiplier in such a way that rounding support is not implemented, thus accommodating more precision. Exponent's addition, significant multiplication, and Results sign calculation are independent and are done in parallel. Xilinx ISE Design Suite 13.3 tool & VHDL programming is used. ISIM tool is used for Simulation process. Xilinx core generator tool is used to generate Xilinx floating point multiplier core. The whole multiplier (top unit) was simulated against the Xilinx floating point multiplier core generated by Xilinx core generator.

Literature review

Various researches have been done to increase the performance on getting best and fast multiplication result on two floating point numbers. Some of which are listed below-

Addanki Puma Ramesh, A. V. N. Tilak, A.M.Prasad [1] the double precision floating point multiplier supports the LEEE-754 binary interchange format. The design achieved the increased operating frequency. The implemented design is verified with single precision
floating point multiplier and Xilinx core, it provides high speed and supports double precision, which gives more accuracy compared to single precession. This design handles the overflow, underflow, and truncation rounding mode resp.
Itagi Mahi P and S. S. Kerur [2] ALU is one of the important components within a computer processor. It performs arithmetic functions like addition, subtraction, multiplication, division etc along with logical functions. Pipelining allows execution of multiple instructions simultaneously. Pipelined ALU gives better performance which will evaluated in terms of number of clock cycles required in performing each arithmetic operation. Floating point representation is based on IEEE standard 754. In this paper a pipelined Floating point Arithmetic unit has been designed to perform five arithmetic operations, addition, subtraction, multiplication, division and square root, on floating point numbers. IEEE 754 standard based floating point representation has been used. The unit has been coded in VHDL. The same arithmetic operations have also been simulated in Xilinx IP Core Generator.

Remadevi R [3] Multiplying floating point numbers is a critical requirement for DSP applications involving large dynamic range. This paper presents design and simulation of a floating point multiplier that supports the IEEE 754-2008 binary interchange format, the proposed multiplier does not implement rounding and presents the significant multiplication result. It focuses only on single precision normalized binary interchange format. It handles the overflow and underflow cases. Rounding is not implemented to give more precision when using the multiplier in a Multiply and Accumulate (MAC) unit.

Rakesh Babu, R. Saikiran and Sivanantham S [4] A method for fast floating point multiplication and the coding is done for 32-bit single precision floating point multiplication using Verilog and synthesized. A floating point multiplier is designed for the calculation of binary numbers represented in single precision IEEE format. In this implementation exceptions like infinity, zero, overflow are considered. In this implementation rounding methods like round to zero, round to positive infinity, round to negative infinity,
round to even are considered. To analyse the working of our designed multiplier we designed a MAC unit and is tested.

Reshma Cherian, Nisha Thomas, Y.Shyju [5] Implementation of Binary to Floating Point Converter
using HDL. Implemented a binary to floating point converter which is based on IEEE 754 single precision format. The unit had been designed to perform the conversion of binary inputs to IEEE 754 32 bit format, which will be given as inputs to the floating point adder/sub block.

Sunil Kumar Mishra, Vishakha Nandanwar, Eskinder Anteneh Ayele, S.B. Dhok[6] FPGA Implementation of Single Precision Floating Point Multiplier using High Speed Compressors. For Mantissa calculation, a 24×24 bit multiplier has been developed by using these compressors. Owing to these high speed compressors, the proposed multiplier obtains a maximum frequency. The results obtained using the proposed algorithm and implementation is better not only in terms of speed but also in terms of hardware used.

Gargi S. Rewatkar [7] Implementation of Double Precision Floating Point Multiplier in VHDL
. Implemented double precision floating point multiplier in VHDL. Double precision floating point multiplier implemented in VHDL may be used applications such as

digital signal processors, general purpose processors and controllers and hardware accelerators.

These results are compared with the previous work done by various authors.

Proposed work

In this paper we implemented a single precision floating point multiplier with exceptions and rounding. Figure shows the multiplier structure that includes exponents addition, significant multiplication, and sign calculation.

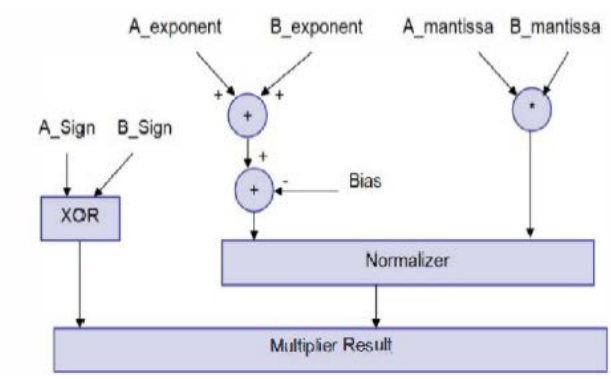


Figure 3 Multiplier structure

1. Exponent

The exponent field represents the exponent as a biased number. It contains the actual exponent plus 127 for single precision, or the actual exponent plus 1023 in double precision. This converts all single precision exponents from -127 to 127 into unsigned numbers from 0 to 254, and all double precision exponents from -1023 to 1023 into unsigned numbers from 0 to 2046. Two Examples shown below for single precision If the exponent is 4, the e-field will be 4+127=132 (100000112). If the e-field contains 8'b01011101 (9310) the actual exponent is 93-127 = 34 Storing a biased exponent means we can compare IEEE values as if they were signed integers.

2. Mantissa

The field f contains a binary fraction. The actual mantissa of the floating-point value is (1 + f). In other words, there is an implicit 1 to the left of the binary point. For example, if f is 01101…, the mantissa would be 1.01101…There are many ways to write a number in scientific notation, but there is always a unique normalized representation, with exactly one non-zero digit to the left of the point. 0.232 * 103 = 23.2 * 101 = 2.32 * 102 = …A side effect is that we get a little more precision: there are 24 bits in the mantissa, but we only need to store 23 of them.

3. Sign

The sign bit is 0 for positive numbers and 1 for negative numbers. But unlike integers, IEEE values are stored in signed magnitude format.

4. Normalizing

The result of the significant multiplication (intermediate product) must be normalizing. Having a leading '1' just immediate to the left of the decimal point is known as a normalized number.

The normalized floating point numbers have the form of $Z= (-1S) * 2 (E - Bias) * (1.M)$. The following algorithm is used to multiply two floating point numbers.

1. Significant multiplication; i.e. (1.M1*1.M2).
2. Placing the decimal point in the result.
3. Exponent's addition; i.e. (E1 + E2 - Bias).
4. Getting the sign; i.e. s1 xor s2.
5. Normalizing the result; i.e. obtaining 1 at the MSB of the results significant.
6. Rounding implementation.
7. Verifying for underflow/overflow occurrence.

Implementation

Dadda Multiplier:

Dadda proposed a sequence of matrix heights that are predetermined to give the minimum number of reduction stages. To reduce the N by N partial product matrix, dadda Multiplier develops a sequence of matrix heights that are found by working back from the final two-row matrix. In order to realize the minimum number of reduction stages, the height of each intermediate matrix is limited to the least integer that is no more than 1.5 times the height of its successor. The process of reduction for a dadda multiplier is developed using the following recursive algorithm

1. Let d1=2 and dj+1 = [1.5*dj], where dj is the matrix height for the jth stage from the end. Find the smallest j such that at least one column of the original partial product matrix has more than dj bits.
2. In the jth stage from the end, employ (3, 2) and (2, 2) counter to obtain a reduced matrix with no more than dj bits in any column.
3. Let j = j-1 and repeat step 2 until a matrix with only two rows is generated.

This method of reduction, because it attempts to compress each column, is called a column compression technique. Another advantage of utilizing Dadda multipliers is that it utilizes the minimum number of (3, 2) counters. {Therefore, the number of intermediate stages is set in

terms of lower bounds: 2, 3, 4, 6, 9 . . .For Dadda multipliers there are N2 bits in the original partial product matrix and 4.N-3 bits in the final two row matrix. Since each (3, 2) counter takes three inputs and produces two outputs, the number of bits in the matrix is reduced by one with each applied (3, 2) counter therefore}, the total number of (3,2) counters is #(3, 2) = N2 – 4.N+3 the length of the carry propagation adder is CPA length = 2.N–2.

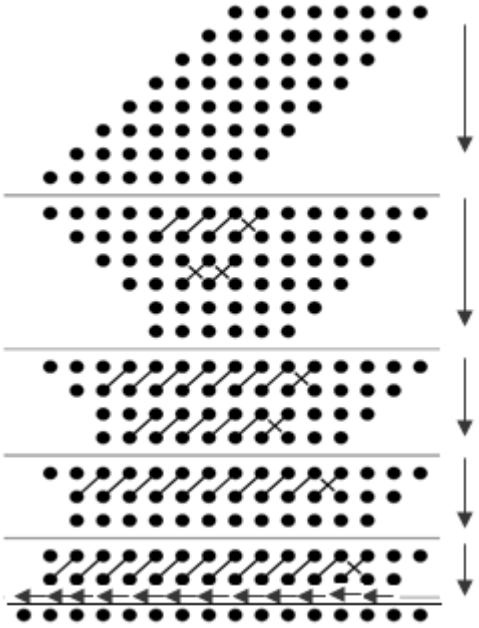


Figure 4 Dot diagram for 8 by 8 Dadda Multiplier

The number of (2, 2) counters used in Dadda's reduction method equals N-1.The calculation diagram for an 8X8 Dadda multiplier is shown in figure 9. Dot diagrams are useful tool for predicting the placement of (3, 2) and (2, 2) counter in parallel multipliers. Each IR bit is represented by a dot. The output of each (3, 2) and (2, 2) counter are represented as two dots connected by a plain diagonal line. The outputs of each (2, 2) counter are represented as two dots connected by a crossed diagonal line. The 8 by 8 multiplier takes 4 reduction stages, with matrix height 6, 4, 3 and 2. The reduction uses 35 (3, 2) counters, 7 (2, 2) counters, reduction uses 35 (3, 2) counters, 7 (2, 2) counters, and a 14-bit carry propagate adder. The total delay for the generation of the final product is the sum of one AND gate delay, one (3, 2) counter delay for each of the four reduction stages, and the delay through the final 14-bit carry propagate adder arrive later, which effectively reduces the worst case delay of carry propagate adder. The decimal point is between bits 45 and 46 in the significant IR. Critical path is used to determine the time taken by the Dadda multiplier. The critical path starts at the AND gate of the first partial products passes through the full adder of the each stage, then passes through all the vector merging adders. The stages are less in this multiplier compared to the carry save multiplier and therefore it has high speed than that.

Example

Let's suppose a multiplication of 2 floating-point numbers A and B, where A=-18.0 and B=9.5

• Binary representation of the operands:

A = -10010.0

B = +1001.1

• Normalized representation of the operands:

A = -1.001x2^4

B = +1.0011x2^3

• IEEE representation of the operands:

A = 1 10000011 00100000000000000000000

B = 0 10000010 00110000000000000000000

(a) Multiplication of the mantissas:

• We must extract the mantissas, adding an1 as most significant bit, for normalization

10010000000000000000000

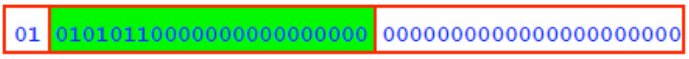
10011000000000000000000

• The 48-bit result of the multiplication is:

0x558000000000

• Only the most significant bits are useful: after normalization (elimination of the most significant 1), we get the 23-bit mantissa of the result. This normalization can lead to a correction of the result's exponent

• In our case, we get:



(b) Addition of the exponents:

• Exponent of the result is equal to the sum of the operands exponents. A 1 can be added if needed by the normalization of the mantissas multiplication (this is not the case in our example)

• As the exponent fields (Ea and Eb) are biased, the bias must be removed in order to do the addition. And then, we must to add again the bias, to get the value to be entered into the exponent field of the result (Er):

Er = (Ea-127) + (Eb-127) + 127

= Ea + Eb – 127

• In our example, we have:

```
Ea    10000011
Eb    10000010
-127  10000001
Er    10000110
```

What is actually 7, the exponent of the result

(c) Calculation of the sign of the result:

• The sign of the result (Sr) is given by the exclusive-or of the operands signs (Sa and Sb):
Sr = Sa □ Sb
• In our example, we get:
Sr = 1 □ 0 = 1
i.e. a negative sign
• Composition of the result: the setting of the 3 intermediate results (sign, exponent and
Mantissa) gives us the final result of our multiplication:

```
1 10000110 01010110000000000000000
```

```
AxB = -18.0x9.5
```

$$= -1.0101011x2^{134-127}$$

$$= -10101011.0 = -171.0_{10}$$

Results

Using Modelsim, simulation is done for Floating Point Multiplication.
In the above analysis two Floating Point Numbers in decimal form are selected and their equivalent IEEE 32 bit binary format is formed resp. Now those 32bit binary formats are to be converted into its equivalent Hexadecimal form as shown in the above e.g. At every rising applied positive edge of the clock output z becomes 0, then result becomes active low and then x and y is computed resp. Here the 2 inputs x and y in their hexadecimal format are applied and the corresponding output z is provided as shown.

x = 0xCC697C1, y = 0x8AC7C609 Output z=17B63413

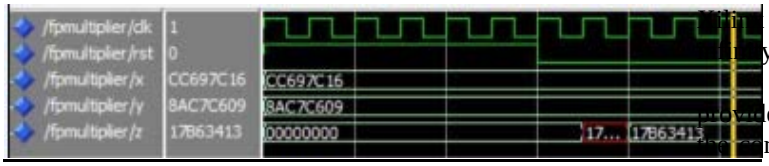


Figure 5 Showing Simulation Results

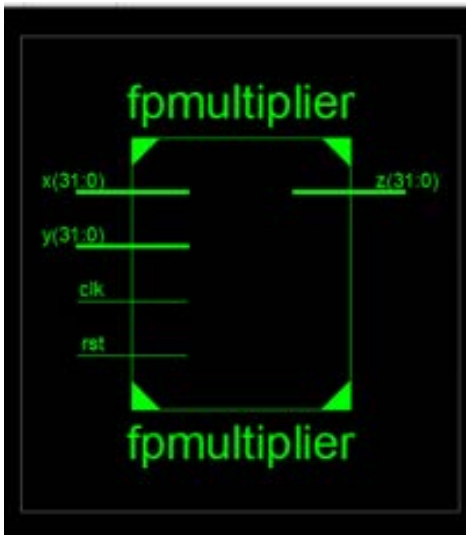


Figure 6 RTL view of Dadda Multiplier

Table 1: Time and Frequency summary of single precision floating point multiplier

Min Propagation delay(ns)	Max Frequency(MHz)
25.096ns	39.847MHz

.

Conclusion

A floating point multiplier is designed for the calculation of binary numbers represented in single precision IEEE format is designed in this project. The above designed floating point multiplier supports the IEEE-754 binary interchange format, targeted on a Xilinx Virtex-6 in FPGA. The design achieved the high operating frequency. The implemented design is verified with single precision floating point multiplier.

It provides high speed and supports double precision, which gives more accuracy compared to single precession. This design handles the overflow, Underflow, and truncation rounding mode. The speed of execution for multiplying two floating point number is increased using FPGA. This multiplier is designed and simulated using ISE v 13.4. In this implementation exceptions like y, zero, NaN, overflow are considered

The design is verified and Simulation results are ded within the thesis. This design is synthesized and rresponding results are also provided. These results are compared with the previous work done by various

authors. Maximum frequency of the design is 39.847MHz and the Maximum period is 25.096ns.

Acknowledgment

We are very grateful to our college of HVPM College of Engineering and Technology to support and other faculty and associates of ENTC department who are directly & indirectly helped me for these paper.

References

[1]. Addanki Puma Ramesh, A. V. N. Tilak, A.M.Prasad, "An FPGA Based High Speed IEEE-754 Double Precision Floating Point Multiplier using Verilog", 978-1-4673-5301-4/13/2013 IEEE.

[2]. Itagi Mahi P and S. S. Kerur, "Design and Simulation of Floating Point Pipelined ALU Using HDL and IP Core Generator", ISSN 2277 – 4106 ©2013 INPRESSCO.

[3]. Remadevi R, "Design and Simulation of Floating Point Multiplier Based on VHDL", Vol.3, Issue 2, March -April 2013.

[4]. A. Rakesh Babu, R. Saikiran and Sivanantham S, "Design of Floating Point Multiplier for Signal Processing Applications", ISSN 0973-4562 Volume 8, Number 6 (2013).

[5]. Gargi S. Rewatkar, "Implementation of Double Precision Floating Point Multiplier in VHDL", Volume.1, Issue 1, April 2014 (IJIREC).

[6]. P.Gayatri, P.Krishna Kumari, V.Vamsi Krishna, T.S.Trivedi, V.Nancharaiah, "Design of Floating Point Multiplier Using Vhdl", Volume 10, Issue 3 (March-2014), IEEE.

[7]. W. Kahan "IEEE Standard 754 for Binary Floating-Point Arithmetic,"1996

[8]. Michael L. Overton, "Numerical Computing with IEEE Floating Point Arithmetic," Published by Society for Industrial and Applied Mathematics,2001.

[9]. D. Narasimban, D. Fernandes, V. K. Raj , J. Dorenbosch , M. Bowden, V. S. Kapoor, "A 100 MHz FPGA based floating point adder", Proceedings of IEEE custom integrated circuits conference,1993.

[10] Jim Hoff; "A Full Custom High Speed Floating Point Adder" Fermi National Accelerator Lab, 1992.

[11]. Subhash Kumar Sharma, Himanshu Pandey, Shailendra Sahni,Vishal Kumar Srivastava, "Implementation of IEEE_754 Addition and Subtraction for Floating Point Arithmetic Logic Unit", Proceedings of International Transactions in Material Sciences and Computer,pp.131-140,vol.3,No.1,2010.

[12]. Shaifali, Sakshi, " FPGA Design of Pipelined 32-bit Floating Point Multiplier", International Journal of Computational Engineering & Management, Vol. 16, 5th September 2013.