ISSN 2348 – 9928
IJAICT Volume 2, Issue 5, May 2015 Doi:01.0401/ijaict.2014.02.20 Published Online 05 (05) 2015

VHDL IMPLEMENTATION OF IEEE 754 FLOATING POINT UNIT

Ms. Anjana Sasidharan
Student,
Vivekanandha College of Engineering for Women,
Namakkal, TamilNadu, India.

Mr. M.K. Arun
Electronics Engineer,
TechnoVision,
Pune, India.

Abstract — IEEE-754 specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic in computer programming world. The implementation of a floating-point system using this standard can be done fully in software, or in hardware, or in any combination of software and hardware. This paper propose VHDL implementation of IEEE -754 Floating point unit .In proposed work the pack, unpack and rounding mode was implemented using the VHDL language and simulation was verified.

Keywords—IEEE754, Floating Point Unit, Pack, Unpack, Rounding.

I. INTRODUCTION

The digital arithmetic operations are very important in the design of digital processors and application-specific systems. Arithmetic circuits play an important role in digital systems. With the vast development in the very large scale integration (VLSI) circuit technology, many complex circuits, have become easily implementable today[1]. Algorithms that are seemed to be impossible to implement now have attractive implementation possibilities for the future. This means that not only the conventional computer arithmetic methods, but also the unconventional ones are worth investigation in new designs. The notion of real numbers in mathematics is convenient for hand computations and formula manipulations.

However, real numbers are not well-suited for general purpose computation, because their numeric representation as a string of digits expressed [2] in, say, base 10 can be very long or even infinitely long. Examples include π, e, and 1/3. In practice, computers store numbers with finite precision. Numbers and arithmetic used in scientific computation should meet a few general criteria

- Numbers should have minimum storage requirements
- Arithmetic operations should be efficient to carry out
- A level of standardization, or portability, is desirable– results obtained on one computer should closely matchthe results of the same computation on other computers.

1.1 floating Point Numbers

The term floating point is derived from the meaning that there is no fixed number of digits before and after the decimal point, that is, the decimal point can float. There was also a representation in which the number of digits before and after the decimal point is set, called fixed-point representations [3]. In general floating point representations are slower and less accurate than fixed-point representations, but they can handle larger range of numbers. Floating Point Numbers are numbers that consist of a fractional part. For e.g. following numbers are the 35, -112.5, ½, 4E-5 etc.

Floating-point arithmetic is considered a tough subject by many people. This is rather surprising because floating-point is found in computer systems[4]. Almost every language support a floating-point data type; computers from PC's to supercomputers have floating-point units; most compilers will compile floating-point algorithms from time to time; and every operating system must respond to floating-point exceptions such as overflow. A number representation (called a numeral system in mathematics) specifies some way of storing a number that may be encoded as a string of digits. In computing, floating point describes a system for numerical representation in which a string of digits (or bits) represents a rational number.

The term floating point refers to the fact that the radix point (decimal point, or, more commonly in computers, binary point) can "float"; that is, it can be placed anywhere relative to the significant digits of the number. This position is indicated separately in the internal representation, and floating-point representation can thus be thought of as a computer realization of scientific notation.

1.2 Floating Point Formats

Several different representations of real numbers have been proposed, but by far the most widely used is the floating-point representation. Floating-point representations have a base b

(which is always assumed to be even) and a precision p. If b = 10 and p= 3 then the number 0.1 is represented as 1.00 × 10^-1. In general, a floating point number will be represented as ± d.dd… d × be where d.dd… d is called the Significant and has p digits. More precisely ± d0 d1 d2 ... dp-1 × be represents the number. The term floating-point number will be used to mean a real number that can be represented in the format under discussion. Two other parameters associated with floating-point representations are the largest and smallest allowable exponents, emax and emin. Since there are bp possible significant, and emax – emin + 1 possible exponents, a floating-point number can be encoded in bits, where the final +1 is for the sign bit. The precise encoding is not important for now [5]. There are two reasons why a real number might not be exactly represented able as a floating-point number. The most common situation is illustrated by the decimal number 0.1. Although it has a finite decimal representation, in binary it has an infinite repeating representation.

Thus when b = 2, the number 0.1 lies strictly between two floating-point numbers and is exactly representable by neither of them. A less common situation is that a real number is out of range, that is, its absolute value is larger than b × bemax or smaller than 1.0 × bemin. Floating-point representations are not necessarily unique. For example, both 0.01 × 101 and 1.00 × 10-1 represent 0.1. If the leading digit is nonzero, then the representation is said to be normalized. The floating-point number 1.00 × 10-1 is normalized, while 0.01 ×101 is not. When b = 2, p = 3, emin = -1 and emax = 2 there are 16normalized floatingpoint numbers. The bold hash marks correspond to numbers whose significand is 1.00. Requiring that a floating-point representation be normalized makes the representation unique.

Unfortunately, this restriction makes it impossible to represent zero! A natural way to represent 0 is with 1.0 × bemin-1, since this preserves the fact that the numerical ordering of nonnegative real numbers corresponds to the lexicographic ordering of their floating-point representations [6]. When the exponent is stored in a k bit field, that means that only 2k - 1 values are available for use as exponents, since one must be reserved to 0.Note that in a Floating Point number is a part of the notation ,and different from a floating point multiply operation

1.3 Floating point unit

Floating-point units (FPU**)** colloquially are a math coprocessor which is designed specially to carry out operations on floating Point number. Typically FPUs can handle operations like addition, subtraction, multiplication and division. FPUs can also perform various transcendental functions such as exponential or trigonometric calculations, though these are done with software library routines in most modern processor When a CPU executes a program that is calling for a floating-point (FP) operation, there are three ways by which it can carry out the operation. Firstly, it may call a floating-point unit emulator, which is a floating-point library, using a series of simple fixed-point arithmetic operations which can run on the integer ALU.

These emulators can save the added hardware cost of a FPU but are significantly slow. Secondly, it may use an add-on FPUs that are entirely separate from the CPU, and are typically sold as optional add-ons which are purchased only when they are needed to speed up math-intensive operations [7]. Else it may use integrated FPU present in the system. The FPU designed by us is a single precision IEEE754 compliant integrated unit. It can handle not only basic floating point operations like addition, subtraction, multiplication and division but can also handle operations like shifting, square root determination and other transcendental functions like sine, cosine and tangential function.

1.4 IEEE 754 Standards

IEEE754 standards a technical standard established by IEEE and the most widely used standard for floating-point computation, followed by many hardware and software implementations.

Formats

The standard defines five basic formats, named using their base and the number of bits used to encode them. There are three binary floating-point formats (which can be encoded using 32, 64, or 128 bits) and two decimal floating-point formats (which can be encoded using 64 or 128 bits). The first two binary formats are the ‘Single Precision’ and ‘Double Precision’ formats of IEEE 754-1985, and the third is often called 'quad'; the decimal formats are similarly often called 'double' and 'quad.

Single Precision

The most significant bit starts from the left exponent and mantissa. The storage layout for single precision

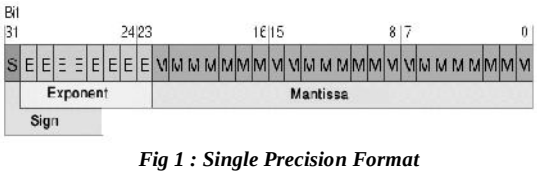

Fig 1 : Single Precision Format

Double Precision

The double precision format helps overcome the problems of single precision floating point. Using twice the space, the double precision format has an 11 bit excess 1023 exponent and a 53 bit mantissa (with an implied H.O. bit of one) plus a sign bit. This provides a dynamic range of about 10±308 and 14-1/2 digits of precision, sufficient for most applications.

Exceptions

The IEEE standard defines seven types of exceptions that should be signaled through a one bit status flag when encountered. They are invalid operation, underflow, overflow, inexact, division by zero, infinity, and zero.

1.5 Rounding mode

Since the result precision is not infinite, sometimes rounding is necessary. To increase the precision of the result .The standard defines the following five rounding rules

- Round to the nearest even which rounds to the nearest value with an even (zero) least significant bit.
- Round to the nearest odd which rounds to the nearest value above (for positive numbers) or below (for negative numbers)
- Round towards positive infinity which is a rounding directly towards a positive infinity and it is also called rounding up or ceiling.
- Round towards negative infinity which is rounding directly towards a negative infinity and it is also called rounding down or floor or truncation.

Guard Digits

One method of computing the difference between two floating-point numbers is to compute the difference exactly and then round it to the nearest floating point number. This is very expensive if the operands differ greatly in size.

II. RELATED WORKS

2.1 Open Floating Point Unit

This was the open source project done by Rudolf Selman. His FPU described a single precision floating point unit which could perform add, subtract, multiply, divide, and conversion between FP number and integer. It consists of two pre-normalization units that can adjust the mantissa as well as the exponents of the given numbers, one for addition/subtraction and the other for multiplication/division operations. . It also has a shared post normalization unit that normalizes the fraction part. The final result after post-normalization is directed to a valid result which is in accordance to single precision FP format. the main drawback is that most of which written in MATLAB and due to this is non synthesizable

2.2 GRFPU

This high Performance IEEE754 FPU was designed at Gaisler Research for the improvement of FP operations of a LEON based systems. It supports both single precision and double precision operands. It implements all FP operations defined by the IEEE754 standard in hardware. All operations are dealt with the exception of demoralized numbers which are flushed to zero and supports all rounding modes.

This advanced design combines low latency and high throughput. The most common operations such as addition, subtraction and multiplication are fully pipelined which has throughput of one CC and a latency of three CC. More complex divide and square root operation takes between 1 to 24 CC to complete and execute in parallel with other FP operations. It can also perform operations like converse and compliment. It supports all SPARC V8 FP instructions. The main drawback of this model is that it is very expensive and complex to implement practically.

III. PROPOSED WORK

The IEEE (Institute of Electrical and Electronics Engineers) has produced a Standard to define floating-point representation and arithmetic the standard brought out by the IEEE come to be known as IEEE 754. The IEEE 754 Standard for Floating-Point Arithmetic is the most widely-used standard for floating-point computation, and is followed by many hardware (CPU and FPU) and software implementations.

In this paper VHDL implementation of some or all arithmetic is carried out using IEEE 754 formats and operations. The current version is IEEE 754-2008, which was published in August 2008; it includes nearly all of the original IEEE 754-1985 (which was published in 1985) and the IEEE Standard for Radix-Independent Floating-Point Arithmetic (IEEE 854-1987.

The standard specifies:

- Basic and extended floating-point number formats.
- Add, subtract, multiply, divide, square root, remainder, and compare operations
- Conversions between integer and floating-point formats.
- Conversions between different floating-point formats.
- Conversions between basic format floating-point numbers and decimal strings.
- Floating-point exceptions and their handling, including non numbers when it comes to their precision and width in bits, the standard defines two groups: basic and extended format.

3.1 *Software used*

In this work VHDL implementation of IEEE 754 standard is used VHDL is a versatile language in which we can effectively done the coding for pack, unpack and rounding and the synthesis result are taken using MODEL SIM XE 11 5.7.In this work coding for pack ,unpack as well s rounding is done a template. Template allows us to indicate specific byte ordering or word ordering. This provides us great deal of flexibility when dealing with external program.

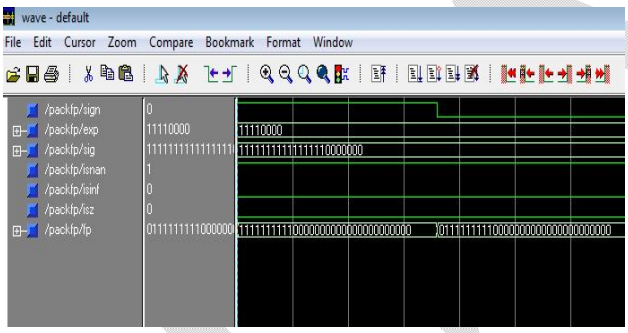

Fig 2 : wave form generated for pack operation

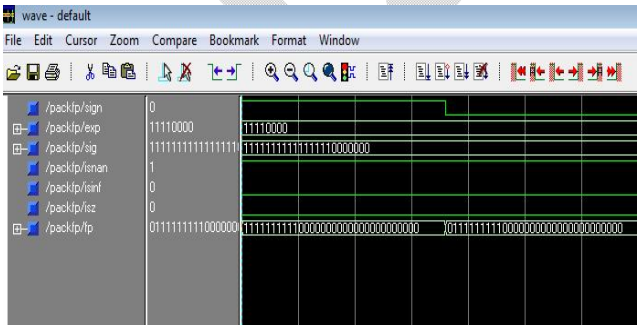

Fig 3 : Wave form generated for unpack function

3.2 Rounding

Simulation result for rounding is shown in the figure. Rounding operations are somewhat more complicated In addition to overflow we can have "underflow" Accuracy can be a big problem – IEEE 754 keeps two extra bits, guard and round for accurate rounding.

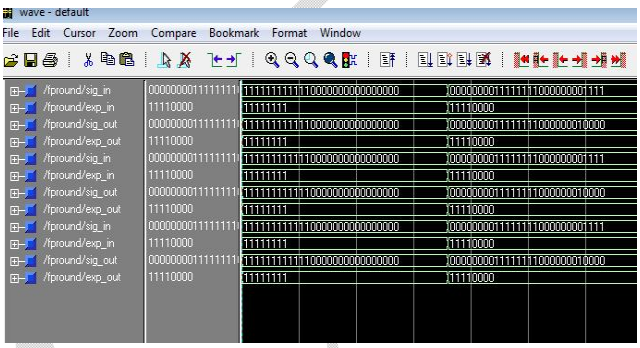

Fig 3 : Waveform generated for rounding

IV. CONCLUSION

Arithmetic unit has been designed to perform Pack, unpack and rounding arithmetic operations, on floating point numbers. IEEE 754 standard based floating point representation has been used. The unit has been coded in VHDL. Code has been synthesized on model sim5.5 and has been implemented and verified on the board successfully. This standard provides a method for computation with floating-point numbers that will yield the same result whether the processing is done in hardware, software, or a combination of the two. The results of the computation will be identical, independent of implementation, given the same input data. Errors, and error conditions, in the mathematical processing will be reported in a consistent manner regardless of implementation.

FUTURE SCOPE

The designed arithmetic unit operates on 32-bit operands. It can be designed for 64- bit operands to enhance precision. It can be extended to have more mathematical operations like addition, subtraction, division, square root etc.

References

[1] Muller, J.-M., Elementary Functions: Algorithms and Implementation, 2nd edition, Chapter 10, ISBN 0-8176-4372-9, Birkhäuser, 2006.

[2] The Unicode Standard, Version 5.0, The Unicode Consortium, Addison-Wesley Professional, 27October 2006, ISBN 0-321-48091-0.54

[3] Stehlé, D., Lefèvre, V., and Zimmermann, P., "Searching worst cases of a one-variable function",IEEE Transactions on Computers, 54(3), pp. 340–346, 2005.

[4] Bruguera, J. D., and Lang, T., "Floating-point Fused Multiply-Add: Reduced Latency for Floating- Point Addition", Proceedings of the 17th IEEE Symposium on Computer Arithmetic, ISBN 0-7695-2366-8,pp. 42–51, IEEE Computer Society, 2005

[5] Cowlishaw, M. F., "Decimal Floating-Point: Algorism for Computers", Proceedings of the 16th IEEE Symposium on Computer Arithmetic, ISBN 0-7695-1894-X, pp. 104–111, IEEE Computer Society,2003.

[6] Overton, M. L., Numerical Computing with IEEE Floating Point Arithmetic, ISBN 0-89871-571-7, Society for Industrial and Applied Mathematics 2001

[7] Demmel, J. W., and Li., X., "Faster numerical algorithms via exception handling", IEEE 1994.