ISSN2394-3777 (Print)
ISSN2394-3785 (Online)
Available online atwww.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology
(IJARTET) Vol. 5, Special Issue 14, April 2018

FLOATING POINT ARITHMETIC UNIT

CYAMIRON SHRESTHA, BTI, Bangalore, India. kyamironstha@gmail.com

BIRENDRA SHRESTHA, BTI, Bangalore, India. Sdhiren24@gmail.com

NABIN DHAKAL, BTI, Bangalore, India. dhakalnabin013@gmail.com

KORASH WAIBA, BTI, Bangalore, India. waibakorash@gmail.com

PROF. KIRAN KUMAR BTI,Bangalore, India. kirankumarec029@gmail.com

ABSTRACT

Floating Point Arithmetic unit plays a vital role in digital systems. With the over view of complex digital circuits it is possible to development Very Large Scale Integration (VLSI) circuit technology for the arithmetic unit. In this paper, algorithm for the arithmetic unit has been implemented on Spartan3 XC3S400 FPGA Board for the Floating point numbers. The arithmetic operations like addition, subtraction and multiplication on Floating Point numbers have been implemented on arithmetic unit. The purpose of this paper is to overcome the delay time of arithmetic operations of floating point numbers like addition, subtraction and multiplication. Floating Point Arithmetic unit is used for execution of the floating point computations. All the algorithm modules are coded by using Verilog HDL and Simulated with Xilinx ISE tools.

Keywords—FPGA; Floating Point Numbers; Verilog;

I. INTRODUCTION

Floating Point (FP) Arithmetic is used for a variety of Digital Signal Processing (DSP)applications and it allows the designer and the user to concentrate on the algorithms and architecture without complex numerical issues. In early daysmany DSP applications used fixed point arithmetic due to high cost in area of utilization, delay and power consumption of Floating Point arithmetic unit. Floating Point also supports the much wider range of values then the fixed point and also have the ability to represent the very small number to the very large number. In DSP applications they have high accuracy and impressive precision with dynamic range.Floating Point operations haveapplications in communication, multimedia system and the signal processing techniques. Realization of complex digital circuits is possible with development in Very Large Scale Integration (VLSI) circuit technology. Therefore various conventional and non-conventional arithmetic methods are required for the Floating Point Arithmetic operation.Initially Floating Point was allowed via coprocessor rather than having different things working together as one unit and in microcomputers time a single microchip was used.
This paper implements algorithm for thearithmetic unit that are specially designed for various operations on Floating Point numbers like addition, subtraction and multiplication.

1bit	8 bit	23 bit
SIGN	EXPONENT	MANTISSA

Figure 1: Representation of 32bit Floating Point Number

Floating Point number for 32 bit format includes 1 sign bit, 8 bits for exponential and 23 bits wide mantissa or fractions. In this algorithm a 32 bit operandisused forthe operations to be performed and operator register for the end result. For the representation of the operation, signals status like overflow, underflow, inexact, exception and invalid is been used in the algorithm.

II. LITERATURE SURVEY

Floating-point operations are needed very frequently in nearly all computing disciplines, and studies have shown floating-point addition to be the most used Floating-Point operation. In this, the operation presents for the floating-point adder that closely follows the IEEE-754 specification for floating-point arithmetic operations.

ISSN2394-3777 (Print)
ISSN2394-3785 (Online)
Available online atwww.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology
(IJARTET) Vol. 5, Special Issue 14, April 2018

[1] This paper presents the systems using digital signal processing multiply-accumulate which is the one of the functions. The whole systems performance depends on the performance of the MAC units. It also presents that the design and implementation of 16 bit Floating Point multiplication and additions. Where MAC unit generally consists of three units like Floating Point multiplier, Adder and an Accumulator. The input takes the half-precision format where there is 1 bit sign, 7-bit exponent and 8 bit mantissa therefore making the performance of operation faster.

[2] This paper presents an open source Floating Point adder and multiplier implemented using a 32-bits custom number format based on radix-16 and optimized for 7 series FPGAs from Xilinx. Although this number format is not identical to the single-precision IEEE-754 format, the Floating Point operators are designed in such a way that the numerical results for a given operation will be identical to the result from an IEEE-754 compliant operator with support for round-to-nearest even, NaNs and Infs, and subnormal numbers. The drawback of this number format is that the rounding step is more involved than in a regular, radix-2 based operator. On the other hand, the use of a high radix means that the area cost associated with normalization and denormalization can be reduced, leading to a net area advantage for the custom number format, under the assumption that support for subnormal numbers is required. The adder can operate at 319MHz and multiplier can operate at a frequency of 305MHz.

[3] The paper presents an arithmetic unit based on IEEE-754 standard for floating-point numbers has been implemented on Spartan3E XC3S500e FPGA Board and follows IEEE single precision format. Various arithmetic operations on Floating-Point number have been performed on arithmetic unit. This can perform 50 Mega floating point operation per second at 50MHz clock. Three stages of all arithmetic operations are- pre-normalize stage, arithmetic core and post-normalize stage. In pre- normalize stage the operands are converted into formats so that they can be handled easily and efficiently. In arithmetic core basic arithmetic operations are done. Lastly in post normalize stage the result is normalized and converted into format specified by IEEE standard.

III. NUMBER SYSTEMS

A. Floating Point Number

A Floating Point number is a number which represents the specific way of encoding a number. A Floating Point number is represented in a standard form

$F = -1^{s} x\ 1.M\ x2^{E}$

Where, S is Sign part, M is Mantissa part, and E is Exponent part of the Floating Point number.

For 32bit floating point number

Bit No	Size	Name
31	1 bit	Sign (S)
23-30	8 bits	Exponent (E)
0-22	23 bits	Mantissa (M)

When the mantissa is leading by 1 and implied in hardware. So, itmeans that for computations the produce a leading should be 0 and fraction must be shifted. The only leading 1 is for gradual underflow. The exponent of the floating point is given by the usual biased format which is given by

$E = E^{true}$ +bias.

Common value of the bias of Floating Point number is given by

$2^{e-1} - 1$

Where, e is the number of bits present in the exponent part. It is done by make comparisons of floating point numbers.

IV. IMPLEMENTATION OF FLOATING POINT UNIT

The arithmetic operations are done in three stages and they are pre-normalize stage, arithmetic operation stage and post-normalize stage. The operands are converted into a formats so that they can be handled more easily and efficiently in pre- normalize stage. Various basic arithmetic operations are done in arithmetic core. At lastly the result is normalized and converted into format which is specified by IEEE standard in post-normalize stage.

ISSN2394-3777 (Print)
ISSN2394-3785 (Online)
Available online atwww.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology
(IJARTET) Vol. 5, Special Issue 14, April 2018

A. Addition/Subtraction

Suppose two floating-point numbers are given, then sum of these numbers are
(M1 x 2^E1) + (M2 x 2^E2) = M x 2^E
Where M is Mantissa
E is Exponent of the Radix(r)
Radix(r) = 2
M1 is Mantissa of first number
M1is Exponent of the Radix(r) of first number
M2 is Mantissa of second number
E2 is Exponent of the Radix(r) of second number
The sum of mantissas is the mantissa part and whereas the exponent part of the sum is remain same. M1 and M2 are properly normalized and if M11 and M2 are not properly normalized than we have to first normalize the mantissa part of sum.

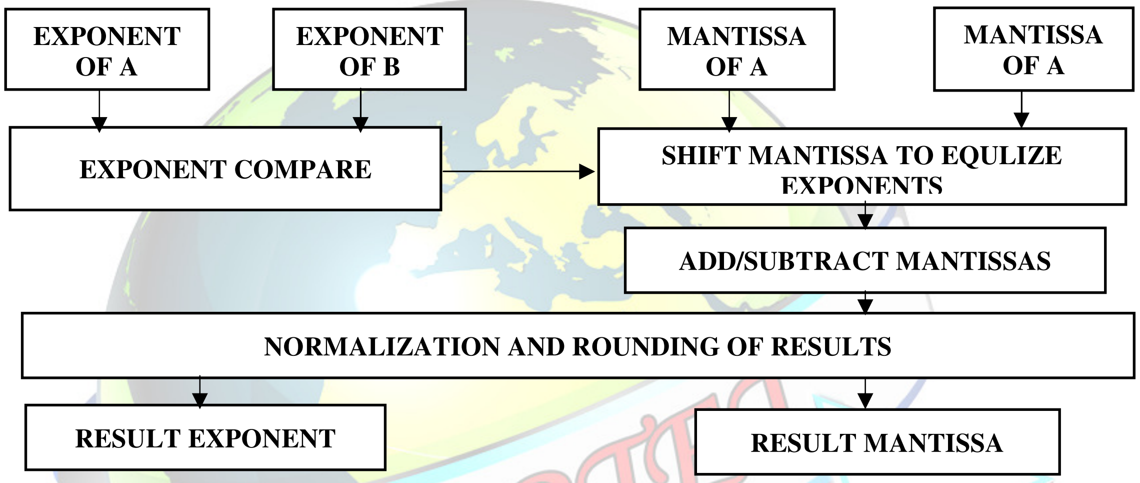
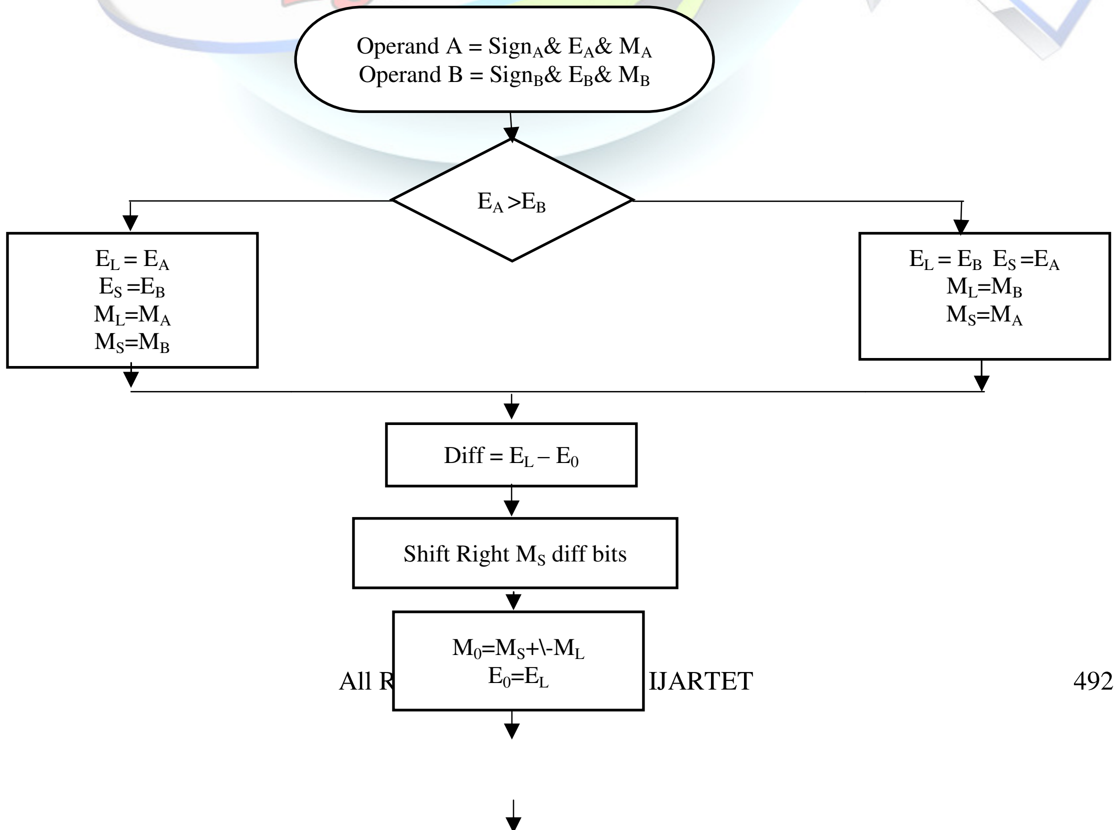


Fig 1. Block Diagram of Floating-Point Addition/Subtraction

B. Addition/Subtraction Algorithm

For the computation of Floating Point addition and subtraction are more complex rather than multiplication.


Operand A = Sign_A& E_A& M_A
Operand B = Sign_B& E_B& M_B
E_A >E_B
E_L = E_A E_S =E_B M_L=M_A M_S=M_B
E_L = E_B E_S =E_A M_L=M_B M_S=M_A
Diff = E_L – E_0
Shift Right M_S diff bits
M_0=M_S+\-M_L E_0=E_L

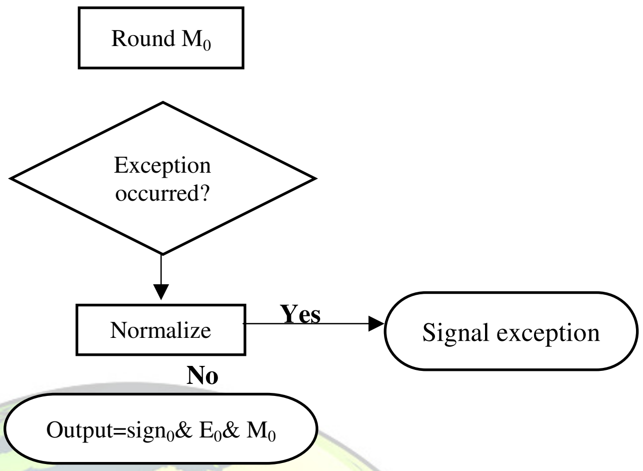


Fig. 2: Flowchart of Addition and Subtraction Computation

Here are the some algorithm for computation of Floating Point addition have been given below:
Suppose X1 and X2 be two numbers and X3 be the resultant. The sum/different of these numbers is given by
X3 = X1 +/- X2
X3 = (M1 x 2^{E1}) +/- (M2 x 2^{E2})

1. When the exponents are the same(i.e. E1=E2) then X1 and X2 can only be added.
2. Assume X1 has the larger absolute value of the two numbers. Then absolute value of X1 should be greater than absolute value of X2. Otherwise swap the values such that absolute value of X1 is greater than absolute value of X2. (i.e. Abs(X1) > Abs(X2)).
3. At first initial value of the exponent should be the larger of the two numbers, since exponent of X1 will be bigger. Therefore, initial exponent result E3 = E1.
4. The difference of exponents is calculated. i.e. diff = (E1-E2).
5. Then left shift the decimal point of mantissa (M2) by the exponent difference. Since the exponents of both X1 and X2 are now same.
6. The sum/difference of the mantissas depending on the sign bit S1 and S2 are calculated.
   If signs part of X1 and X2 are equal (S1 == S2) then the mantissas are added.
   Otherwise the mantissas are subtracted (i.e.S1! =S2).
7. Then the resultant mantissa (M3) is normalized if it is needed. Then the 1.M3 format and the initial exponent result E3=E1 have to be adjusted according to the normalization of mantissa part.
8. If anyone of the operands is infinity or if resultant exponent is greater than maximum exponent (i.e. E3 >Emax), then overflow have occurred. Therefore, the output should set to infinity. If resultant exponent is less than minimum exponent (i.e. E3 <Emin) then it is an underflow. Therefore, the output should set to zero.
9. Not a numbers (NaN) are not supported in Floating Point operations.

C. Multiplication

In this designed the single-precision multiplier for floating-point numbers. We have used 23 bit mantissa part and 9 bit for exponent part. If there is a negative numbers then that number should be taken in 2's complement format.
We supposed, the two Floating Point numbers and taken for multiplication operation. The product of these two number is given by
(M1×2^E1) × (M2 × 2^E2) = (M1 × M2) × 2^ (E1+E2) = M × 2^E
Where M is Mantissa
E is Exponent of the Radix(r)
Radix(r) = 2
M1 is Mantissa of first number
E1is Exponent of the Radix(r) of first number
M2 is Mantissa of second number

E2 is Exponent of the Radix(r) of second number

The mantissa part of the product of these two numbers is the product of mantissas, and the exponent part of the product of these two numbers is the sum of exponents. We assumed that M1 and M2 are properly normalized. If they are not normalized than first normalize the mantissa part of product of these two numbers.

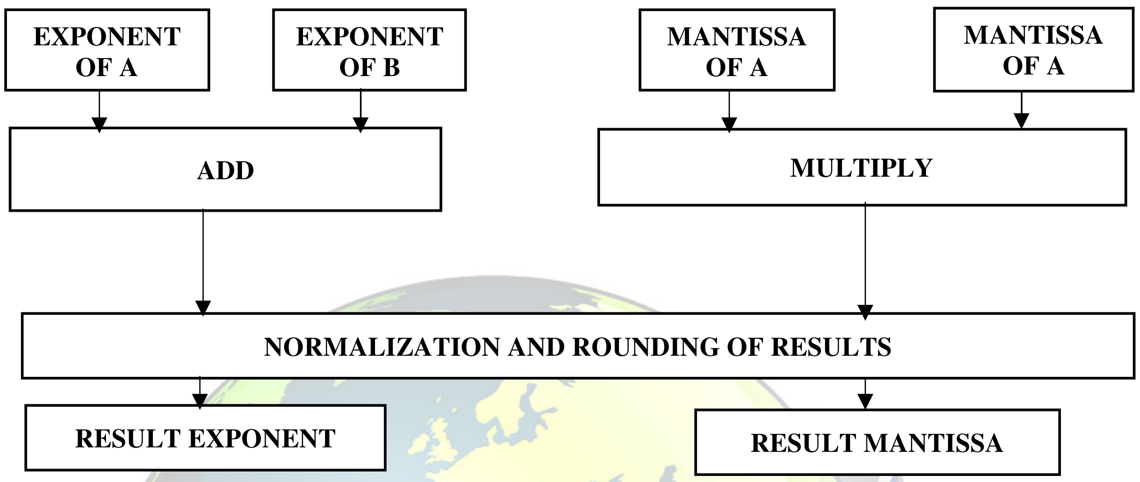


Fig.3: Block Diagram of Floating Point Multiplication.

a. Floating point multiplication algorithm

• Addition of exponents and then subtract bias.
• The mantissa parts of these numbers are multiplied and also the sign of result is determined.
• Then the resulting value of the product is normalized.
• When the mantissa of the product is reduced by half than the rounding of numbers occurs in floating point multiplication.
• The algorithm for the multiplication is done in 3 steps. For example let us take two 5- digits FP numbers to multiply:

$$(2^{100} \times 1.1001) \times (2^{110} \times 1.0010)$$

(a) Multiply fractions and calculate the result exponent.

$$1.1001 \times 1.0010 = 1.11000010$$
$$\text{So, } M_0 = 1.11000010 \text{ and}$$
$$E_0 = 2^{100+110-bias} = 2^{83}$$

(b) Round the fraction to nearest-even

$$M_0 = 1.1100$$

(c) Result = $2^{83} \times 1.1100$

V. IMPLEMENTATION RESULTS

TABLE I. COMPARISON

	In Our Paper	**In Reference Paper[1]**
Slices For Addition	155 out of 3584 (4%)	157 out of 2448 (6%)
Slices For Subtraction	155 out of 3584 (4%)	157 out of 2448 (6%)
LUTs For Addition	295 out of 7168 (4%)	279 out of 4896(5%)
LUTs For Subtraction	295 out of 7168 (4%)	279 out of 4896(5%)

From the table I shows the comparison for slices and LUTs of floating point operation like addition and subtraction.

REFERENCES

[1]International journal of modern trends in engineering and science on"Design of 16-bit Floating Point Multiply and Accumulate Unit" by Saravana R, Balaji P,Prabu R
[2]"Area Efficient Floating-Point Adder and Multiplier with IEEE-754 Compatible Semantics" 2014 IEEE.
[3] Prateek Sing Instrumentation and Control Department College of Engineering Pune, India 41105 "Optimizied Floating Point Arithmetic Unit".
[4]Koren, Israel, Computer Arithmetic Algorithms, 2nd Edition, A.K. Peters, Ltd., Natick, MA, 2002.
[5] G. Lienhart, A. Kugel, and R. Manner, "Using floating-point arithmetic on FPGAs to accelerate scientific n-body simulations," in *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*. IEEE, 2002, pp. 182–191.
[6] D. G. Bailey, "Space efficient division on fpgas," in *Electronics New Zealand Conference (EnzCon'06)*, 2006, pp. 206–211.
[7] N. Sorokin, "Implementation of high-speed fixed-point dividers onFPGA," *Journal of Computer Science & Technology*, vol. 6, 2006.
[8] M. Haselman, M. Beauchamp, A. Wood, S. Hauck, K. Underwood, and K. S. Hemmert, "A comparison of floating point and logarithmic number systems for fpgas," in *Field-Programmable Custom ComputingMachines, 2005. FCCM 2005. 13th Annual IEEE Symposium on*. IEEE,
2005, pp. 181–190.
[9] R. Goldberg, G. Even, and P.-M. Seidel, "An fpga implementation of pipelined multiplicative division with ieee rounding," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15thAnnual IEEE Symposium on*. IEEE, 2007, pp. 185–196.
[10] "Digital Signal Processing with Field Programmable Gate Array" by Dr. Uwe Meyer-Baese.
[11] "Digital Systems Design using VHDL" by Charles H. Roth.
[12] "Design Through Verilog HDL" by T.R.Padmanabhan and B.Bala Tripura Sundari.
[13] Peter-Michael seidel and Guy Even.On the Design of Floating-Point Adders.In proceedings of the 15th IEEE symposium of Computer Arithmetic.