Algorithm Research and Hardware Implementation of High Precision Floating Point Exponential Function

Xiafeng Zou*, Mingjiang Wang
Integrated circuit engineering, Harbin Institute of Technology (Shenzhen), Shenzhen, Guangdong, 518000, China
*Corresponding author’s e-mail: 1216649833@qq.com

Abstract. Exponential function is of great significance for high-precision calculation in computer technology. The study of exponential function plays a very important role in modern communication and signal processing. With the rapid increase of the number of digits in computer computing, the fixed-point representation range is small and limited in practical applications. The floating-point number indicates a wide range of teaching and has a wider range of applications. Therefore, the research direction of this article is the hardware algorithm research and implementation of high-precision floating-point exponential function. The goal is to implement the 64-bit high-precision exponential function hardware implementation in accordance with IEEE754 standard. In this paper, the vector mode in the CORDIC algorithm is used to improve the efficiency of the algorithm by improving the iterative method. The 64-bit floating-point exponential function operation is realized by FPGA, and the accuracy is achieved when compared with the PYTHON operation result.

1. Introduction
Transcendental functions (trigonometric functions, exponential functions, etc.) are generally implemented in software. However, it is difficult to meet the modern requirements of signal processing by using traditional DSP for its slow speed. The hardware method to implement the transcendental function can solve this problem well. Among them, the study of exponential function plays a very important role in communication and signal processing, and the study of transcendental functions is particularly important. At present, the hardware methods for implementing the exponential function mainly include Taylor expansion method [1], Table addition method [2], Polynomial approximation based on lookup table [3], and CORDIC algorithm [4]. Currently the most commonly used method is CORDIC algorithm. In practical applications, the fixed-point number representation range is limited, and it is unable to adapt to the increasingly large amount of data calculation. The floating-point number indicates a large range, so floating-point number can better adapt to high-precision calculation. The research content of this paper is the algorithm research and hardware implementation of floating-point exponential function [5]. The CORDIC iterative algorithm is used to implement the hardware implementation of 64-bit high-precision floating-point [6] exponential function.

2. Detailed algorithm
The first section introduces the overall idea of the algorithm, the second section is the basic CORDIC algorithm principle, and the last section is the improved algorithm.

2.1. Algorithm principle
The index part of the floating-point number is supposed to be $M \cdot 2^E$. The mark will be discussed in the follow-up study.

Make the following assumptions, assuming the result is:

$$e^{M \cdot 2^E} = X \cdot 2^Y$$  \hspace{1cm} (1)

Among them, $0.5 < X < 1$. Then get:

$$0.5 < \frac{e^{M \cdot 2^E}}{2^Y} < 1$$

$$2^{Y-1} < e^{M \cdot 2^E} < 2^Y$$

$$Y - 1 < M \cdot \frac{2^E}{\ln 2} < Y$$  \hspace{1cm} (2)

$Y$ is an integer so $Y$ can be obtained according to (2).

Next is the solution to $X$. Suppose $2^Y = e^Z$, $Z = Y \ln 2$. Then get:

$$X = e^{M \cdot 2^E - Y \ln 2}$$  \hspace{1cm} (3)

$Y$ can be calculated, according to the image of the exponential function, it will definitely fall in the convergence area of CORDIC [-1.1182, 1.1182][7]. Therefore, (3) can be used to calculate $X$ by the CORDIC algorithm.

2.2. Traditional CORDIC model

This paper will use the hyperbolic model of the CORDIC algorithm to solve.

![Hyperbolic model of CORDIC](image)

The mathematical relationship used in the hyperbolic model [8] is:

$$
\begin{bmatrix}
X_{n+1} \\
Y_{n+1}
\end{bmatrix}
= \begin{bmatrix}
\cosh \theta_n \sinh \theta_n \\
\sinh \theta_n \cosh \theta_n
\end{bmatrix}
\begin{bmatrix}
X_n \\
Y_n
\end{bmatrix}
$$  \hspace{1cm} (4)

According to the definition of the hyperbolic trigonometric function, it can be reduced to:

$$
\begin{bmatrix}
X_{n+1} \\
Y_{n+1}
\end{bmatrix}
= \cosh \theta_n \begin{bmatrix} 1 & \tan \theta_n \\tan \theta_n, 1 \end{bmatrix}
\begin{bmatrix}
X_n \\
Y_n
\end{bmatrix}
$$  \hspace{1cm} (5)

Through the definition of the hyperbolic function: $e^\theta = \sinh \theta + \cosh \theta$

Make $X_0 = 1, Y_0 = 0$, get: $X_{n+1} = \cosh \theta_n Y_{n+1} = \sinh \theta_n X_n$. So get $e^\theta = X_n + Y_n$.

The iterative method used in the CORDIC algorithm to perform transcendental function calculations (iteration from $i=1$, repeated iterations at $i=1, 4, 13, 40, 3K+1...$ to ensure convergence[9]), at some fixed angle ($\tanh \theta_j = 2^{-j}$) to approximate the required value. The above equation is utilized in the iterative process. The multiplication $\prod_{i=1}^{n} \cosh \theta_i$ in the iteration is called the scaling factor $K$, which converges
to 1.20750 when the number of iterations is more than 16 times [10]. To this end, it can be proposed that the scaling factor of the CORDIC algorithm is:

\[
\begin{bmatrix}
X_{n+1} \\
Y_{n+1}
\end{bmatrix} = \frac{1}{\sigma_n} \begin{bmatrix} 2^{-n} & \sigma_n \\sigma_n \end{bmatrix} \begin{bmatrix} X_n \\
Y_n
\end{bmatrix}
\] (6)

Introduced \(\sigma_n\) to identify the distance between the angle of rotation and the angle required. The CORDIC model will be divided into two models: a vector model (\(\sigma_j = -\text{sign}(y_j)\)) and a circumferential model (\(\sigma_j = \text{sign}(z_j)\)). The general relationship of each iteration obtained by these definitions is [3]:

\[
\begin{align*}
x_{i+1} &= x_i + \sigma_i 2^{-i} y_i \\
y_{i+1} &= y_i + \sigma_i 2^{-i} x_i \\
z_{i+1} &= z_i - \sigma_i \hat{\theta}_i
\end{align*}
\] (7)

The exponential function is calculated by the circular model of the hyperbolic model. The hyperbolic model of the CORDIC algorithm can perform the calculation of the value between \([-1.1182, 1.1182]\), and the input \(M \cdot 2^k - Y \ln 2\) is in this group interval, so it can be obtained after N iterations:

\[
\begin{align*}
x_{n+1} &= K^{-1}(X_0 \cosh Y_0 \sinh \theta) \\
y_{n+1} &= K^{-1}(Y_0 \cosh \theta + X_0 \sinh \theta) \\
z_{n+1} &= 0
\end{align*}
\] (8)

Make \(X_0 = 1, Y_0 = 0\), Get X by iteration.

2.3. Improved CORDIC algorithm

The traditional CORDIC algorithm can only be iterated once for each clock signal, and the efficiency is low. The selection of \(\theta\) cannot be made as a whole because the \(\theta\) value of each iteration can only be selected according to the following \(Z\) value, making the error large. Therefore, the improved algorithm predicts four \(\theta\) values in one clock, and the efficiency is increased by four times, and each \(\theta\) value is judged according to the four predicted values and the \(Z\)-value interpolation, thereby effectively reducing the error.

Use traditional CORDIC model equation (7). Predict four \(\sigma\) values at a clock (each clock predicts four values backwards), the equation becomes:

\[
\begin{align*}
x_{i+4} &= [1 + \sigma_{i+3}\sigma_{i+2}\sigma_i\sigma_i] \* 2^{-(4i+6)} + (16\sigma_j\sigma_j + 8\sigma_j\sigma_i + 4\sigma_j\sigma_i + 4\sigma_j\sigma_i) \* 2^{-(2i+5)} \] \* x_i \\
&+ ([8\sigma_i + 4\sigma_{i+2} + \sigma_{i+3}] \* 2^{-(i+3)}) \* y_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(3i+6)} \] \* y_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(2i+5)} \] \* x_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(3i+6)} \] \* y_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(2i+5)} \] \* x_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(3i+6)} \] \* y_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(2i+5)} \] \* x_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(3i+6)} \] \* y_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(2i+5)} \] \* x_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(3i+6)} \] \* y_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(2i+5)} \] \* x_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(3i+6)} \] \* y_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(2i+5)} \] \* x_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(3i+6)} \] \* y_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(2i+5)} \] \* x_i \\
&+ ([8\sigma_{i+2}\sigma_{i+3}\sigma_{i+3}\sigma_i + 4\sigma_{i+2}\sigma_{i+3}\sigma_i + 2\sigma_{i+3}\sigma_{i+3}\sigma_i + \sigma_{i+3}\sigma_{i+3}\sigma_i] \* 2^{-(3i+6)} \] \* y_i
\end{align*}
\] (9)

Where \(\sigma_i\) is the angle of the \(i\)-th rotation, and each predicted \(\sigma_i \sigma_{i+1} \sigma_{i+2} \sigma_{i+3}\) is the value that minimizes \(Z_{i+4}\). This greatly improves efficiency by predicting four values at a clock.
2.3.1. Algorithm simplification. It seems that the formula is very verbose, but each \( \sigma_i \sigma_{i+1} \sigma_{i+2} \sigma_{i+3} \) has only two values, 1 or -1.

When \( \sigma_i, \sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3} = 1 \) it rotates in the positive direction, approaching the required angle.

Under this situation: \( Z_{i+1} = Z_i - \sigma_i \theta_i \); When \( \sigma_i, \sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3} = -1 \), it rotates in the opposite direction, that is, away from the required angle. Under this situation: \( Z_{i+1} = Z_i + \sigma_i \theta_i \).

Taking x as an example, let \( \sigma_i \sigma_{i+1} \sigma_{i+2} \sigma_{i+3} = [-1-1-1-1] \), and write the program solution coefficient or hand calculation. Get the coefficient matrix:

| \( \sigma_i \) | \( \sigma_{i+1} \) | \( \sigma_{i+2} \) | \( \sigma_{i+3} \) | \( 2^{(4i+6)} \) coefficient | \( 2^{(2i+5)} \) coefficient | \( 2^{-(i+3)} \) coefficient | \( 2^{-(3i+6)} \) coefficient |
|--------------|--------------|--------------|--------------|-----------------|-----------------|-----------------|-----------------|
| -1           | -1           | -1           | -1           | 1               | 35              | -15             | -15             |
| -1           | -1           | 1            | 1            | 1               | 21              | -13             | -1              |
| -1           | -1           | 1            | -1           | -1             | 9               | -11             | 7               |
| -1           | -1           | 1            | 1            | 1               | -1             | 9               | 9               |
| -1           | 1            | -1           | -1           | -1             | 9               | -7              | 11              |
| -1           | 1            | -1           | -1           | 1               | -15            | -5              | 5               |
| -1           | 1            | -1           | 1            | -1             | -19            | -3              | -3              |
| -1           | 1            | 1            | 1            | 1               | -21            | -1              | -13             |
| 1            | -1           | -1           | -1           | 1               | 21              | 1               | 13              |
| 1            | -1           | -1           | 1            | 1               | -19            | 3               | 3               |
| 1            | -1           | 1            | -1           | 1               | -15            | 5               | -5              |
| 1            | -1           | 1            | -1           | 1               | -9             | 7               | -11             |
| 1            | 1            | -1           | -1           | 1               | 9               | -9              | -9              |
| 1            | 1            | -1           | -1           | 1               | 11             | -5              | 5               |
| 1            | 1            | 1            | -1           | -1             | 21              | 13              | 1               |
| 1            | 1            | 1            | 1            | 1               | 35              | 15              | 15              |

According to the x, y iterative formula, this coefficient table is applicable to two iterative formulas. Thus, after determining the value of \( \sigma_i \sigma_{i+1} \sigma_{i+2} \sigma_{i+3} \) by the minimum Z value, we only needs to correspond to the above table coefficients, calculate the corresponding \( x_{i+4}, y_{i+4} \), which greatly reduces the difficulty of the algorithm.

3. Algorithm architecture
The core algorithm structure is as the picture 2 shows:

The overall structure name is CORDIC-exp, and the function of each module is as follows.

The input signal is a required 64-bit floating point number called Input_num, a clock signal clk, a reset signal rst;

The output signal is the result of the exponent calculation of the 64-bit floating point number exp_result.
3.1. Module Pre
Module function:

The input 64-bit floating point number Input_num is divided into a sign bit sign, an index E(11 bit), and a mantissa M(52 bit);

Defines the 64-bit output signal num_1:

\[ \text{num}_1 = M \times 2^E - Y \times \ln 2 + \ln 2 \]  \hspace{1cm} (10)

Get: \( 2X = e^{\text{num}_1} \); Since the X value range is (0.5, 1), the 2X value range (1, 2), so the num_1 value can be obtained from the exponential function image in \((0, e^{1.1182})\), in this area the CORDIC algorithm converges.

3.2. Module Pre_deal
Module function:

The first 60 bits of the input 64-bit num_1 are named num_2, which is convenient for operation with subsequent lookup tables.

3.3. Module Cordic_In_core
Module function:

Algorithm iterative process implementation.

3.3.1. State machine. The iterative method used in the CORDIC algorithm to perform the transcendental function calculation, in order to ensure the convergence of the algorithm, repeated iterations at \( i = 1, 4, 13, 40, 3K+1[11] \)...But the iteration is four times in the text, and the actual value of \( i \) is \( i = 1, 5, 9, \) and 13. 17...41...113, when \( i = 5, 13, 41 \), the value is closest to the ideal iteration \( i \) value, so repeating the iteration at \( i = 5, 13, 41 \) can ensure convergence.

Repeat \( i = 13 \) is the same as the ideal \( i \) value, and another repeated iteration is \( i = 5, 41 \). It is necessary to subtract 1 from the value of \( i \) (the same value as the ideal iteration \( i \)), and repeat the iteration.

The iteration begins with \( i = 1 \) and ends with \( i = 113 \).

Therefore, the state machine can be used to implement the above iterative function.

Assume four states:

00: Initial state, given initial value, \( i = 1 \), empty \( x_{\text{next}}, y_{\text{next}}, z_{\text{next}} \);
01: Iterative process, each time the clock makes \( i = 1 + 4 \),
If \( i = 5 \) or \( i = 41 \), the state enters 10;
If \( i = 13 \), the state enters 11;
If $l=113$, the iteration is ended, and the $x, y, z$ values after the iteration are output.

10: -1 repeat state. By judging the output value $S[0]$ of the module $z_{\text{pre}}$, the sign of the fourth iteration is determined, let $i=i-1$, repeat the iteration once. If $S[0]=1$, the $\sigma$ of the fourth iteration is 1. If $S[0]=0$, the fourth iteration $\sigma$ is -1. Use the basic iteration formula (7) to calculate. Go back to the 01 state.

11: Repeat state, By judging the output value $S[3]$ of the module $z_{\text{pre}}$, the sign of the 13th iteration is determined. Repeat the iteration once. If $S[3]=1$, the $\sigma$ of the 13th iteration is 1. If $S[3]=0$, the 13th iteration $\sigma$ is -1. Use the basic iteration formula (7) to calculate. Go back to the 01 state.

3.3.2. State machine. Architecture of iteration in the cordic\_ln\_core module is showed in figure 4. In the configuration, the output $s$ of the $Z_{\text{pre}}$ module is used as an input judgment signal of $X_{\text{pre}}, Y_{\text{pre}}$ to judge the state of $X_{\text{next}}, Y_{\text{next}}$.

![State machine diagram](image)

**Figure 3. State machine.**

Module $Z_{\text{pre}}$ function:
Calling the lookup table lut, the lut table will give the adjacent four angles corresponding to each different cnt time. $Z_{\text{pre}}$ obtains the angle values of four iterations, respectively assigned to $z_1, z_2, z_3, z_4$. Let:

$$z_{\text{next}} = z_{\text{in}} - \sigma_1 z_1 - \sigma_{i+1} z_2 - \sigma_{i+2} z_3 - \sigma_{i+3} z_4$$  \hspace{1cm} (11)
Calculate all possible situations corresponding to different \( \sigma_i \sigma_i^2 \sigma_i^3 \). The value of \( \sigma_i \sigma_i^2 \sigma_i^3 \) making the smallest value of \( z_{\text{next}} \) will be this iteration value.

Define \( S = \sigma_i \sigma_i^2 \sigma_i^3 \text{ & } 1111 \). \( S \) is used as a judgment signal as input to module X_pre and module Y_pre.

Module X_pre (or Y_pre) function:

It can be seen in the structure diagram that the input signal of module X_pre (Y_pre) is the clock signal \( \text{clk} \), and the \( x \) and \( y \) values. According to the value of the judgment signal \( S \), the result \( x_{\text{out}}(y_{\text{out}}) \) is the four iterations value corresponding to \( \sigma_i \sigma_i^2 \sigma_i^3 \).

3.4. Module normalize

Splicing input values is \( x_{\text{out}}, y_{\text{out}}, Y \), 64-bit floating point is the output.

4. Result

Enter test data, the CORDIC-exp algorithm results and PYHTON results are as follows:

| Input \( (D) \) | PYHTON Exp Results(H) | FPGA Exp Results | Precision (bit) |
|----------------|----------------------|------------------|----------------|
| \( 2^{-6} \)   | 0000104080ab55de     | 0000104080ab55d8 | 51             |
| 1              | 00115bf0a8b14576     | 00115bf0a8b14576 | 53             |
| 5              | 007128d389970338     | 007128d38997037c | 50             |
| \( 2^4 \)      | 01710f2ebd0a8002     | 01710f2ebd0a8007 | 51             |
| \( 2^8 \)      | 167141c7a8814beb     | 167141c7a8814ba9 | 49             |

In the case of 64-bit floating-point numbers, the effective mantissa is 53bit. The result shows algorithm in this article can guarantee the accuracy of exponential functions of floating-point number.

5. Summary

Using FPGA to realize exponential function can increase the speed of operation and improve accuracy at the same time. The CORDIC realizing exponential function provides a new solution for high-precision computing of computers.

References

[1] Sidahao, N., Constantinides, G.A., Chelng, P.Y.K. (2003) Architectures for Function Evaluation on FPGAs. In: International Symposium on Circuits & Systems. Bangkok. 804-807.

[2] Muller, J.M. (1999) A Few Results on Table-Based Methods. Reliable Computing, 5(3):279-288.

[3] Nagayama, S., Sasao, T. (2006) Programmable Numerical Function Generators Based on Quadratic Approximation, In: Proceedings of Asia South Pacific Design Automation Conference. Japan. 378-383.

[4] Walther, J.S. (1971) A Unified Algorithm for Elementary Function. In: Spring Joint Computer Conf. New York. 379-385.

[5] Li, L.S. (2012) Design and Implementation of Floating-Point Exponential Function. Microelectronics, 42: 703–709.

[6] Zuras, D., Cowlishaw, M., Aiken, A. (2008) IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2008, (2008):1-70.

[7] Hu, X., Harber, R.G., Bass, S.C. (1991) Expanding the range of convergence of the CORDIC algorithm. IEEE Transactions on Computers, 40(1):13-21.

[8] Andraka, R. (1998) A Survey of CORDIC Algorithms for FPGA Based Computers. ACM, 2:191-200.
[9] Zou, X. (2008) FPGA Implementation of Exponent Function Based on CORDIC. Public Technology, 10: 36-37.

[10] Zhai, S.M., (2006) The CORDIC Iterative Initial Value Selection Strategy and Its Hardware Implementation. journal6, 43: 79-80.

[11] Shi, X.W., Wang, C., Zhang, C.L. (2017) Research and Implementation of Floating-Point Exponential Function Algorithm Based on FPGA. Journal of Computer Measurement and Control, 10: 226-228.