Table of Contents

Introduction

Single-precision floating-point multiplication is a mathematical operation performed on two single-precision floating-point numbers. Also known as floats [1], are a common way of representing real numbers in computers and are used in various applications such as scientific simulations, engineering calculations, and graphics rendering.

Represented using 32 bits, with one bit used to represent the sign, eight bits used to represent the exponent, and 23 bits used to represent the significand (also known as the mantissa) [2]. The exponent represents the power of 2 by which the significand is multiplied, and the sign bit indicates whether the number is positive or negative.

Multiplying two single-precision floating-point numbers involves multiplying their significands and adding their exponents. However, since the result of the multiplication may have more bits than can be represented in a single-precision floating-point number, the result needs to be rounded and normalized to fit into the 32-bit format [3]. The rounding and normalization process involves shifting the significand and adjusting the exponent to maintain the correct value.

The multiplication operation is a fundamental operation in many numerical algorithms and is implemented in hardware in modern CPUs and GPUs. Efficient implementation of single-precision floating-point multiplication is crucial for achieving high performance in these applications.

IEEE 754 Single-Precision Floating Point Format

The IEEE 754 standard defines the format for representing single-precision floating-point numbers using 32 bits [4]. The format consists of three fields: the sign bit, the exponent field, and the significand field.

The sign bit is the leftmost bit, with a value of 0 for positive numbers and 1 for negative numbers. The exponent field consists of the next 8 bits and represents the exponent of the number. The exponent is biased by 127, meaning that a stored exponent of 0 represents a true exponent of -127 and a stored exponent of 255 represents a true exponent of +128. This biasing allows for efficient comparison of exponents, as the comparison can be performed on the stored exponent values directly.

The significand field consists of the remaining 23 bits and represents the fractional part of the number. The significand is normalized, meaning that the leftmost bit is always 1, and is represented as a binary fraction with the decimal point immediately to the right of the leftmost bit. The remaining bits represent the fractional part of the number to varying degrees of precision, with the precision decreasing as the distance from the leftmost bit increases.

The value represented by a single-precision floating-point number can be calculated using the following formula: (-1)^s * 2^(e-127) * (1 + f) where s is the sign bit (0 for positive, 1 for negative), e is the exponent field biased by 127, and f is the fractional part represented by the significand field.

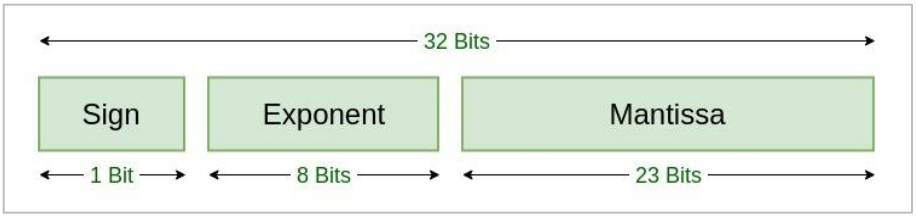


Figure 1(IEEE-754 Floating Point Representation.

Floating-point arithmetic operations

1. **Addition**: The addition of two floating-point numbers involves aligning the decimal points of the numbers and adding their significands while considering their exponents. If the exponents are different, one of the significands is first shifted to align the decimal points. After the addition is performed, the result is typically rounded and normalized to fit into the specified format.

2. **Subtraction**: The subtraction of two floating-point numbers is similar to addition, but with the sign of one of the numbers flipped before the addition is performed.

3. **Multiplication**: The multiplication of two floating-point numbers involves multiplying their significands and adding their exponents. Since the product of the significands may have more bits than can be represented in the specified format, the result needs to be rounded and normalized to fit into the format.

4. **Division**: The division of two floating-point numbers involves dividing their significands and subtracting their exponents. Like multiplication, the result may need to be rounded and normalized to fit into the specified format.

5. **Square root**: The square root of a floating-point number is calculated using an iterative algorithm that approximates the square root to a specified degree of accuracy.

6. **Exponentiation**: The exponentiation of a floating-point number involves raising it to a specified power, which can be either a floating-point or an integer value. The result is typically rounded and normalized to fit into the specified format.

Each of these operations can introduce errors due to limitations in the precision of the floating-point format. To minimize the impact of these errors, it is important to choose an appropriate format and use appropriate techniques for error analysis and mitigation.

The process involves:

A. Aligning the binary points of the mantissas.
B. Adjusting the exponents.
C. Performing the operation.
D. normalizing the result
E. rounding if necessary.

Single-precision Floating-point Multiplication design representation

Floating-point multiplication is the most frequently utilized arithmetic operation and is a crucial component in various engineering applications, including signal processing, video processing, image processing, and more. In Addition, accounts for approximately 37% of the floating-point operations in benchmark applications.

involves combining hardware and software components to implement the algorithm for performing single-precision floating point multiplication, including aligning the mantissas, adding the exponents, multiplying the mantissas, normalizing the result, and rounding if necessary.

The design must consider special cases like overflow, underflow, and NaN inputs. Additionally, it mentions various design considerations such as precision, accuracy, performance, hardware complexity, power consumption, and cost.

Multiplication algorithm:

1. **Sign bit:** Determine the sign of the result by performing an XOR operation on the signs of the two operands.

2. **Exponent**: Add the exponents of the operands together.

3. **Mantissa:** Multiply the mantissas of the operands together.

4. Normalization: Adjust the exponent and mantissa so that the binary point is in the correct position. If the mantissa overflows, shift the binary point to the right and increment the exponent. If the mantissa underflows, shift the binary point to the left and decrement the exponent.

5. **Rounding:** Round the result to fit within the specified format.

Example:

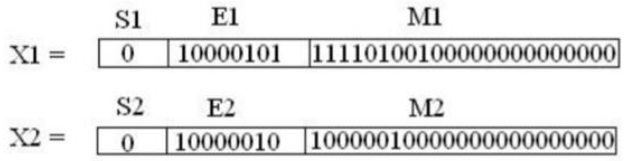


Fig 10

1) Find the sign bit by xor-ing sign bit of A and B

i.e. Sign bit = > (0 xor 0) => 0

2) Multiply the mantissa values including the "hidden one". The Resultant product of the 24 bits mantissas (M1 and M2) is 48bits (2 bits are to the left of binary point)

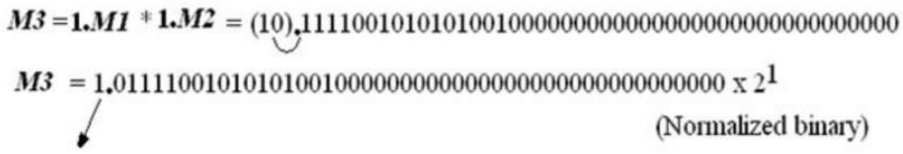


Fig 11

If M3 (48) = "1" then left shift the binary point and add "1" to the exponent else don't add anything. This normalizes the mantissa. Truncate the result to 24 bits. Add the exponent "1" to the final exponent value.

3) Find exponent of the result. = E1 + E2 -bias + (normalized exponent from step 2) = (10000101)2 + (10000010)2 - bias +1 = 133 + 130 - 127 + 1 = 137.

Add the exponent value after normalization to the biased exponent obtained in step 2. i.e. 136+1 = 137 => exponent value.

Note: The normalization of the product is simpler as the range of M_A and M_B is between 1 - 1.9999999.and the range of the product is between (1 - 3.9999999)Therefore a 1 bit shift is required with the adjust of exponent. So we have found mantissa, sign, and exponent bits.

4) We have our final result i.e.

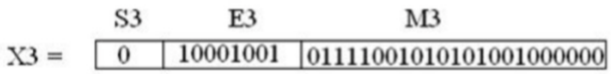


Fig 12

If we convert this to decimal we get X=1509.3203125

Initial Multiplication design:

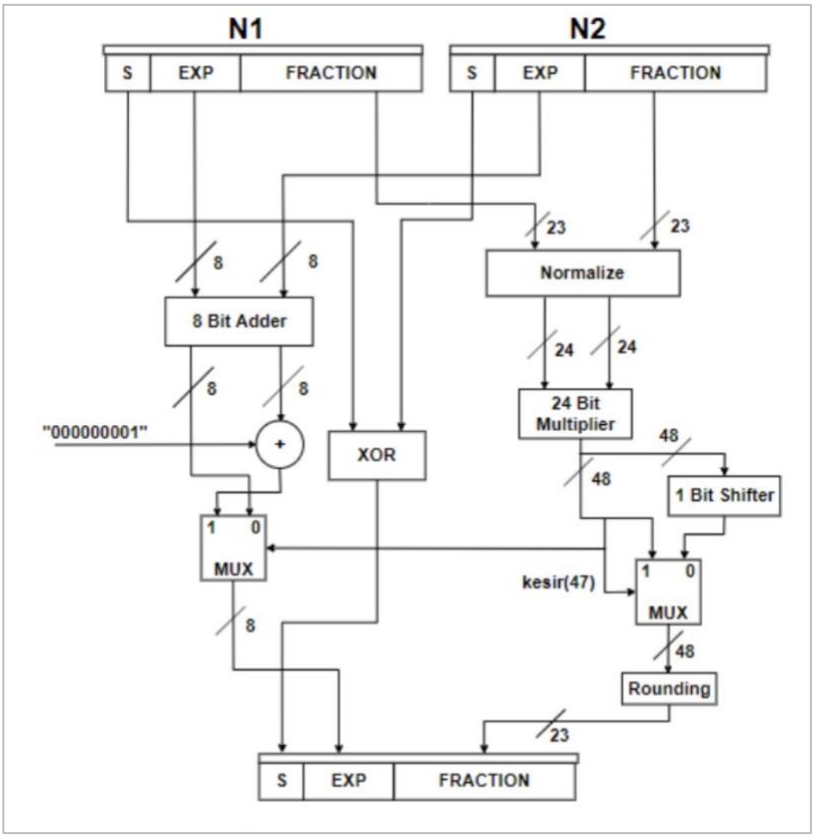


Figure 2 Multiplication Design [3].

Floating-point multiplication is performed as follows:

1. Checked whether N1 and N2 have a special case of being zero or infinity.
2. For pre-normalization, ‘1’ is added to the far left of the fractions.
3. The 24-bit multiplier module give a 48-bit number output as a result of multiplication.
4. Exponent values of the numbers are added on the Adder.
5. Subtract “01111111” from the sum of exponents.
6. Sign bits of the numbers are passed through the XOR gate.
7. If the MSB of multiplier result is bit ‘1’, “00000001” is added to exponent value of the result and fraction value is shifted 1 bit to the right, and 0 is continued without doing anything.
8. Rounding fraction value.

The implementation:

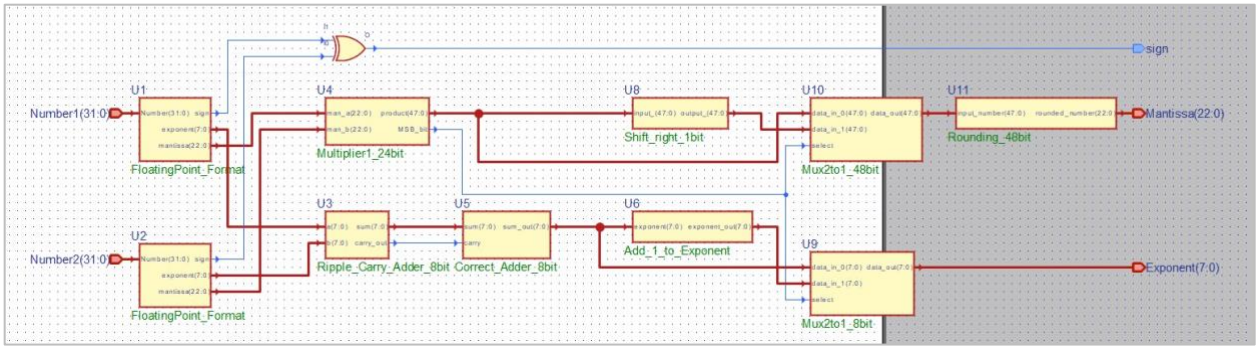
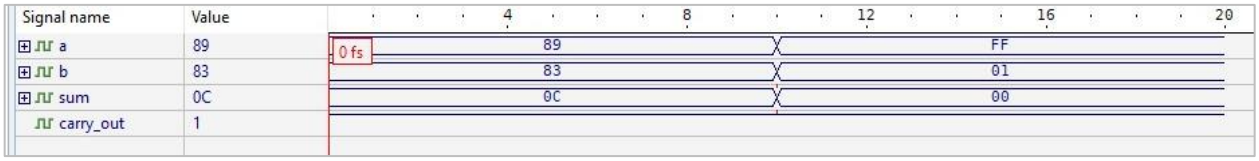


Figure 3: IIIE 754 Single-precision Floating-point Multiplication Design.

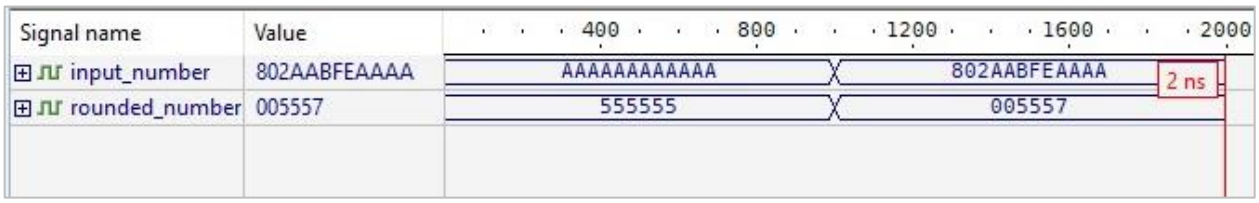
- "**Number1**" and "**Number2**" are 32-bit binary IEEE754 single-precision floating-point numbers.
- **XOR gate:** It takes the "sign" bits from the two floating-point numbers and obtains the final sign result.
- **Multiplier 24-bit:** It takes the two mantissa numbers and multiplies them after adding the hidden bit.
- **Ripple Carry Adder 8-bit:** It adds two exponents and obtains the sum and carry.
- **Correct Adder 8-bit:** It corrects the overflow results from the Ripple Carry Adder and subtracts "01111111" from the sum of exponents if there is a carry.
- **Shift Right 1-bit and Add 1 to Exponent:** If the MSB (Most Significant Bit) of the multiplier result is '1', it adds "00000001" to the exponent value of the result and shifts the fraction value 1 bit to the right, without changing the value of 0.
- **Rounding:** It rounds the mantissa value up or down based on bit 46.

Simulations:

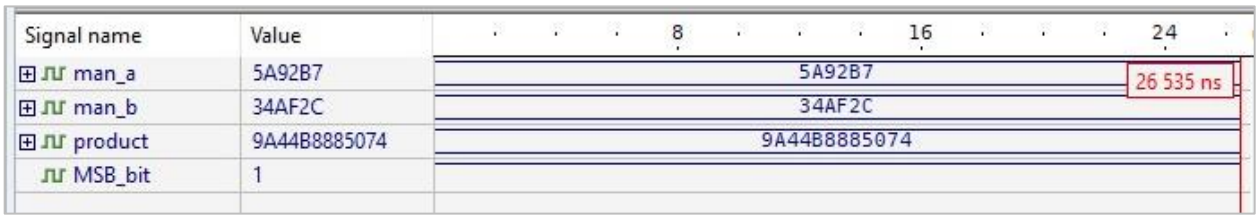
Ripple Carry Adder



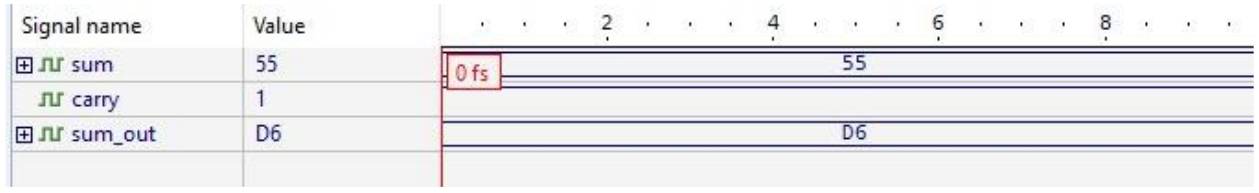
Rounding 48-bit



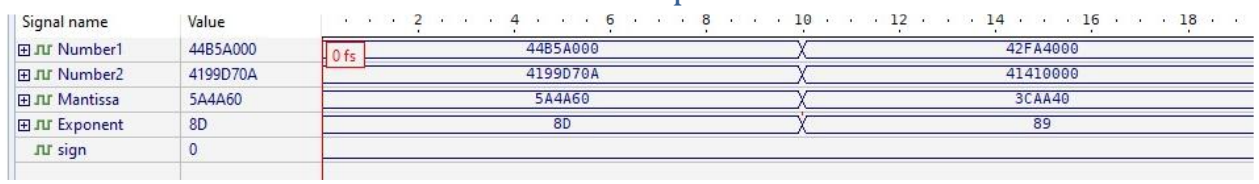
24bit Multiplier



Correct Adder



IEEE754 Multiplication



Conclusion:

Single-precision floating-point multiplication is a common operation in computer arithmetic that involves multiplying two single-precision floating-point numbers.

There are several challenges that need to be addressed to ensure accurate and reliable results. These challenges include handling special cases such as NaN (Not-a-Number), infinity, overflow, and underflow.

Due to the limited precision of single-precision floating-point numbers, this operation can introduce rounding errors and other numerical artifacts that can affect the accuracy of the results.

References:

[1] L. G. Marcus and S. T. Geetam, "Comparative Review of Floating-Point Multiplier," 2019. [Online]. Available: https://gvpress.com/journals/IJHIT/vol12_no2/4.pdf.

[2] G. Aman, M. Satyam, J. M. Vincent, L. Keck-Voon, B. Arindam, J. Henry and T. Budianto, "Low Power Probabilistic Floating Point Multiplier Design," 18 August 2011. [Online]. Available: https://ieeexplore.ieee.org/document/5992502/authors#authors.

[3] "Implementation and Design of 32 Bit Floating-Point ALU on a Hybrid FPGAARM Platform," 2021. [Online]. Available: https://www.acapublishing.com/dosyalar/baski/BEN_2019_16.pdf.

[4] "IEEE 754 Standard Floating Point Numbers," [Online]. Available: https://www.rfwireless-world.com/Tutorials/floating-point-tutorial.html.