ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Implementation of Single Precision Floating Point Multiplier Unit Using Verilog HDL

J.Ranjani
Department of Electronics & Communication Engineering, St. Mary's College of Engineering & Technology, Hyderabad, Telangana-502 319, India.

K.Aksa Rani
Department of Electronics & Communication Engineering, St. Mary's College of Engineering & Technology, Hyderabad, Telangana-502 319, India.

M.S.Shyam
Department of Electronics & Communication Engineering, St. Mary's College of Engineering & Technology, Hyderabad, Telangana-502 319, India.

ABSTRACT:
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. Shows the IEEE 754 single precision binary format representation; 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 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. Multiplying two numbers in floating point format is done by 1- adding the exponent of the two numbers then subtracting the bias from their result, 2- multiplying the significant of the two numbers, and 3- calculating the sign by XORing the sign of the two numbers. In order to represent the multiplication result as a normalized number there should be 1 in the MSB of the result (leading one). Floating-point implementation on FPGAs has been the interest of many researchers.

Keywords:
Floating point number, normalization, exceptions, overflow, underflow, etc.

I. INTRODUCTION:
Floating point numbers are one possible way of representing real numbers in binary format; the IEEE 754 [1] 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. Fig. 1 shows the IEEE 754 single precision binary format representation; 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 significand1. If the exponent is greater than 0 and smaller than 255, and there is 1 in the MSB of the significand then the number is said to be a normalized number; in this case the real number is represented by (1)

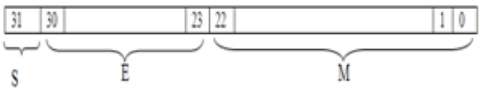


Figure 1. IEEE single precision floating point format

$Z = (-1S) * 2 (E - Bias) * (1.M)$__
Where M = m22 2-1 + m21 2-2 + m20 2-3+…+ m1 2-22+ m0 2-23;
Bias = 127.

Cite this article as: J.Ranjani, K.Aksa Rani & M.S.Shyam, "Implementation of Single Precision Floating Point Multiplier Unit Using Verilog HDL", International Journal & Magazine of Engineering, Technology, Management and Research, Volume 5, Issue 1, 2018, Page 90-94.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Multiplying two numbers in floating point format is done by 1- adding the exponent of the two numbers then subtracting the bias from their result, 2- multiplying the significand of the two numbers, and 3- calculating the sign by XORing the sign of the two numbers. In order to represent the multiplication result as a normalized number there should be 1 in the MSB of the result (leading one). Floating-point implementation on FPGAs has been the interest of many researchers. In [2], an IEEE 754 single precision pipelined floating point multiplier was implemented on multiple FPGAs .

II. FLOATING POINT MULTIPLIER UNIT:

In this project we present a floating point multiplier in which rounding support isn't implemented. Rounding support can be added as a separate unit that can be accessed by the multiplier or by a floating point adder, thus accommodating for more precision if the multiplier is connected directly to an adder in a MAC unit. Fig. 2 shows the multiplier structure; Exponents addition, Significand multiplication, and Result's sign calculation are independent and are done in parallel. The significand multiplication is done on two 24 bit numbers and results in a 48 bit product, which we will call the intermediate product (IP). The IP is represented as (47 downto 0) and the decimal point is located between bits 46 and 45 in the IP. The following sections detail each block of the floating point multiplier

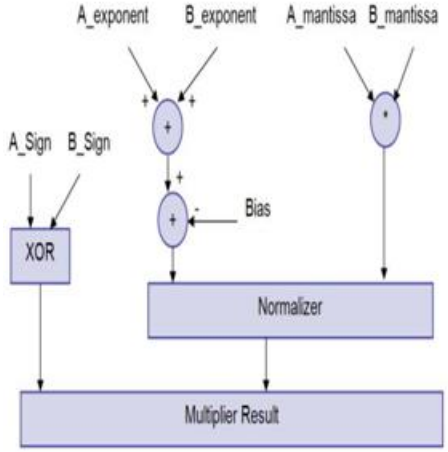


Figure 2. Floating point multiplier block diagram

1) Algorithm:

Figure 2 shows a algorithm flow chart for multiplier. The mantissa of two numbers are multiplied, and the exponent are added. For floating-point multiplication, a easy algorithm is proposed

1. Add the exponents portion and subtract bias portion.
2. Multiply the mantissas portion and calculate the sign bit.
3. The output will be normalized to the prefer number of bits.

2) Multiplier Flow:

Multiplication of floating point number can be carried out in 3 parts [5] In the I st part, the sign product will be perform a XOR operation. In the 2nd part, the exponent bits operands are passed to an adder stage and a bias 127 is subtracted from the output. 8-bit kogge-stone adder is used for implement the addition and 2s complement addition for subtraction operations.

Sign Bit Calculation:

Multiplying two numbers results in a negative sign number If one of the multiplied numbers is of a negative value. By the aid of a truth table we find that this can be obtained by XORing the sign of two inputs.

Koggestone Adder:

Kogge stone adder [7] is a related prefix form of carry look ahead adder. It create the carry in a logarithmic order. since logarithmic order it fastest adder when compared to other and also taking extra area but has lesser fan out at each stage which make better performance of adder. Order of kogge stone adder is 0 (logn) [8].Figure 3 shows the structure of KoggeStone adder.

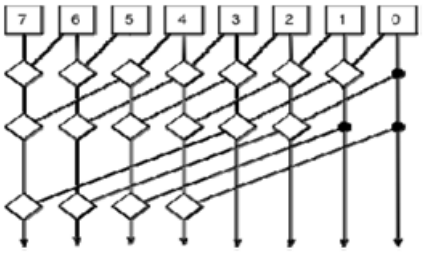


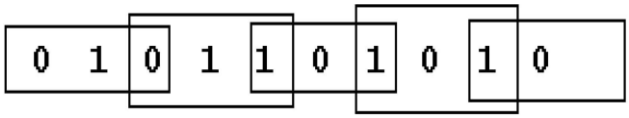
Figure 3: Koggestone adder

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

In the 3rd stage, find the product of the mantissa portion and the multiplication of mantissa portion is carry out in the following steps.

A. Partial product generator:

For a given multiplier [6] there are many ways to generate partial products. The radix-4 booth programming was found to be quicker in which we had found out, so it will be put into operation in the final multiplier architecture. Twelve partial products are the output of this stage. Radix 4 booth encoder To recode the terms, divide it into block of three and in that every one block overlaps the prior block by one bit. The bits are grouped from the LSB, and I st block only takes 2 bits of the multiplier for grouping (no prior block to overlap): Two bits the multiplier have been in use by the least significant block, and consider a 0 for the third bit.



B. Partial result accumulator:

The partial result obtained from the proposed multiplier will be used in the Wallace tree structure which is 4:2compressors. The propagation carry time is less in 4:2 compressor technique, used in carry save adders.

Wallace tree structure:

Partial results is added in Wallace tree which is a new technique to propagate the carry that is ;sed in the carry save adders [4]. The 4:2 compressor structures compresses five partial products bits into three. Figure 4 shows the block diagram for the data distribution among a tree architecture that utilizes 4:2 compressors.

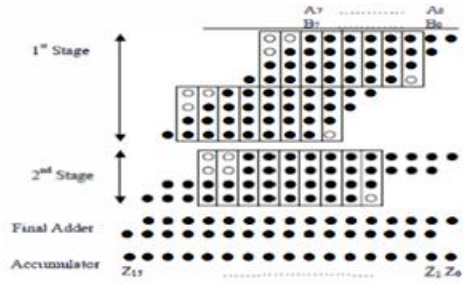


Figure 4.Data distribution among tree architecture

Each packet consists of the bits that fed into a 4:2 compressor group. The partial products result can be reduce by ratio of 2: 1 for two stages of 4:2 compressors. Figure 5 show the cutback tree of 8 partial products to form two operands, added in to form a final product by using a speedy carry propagate adder.

C. Final stage adder:

The products of mantissas are specified by the 48-bit sum and carry outputs obtained from the partial product are added in the final stage adder. This stage adders should have a small amount delay and high speed. After research, comparing and implementing the power and delay uniqueness of various adders, we found out that the KoggeStone adder is the fastest among of all the adder.

D. Normalization and rounding:

The product of mantissas is normalized and round off. The excess one is detected and the exponent is adjusted for normalization. We are reducing the implied bit which is foremost one [3]. The left over bits are reduced to a 26-bit value. For precision a few extra bits is added to the reduced value. The reduced value is finally rounded off using the rounding to nearest value to give the 23 bit mantissa of the product. A zero detect block is used in the multiplier architecture to avoid unnecessary calculations of zero in the input.

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.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

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 by (6)

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.

III. 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).

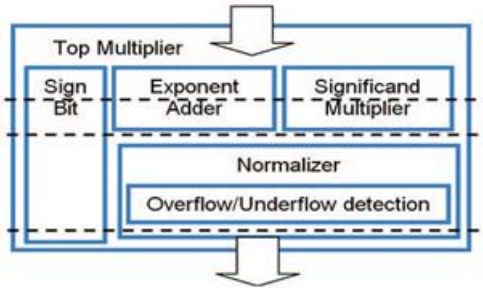


Figure 5 shows the pipelining stages as dotted lines.

Three pipelining stages mean that there is latency in the output by three clocks. The synthesis tool "retiming" option was used so that the synthesizer uses its optimization logic to better place the pipelining registers across the critical path.

IV. RESULTS:

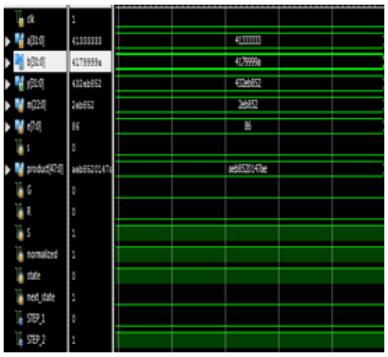


Figure 6. Simulation results for Floating point multiplication

Here
a=11.2(41333333(Hexadecimal)) and
b=15.6(4179999a(Hexadecimal) then result
y=174.72(432EB852(Hexadecimal).

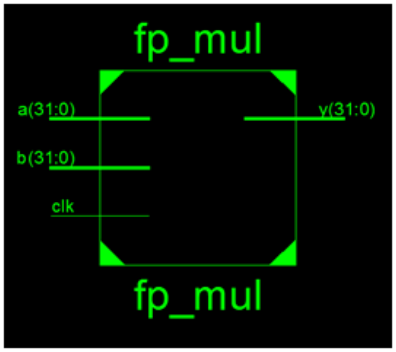


Figure 7 Top module Schematic of FPMUL

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

V. CONCLUSION:

This paper presents an implementation of a floating point multiplier that supports the IEEE 754-2008 binary interchange format; the multiplier doesn't implement rounding and just presents the significand multiplication result as is (48 bits); this gives better precision if the whole 48 bits are utilized in another unit; i.e. a floating point adder to form a MAC unit. The design has three pipelining stages and after implementation.

REFERENCES:

[l] Anna Jain, Baisakhy Dash, Ajit Kumar Panda,"FPGA Design of a Fast 32-bit Floating Point Multiplier Unit", International Conference on Devices Circuits and Systems(lCDCS)-2012.

[2] BJeevan, S.Narender, Dr.C.V Krishna Reddy, K.Sivani, "A high speed binary floating point multiplier using Dadda algorithm" Automation Computing , Communication, Control and Compressed sensing - International Multi Conference -2013

[3] Mohammed AI Ashrafy, Asharf Salem, Wagdy Anis, "An efficient implementation of floating point multiplier" Electronics, Communications and Photonics Conference (SIECPC)-2012

[4] Jung-Yup Kang, Jean-Luc Gaudiot, "A Simple High Speed Multiplier" IEEE Transactions On Computers, Vol. 55, No. 10, October 2006

[5] S.Paschalakis, P.Lee, "Double Precision Floating-Point Arithmetic on FPGAs", In Proc. 2003, 2nd IEEE International Conference on Field Programmable Technology (FPT '03), Tokyo, Japan, Dec. 15-17, pp.352-358, 2003.

[6] Hamacher, Carl, Vranesic, Zvonko, Zaky, SafWat, "Computer Organization" Fifth Edition, pp. 367-390.

[7] Neil H Weste, David Harris,Ayan Bamujee, "CMOS VLSI DESIGN", Pearson Education,3'd Edition,2009

[8] Gong Renxi, Zhang Hainan, Meng Xiaobi,Gong Wenying, "Hardware Implementation of a High Speed Floating Point Multiplier Based on FPGA,"2009 4th International Conference on Computer Science & Education.