INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 5, Issue 1 NOV 2015

AN OPTIMIZED IMPLEMENTATION OF IEEE-754 FLOATING POINT MULTIPLIER FOR DSP APPLICATIONS

CH.VENKATA CHARY[1] Venkatachary21ch@gmail.com[1]

P.V.VARA PRASAD RAO[2] varaprasad.puli@gmail.com[2]

[1]PG Scholar, Dept of ECE, SLC's Institute of Engineering and Technology, Hayathnagar, Rangareddy, Telangana.

[2]Associate Professor, HOD, Dept of ECE, SLC's Institute of Engineering and Technology, Hayathnagar, Rangareddy, Telangana.

***Abstract*:** Arithmetic circuits form an important class of circuits in digital systems. With the remarkable progress in the very large scale integration (VLSI) circuit technology, many complex circuits, unthinkable yesterday have become easily realizable today. Algorithms that seemed impossible to implement now have attractive implementation possibilities for the future. In this paper an arithmetic unit based on IEEE standard for floating point numbers has been implemented on FPGA Board. Here FPU follows IEEE double precision format. The arithmetic unit implemented has a 64-bit processing unit which allows various arithmetic operations such as, Addition, Subtraction, Multiplication and Division on floating point numbers. Each operation can be selected by a particular operation code. We see that the overhead for double precision is less than that for single precision. The unit comprises of rounding and exception unit as specified in format. The FPU design achieved the operating frequency of 107MHz.

Keywords- Double precision, Floating point, FPGA, IEEE-754 Standard, Rounding Unit.

I. INTRODUCTION

Floating point number system is a common choice for many scientific computations due to its wide dynamic range feature. For instance, floating point arithmetic is widely used in many areas, especially in scientific computation, numerical processing, image processing and signal processing. The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point, that is, the decimal point or binary point can float. There are also representations in which the number of digits before and after the decimal or binary point is fixed; called fixed-point representations. The advantage of floating-point representation over fixed point representation is that it can support a much wider range of values. The floating point numbers is based on scientific notation. A scientific notation is just another way to represent very large or very small numbers in a compact form such that they can be easily used for computations. The floating point multiplication operations are greatly affected by how the floating point multiplier is designed.

Floating point number consists of three fields:

1. Sign (S): It used to denote the sign of the number i.e. 0 represent positive number and 1 represent negative number.
2. Significand or Mantissa (M): Mantissa is part of a floating point number which represents the magnitude of the number.
3. Exponent (E): Exponent is part of the floating point number that represents the number of places that the decimal point (binary point) is to be moved.

The IEEE 754 standard provides the format for representation of binary floating point numbers [7]. The Binary Floating point numbers are represented in Single and Double formats. The Single consist of 32 bits and the Double consist of 64 bits. The Fig1 shows the structure of Single and Fig2 shows

the structure Double formats of IEEE 754 standard. The formats are composed of 3 fields; Sign, Exponent and Mantissa. For single precession format, the Mantissa is represented in 23 bits and 1bit is added to the MSB for normalization, Exponent is represented in 8 bits which is biased to 127 and MSB of Single is reserved for Sign bit. For Double precision format, the Mantissa is represented in 52 bits, the Exponent is represented in 11 bits which is biased to 1023 and the MSB of Double is reserved for sign bit. For both 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.

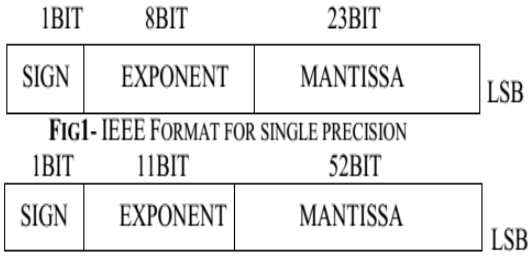


Fig1-IEEE Format for double precision

Double Precision:

In case of double precision 64 bits format, Mantissa is represented in 52 bits, 1 bit is added to the MSB for normaliz ation, Exponent is represented in 11 bits which is biased to 1023 and MSB of double precision is reserved for sign bit as shown
in Figure 1

The value of the floating point number represented in double precision format is

$$F= (-1)^S 1.M\ 2^{E-1023}$$

where 1023 is the value of bias in double precision data format. Exponent E ranges between 1 to 2046, and E = 0 and E = 2047 are reserved for special values. The double precision format offers range from $2^{-1023}$to 2^{+1023}, which is equivalent 10^{-308}to10^{308}.

II.Floating Point Multiplier

Multipliers are key components of many high performance systems such as FIR filters, Microprocessors, Digital Signal Processors etc. System ′ s performance is generally determined by the performance of the multiplier, because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a critical issue for an effective systems design.

A. Floating Point Multiplication Algorithm

Multiplication of floating point numbers F1 and F2 is a seven step process. The algorithm of IEEE compatible floating point multiplier is listed below. To multiply two floating point numbers the following is done:

1. Obtaining the sign; i.e. S1 xor S2.
2. Adding the exponents and subtracting the bias value; i.e. (E1 + E2 – Bias).
3. Multiplying the significand; i.e. (1.M1*1.M2).
4. Placing the decimal point in the significand result.
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.

*Pipelined Double Precision Floating Point Multiplier***:**

The aim in developing a floating point multiplier was to be pipeline that is each module in order to produce result at every clock cycle. In order to enhance the performance of the multiplier, three pipelining stages are used to divide the critical path thus increasing the operating fr equency of the multiplier. As the number of pipeline stages is increased, the path delays of each stage are decreased and the overall performance of the circuit is improved. By pipelining the floating point multiplier, the

speed increased, however, the area increased as well. Different coding structures were tried in VHDL code in order to minimize size. Multiplying two numbers in floating point format is done by three main module i.e. multiplier module, rounding module and exceptions module. Figure 2 shows the Pipelined floating point multiplier with rounding and exceptions module. The multiplier, rounding and exceptions module that are implement independent and are done in parallel. All the modules in the FPM are realized using VHDL. In this paper, pipelined floating point multiplier structure organized in a three-stage. Stage 1 performs the addition of the exponents, the multiplication of the mantissas, and the exclusive-or of the signs. Stage 2 takes these results from stage 1 as inputs, performs the rounding operations. Stage 3 checks the various exceptions conditions like overflow, underflow, invalid and inexact.

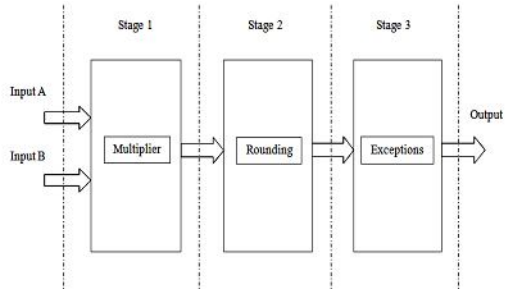


Figure 2. Pipelined Floating Point Multiplier with Rounding and Exceptions

III. Hardware of Floating Point Multiplier

A. 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

.

B. exponent addition

This unsigned adder is responsible for adding the exponent of the first input to the exponent of the second input and subtracting the Bias (1023) 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, and there is no need for a quick result because most of the calculation time is spent in the significand multiplication process (multiplying 53 bits by 53 bits); thus we need a moderate exponent adder and a fast significand multiplier.

An 11-bit ripple carry adder is used to add the two input exponents. As shown in Figure 3 a ripple carry adder is a chain of cascaded full adders and one half adder; each full adder has three inputs (A, B, Ci) and two outputs (S, C). The carry out (C) of each adder is fed to the next full adder (i.e each carry bit "ripples" to the next full adder). The addition process produces an 11 bit sum (S10 to S0) and a carry bit (C11). These bits are concatenated to form a 12 bit addition result (S12 to S0) from which the Bias is subtracted.

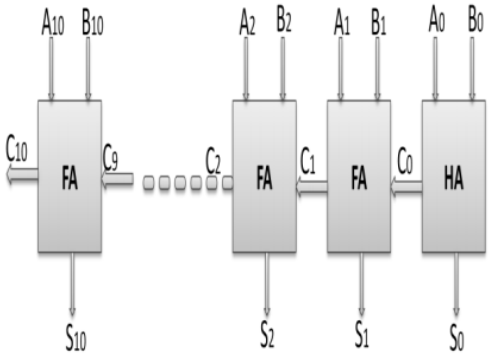


Figure. 3 Ripple Carry Adder

The Bias is subtracted using an array of ripple borrow subtractors. A normal subtractor has three inputs (minuend (S), subtrahend (T),

Borrow in (Bi)) and two outputs (Difference (R), Borrow out (B)). The subtractor logic can be optimized if one of its inputs is a constant value which is our case, where the Bias is constant (1023|10 = 001111111111|2).Table I shows the truth table for a 1-bit subtractor with the input T equal to 1 which we will call "one subtractor (OS)". Table II shows the truth table for a 1-bit subtractor with the input T equal to 0 which we will call "zero subtractor (ZS)"

Table I.
1-Bit Subtractor with the input T = 1

S	T	B_i	Difference(R)	B
0	1	0	1	1
1	1	0	0	0
0	1	1	0	1
1	1	1	1	1

Table II
1-Bit Subtractor with the input T = 0

S	T	B_i	Difference(R)	B
0	0	0	0	0
1	0	0	1	0
0	0	1	1	1
1	0	1	0	0

Figure 4 shows the Bias subtractor which is a chain of 10 one subtractors (OS) followed by 2 zero subtractors (ZS); the borrow output of each subtractor is fed to the next subtractor. If an underflow occurs then Eresult < 0 and the number is out of the IEEE 754 single precision normalized numbers range; in this case the output is signaled to 0 and an underflow flag is asserted.

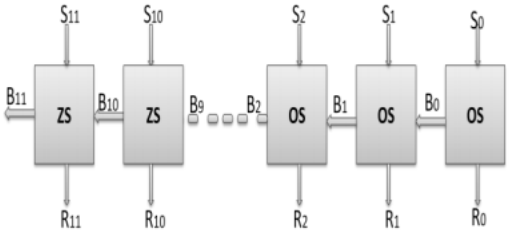


Figure .4 Ripple Borrow Subtractor

Rounding

In floating point arithmetic, rounding errors occur as a result of the limited precision of the mantissa. For example, consider the average of two floating point numbers with identical exponents, but mantissas which differ by 1. Although the mathematical operation is well-defined and the result is within the range of represent able numbers, the average of two adjacent floating point values cannot be represented exactly. In this paper we implemented all four rounding modes.

The IEEE FPS defines four rounding rules for choosing the closest floating point when a rounding error occurs:

– RN - Round to Nearest. Break ties by choosing the least significant bit = 0.

– RZ - Round toward Zero. Same as truncation in sign-magnitude.

– RP - Round toward Positive infinity.

– RM - Round toward Minus infinity. Same as truncation in integer 2's complement arithmetic.

Under Flow/Over Flow 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 11 bits in size, and must be between 1 and 2046 otherwise the value is not a normalized one. An overflow may occur while adding the double precision floating point multiplier code was a checked using Design Xilinx targeting on Virtex-6 xc5vlx110-3ff1760. Figure 5 shows the simulation results of high speed double precision floating point multiplier 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. Table III shows the normalization effect on result's exponent and overflow/underflow detection.

Table III
Normalization effect on result's exponent and overflow/underflow detection

E_{result}	Category	Comments
$-1021 \leq E_{result} < 0$	Underflow	Can't be compensated during normalization
$E_{result} = 0$	Zero	May turn to normalized number during normalization (by adding 1 to it)
$1 \leq E_{result} < 2046$	Normalized number	May result in overflow during normalization

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 results exponent is calculated by

$$E_{result} = E1 + E2 - 1023$$

E1 and E2 can have the values from 1 to 2046; resulting in E result having values from -1021 (2-1023) to 3069 (4092-1023); but for normalized numbers, Eresult can only have the values from 1 to 2046.

Simulation & Synthesis Results:

Double precision floating point multiplier has two 64 bits inputs and one 64 bits output output_FPM. It has also two bits r mode input signal which is used for rounding mode. Output signals of underflow, overflow, inexact, exception, and invalid will be asserted if the conditions for each case exist

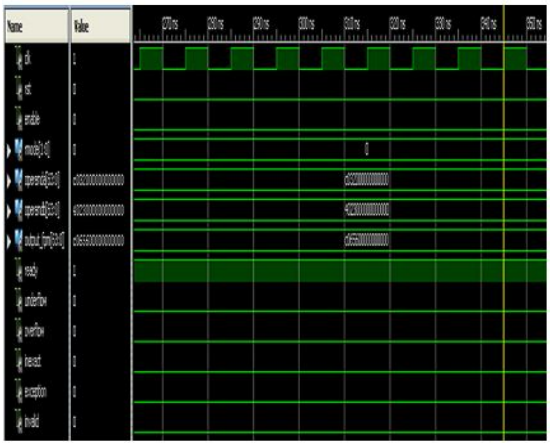
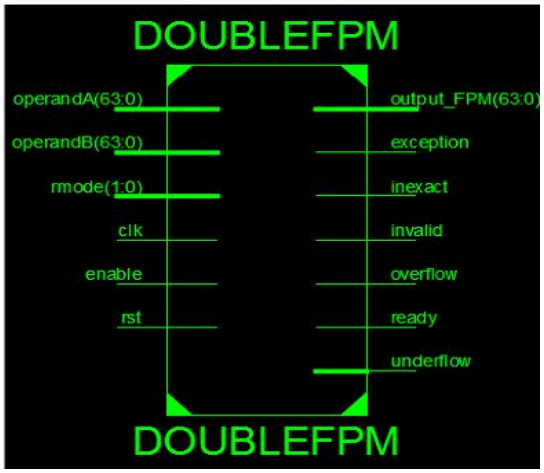


Fig 5. Double precision floating point multiplier

RTL Schematic:

Top module of double precision Floating point multiplier consist of three modules multiplier, rounding and exceptions module.



Conclusion

In this paper, double precision floating point multiplier has been coded with VHDL. Pipelining technique is used for synthesis of the double precision floating point multiplier. Double precision floating point multiplier using three stage pipelining technique achieved the

maximum frequency of 489.045 MHz with minimum delay 2.045 ns and area of 888 slices. The pipelined double precision FPM supports the IEEE -754 binary interchange format, targeted on a Xilinx Virtex-6 xc6vlx75t-3ff484 device. Pipelined double precision floating point multiplier is compared with other floating point multiplier it pr ovide high speed with minimum delay. Pipelined double precision floating point multiplier also support rounding mode and handle various exceptions like under flow, overflow and invalid. The double precision floating point multipliers was simulated in ISE simulator and synthesized using Xilinx (integrated software environment) ISE 13.2 tool.

Future Scope:

In this work, pipelined double precision floating point multiplier operates on 64-bit operands. It can be designed for quadruple precision floating point multiplier operates on 128-bit operands to enhance precision. Future work can also further extend to increase the more speed and reduce area. It can be extended to have more mathematical operations like adder/ subtractor, divider and exponential functions.

References

[1] Addanki Puma Ramesh, A.V.N. Tilak and A.M. Prasad, "An FPGA Based High Speed IEEE - 754 Double Precision Floating Point Multiplier using Verilog", IEEE International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), Tiruvannamalai, pp. 1– 5, January 2013.

[2] Riya Saini and R.D.Daruwala, "Efficient Implementation of Pipelined Double Precision Floating Point Multiplier", International Journal of Engineering Research and Applications, Vol. 3, Issue 1, pp.1676-1679, January -February 2013.

[3] Manish Kumar Jaiswal and Ray C.C.Cheung, "Area-Efficient Architectures for Double Precision Multiplier on FPGA, with Run-Time-Reconfigurable Dual Single Precision Support", Elsevier Microelectronics Journal, pp. 421–430, March 2013.

[4] Anna Jain, Baisakhy Dash and Ajit Kumar Panda, "FPGA Design of a Fast 32-bit Floating Point Multiplier Unit", IEEE Conference Publications, pp. 545 – 547, 2012.

[5] Addanki Purna Ramesh and Rajesh Pattimi, "High Speed Double Precision Floating Point Multiplier", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 1, Issue 9, pp. 647 – 650, November 2012.

[6] Xia Hong and JiaJinging, "Research and Optimization on Rounding Algorithms for Floating-Point Multiplier",IEEE Conference Publications, International Conference on Computer Science and Electronics Engineering, pp. 137 – 142, 2012.

[7] Puneet Paruthi, Tanvi Kumar and Himanshu Singh, "Simulation of IEEE 754 Standard Double Precision Multiplier using Booth Techniques", International Journal of Engineering Research and Applications, Vol. 2, Issue 5, pp. 1761-1766, September- October 2012.4

[8] B.Sreenivasa Ganesh, J.E.N.Abhilash and G. Rajesh Kumar, "Design and Implementation of Floating Point Multiplier for Better Timing Performance", International Journal of Advanced Research in Computer Engineering & Technology, Vol. 1, Issue 7, September 2012.

[9] Aniruddha Kanhe, Shishir Kumar Das and Ankit Kumar Singh,"Design and Implementation of Floating Point Multiplier based on Vedic Multiplication Technique", International Conference on Communication, Information & Computing Technology (ICCICT), Oct. 19-20, Mumbai, India, pp. 1 – 4, 2012.

[10] Mohamed Al-Ashrafy, Ashraf Salem and Wagdy Anis, "An Efficient Implementation of Floating Point Multiplier", IEEE Electronics, Communications and Photonics Conference (SIECPC), Saudi International, pp. 1 - 5, 2011.