# Design Of 32 Bit Floating Point Addition And Subtraction Units Based On IEEE 754 Standard

Ajay Rathor, Lalit Bandil
Department of Electronics & Communication Eng.
Acropolis Institute of Technology and Research
Indore, India

## Abstract

Floating point arithmetic implementation described various arithmetic operations like addition, subtraction, multiplication, division. In science computation this circuit is useful. A general purpose arithmetic unit require for all operations. In this paper single precision floating point arithmetic addition and subtraction is described. This paper includes single precision floating point format representation and provides implementation technique of addition and subtraction arithmetic calculations. Low precision custom format is useful for reduce the associated circuit costs and increasing their speed. I consider the implementation of 32 bit addition and subtraction unit for floating point arithmetic.

Keywords: Addition, Subtraction, single precision, doubles precision, VERILOG HDL

## 1. Introduction

This is invaluable tools in the implementation of high performance systems, combining the reprogramability advantage of general Purpose processors with the speed and parallel processing. At some point, require general purpose arithmetic processing units which are not standard components of fpga devices [10].
More recently, the increasing size of fpga devices allowed researchers too efficiently Implement operators in the 32-bit single Precision format .Single precision format, the most basic format of the ANSI/IEEE 754-1985 binary floating-point arithmetic standard. Double precision and quad precision described more bit operation so at same time we perform 32 and 64 bit of operation of arithmetic unit. Floating point includes single precision, double precision and quad precision floating point format representation and provide implementation technique of various arithmetic calculation. Normalization and alignment are useful for operation and floating point number should be normalized before any calculation [3].

## 2. Floating Point format Representation

Floating point number has mainly three formats which are single precision, double precision and quad precision.
Single Precision Format: Single-precision floating-point format is a computer number format that occupies 4 bytes (32 bits) in computer memory and represents a wide dynamic range of values by using floating point .As mentioned in Table 1 the single precision format has 23 bit for significand (1 represent implied bit), 8 bit for exponent and 1 bit for sign. The IEEE standard specifies that Single precision floating-point numbers are comprised of 32 bits, i.e. a sign bit (bit 31), 8 bits for the exponent *E* (bits 30 down to 23) and 23 bits for the fraction f (bits 22 to 0). *E* is an unsigned biased number and the true exponent *e* is obtained as $e=E-E\text{bias}$ with $E\text{bias}=127$ the leading 1 of the

significand, is commonly referred to as the "hidden bit". This is usually made explicit for operations, a process usually referred to as "unpacking".

The IEEE standard specifies that double precision floating-point numbers are comprised of 64 bits, i.e. a sign bit (bit 63), 11 bits for the exponent *E* (bits 62 down to 52) and 52 bits for the fraction *f* (bits 51 to 0). *E* is an unsigned biased number and the true exponent *e* is obtained as *e=E–E*bias with *E*bias=1023. The fraction *f* represents a number in the range [0,1) and the significand *S* is given by *S*=1 [8].

The IEEE standard specifies that quad precision floating-point numbers are comprised of 128 bits, i.e. a sign bit (bit 127), 15 bits for the exponent *E* (bits 126 down to 112) and 112 bits for the fraction *f* (bits 111 to 0). *E* is an unsigned biased number and the true exponent *e* is obtained as *e=E–E*bias with *E*bias=16383 [2, 10].

| **Precision** | **sign** | **Exponent** | **Significand** |
|---|---|---|---|
| Single precision | **1** | **8** | **23+1** |
| Double precision | **1** | **11** | **52+1** |

**Table 1:** Floating point Format

Table 1 show single precision and double precision floating point number specification. Single precision for 32 bit and double precision for 64 bit used different exponent and significand. Quad precision format used for 128 bit and described different significand and exponent. In this paper single precision floating point format used. Single precision addition and subtraction floating point operation used.

## 3. Addition/Subtraction

If two numbers x and y which described floating point numbers and we perform calculation sum or difference *of* two numbers. Firstly, we check for zero of two values and then require next step calculate the difference of the two exponents, so need operation of alignment. Align the significand *Ex–Ey =0* and put exponent *ER* is result of the two exponents. Now add or subtract the two significands *Ex* and *Ey*, according to the effective operation. Normalize the result *SR*, adjusting *ER* as appropriate. Step by step we can perform addition and subtraction of two floating point numbers.

In the first step, the floating point operands *x* and *y* are unpacked and checks for zero, infinity. If we can assume that neither operand is infinity. The relation between the two operands *x* and *y* is determined based on the relation between *Ex* and *Ey* and by comparing the significands *sx* and *sy*, which is required if *Ex=Ey*. Swapping *Sx* and *Sy* is equivalent to swapping *x* and *y* and making an adjustment to the sign *sr*. this swapping requires only multiplexers.

In the next step, alignment of two significand used. The significand alignment shift is performed and the effective operation is carried out. This step useful for equal the significand numbers. Alignment achieved by shifting either smaller number to right. Now we can say that increasing it exponent .if alignment has been completed then we go for next step normalization. Alignment is useful for next step of normalization. Without this step result not be calculated.

Normalization is last step for floating point arithmetic operation. *Sr* is normalized by after the alignment and shifter, which can perform by right shifts. If *sr* is normalized, then it shows result of two floating point number of addition and subtraction.

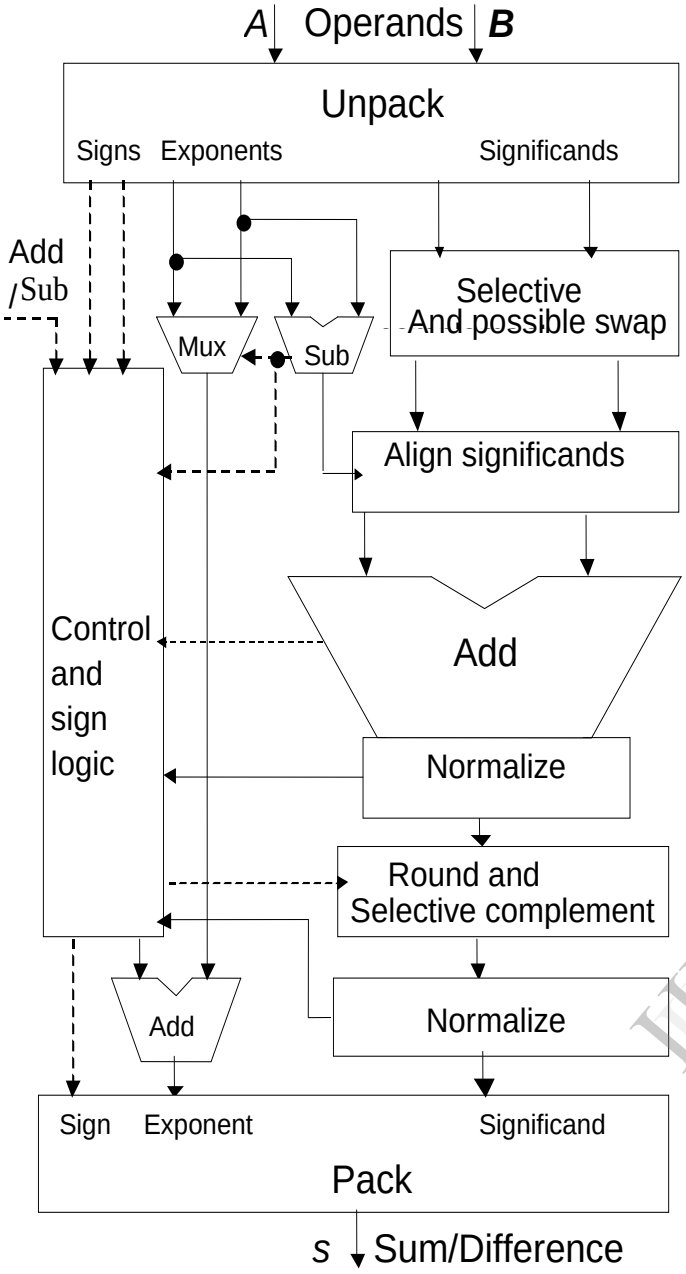


**Figure 1.** floating-point adder/subtractor

Addition and subtraction of two floating point number described in following block diagram. Step by step we perform calculation of two floating point number addition and subtraction. Firstly if we taken two floating point number for addition and subtraction then in starting both number will be unpack. After that we identify sign, exponent and significand of both numbers. This process depends on single precision and double precision and quad precision format. According to this format we taken sign, exponent and significand bit. The IEEE standard specifies that format. Normalization process will be completed after specify IEEE standard. In this step normalize the number means move binary point so that it is one bit from left. Shifting the mantissa left by one bit decrease the exponent by one or shifting the mantissa right by one bit increases the exponent by one. Before subtracting, compare magnitude and change sign bit if order of operands is changed .Alignment can be done after that process and finally we get result of addition and subtraction of two floating point numbers.

## 4. Result and discussion

The implementation of floating point addition and subtraction design unit for 32 bit floating point number are described in simulation report. Simulation report explains result of addition and subtraction of two floating point number. A direct comparison with other floating-point unit implementations is very difficult to perform, not only because of floating-point format differences, but also due to other circuit characteristics, e.g. all the circuits presented here incorporate I/O registers, which would eventually be absorbed by the surrounding hardware. In conclusion, the circuits provide an indication of the costs of FPGA floating-point operators using a long format. 8.5 And 1.25 floating point additions provide 9.75 which apply input as a single precision format.

Two floating point number in_1 and in_2 shows input data. And we get result in out which is result or output. This is shown in simulation waveform figure 3. This figure show single precision addition of two floating point numbers. In this implementation use clock pulse and load pulse for getting result. Resets also apply for new calculation. In this result single

precision 32 bit calculation shows. Floating point subtraction simulation result show in figure 4 .Two floating point number in_1 and in_2 shows input data. And we get result in out which is result or output. 8.5 And 1.25 floating point subtractions provide 7.25 which apply input as a single precision format. In this implementation use clock pulse and load pulse for getting result.

| | Adder | Subtractor |
|---|---|---|
| Total logic elements | 184/15,408 ( 1% ) | 304/15,408 ( 2% ) |
| Total combinational functions | 182/15,408 ( 1% ) | 302/15,408 ( 2% ) |
| Dedicated logic registers | 127/15,408 (<1%) | 193/15,408 (1%) |
| Total pin | 99/161 (61%) | 131/161 (81%) |
| Total memory bit | 0/516,096 ( 0% ) | 0/516,096 ( 0% ) |

**Table 2:** Operator statistics on Quartus II EP3C16Q240C8 Cyclone III FPGA device

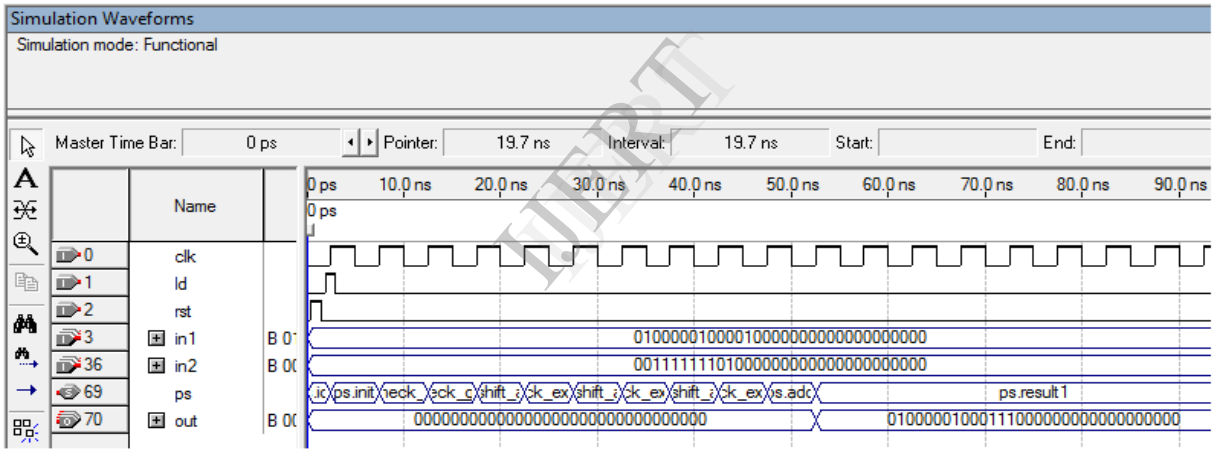


**Figure 2.Simulation waveform of floating point Adder**

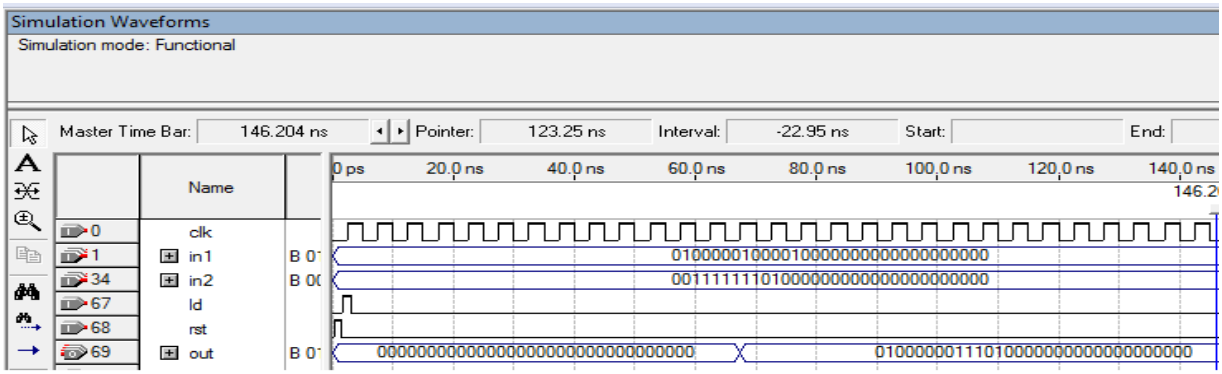


**Figure 3.Simulation waveform of floating point Subtractor**

## 5. Conclusion

Arithmetic unit has been designed to perform two basic arithmetic operations addition, subtraction for single precision floating point numbers. IEEE 754 standard based floating point representation has been used. The unit has been coded in Verilog. Code has been synthesized for the CYCLONE III Family using Quartus II Simulator and has been implemented and verified.

## References

[1] B. J. Hickmann, A. Krioukov, M. A. Erle, and M. J. Schulte, "A Parallel IEEE P754 Decimal Floating-Point Multiplier," in 25th IEEE International Conference on Computer Design .IEEE Computer Society, October 2007, pp. 296–303.

[2] M. F. Cowlishaw, "Decimal Floating-Point: Algorism for Computers," in 16th IEEE Symposium on Computer Arithmetic. IEEE Computer Society, June 2003, pp. 104–111.

[3] ANSI/IEEE STD 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic"1985.

[4] Loucas, L., Cook, T.A., Johnson, W.H., "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs", InProc. IEEE Symposium on FPGAs for Custom Computing Machines, 1999.

[5] Belanovic. P., Leeser, M., "A Library of Parameterized Floating-Point Modules and Their Use", In Proc. Field Programmable Logic and Applications, 2002, pp. 657-666.

[6] Shirazi, N., Walters, A., Athanas, P., "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines", In Proc. IEEE Symposium on FPGAs for Custom Computing Machines 1995, pp. 155-162.

[7] Li, Y., Chu, W., "Implementation of Single Precision Floating Point Square Root on FPGAs", In Proc. 5th IEEE Symposium On Field Programmable Custom Computing Machines 1997, pp .226-232.

[8] Goldberg, D., "What Every Computer Scientist Should Know About Floating-Point Arithmetic", ACM Computing Surveys.

[9] Paschalakis, S., Lee, P., "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.