https://doi.org/10.37420/j.eeer.2023.001

A Fast Fixed-Point Logarithmic Functions Based on ARM Microprocessor

Ziyun Li[1], Qiang Zhang [2,*], Huifang Yang[3]

[1] Zhengzhou University-Apex Institute of Integrated Circuit Design and Application, Zhengzhou University, Zhengzhou, Henan,China, 1778389206@qq.com

[2] Zhengzhou University-Apex Institute of Integrated Circuit Design and Application, Zhengzhou University, Zhengzhou, Henan,China, zq_28@126.com

[3] Zhengzhou University-Apex Institute of Integrated Circuit Design and Application, Zhengzhou University, Zhengzhou, Henan,China,2961441566@qq.com

***Corresponding author,E-mail: zq_28@126.com**

Abstracts

The fixed-point mathematical library refers to a highly optimized collection of high-precision mathematical functions primarily used for fast and high-precision real-time calculations. It can execute equivalent code written in the C language with floating-point format faster, while maintaining considerable accuracy. Current mainstream fixed-point mathematical libraries face issues such as being non-open source, having unknown models, incomplete basic mathematical operation functions, and insufficient precision. Therefore, designing an open-source, fast, more optimized, and high-precision fixed-point mathematical library will have a groundbreaking impact. It can not only play a crucial role in industrial control algorithms but also eliminate the dependency of domestic fixed-point MCUs on foreign library functions.

This paper conducts research on fixed-point arithmetic for logarithmic functions, designs, and improves the implementation of fast logarithmic functions.Taking Q12 as an example, Simulation experiments and MCU experiments show that, in the Q12 format, the highest precision that can be represented by a 32-bit fixed-point number is 0.000244141. From the test results, it can be observed that for 98.15% of the test data, the computational errors of the two mathematical libraries are both smaller than the highest precision value. Both two logarithmic function computations can maintain good computational accuracy within the ARM microprocessor. In terms of computational speeds, the average computation cycle of the fixed-point logarithmic function designed in this paper is reduced by 31.88% compared to the counterpart. This substantial improvement in computational speed, while ensuring computational accuracy, enhances the performance of fixed-point logarithmic operations based on ARM microprocessors.

Keywords: Logarithm Function, Fast Algorithm, Fixed-point Processors

1 INTRODUCTION

Chips are miniature, large-scale integrated circuits that are the core technology and main driving force of the information revolution. For a long time, China has had a low self-sufficiency rate in chips, with the majority of high and mid-end chips being imported from abroad. There is a high dependence on foreign core technologies, and the chip industry lacks international competitiveness. Despite the increased efforts in recent years to boost chip technology research and development in China, achieving effective transformation of many technological achievements, the country still faces the current situation of "high-end industry, low-end technology," and a bottleneck in the development of core chips. There are still noticeable shortcomings in the development of the chip industry. With the deep implementation of China's innovation-driven development strategy and in order to accelerate the escape from the predicament of being technology-dependent in chip technology, and achieve the goal of independent manufacturing of high-tech chips, the Chinese government has formulated and introduced a series of encouraging and supportive policies. A policy system to serve the development of the chip industry has been basically established(Yang,K.R.,Yan,C.L., & Shi,K.2023).

For a microcontroller unit (MCU), it needs to possess stronger processing and computational capabilities. Integrated software and hardware are the current trend, and it is not only reliant on hardware design but also crucially dependent on software support (Song, W.N.,Xu, D.J.,&Chen, L. 2023). In the process of applying fixed-point MCUs, the improvement in computational capabilities primarily relies on fixed-point mathematical libraries. A fixed-point mathematical library refers to a highly optimized collection of high-precision mathematical functions. It is mainly used for fast and high-precision real-time calculations, allowing for faster execution compared to equivalent code written in the floating-point format of the C language. Moreover, it maintains considerable accuracy, which is particularly important for real-time control systems with demanding design requirements (Ding,Y.,&Ji,P.F. 2023). Through the use of fixed-point mathematical libraries, C language programmers can seamlessly migrate floating-point algorithms to fixed-point arithmetic, significantly reducing development time for applications such as DSP (Digital Signal Processing).

Fixed-point mathematical libraries typically provide various mathematical functions and algorithms, such as addition, subtraction, multiplication, division, trigonometric functions, exponential functions, logarithmic functions, etc. These functions and algorithms are specifically designed for processing fixed-point numbers to ensure both precision and efficiency. As software support for fixed-point microprocessors, the source code and high-precision, high-speed fixed-point algorithms for fixed-point mathematical libraries have already been mastered abroad. However, domestically, commonly used fixed-point mathematical libraries are dynamic link libraries (DLLs) with invisible source code and unclear computational principles, resulting in core technologies still being controlled by others. In order to achieve the domestic substitution of "bottleneck" technologies, it is imperative to design and develop high-precision, fast fixed-point mathematical library functions based on domestically produced microprocessors. Fixed-point mathematical operations depend on the implementation of fixed-point mathematical libraries, and in fixed-point mathematical operations, exponentiation, exponentials, and logarithmic operations are relatively complex. General fixed-point microprocessors lack hardware support for division, requiring a series of shift and conditional subtraction operations, or implementing related operations through lookup tables. The former has a long computational cycle, consuming a considerable amount of software runtime, while the latter, through space memory trade time, leads to large memory usage by the microprocessor. Both methods significantly impact the performance of fixed-point operations. Therefore, designing high-precision and fast fixed-point mathematical library functions for exponentiation, exponentials, and logarithmic operations holds significant theoretical value and practical significance.

2 MATERIALS AND METHODS

2.1 Fixed-Point Numbers and Floating-Point Numbers

Fixed-point numbers are a representation with a fixed position for the decimal point, meaning a number in

binary uses a fixed number of bits to represent digits before and after the decimal point. The precision and range of fixed-point numbers are constrained by the number of bits, and the position of the decimal point usually needs to be manually specified. The fixed-point data format is illustrated in Figure 1, where the first bit of a 32-bit fixed-point number is the sign bit, with 0 indicating a positive number and 1 indicating a negative number. The subsequent data section consists of the binary integer part and the binary fractional part. The entire numerical value represents the sum of the integer and fractional parts.

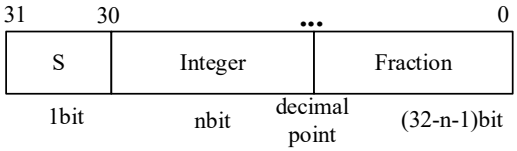

Figure 1. 32-bit fixed-point data format.

Since the position of the decimal point in fixed-point numbers can vary, fixed-point numbers can be defined in different data formats based on the position of the decimal point, which is also referred to as scaling. Data scaling includes Q format and S format. Q represents the bit width occupied by the fractional part, while S represents the bit width occupied by both the integer and fractional parts. Taking a 32-bit fixed-point number as an example, the storage format for Q format is denoted as Qn, where n represents the number of fractional bits; the storage format for S format is denoted as Sm.n, where m represents the number of integer bits, n represents the number of fractional bits, and the sum of m and n must be 31 (Hou,L.Z.2020).

The numerical range and precision of values represented in fixed-point formats are contradictory; the larger the numerical range, the lower the precision. For a 32-bit data, the format with the largest numerical range is the Q0 format, where the decimal point is placed after the 0th bit, suitable for representing pure integers. The format with the highest precision is the Q31 format, where the decimal point is placed after the sign bit, suitable for representing pure decimals. The numerical ranges and precisions of other Q formats are shown in Table 1. The choice of data format is essentially a trade-off between the actual application requirements, seeking a compromise between numerical range and precision.

Table 1. Range of fixed-point data representation.

Q	S	Numerical representation range (decimal)	
		Minimum value	Maximum value
30	1.30	-2	1.999 999 999
29	2.29	-4	3.999 999 998
28	3.28	-8	7.999 999 996
…	…	…	…
3	28.30	-268435456	268435455.875 000 000
2	29.20	-536870912	536870911.750 000 000
1	30.10	-1073741824	1073741823.500 000 000
0	31.00	-2147483648	2147483648.000 000 000

Q format is used for data scaling in this paper. Microprocessors store fixed-point numbers in two's complement form, where the original code of a positive number is equal to its two's complement. For a negative number, its two's complement is obtained by inverting each bit of the numerical part while keeping the sign bit unchanged and then adding 1. Therefore, in the Qn format, the numerical value of a 32-bit fixed-point number from section 2.1 is represented as shown in equation (1).

$$X = \sum_{i=0}^{30} X_i \cdot 2^{i-n} - S \cdot 2^{31} \#(1)$$

Floating-point numbers use scientific notation, where a number is represented in the form of an exponent

and a mantissa. The exponent indicates the number of positions the decimal point is moved to the left or right, while the mantissa represents the value before the decimal point. Floating-point numbers can represent a wide range of real values and have higher precision, but their computational speed is relatively slower, and they require more storage space.

Using the following floating-point representation format to express a 32-bit floating-point number, this format is a subset of the IEEE 745 standard. In addition to meeting the requirements for real number representation, it also simplifies processing. The total bit width is , and the format is shown in Figure 2. A number N is represented by a triplet {S, E, M}.

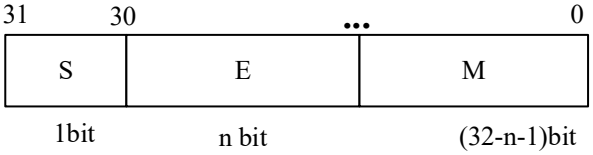

Figure 2. Example of triplet representation.

S (Sign) represents the sign bit of N, with a bit width of 1 bit. The corresponding value s satisfies: when ; when ; when , s can be 0 or 1. E (Exponent) represents the exponent bits of N, located between S and M with a default bit width of . E adopts biased representation with a fixed bias of 16. The numerical range of E is 0 to 31, but its actual representation range is -16 to 15. M (Mantissa) represents the mantissa bits of N, located at the end of N with a default bit width of 10 bits. The decimal point is at the far left, and there is an implicit bit to the left, which is fixed at 1. Assuming M is "0101100111," its corresponding actual value in binary is "1.0101100111." The numerical calculation formula for N is as follows (Hou,L.Z. 2020):

$$N = (-1)^s \cdot \left(1 + \frac{M}{2^{m-n-1}}\right) \cdot 2^{E-16} \#(2)$$

As seen, its minimum positive value is $min = 1.0 \times 2^{-16} \approx 1.5259 \times 10^{-5}$ and the maximum positive value is $max = (1.1111111111)_{bin} \times 2^{15} = 65504$.

The precision and range of fixed-point numbers are limited by the number of bits, and the position of the decimal point usually needs to be manually specified. Compared to floating-point numbers, fixed-point numbers have faster computational speed and require less storage space. However, they have a smaller numerical range. For extremely large or small values, scientific notation or exponential notation may be needed for representation. Therefore, floating-point and fixed-point numbers each have their own advantages and disadvantages, suitable for different application scenarios. In situations where fast numerical calculations are needed, and the numerical range is small, fixed-point numbers are a preferable choice. Especially for MCUs without FPU units in the market, fixed-point arithmetic has significant advantages over traditional mathematical libraries.

2.2. Algorithm Research of Logarithmic Functions

Logarithmic operations are widely used in fields such as particle filtering, RBF neural networks, image processing, signal processing, etc. Currently, there are several methods for implementing logarithmic operations, including table lookup, Taylor series iteration, and polynomial approximation.

(1)Look-Up Table

Similar to the exponential function, this method involves deriving a formula that establishes the relationship between input values and the normalized range. The final result is obtained through simple calculations and table lookup. This algorithm requires balancing table size and precision based on the applicable scenario.

(2)Taylor series iteration method

Similar to the exponential function, if the input data for the logarithmic function is a very small value, the convergence speed of the polynomial is fast. However, if the input value is close to 1, achieving accurate results requires a high degree of polynomial expansion, involving a considerable amount of multiplication and

addition operations, making the implementation complex. If the logarithmic value is calculated using the Taylor series expansion, since , then . The input value needs to be transformed through operations to the interval [0, 2], and then the Taylor expansion can be applied.

(3)Linear approximation method

Linear approximation is a method of fitting and calculating a nonlinear function using a simple linear function. For example, approximating the value of with a pure fraction . The key to linear approximation lies in establishing an appropriate linear function, determining the interval divisions, and compensating for errors. MATLAB can be used for simulation analysis to obtain a suitable approximation model.

(4)Cordic algorithm

The Cordic algorithm is a universal iterative algorithm that achieves various elementary function evaluations by controlling vector rotations and orientation operations in linear, circular, and hyperbolic coordinate systems (Mopuri, S.,Acharyya, A.2020). The Cordic algorithm for logarithmic operations is based on the hyperbolic system, as illustrated in Figure 3.

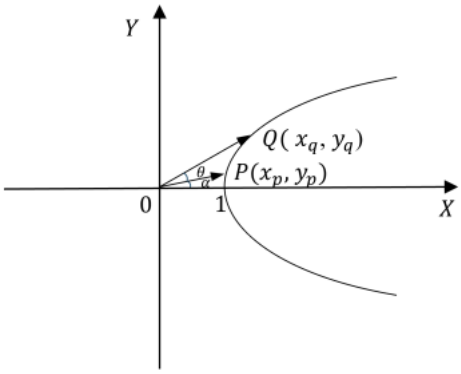

Figure 3. Vector rotation diagram of rotation mode for hyperbolic systems.

The table lookup method is simple to implement, but the required storage units grow exponentially with increasing precision or input range. It consumes a significant amount of storage resources for scenarios requiring high precision and a wide input range. Taylor series iteration involves complex computations and numerous multiplications, making it intricate in design and not conducive to engineering implementation. The linear approximation method is complex, has limited calculation precision, and requires error correction, making it unsuitable for high-precision applications. Traditional Cordic algorithm, due to its simplicity and ease of implementation, converges slowly in cases where high precision is required. Therefore, achieving resource-efficient, high-precision logarithmic operations through simple calculations requires further exploration by combining classical algorithms with the latest research findings.

A logarithmic algorithm proposed by Clay S. Turner, which utilizes repeated squaring and shifting to find the binary logarithm. In the initial iteration phase of the algorithm, it assumes that is within the range and requires normalization of . This normalization is achieved through simple consecutive division/multiplication by 2, implemented by binary shifting. The calculations of division/multiplication yield the logarithmic result (integer part). Adding the decimal part obtained through repeated squaring completes the entire logarithmic result (Turner, C.S.2010). The main process of this algorithm is illustrated in Figure 4.

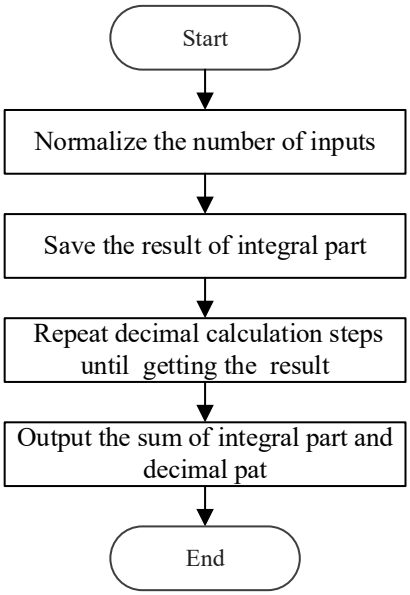

Figure 4. Flow chart of Clay Turner logarithmic algorithm.

Li Gang et al. proposed an improved algorithm based on the Cordic algorithm, which decomposes the logarithmic function into a series of addition, subtraction, and shifting operations. This algorithm is well-suited for the implementation of binary logarithmic operations, ensuring convergence while expanding the input range by introducing negative iterations (Li,G., Wan,L.,& Lin,L.(2008. Initially, the algorithm initializes , where equals the input data plus 1, equals the input data minus 1, and equals 0. After preprocessing, and undergo amplification (achieved through left shifts). Based on the configured number of iterations, the algorithm performs shifts, addition, and subtraction operations on , and . The process of obtaining involves the use of a set of regular fixed values of inverse hyperbolic tangent functions, which can be quantified through Matlab calculations. After the iterations are completed, the computed result is multiplied by 2, yielding the logarithmic result. The main process of this algorithm is depicted in Figure 5.

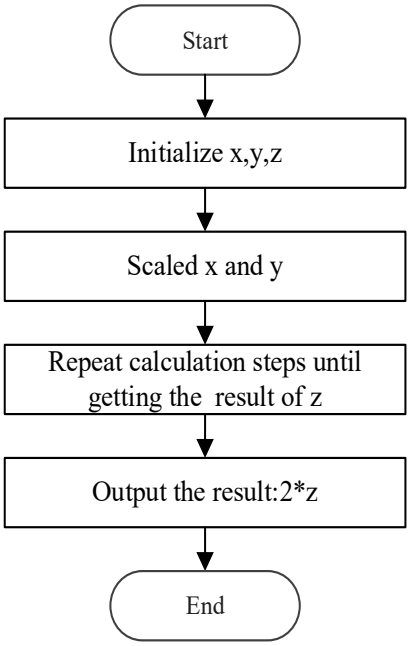

Figure 5. Flow Chart of Improved Cordic Logarithmic Algorithm.

Wang Yuqing et al. proposed a two-level lookup table polynomial logarithm algorithm. Building upon the original single-level lookup algorithm, they addressed the issue of excessive errors by introducing a sec-

ond-level lookup table and Taylor expansion as a supplementary algorithm(Wang,Y.Q.;Lei,Y.W.2017). The algorithm starts by transforming the input to a specific range [1, 2), then approximates elementary functions within this narrowed interval. A secondary approximation is performed within the first segment, and Taylor expansion is employed for the remaining parts that do not meet the accuracy requirements. The main process of this algorithm is illustrated in Figure 6.

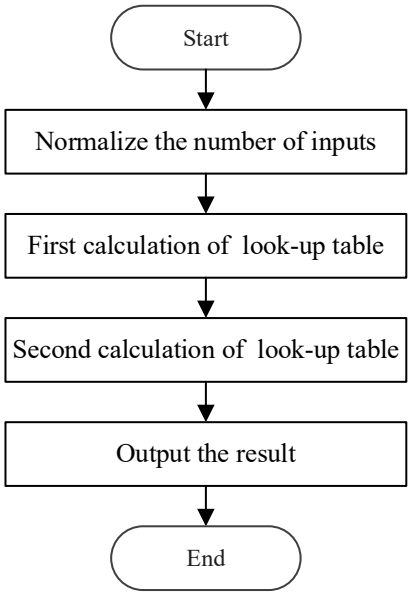

Figure 6. Flow chart of two-level table lookup logarithmic algorithm.

2.3. Simulation testing

The three selected algorithms from Section 3.2 were implemented separately in MATLAB. A set of data with Q value equal to 12 was chosen for testing. The error comparison results for the three algorithms are shown in Figure 7. Here, log1 refers to the Clay Turner logarithmic algorithm, log2 represents the improved Cordic logarithmic algorithm, and log3 corresponds to the two-level lookup table logarithmic algorithm.

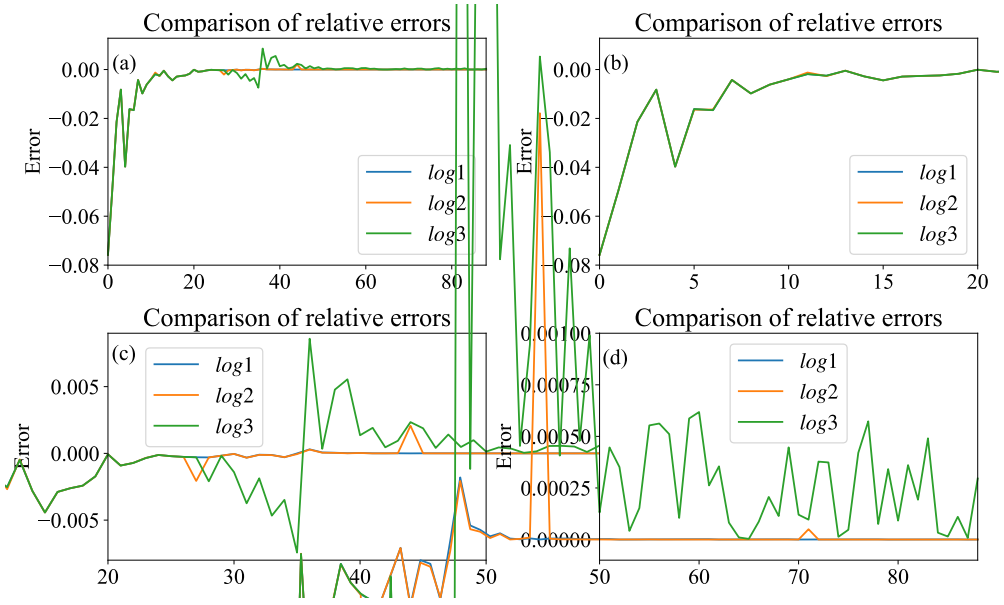

Figure 7. Error comparison of three log algorithms. (a) All data; (b) Output less than -4; (c) Output range is (-4, 3); (d) Output greater than 3.

From Figure 7(b), it can be observed that within the range of output values less than -4, the relative errors of the three algorithms essentially overlap, showing no significant differences. Figure 7(c) reveals that within the output range of (-4, 3), both the Clay S. Turner logarithmic algorithm and the improved Cordic logarithmic algorithm exhibit significantly lower relative errors, indicating higher precision. Figure 7(d) illustrates that within the range of output values greater than 3, the relative errors of the Clay S. Turner logarithmic algorithm and the improved Cordic logarithmic algorithm remain consistently low, demonstrating higher precision. Overall, the Clay S. Turner logarithmic algorithm consistently maintains the lowest relative error, with relatively stable error fluctuations.

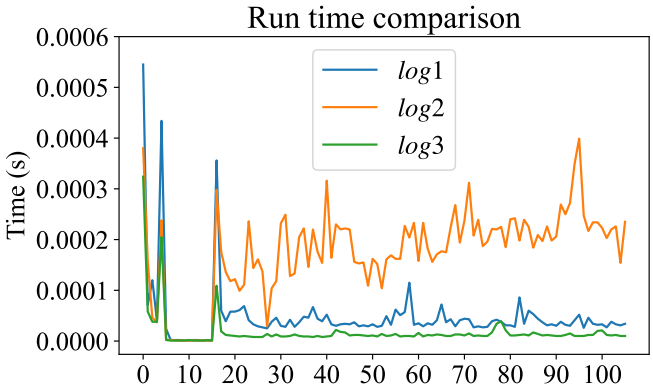

Figure 8. Comparison of runtime of three log algorithms.

From Figure 8, it can be observed that the two-level lookup table logarithmic algorithm requires the least amount of runtime, followed by the Clay S. Turner logarithmic algorithm, while the runtime of the improved Cordic logarithmic algorithm is noticeably greater than the other two algorithms.

In terms of required spatial resources, the Clay S. Turner logarithmic algorithm involves simple calculations without the need for table lookups, while the improved Cordic logarithmic algorithm requires pre-stored values for, with a lookup table capacity of approximately 1 KB. The two-level lookup table logarithmic algorithm requires the largest lookup table capacity, approximately 3 KB. In terms of algorithm complexity, all three algorithms are simple and easy to implement. The comparative analysis results of the three implementations of fixed-point logarithmic functions in MATLAB simulation tests, considering aspects such as precision (relative error), stability, runtime, storage space occupancy, and algorithm complexity, are summarized in Table 2. The radar chart is presented in Figure 9.

Table 2. Comprehensive performance of three log algorithms.

	Clay Turner method	**Improved Cordic method**	**Two level lookup table method**
accuracy	High	High	Low
stability	High	Medium	Low
running speed	Medium	Low	High
Occupy storage space	Less	Medium	High
algorithm complexity	Low	Low	Low

Based on the previous comparative testing of three algorithms for the logarithmic function, analyzing their performance in terms of precision, execution time, algorithm complexity, storage space occupancy, and stability, we can draw the conclusion that the Clay S. Turner logarithmic algorithm consistently maintains high precision and short execution times within the input range. Additionally, it is a simple algorithm with minimal space requirements, making it the optimal choice. In accordance with this conclusion, this paper selects the Clay S. Turner logarithmic algorithm for the design of a fast, high-precision, fixed-point logarithmic algo-

rithm. To address the slight deficiency in the algorithm's runtime, an improvement plan is proposed, aiming to further optimize the algorithm and enhance the speed of the fixed-point logarithmic algorithm.

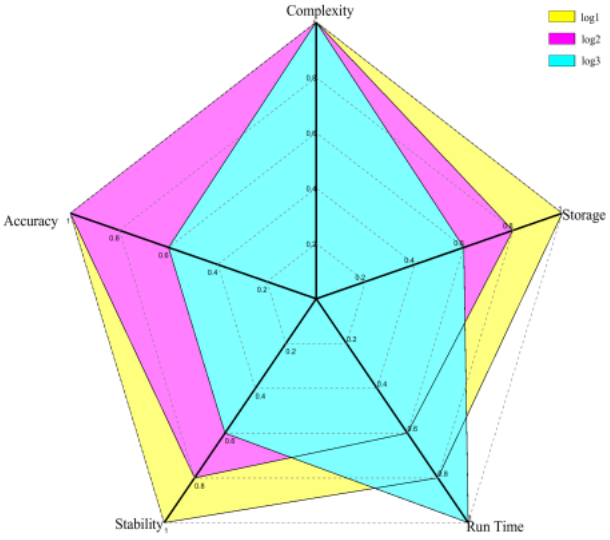

Figure 9. Comprehensive performance comparison of three log algorithms.

2.4 Algorithm improvement

In the Clay S. Turner algorithm, the logarithmic solution is divided into two parts. The first part involves obtaining the integer part of the logarithmic result. This is achieved by performing multiplication or division operations on the input (implemented through left shifts or right shifts), thereby normalizing within the [1, 2) interval.

$$y = log_2 x \#(3)$$

convert to

$$x = 2^y \#(4)$$

If , set . If , set . Repeat this step until normalization is completed, counting multiplication or division during this loop process, where represents the integer part of the logarithm. The second part involves obtaining the decimal part of the logarithmic result. With normalized within the [1, 2) interval, where , expand y into binary series form:

$$y = y_1 2^{-1} + y_2 2^{-2} + y_3 2^{-3} + y_4 2^{-4} + ... \#(5)$$

Equation (5) is further transformed into:

$$y = 2^{-1}(y_1 + y_2 2^{-1} + y_3 2^{-2} + y_4 2^{-3} + ... \#(6)$$

Substituting equation (6) into equation (4) yields:

$$x = 2^{2^{-1}(y_1 + 2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + ...))))} \#(7)$$

Next, let's see how to sequentially extract the values of by squaring :

$$x^2 = 2^{y_1} \times 2^{2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + ...))))} \#(8)$$

We observe that the right side of (8) is the product of two factors, and the left-side factor is equal to 1 or 2, depending on the value of . Therefore, there are two cases:

$$y_1 = 1 : \quad x^2 = 2 \times 2^{2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + ...))))} \#(10)$$

$$y_1 = 1 : \quad x^2 = 2 \times 2^{2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + ...))))} \#(10)$$

Due to the common factor $2^{2^{-1}(y_2+2^{-1}(y_3+2^{-1}(y_4+\ldots)))}$, which is greater than or equal to 1 and less than 2, appearing in both (9) and (10), we observe that $y_1 = 1$ only when the square of x is greater than or equal to 2, otherwise $y_1 = 0$. Therefore, the next step after squaring x involves comparing the square result; if x^2 is greater than or equal to 2, the mantissa bit is set to "1," and x^2 is divided by 2. Otherwise, the mantissa bit is set to "0," and remains unchanged. After completing this step, we obtain (9), and it can be noticed that (9) and (7) have the same format. Therefore, by repeating this step, we can extract the remaining fractional bits of y. Once the necessary bits for the mantissa are obtained, adding them to the previous integer part yields the complete result of the logarithmic operation.

If a logarithm with a base other than 2 is required, we can utilize the following property of logarithms:

$$log_a x = \frac{log_2 x}{log_2 a} \#(11)$$

This can be converted to division by $log_2 a$ of a by multiplying by the fixed value $1/log_2 a$.

The above is the principle and derivation of the Clay S. Turner algorithm. The specific implementation steps of the algorithm are as follows:

(1) Initialize the result to 0, that is, y = 0.

(2) Initialize the fractional value of the mantissa to 0.5, b = 1/2.

(3) If $x \geq 2$, set $x=x/2$, $y=y+1$; If $x < 1$, set $x=2x$, $y=y-1$. Repeat this step until $1 \leq x < 2$.

(4) Square x, $x=x \cdot x$.

(5) If is greater than or equal to 2, set $x=x/2$, $y=y+b$, $b=b/2$. Otherwise, keep x unchanged and repeat until the desired fractional bits are obtained.

By analyzing the specific steps of the Clay S. Turner algorithm, we can observe that in the first part of the algorithm, which is the process of calculating the integer part of the logarithm, if the input value x is large, it requires a larger number of iterations. For a 32-bit system, it may require a maximum of 30 iterations. Based on the characteristics of fixed-point numbers, this paper uses a binary search approach to obtain the leading zeros and determine the integer part of the logarithm. At most, five iterations are needed, and the number of shifts for x can be determined. The specific implementation is as follows: First, if the input x is 0, return 32; Let n=1, if x right-shifted by 16 bits equals 0, $n=n+16$, $x=x \gg 16$; if x right-shifted by 24 bits equals 0, $n=n+8$, $x=x \gg 8$; if x right-shifted by 28 bits equals 0, $n=n+4$, $x=x \gg 4$; if right-shifted by 30 bits equals 0, $n=n+2$, $x=x \gg 2$; finally, $n=n-i \gg 31$. At this point, based on the fixed-point Q value, we can obtain the integer part of the logarithmic result $y=32-Q-n$, and should be left-shifted by n.

3 RESULTS AND DISCUSSION

3.1 Code Design

From the study of logarithmic functions in Section 2, a design for fixed-point logarithmic functions under all Q formats was created after algorithm optimization. The flowchart for the code design is as follows:

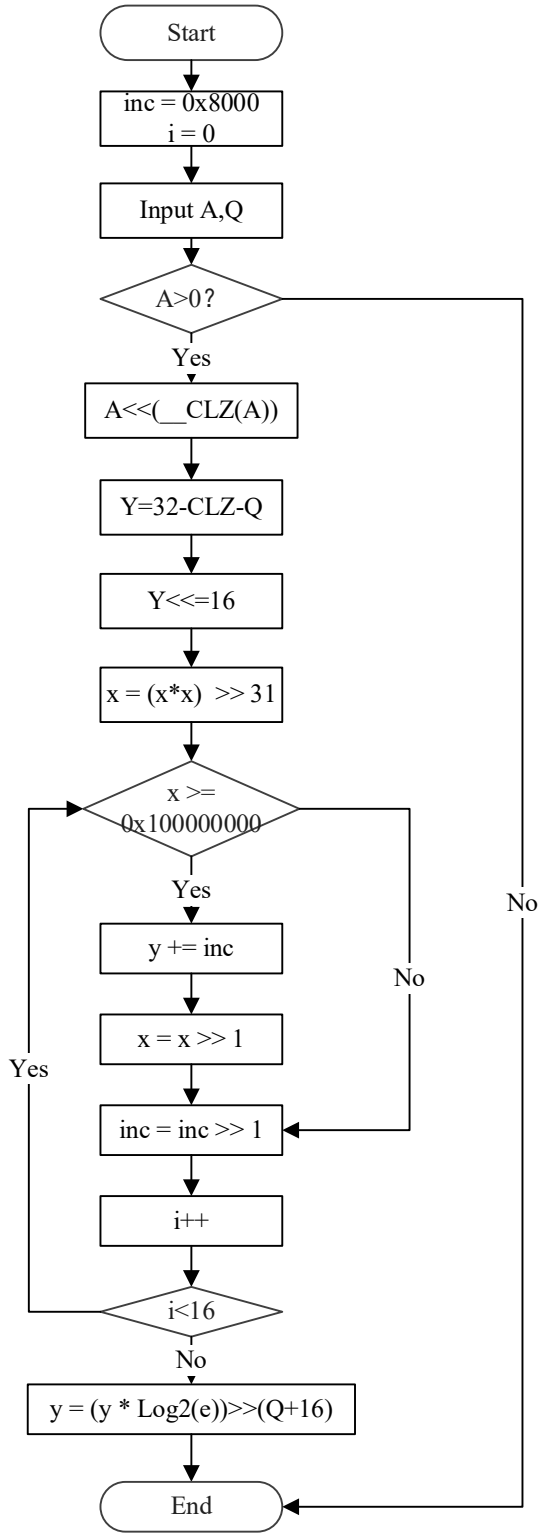

Figure 10. The flowchart of the code.

3.2 Performance testing Results

The testing environment established in this paper is divided into two parts: software environment and hardware environment. The overall architecture is illustrated in Figure 11. The software environment is built on

the Keil uVision5 development platform, and the engineering project used for testing is configured according to the selection of the microprocessor, targeting the corresponding environment and runtime environment. The hardware environment is based on the ARM Cortex-M4 core, operating at a main frequency of 168MHz, supporting DSP instructions, and comprehensively considering characteristics such as high computing power, high real-time performance, and user-friendliness. It possesses excellent CPU computing performance(Yiu, J., &Paul B., 2014). Additionally, for a more intuitive reading of the mathematical library function's computation results and cycle, the microprocessor needs to configure I/O ports to establish serial communication with peripheral devices and computers.

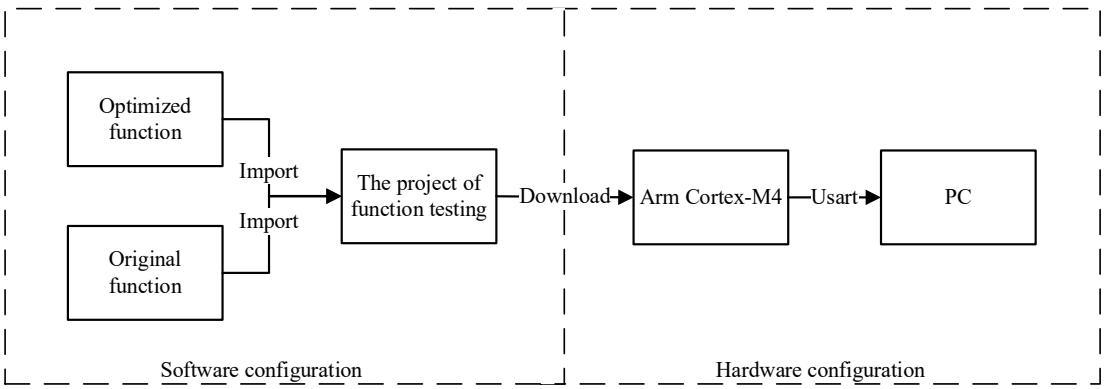

Figure 11. The composition of the test environment.

During the testing process, a comparative analysis was conducted between the fixed-point logarithmic functions designed in this paper and the logarithmic functions before improvement, using the same data and testing methods. The results of the function testing are presented in Table 3, which includes the computation cycles and precision for ARM microprocessor-based calculations.

Table3. Partial test results

Input	Accurate result	Before optimization			After optimization		
		Run result	Run time	Arithmetic error	Run result	Run time	arithmetic error
1.000000	0.00	0.000000	583	0.000000	0.000000	376	0.000000
1.284025	0.25	0.249756	580	0.000244	0.249756	392	0.000244
1.648721	0.50	0.499756	565	0.000244	0.499756	386	0.000244
2.117000	0.75	0.749756	589	0.000244	0.749756	392	0.000244
2.718282	1.00	0.999756	577	0.000244	0.999756	390	0.000244
3.490343	1.25	1.249756	574	0.000244	1.249756	392	0.000244
4.481689	1.50	1.499756	592	0.000244	1.499756	394	0.000244
5.754603	1.75	1.749756	586	0.000244	1.749756	386	0.000244
7.389056	2.00	1.999756	586	0.000244	1.999756	390	0.000244
9.487736	2.25	2.249756	586	0.000244	2.249756	398	0.000244
12.182494	2.50	2.499756	583	0.000244	2.499756	392	0.000244
15.642632	2.75	2.749756	583	0.000244	2.749756	396	0.000244
20.085537	3.00	2.999756	583	0.000244	2.999756	394	0.000244
25.790340	3.25	3.249756	580	0.000244	3.249756	386	0.000244
33.115452	3.50	3.499756	573	0.000244	3.499756	388	0.000244
42.521082	3.75	3.749756	588	0.000244	3.749756	392	0.000244
54.598150	4.00	3.999756	591	0.000244	3.999756	392	0.000244
70.105412	4.25	4.249756	585	0.000244	4.249756	388	0.000244
90.017131	4.50	4.499756	582	0.000244	4.499756	398	0.000244

115.584285	4.75	4.749756	567	0.000244	4.749756	390	0.000244
148.413159	5.00	4.999756	591	0.000244	4.999756	392	0.000244
190.566268	5.25	5.249756	579	0.000244	5.249756	396	0.000244
244.691932	5.50	5.499756	576	0.000244	5.499756	390	0.000244
314.190660	5.75	5.750000	594	0.000244	5.749756	390	0.000244
403.428793	6.00	5.999756	588	0.000244	5.999756	396	0.000244

For multiple sets of test data, fixed-point logarithmic calculations were performed in Q12 format. The statistical analysis of computational errors is presented in Figure 12(a), while the runtime statistics are illustrated in Figure 12(b).

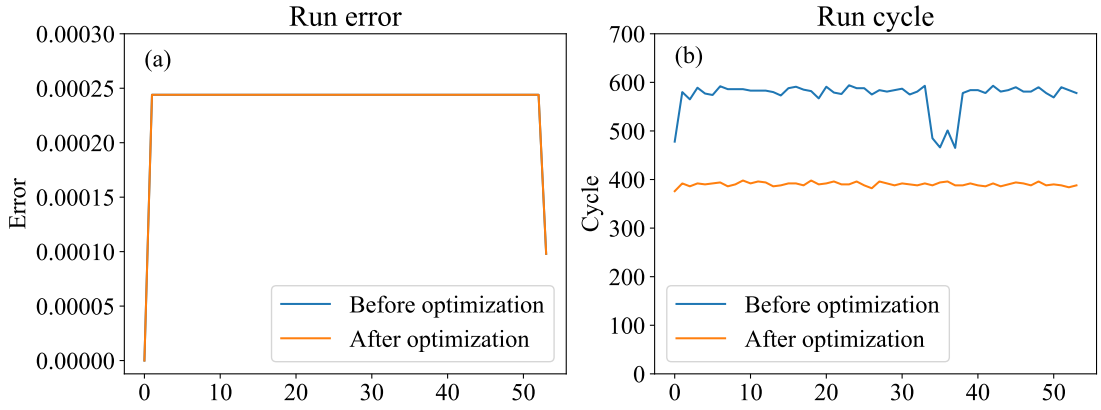

Figure 12. Comparison of running results. (a) Error comparison; (b) Time comparison

In Q12 format, a 32-bit fixed-point number can represent a maximum precision of 0.000244141. From the test results, it can be observed that for 98.15% of the test data, the computational errors of both logarithmic functions in the two mathematical libraries are less than the highest precision value. Both logarithmic functions maintain good computational accuracy within the ARM microprocessor. In terms of computational speed, the average computation cycle of the fixed-point logarithmic function designed in this paper is reduced by 31.88% compared to the previous version. This significant improvement in computational speed, while ensuring computational accuracy, enhances the performance of fixed-point logarithmic calculations on ARM microprocessors.

4 CONCLUSIONS

By studying the fixed-point implementation methods of logarithmic functions, this paper proposes a fast and efficient fixed-point logarithmic function algorithm based on ARM microprocessors. The algorithm aims to reduce the computational time of logarithmic operations while maintaining computational accuracy. It utilizes a binary search method to obtain the leading zeros, quickly determining the integer part of the logarithmic result, and simplifying the steps for the fractional part. Subsequently, a testing environment is set up, and the test results are compared and analyzed against the previous algorithm to determine the corresponding computation cycles and errors. Experimental results demonstrate that the designed and implemented fixed-point logarithmic function in this paper exhibits superior performance on domestically produced ARM microprocessors: the improved algorithm reduces the average computation cycle by 31.88%, with computation errors below the highest precision value of 0.000244141 for 32-bit fixed-point numbers in Q12 format. The proposed fast fixed-point logarithmic function algorithm is well-suited for embedded systems without floating-point mathematical instructions and some DSP applications.

ACKNOWLEDGMENTS

This work was supported by Songshan Laboratory, Spatial Fusion Intelligent Perception Technology and

Precision Reconstruction System (221100211000).

REFERENCES

Ding,Y.,&Ji,P.F.(2023). Technical and Market Analysis of MCU Industry at Home and Abroad. China lntegrated Circuit,32(11),22-27

Hou,L.Z.(2020). Floating Point and Fixed Point Comparison in DSP. Telecom Power Technolog y, 37(1),105-106.

Li,G., Wan,L.,& Lin,L.(2008). Design and Implementation of FPGA-Based Logarithmic Converter. Electronic Engineering & Product World, (08),86-88+91.

Mopuri, S.,Acharyya, A.(2020).Configurable Rotation Matrix of Hyperbolic CORDIC for Any Logarithm and Its Inverse computation.Circuits, systems, and signal processing: CSSP, 39(5),2551-2573

Song, W.N.,Xu, D.J.,&Chen, L. (2023).Overview of DSP architecture development. Microelectronics & Computer ,40(4),1-7.

Turner, C.S.(2010).A Fast Binary Logarithm Algorithm. IEEE Signal Processing Magazine, 27(5),124-140.

Wang,Y.Q.;Lei,Y.W.;Peng,Y.X. Logarithmic algorithm and hardware implementation based on two-level lookup table polynomial,Computer software and computer applications,The 21st Annual Conference on Computer Engineering and Technology and the 7th Microprocessor Technology Forum, Xiamen, China,2017-08-17; Computer Engineering and Techniques.

Yang,K.R.,Yan,C.L., &Shi,K.(2023). Analysis of Chip Industry Policies between China and Americain a Cross-National Comparative Perspective – Based on Text Quantification.Information Technology and anagement Application,2(5),23-39.

Yiu, J., &Paul B.(2014). The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, 3th ed., Waltham: Newnes.