Proceedings of the 2023 International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/8/20230092

Research and analysis of floating-point adder principle

Fengyuan Yang

School of Materials Science and Engineering, Northeastern University, Shenyang, Liaoning Province, China, 110819

171589845@qq.com

Abstract. With the development of the times, computers are used more and more widely, and the research and development of adder, as the most basic operation unit, determine the development of the computer field. This paper analyzes the principle of one-bit adder and floating-point adder by literature analysis. One-bit adder is the most basic type of traditional adder, besides bit-by-bit adder, overrun adder and so on. The purpose of this paper is to understand the basic principle of adder, among them, IEEE-754 binary floating point operation is very important. So that the traditional fixed-point adder is the basis of the floating-point adder, which can have a new direction in the future development of floating-point adder optimization. This paper finds that the floating-point adder is one of the most widely used components in signal processing systems today, and therefore, the improvement of the floating-point adder is necessary.

Keywords: One-bit adder, floating-point adder, IEEE-754.

1. Introduction

Nowadays, human society has entered the information age, and various information computing and storage technologies are the basis for the development of the information age. Computers, microelectronics and communication technologies related to information technology are already the core technologies that drive social progress.

In the microelectronic processing system, the basic quadratic operations (plus, subtract, multiply, divide) can all be reduced to addition operations, so the adder is a very important arithmetic unit in computer logic computing. In addition to this, the adder can also perform program counting and calculate the effective address[1]. During the operation, the data is operated and stored in the form of 0 and 1. The data types are divided into fixed-point and floating-point. A floating-point representation is widely used. According to Stuart F.Oberman, in floating-point operations, more than 55% of the basic operations are floating-point addition operations, so the floating-point adder is one of the most significant components of the microprocessor[2]. Fixed-point adders are the most basic and commonly used part of various digital systems and are also fully used in floating-point operations. The tail module in floating-point addition is essentially an addition operation of fixed-point numbers. Therefore, understanding the fixed-point adder and designing a high-speed fixed-point adder is essential to improve the performance of floating-point adders.

This paper introduces the most basic one-bit adder in fixed-point adder by literature method and summary and induction method to understand its internal principle and have a preliminary understanding of the principle of adder. In the third part, the floating-point adder is introduced, focusing on IEEE-754 binary floating-point operation standard. The research in this thesis can help beginners to understand floating point adders and help in the subsequent research of high performance floating point adders.

2. One-position adder

One-position adder is the most basic type of adder, and other higher performance adders are studied based on this adder, which includes half adder and full adder.

2.1. Half adder

The input of the half adder does not take into account the feed from the lower bit and the output has the feed from the higher bit.S0 is the sum of the inputs and R1 is the incoming output. Its logical expressions are S0=A0 ⊕ B0, R1= A0 ×B0.

Table 1. Half adder truth table[3].

A0	B0	S0	R1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2.2. Full adder

The full adder is summed from the input from the lower bit Ri and the two inputs Ai and Bi to get a higher bit Ri+1and the current sum Si. Its logical expressions are Si = Ri ⊕ Ai ⊕ Bi, Ri+1= Ri × Ai+ Ri ×Bi+ Ai×Bi.

Table 2. Full adder truth table[4].

Ri	Ai	Bi	Ri+1	Si
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

3. Floating point adder

3.1. The representation of floating point numbers

There are two methods of representing data in the logical operations of computers: fixed-point numbers and floating-point numbers. The reason why floating point numbers are widely used is that floating point numbers use the change of exponents within a certain range to change the position of the decimal point as needed, thus representing larger real number range than the range represented by fixed point numbers[5].

The true value of a floating-point number consists of the ordinal base R (with the implied convention of 2), the exponent-marker E and the mantissa M. The number sign indicates the positive or negative of the number. The exponent-marker E is a fixed-point integer, represented by a complement or shift code, whose number of bits determines the range of the value. The mantissa M is a fixed-point decimal number, represented by the original or complementary code, and its digits determine the precision of the number.

The IEEE-754 binary floating-point standard was developed by the Institute of Electric and Electronics Engineers in 1985 and has been the industry standard for floating-point operations ever since[6].

3.2. IEEE-754 binary floating point operation

3.2.1. Four formats. The IEEE-754 standard has four formats: Single precision floating-point numbers, Double precision floating-point numbers, extended double-precision floating-point (SPARC), and extended double-precision floating-point (x86).

Table 3. Bit distribution of four types of float[5]

Format	Total number of bits	Symbol bit S	Exponent-marker E	Mantissa M
Single-precision floating-point	32 bit	1 bit	8 bit	23 bit
Double-precision floating-point	64 bit	1 bit	11 bit	52 bit
Extended double-precision floating-point (SPARC)	128 bit	1 bit	15 bit	112 bit
Extended double-precision floating-point (x86)	80 bit	1 bit	15 bit	63 bit+1 bit (Explicit leading significant digits)

3.2.2. Single-precision floating-point. Single-precision floating-point numbers have three intervals: the 23-bit mantissa M, the 8-bit exponent-marker E, and the 1-bit sign S. Bits 0 to 22 in the 32-bit field are the mantissa M, where bit 0 is the least significant bit of the mantissa, and the first bit to the left of the mantissa decimal point must be 1. In floating-point addition calculations, the calculation is generally achieved by a specific frame shift. Since the first digit of the specification is 1, the 1 before the decimal point can be ignored when saving the trailing digit, and an extra binary bit is added to the mantissa part to improve accuracy. Bits 23 to 30 are the exponent-marker E, where bit 23 is the least significant bit of the exponent-marker. 8 bits of the exponent-marker can represent exponential values between 0 and 255. The exponent can be either positive or negative. When the exponent is negative, a deviation value (Bias=127) is introduced and the sum of it and the exponent value is used as the value stored in the exponent field. The 31st bit is the sign bit, where negative numbers are represented by 0 and positive numbers are represented by 1.

3.2.3. Double-precision floating-point. A double precision floating point number has 64 bits, including 53 bits for the mantissa M, 11 bits for the exponent-marker E, and one bit for the sign S. Bits 0 through 51 are the mantissa M, and bit 0 is the least significant bit of the mantissa. Bits 52 to 62 are the 11-bit exponent-marker E, and bit 52 is the least significant bit of the exponent-marker. 11 bits of the exponent can represent exponent values between 0 and 2047. When the exponent is negative, a deviation value (Bias=1023) is introduced, and the sum of the deviation value and the exponent value is used as the value stored in the exponent field. The 63rd bit is the sign bit, where positive numbers are expressed as 0 and negative numbers are expressed as 1.

3.2.4. Extended double precision floating point (SPARC). The SPARC floating-point format is a quadruple precision format that occupies four 32-bit fields: 112 mantissa bits, 15 exponent-marker bits, and 1 symbol bit. Bits 0 through 11 are the mantissa M, bits 112 through 126 are the exponent-marker E, and bit 127 is the sign bit S.

3.2.5. Extended double precision floating point(x86). The extended double precision floating point format (x86) numbers three consecutive 32 bits by 96 bits. Bits 0 through 63 store the 64-bit mantissa, the 15-bit exponent-marker is stored in bits 64 to 78, and the sign bit is stored in bit 79. However, the format actually uses only 80 bits, i.e., the higher 16 bits of the highest 32 bits of the address are not used by the structure.

3.3. Traditional floating point addition algorithm

The basic algorithm of floating-point addition is to perform the operation of adding the trailing numbers while ensuring that the exponent markers involved in the operation are the same size.

Suppose there are two floating point numbers A and B in double precision floating point format. The operation flow is as follows.

a. Exponential subtraction: Compare the exponents of A and B and subtract them to get the absolute value of the difference d. The absolute value d is the number of digits shifted by the smaller mantissa in the following exponential alignment.

b. Exponential Alignment: Shift the mantissa of the smaller number to the right by d places so that the exponents of the two numbers are aligned, i.e., the two numbers have the same exponent[7].

c. Add the valid digits of the mantissa: Add the valid mantissa of the 2 numbers according to their signs, provided that A and B have the same exponent.

d. Conversion: The result obtained from the calculation is expressed in the form of a complementary code. When the effective digit is negative, it is converted to the form of negative sign - mantissa, and the trailing digit is expressed in the true code.

e. Mantissa specification: Shift the mantissa left or right according to the position of the first 1 in the higher digit, and adjust the ordinate so that the mantissa format becomes 1.f[7].

f. Rounding: Round the final result, if the ingestion causes non-specification, the valid bit needs to be shifted 1 bit to the right, and the ordinal code of the larger number is added 1.

4. Conclusion

This paper mainly discusses the principle of one-bit adder and IEEE-754 binary floating-point arithmetic standard and analyzes the traditional algorithm of floating-point addition, especially the four formats in floating-point arithmetic and their respective specific formats and differences between each other. However, this thesis only introduces the basics of these arithmetic standards through a brief summary of other literature and books. Still, due to time constraints, the actual research design of the floating-point adder has not been mentioned yet. In future research, people can specifically explore improving the existing floating-point adder and research a high-performance, low-power floating-point adder.

References

[1] WANG Dong,LI Zhentao,MAO Erkun,LI Baofeng. CMOS VLSI Design (3rd Edition), Beijing: China Electric Power Press, 2008
[2] Stuart F.Oberman.Design issues in high performance floating point arithmrtic Units [D],Standford University, Degree of Doctor of Philosophy, 1996.

[3] JI Chao, LI Tuo, ZOU Xiaofeng & ZHANG Lu. (2022). Design of combinatorial logic circuit based on memristor. Semiconductor Technology(08),649-659. doi:10.13290/j.cnki.bdtjs.2022.08.010.

[4] Dai Guangzhen, Zhao Zhenyu, Song Xingwen, Han Mingjun & Ni Tianming. (2023). Memristor hybrid logic circuit design and its application. Science in China: Information Science (01), 178-190. doi:.

[5] WANG Dayu. (2012). Research and Design of High-Performance Floating-Point Adders (Master's Thesis, Nanjing University of Aeronautics and Astronautics https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD201301&filename=1012041598.nh).

[6] IEEE Std 754-1985:IEEE Standard for Binary Floating point Arithmatic,IEEE,1985.

[7] FENG Wei. (2009). Optimization Design of a Fast Floating-Point Adder (Master's Thesis, University of Science and Technology of China).https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD2010&filename=2010018994.nh