Scientific Journal of Impact Factor (SJIF): 4.14

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 3, Issue 6, June -2016

Implementation of single precision floating point unit using VHDL

[1] *Ms.Mayuri Pravin Nirgude* ,[2] *Ms.Maya Narayan Phadtare* ,[3] *Ms.Priti Dilip Phadtare* ,
[4] *Ms.Trupti Shankar Shinde* ,[5] *Prof. ANIL GOPAL SAWANT.*

[1,2,3,4,5] *Department of Electronics & Telecommunication,*
Trinity College of Engineering & Research,
Pune,India.

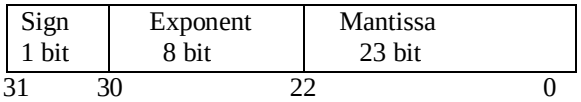
Abstract: *We introduce 32 bit single precision floating point unit with arithmetic operation i.e addition, subtraction ,multiplication & division in single precision floating point number using the IEEE 754 format. Floating point unit increase the speed of floating point arithmetic operation & reduce the complexity of ALU. Floating point having various application like supercomputer, coprocessor in every computer. large range is required to represent large and small value, therefore floating point representation is used to represent those value.*

Keywords- *Very High Speed Integrated Circuit Hardware Description Language (VHDL), Floating Point Unit(FPU).*

Introduction

Floating point processor is used in various field like high speed computing system, digital signal processing,climate research because of its high speed & accuracy.Floating point operation can be performed in IEEE 754 in single precision & double precision. In a fixed point, number of bits are reserved for left side i.e integer part & number of bit are reserved for right side i.e fraction part of the decimal point.In a floating point,number of bits are not reserved after & before of the decimal point. The floating point number can be represented in sign,exponent & mantissa bit. Floating point processor reduce hardware complexity designing is using FPGA it has advantages of low power consumption & cost.

1.1 Floating Point Format:



Sign bit : Sign bit represent the number is positive or negative. If it is '1' then number is negative if sign bit is '0' then number is positive.

Exponent : Exponent is 8 bit field. We add the bias to the actual exponent to get a stored exponent. We are able to represent a number in positive or negative value. For single precision the value of bias is 127.the reason behind this, exponent is 8 bit so the value of bias 2^k-1 , k=8, 2^8-1 =2^7-1=127.

Mantissa:Mantissa field is 23 bit & it consist of the implicit bit &fractional part & it represent in 1.fffff…….ff format.

Normalization:The floating point number when adjusted such that , the implied leading bit is 1, then it is called as normalized number.When we representing floating point number, the implied leading bit is not stored to increase accuracy. It is retrieved while performing operation. When the implied bit is '0' then the number is reffered as de-normalized number.

Exceptions:It occur when result of floating point operation is unclear & undesirable.

i)Zero:If the exponent field is all zero & fractional part is other than zero then zero exception will be occur, and result is zero.

ii)De-normalized:If exponent field is zero & fraction field is not zero, then de-normalized number does not consist '1' leading bit before the fraction part.

Underflow:Underflow occur when the result is too small of the defined range in a negative direction(1).

Overflow:Overflow occur when result has an exponent is too large in positive direction(255).

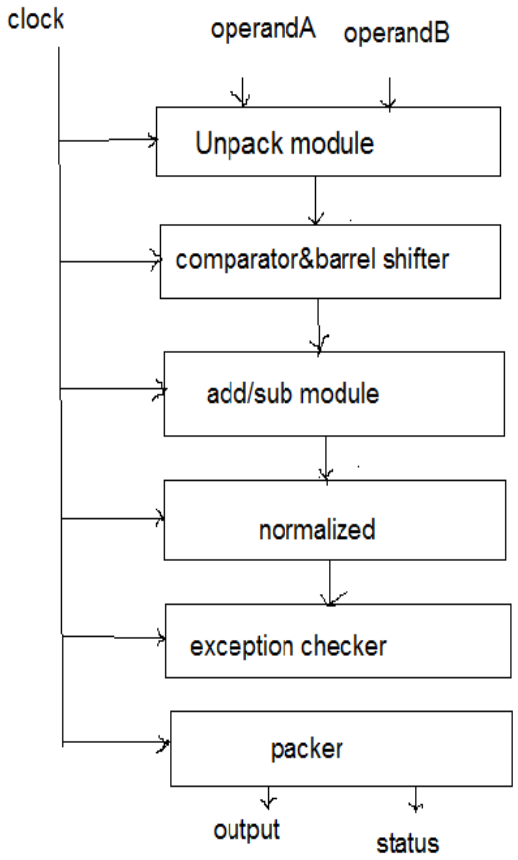
Infinity:Infinity occur when exponent is all 1's & fraction is all zero. Sign bit distinguish between positive infinity & negative infinity.

Nan (Not a number):Nan occur when exponent field is all 1's & fraction is other than zero. There are two types of Nan i.e Quite Nan(QNan) & signaling Nan.

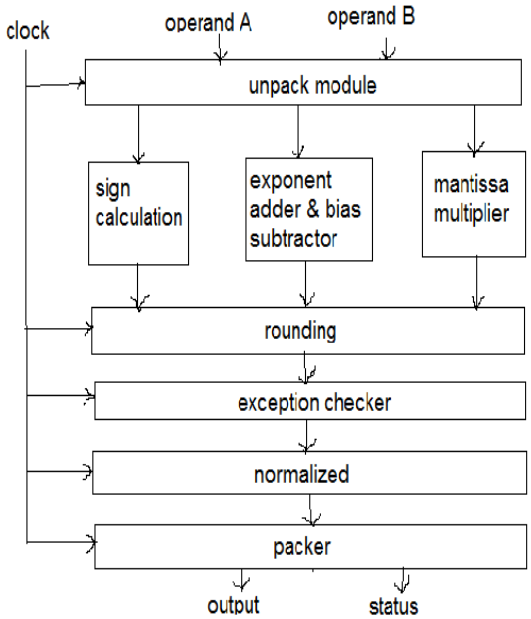
Block diagram

i)add/sub:

The unpack module will separate the mantissa,exponent and sign bit of given operand and also add the implied bit to the mantissa. The comparator module compares the exponent of two operand and shift the smaller operand by the difference bit. The actual add/ sub will carried by the add/sub module depending upon their sign bit. normalize module will normalize the final result it shift the result until MSB becomes one. exception checker will check exception like overflow,underflow, result zero. packer module will combine the sign bit,exponent and mantissa bit and drop the implied bit from output mantissa.

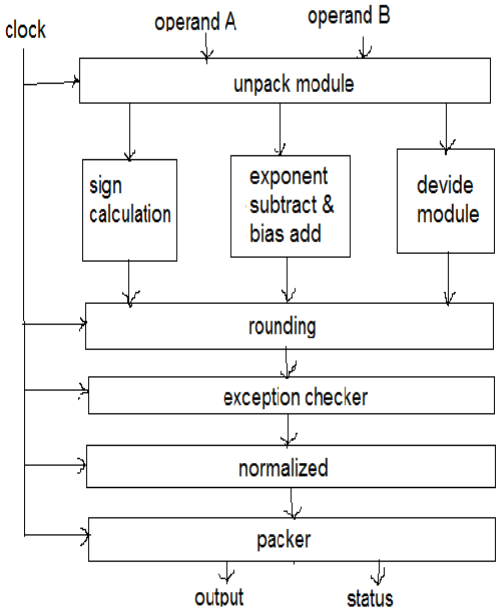


ii)Multilication



Unpack module will separate the sign, exponent, mantissa of the given operand and add the implied leading bit to the mantissa. Sign of the resultant is calculated by XORING of two sign bits. resultant exponent is the addition of two exponent with bias subtraction i.e. E1+E2-127 here 127 is bias. In mantissa multiplier module the actual multiplication of two mantissa is carried out. After multiplication the result is 32 bit . This result is round to fit into single precision format. Then normalize the final result to bring it in the format like 1.xxx… The packer will combine the resultant sign, exponent, mantissa bit.

iii)Division:



The unpack module separates sign, exponent, and mantissa and it also add leading implied bit to the mantissa. Resultant sign is calculated by XORING of sign bit of two operand. Resultant exponent is calculated by subtracting two exponent and adding bias to it i.e. E1-E2+127 here 127 is bias. Division of two mantissa is carried out in divide module. Rounding of output is done to fit the result in single precision format. The exceptions like overflow, underflow, infinity, zero etc is checked by the exception checker. Finally the result is normalize to bring the result in 1.xxx…. format. Packer module is used to combine the resultant sign, exponent, mantissa and it drop the implied leading bit of the mantissa.

Result

Simulation result of Addition :

In this section we analyze the result of addition of two operand. A=01101011111100111010000011000011

B=01101011100011100101111100011100

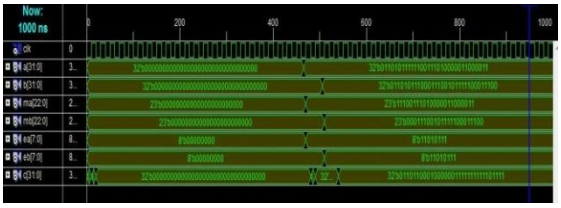
Exponent of A &B=11010111

Mantissa of A=11100111010000011000011

Mantissa of B=00011100101111100011100

Finalresult=

01101100010000000111111111101111



Simulation result of subtraction:

Operand A=01000000111111000000000000000000

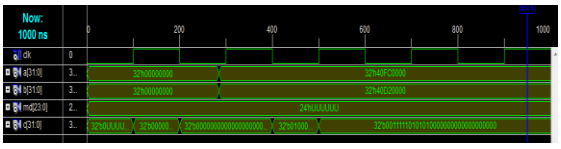
B=01000000110100100000000000000000

Exponent of A &B=10000001

Mantissa of A=11111000000000000000000

Mantissa of B=10100100000000000000000

Final result = 00111110101010000000000000000000



Simulation result of division:

Operand

A =11000001100100000000000000000000

B=01000001000110000000000000000000

Exponent of A=10000011

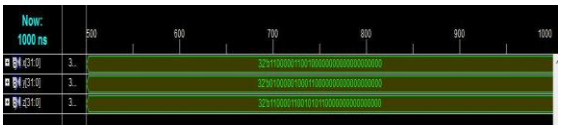
Exponent of B=10000010

Mantissa of A=00100000000000000000000

Mantissa of B=00110000000000000000000

Final result=

11000011001010110000000000000000



Simulation result of Division:

Operand A=01000000000110000000000000000000

B=01000001000001000000000000000000

Exponent of A=10000000

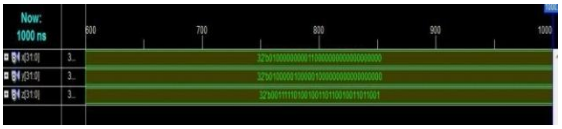
Exponent of B=10000010

Mantissa of A=00110000000000000000000

Mantissa of B=00001000000000000000000

Final result=

00111110100100110110010011011001



Conclusion

Floating point unit is used in various application areas like high performance computing unit in Mobiles, laptops, computers. It is also used in digital signal processing.

The Floating point operation cannot overflow easily although large memory room is required. Floating point format is needed in some high precision operation. By the rapid development of FPGA, there are millions of logic cells and many DSP modules inside, which make the implementation of FP arithmetic in FPGA possible.

In this project we can implement four basic operation which is addition ,subtraction, multiplication and division. in-depth study of floating-point exceptions and specific implementation standards of them from the view of circuit design. Besides, we propose scientific and reasonable methods of handling exceptions in different floating-point operations.

In future we can upgrade single precision floating point unit into double precision floating point unit.

Reference

1) Lamiaa S. A. Hamid, Khaled A. Shehata, Hassan El-Ghitani Mohamed ElSaid- Design of Generic Floating Point Multiplier and Adder/Subtractor Units 978-0-7695-4016-0/10 $26.00 © 2010 IEEE.

2) Guillermo Marcus,Patricia Hinojosa, Alfonso Avila and Juan Nolazco-Flores-A Fully Synthesizable Single-Precision, Floating-Point Adder/Substractor and Multiplier in VHDL for General and Educational Use 0-7803-8777-5/04/$20.00 © 2004 IEEE

3) Manisha Sangwan,Anita Angeline-Design and Implementation of Single Precision Pipelined Floating Point Co-Processor978-1-4799-1441-8/13/$31.00 2013 IEEE
4) Mamu Bin Ibne Reaz, , Md. Shabiul Islam, Mohd. S. Sulaiman-Pipeline Floating Point ALU Design using VHDL 0-7803-7578-5/02/$17.00 *02002* IEEE

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

6) Yunjuan Cui, Baixiao Chen, Shouhong Zhang -Design of Floating-point Operation Based on FPGA and it's Application 7803-9737-1/06/$20.00 ©2006 IEEE

7) Xia Hong Wang Chongyang Yan Jiangyu - Analysis and Research of Floating-Point Exceptions 978-1-4244-7618-3 /10/$26.00 ©2010 IEEE
8) Ms.AnjanasasidharanMrP.Nagarajan- Vhdl Implementation Of IEEE 754 Floating Point Unit ISBN No. 978-1-4799-3834-6/14/$31.00©2014 IEEE
9) Lasith K K, Anoop Thomas, Asst.Professor - Efficient Implementation Of Single Precision Floating Point Processor In FPGA 978-1-4799-5202-1/14/$31.00 2014 IEEE
10) guy even and wolfgang j paul – on the design of IEEE complaint floating point units 0018-9340/00 $ 10.00 2000 IEEE

