Implementation of IEEE754 Floating Point Multiplier

A Kumutha[1] Shobha. P[2]

[1]*MVJ College of Engineering,*

Near ITPB, Channasandra, Bangalore-67.

[2]*MVJ College of Engineering,*

Near ITPB, Channasandra, Bangalore-67.

Abstract

Floating point is very important in real applications likes automotive power train and body control applications, imaging such as scaling, transforms and font generation in printing, 3D transforms, FFT and filtering in graphics image and Digital signal processing. This paper presents floating point multiplication and division of IEEE754 format. The floating point multiplication and division which improves the performance of the processor speed and area. In this paper we implemented 16X16 floating point multiplier using Xilinx ISE13.2 and modelsim simulator and hardware implementation on Spartan3.

I Introduction

In this paper, suggested a technique for implementing a floating point multiplier integer to floating-point and conversion of floating-point to integer. And further shown How these functions can be implemented, and verified. Here we redesign the floating-point unit. It includes all the software Xilinx13.2 and modelsim, hardware implementation is SPARTAN3 needed to generate custom verilog coded floating-point arithmetic unit. In general, it can be assumed that fixed-point implementations have higher speed and lower cost, while floating-point has higher dynamic range and no need for scaling, which may be attractive for more complicated algorithms.

IEEE 754 Floating Point Standard

IEEE 754 [1] floating point standard is the most common representation today for real numbers on computers. The IEEE has produced a Standard to define floating-point representation and arithmetic. Although there are other representations, it is the most common representation used for floating

point numbers. The standard brought out by the IEEE come to be known as IEEE 754. The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely- used standard for floating point computation, and is followed by many CPU and FPU implementations. The standard defines formats for representing floating-point numbers including negative numbers and denormal numbers special values i.e. infinities and NAN's together with a set of floating-point operations that operate on these values. It also specifies four rounding modes which are round to zero, round to nearest, round to infinity and round to even and five exceptions including when the exceptions occur, and what happens when they do occur. Dealing with fixed-point arithmetic will limit the usability of a processor. If operations on numbers with fractions (e.g. 10.2445), very small numbers (e.g. 0.000004), or very large numbers (e.g. 42.243x105) are required, then a different one representation is in order is the floating-point arithmetic. The floating point is utilized as the binary point is not fixed, as is the case in integer (fixed-point) arithmetic. In order to get some of the terminology out of the way, let us discuss a simple floating-point number, such as -2.42x103. The '-' symbol indicates the sign component of the number, while the '242' indicate the significant digits component of the number, and finally the '3' indicates the scale factor component of the number. It is interesting to note that the string of significant digits is technically termed the *mantissa* of the number, while the scale factor is appropriately called the *exponent* of the number. The general form of the representation is the following:

$$(-1)S * M * 2E(1)$$

Where,

S represents the sign bit.

M represents the mantissa.

E represents the exponent.

Floating point arithmetic :

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely- used standard for floating-point computation, and is followed by many CPU and FPU implementations. The standard defines formats for representing floating-point number (including ±zero and denormals) and special values (infinities and NaNs) together with a set of floating-point operations that operate on these values. It also specifies four rounding modes and five exceptions. IEEE 754 specifies four formats for representing floating-point values: single-precision (32-bit), double-precision (64-bit), single-extended precision (≥ 43-bit, not commonly used) and double-extended precision (≥ 79-bit, usually implemented with 80 bits). Many languages specify that

IEEE formats and arithmetic be implemented, although sometimes it is optional. For example, the C programming language, which pre-dated IEEE 754, now allows but does not require IEEE arithmetic (the C float typically is used for IEEE single-precision and double uses IEEE double-precision).

II Methodology

A multiplication of two floating-point numbers is done in four steps:

- Non-signed multiplication of mantissas: it must take account of the integer part, implicit in normalization. The number of bits of the result is twice the size of the operands.
- Normalization of the result: the exponent can be modified accordingly.
- Addition of the exponents, taking into account the bias.
- Calculation of the sign

Example:

- Let's suppose a multiplication of 2 floating-point numbers A and B, where A=-18.0 and B=9.5.
- Binary representation of the operands:

  A = -10010.0 B = +1001.1.
- Normalized representation of the operands:

  A = -1.001x24 B = +1.0011x23.
- IEEE representation of the operands:

  A=11000001101000000000000000000000

  B=01000001000110000000000000000000

Multiplication of the Mantissas:

We must extract the mantissas, adding an1 as most significant bit, for normalization

10010000000000000000000

10011000000000000000000

- The result of the multiplication is:0x558000000000
- Only the most significant bits are useful: after normalization (elimination of the most significant 1), we get the 23-bit mantissa of the result. This normalization can lead to a correction of the result's exponent
- In our case, we get:

  01010101100000000000000000000000000000000000

Addition of the Exponents:

Exponent of the result is equal to the sum of the operands exponents. A 1 can be added if needed by the normalization of the mantissas multiplication (this is not the case in our example).

- As the exponent fields (Ea and Eb) are biased, the bias must be removed in order to do the addition. And then, we must to add again the bias, to get the value to be entered into the exponent field of the result (Er):

$$Er = (Ea-127) + (Eb-127) + 127$$
$$= Ea + Eb – 127$$

- In our example, we have: what is actually 7, the exponent of the result

  Ea 10000011 Eb 10000010 -127 10000001

  Er 10000110

Calculation of the sign of the result:

- The sign of the result (Sr) is given by the exclusive-or of the operands signs (Sa and Sb):

  $$Sr = Sa \text{ XOR } Sb$$

- In our example, we get:

  $$Sr = 1 \text{ XOR } 0 = 1$$

  i.e. a negative sign

- Composition of the result:

The setting of the 3 intermediate results (sign, exponent and mantissa) gives us the final result of our multiplication:

1 10000110 01010110000000000000000

AxB = 18.0x-9.5 = -1.0101011x2134-127 = -10101011.0 = -171.0_{10}

III Result and discussion:

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	27	768	3%
Number of 4 input LUTs	51	1536	3%
Number of bonded IOBs	140	124	112%
Number of MULT18X18s	4	4	100%

Fig 1: Synthesis Table for Floating point multiplier using IEEE 754.

RTL schematic :

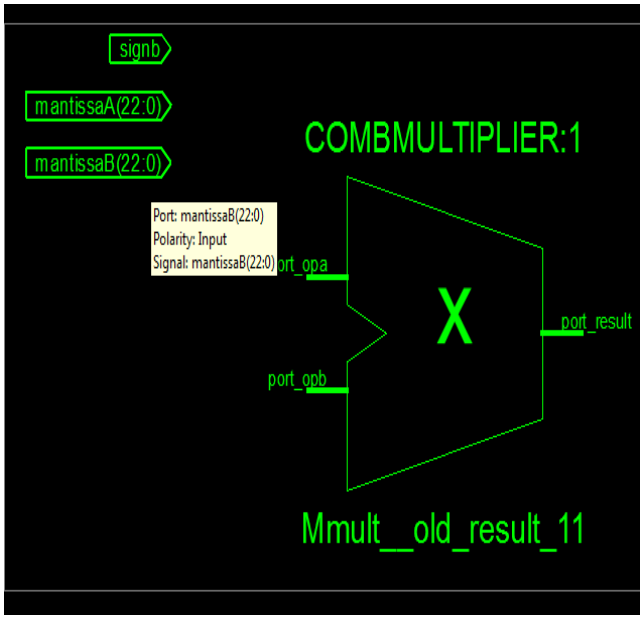

Fig 2: RTL schematic of floating point Multiplier.

Simulation Result :

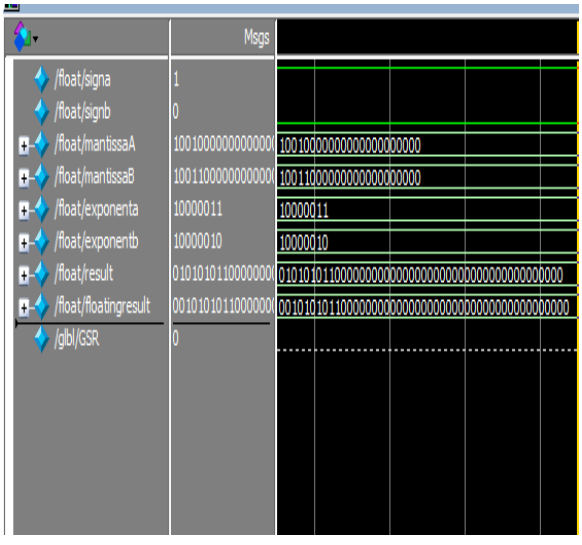

Fig 3: Simulation results of Floating point Multiplier.

IV Conclusion

Floating point multiplier is designed and implemented using Xilinx in this paper. The designed multiplier conforms to IEEE 754 single precision floating point standard. In future work will be implement for scientific calculation.

V Reference

[1] Design and implementation of efficient 32 bit floating point multiplier using Verilog, "*International journal of Engineering and computer science*", ISSN: 2319-7242 Volume 2 Issue 6 June 2013, Page No. 2098-2101.

[2] The VLSI Implementation of A Square Root Algorithm, *Proc. of IEEE Symposium on Computer Arithmetic*, IEEE Computer Society, Press, 1985. Page No. 159-165.

[3] IEEE Floating Point Representation of Real Number, Fundamentals of Computer Science.

[4] P. Karlstrom, A. Ehliar, "High Performance Low Latency Floating Point Multiplier", November 2006.

[5] IEEE standard for binary-floating point arithmetic, ANSI/IEEE Std 754-1985, The Institute of Electrical and Electronic Engineers Inc., New York, August 1985.

[6] An ANSI/ IEEE Standard for Radix-Independent Floating Point Arithmetic, On microprocessor of IEEE computers, October, 1987.

[7] P. Karlstrom, A. Ehliar, High Performance Low Latency Floating Point Multiplier, November 2006.

[8] John. P. Hayes, 'Computer Architecture and Organization', McGraw Hill, 1998.