A CORDIC-Based Architecture with Adjustable Precision and Flexible Scalability to Implement Sigmoid and Tanh Functions

Hui Chen*, Lin Jiang*, Yuanyong Luo*, Zhonghai Lu†, Yuxiang Fu*, Li Li* and Zongguang Yu*
*School of Electronic Science and Engineering, Nanjing University, China
†KTH Royal Institute of Technology, Stockholm, Sweden

***Abstract*—In the artificial neural networks, $tanh$ (hyperbolic tangent) and $sigmoid$ functions are widely used as activation functions. Past methods to compute them may have shortcomings such as low precision or inflexible architecture that is difficult to expand, so we propose a CORDIC-based architecture to implement $sigmoid$ and $tanh$ functions, which has adjustable precision and flexible scalability. It just needs shift-add-or-subtract operations to compute high-accuracy results and is easy to expand the input range through scaling the negative iterations of CORDIC without changing the original architecture. We adopt the control variable method to explore the accuracy distribution through software simulation. A specific case ($ARCH.(1, 15, 18)$, RMSE: 10^{-6}) is designed and synthesized under the TSMC 40nm CMOS technology, the report shows that it has the area of 36512.78μm^2 and power of 12.35mW at the frequency of 1GHz. The maximum work frequency can reach 1.5GHz, which is better than the state-of-the-art methods.**

***Index Terms*—CORDIC, $tanh$, $sigmoid$, adjustable precision, flexible scalability**

I. Introduction

Artificial neural networks (ANNs) have a broad range in the applications of pattern recognition, image classification, biological systems and so on [1]. The activation functions introduce nonlinear factors to neurons, so that the ANNs can approximate any nonlinear functions and be applied to the nonlinear model. Therefore, the efficient implementation of nonlinear neuron activation functions such as $tanh$ and $sigmoid$ are highly of interest. The $tanh$ and $sigmoid$ functions can both produce a curve with an "S" shape, where the $tanh$ output varies between (-1, 1) and the $sigmoid$ output varies between (0, 1). Mathematically, they have a functional relationship as follows: ($S(x)$ represents the $sigmoid$ function and $T(x)$ stands for the $tanh$ function)

$$T(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 1 - \frac{2}{1 + e^{2x}} = 1 - 2S(-2x). \quad (1)$$

Therefore, $T(x)$ can be easily implemented based on $S(x)$, which can save one exponential block in hardware implementation. From the above formulas, we know that division and exponential operations are the main challenges for high accurate computation of $S(x)$ and $T(x)$ because of their complex hardware implementation. Up to now, some methods have been put forward and they are mainly divided into the following types: piecewise linear based methods [2], [3], piecewise nonlinear methods [4], [5], and look-up table based methods [6], [7].

Generally speaking, table-based methods are appropriate for fixed and few computations, but not feasible for the arbitrary and numerous calculations. Besides, its accuracy is also very low. High precision means high storage, which is constrained in prohibitive hardware. Piecewise linear based methods are also suitable for low-precision computations, but it stores fewer data than table-based methods. High precision requires more segments, and the hardware area will be large. Meanwhile, different curvilinear functions and various ranges of input data also can change the way linear or nonlinear segments are structured, which makes it difficult to flexibly scale the architecture. By contrast, the CORDIC-based method can achieve a higher accuracy by simple shift-add-or-subtract operations and also have a great tradeoff between precision (or speed) and area [8].

The main contributions of this paper are as follows:

(1) We propose a new architecture with adjustable precision and flexible scalability to compute $S(x)$ and $T(x)$, which can overcome the disadvantages of the above methods. Technically, we utilize the rotation mode of hyperbolic CORDIC (RHC) to compute the natural exponential function and the vector mode of linear CORDIC (VLC) to perform the division operation.

(2) Based on the above architecture, we provide a method to extend the input range of computing $S(x)$ or $T(x)$, which not only does not change the original architecture, but also has lower hardware overhead and great flexibility compared with the state-of-the-art methods.

(3) Based on the above architecture, we present a relationship between the accuracy distribution and the iterations of CORDIC (RHC and VLC), which proves that the architecture has higher and more easily adjustable precision compared with the state-of-the-art methods.

The rest of this paper is organized as follows. Section II introduces the basic theory of RHC and VLC. Section III firstly proposes the novel method to compute $S(x)$ and $T(x)$, then it shows the method to expand the range of input variables, including the accuracy distribution in software simulation. Section IV details the hardware implementation and evaluates a specific case. Finally, Section V draws the conclusions.

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

II. Overview of RHC and VLC

Volder first invented CORDIC in 1959 to evaluate trigonometric functions, multiplication and division [9]. Then, Walter extended its computation capacities to calculate logarithms, exponentials and square roots [10]. Since we just use RHC and VLC in this paper, the introduction to other CORDIC is omitted. Their iterative formulas are shown below.

The iterative formulas of RHC:

$$\begin{aligned} x_{k+1} &= x_k + sign(z_k)(2^{-k}y_k), \\ y_{k+1} &= y_k + sign(z_k)(2^{-k}x_k), \\ z_{k+1} &= z_k - sign(z_k)tanh^{-1}(2^{-k}), \end{aligned} \tag{2}$$

where $k \geq 1$ and k is an integer. Especially, when the iteration number $k = 4, 13, 40, \ldots, n$, one more iteration of RHC should be required. Otherwise, RHC cannot converge [11].

The iterative formulas of VLC:

$$\begin{aligned} x_{k+1} &= x_k, \\ y_{k+1} &= y_k - sign(y_k)(2^{-k}x_k), \\ z_{k+1} &= z_k + sign(y_k)2^{-k}, \end{aligned} \tag{3}$$

where $k \geq 0$ and k is an integer.

After a few iterations, RHC and VLC will converge to some common functions, which are shown in Table I.

TABLE I
Outputs of RHC and VLC

Mode of CORDIC	Outputs
RHC	$x_n = \Gamma(x_0 \cdot cosh z_0 + y_0 \cdot sinh z_0)$
	$y_n = \Gamma(y_0 \cdot cosh z_0 + x_0 \cdot sinh z_0)$
	$z_n = 0$
VLC	$x_n = x_0$
	$y_n = 0$
	$z_n = z_0 + y_0/x_0$

Γ is computed by $\prod_{k=1}^{n}(\sqrt{1-2^{-2k}})$ and called the scale-factor, where the terms ($k = 4, 13, 40, \ldots$) should be multiplied again. When the iteration number of RHC is determined, Γ will be a constant value, so we don't need to compute it in actual applications. From Table I, we know that RHC has the ability of calculating hyperbolic sine ($sinh$) and hyperbolic cosine ($cosh$) if we initialize the inputs as $x_0 = 1/\Gamma$ and $y_0 = 0$, VLC can implement division operation if z_0 is initialized to 0. They all lay the foundation to compute $sigmoid$ and $tanh$ function.

III. A Novel Method to Compute Sigmoid or Tanh Function and Software Test

We first detail the new method to compute $S(x)$ or $T(x)$. Then, we show the way to expanding the range of input variables. Finally, we conduct software simulation to explore the relationship between the accuracy distribution of computing $S(x)$ or $T(x)$ and the iterations of RHC and VLC.

A. Introduction to the New Method

The proposed method of computing $S(x)$ contains two steps. One is the exponential operation of e^{-x}, which can be implemented by RHC. Because $cosh(x) + sinh(x) = e^x$, we should initialize the inputs of RHC as follows:

$$x_0 = 1/\Gamma, y_0 = 0, z_0 = -x. \tag{4}$$

Then, we can get the result of e^{-x} through x_n plus y_n.

Another is the division operation, which can be calculated by VLC. To this end, the inputs of VLC are initialized to

$$x_0 = 1 + e^{-x}, y_0 = 1, z_0 = 0, \tag{5}$$

so the output z_n will converge to $\frac{1}{1+e^{-x}}$ and it is exactly the result of $S(x)$. From the relationship in (1), we can design the architecture shown in Fig. 1 to calculate $S(x)$ or $T(x)$. "t" represents the functionality to be implemented by the current architecture, $t = 0$ means computing $S(x)$, and $t = 1$ stands for computing $T(x)$.

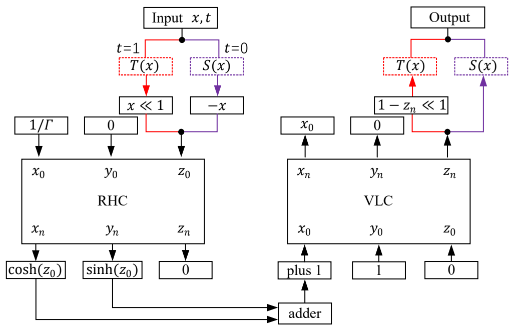

Fig. 1. The architecture of computing $S(x)$ and $T(x)$ based on RHC and VLC. The red path is for computing $T(x)$, the purple path is for computing $S(x)$, and the black path is shared.

Next, we make an analysis on the range of input variables x, which is closely related to the convergence range of RHC and VLC from Fig. 1. As for the standard CORDIC, the convergence range of RHC is $|z_0| \leq 1.1182$, and that of VLC is $\frac{y_0}{x_0} \leq 1$ [11]. Because the output of $T(x)$ varies between [-1, 1] and the $S(x)$ output varies between [0, 1], their ranges are just right for VLC and we needn't to expand the converge range of VLC. Then, we analyze the range of the variable x when we use RHC to compute e^{-x} or e^{2x}. Obviously, x belongs to [-1.1182, 1.1182] for computing $S(x)$ and x only belongs to [-0.5591, 0.5591] for computing $T(x)$, which are both very limited and not suitable for practical applications.

Therefore, we should expand the range of input variables z_0 of RHC. Hu [11] shows that if we extend the iteration index set from $k = 0, 1, \cdots, n$ to $k = -m, -m+1, \cdots, 0, \cdots, n$, the converge range of RHC will be expanded. However, the iterative formulas will be changed when $k \leq 0$, where the terms 2^{-k} need to be replaced by $1 - 2^{-2^{-k+1}}$.

The convergence range of the new RHC depends on the maximum sum of rotation angles, which is defined as

$$\theta_{max}^{(m,n)} = \sum_{k=-m}^{0} tanh^{-1}(1-2^{-2^{-k+1}}) + \sum_{k=1}^{n} tanh^{-1}(2^{-k}), \tag{6}$$

where the terms of iterative number $k = 4, 13, 40, \cdots$ should be accumulated for 2 times.

Therefore, the convergence range of RHC will be updated to $z_0 \leq \theta_{max}^{(m,n)}$, and we can enlarge or reduce the range through changing the value of m. Now, we can mainly change the value of m (n is assumed to be 15) to get different convergence ranges of RHC, including the corresponding input ranges for $T(x)$ and $S(x)$, as shown in Table II.

TABLE II
AN EXTENSION OF THE INPUT RANGES

m	$\theta_{max}^{(m,15)}$	$S(x)$	$T(x)$
0	2.028	[-2.028, 2.028]	[-1.014, 1.014]
1	3.745	[-3.745, 3.745]	[-1.872, 1.872]
2	6.863	[-6.863, 6.863]	[-3.431, 3.431]
3	12.755	[-12.755, 12.755]	[-6.377, 6.377]
4	24.192	[-24.192, 24.192]	[-12.096, 12.096]
5	$\rightarrow \infty$	$\rightarrow \infty$	$\rightarrow \infty$

Because the iterations of the improved RHC are related to two parameters (let them be m and n) and the iterations of VLC are relevant to one parameter (let it be p), we denote the architecture of computing $S(x)$ and $T(x)$ as $ARCH(m, n, p)$.

B. Software Test for the Proposed Method

Before we model the proposed architecture in Verilog, it is necessary to conduct a software simulation for validation and explore the accuracy. First, we input different data x to verify the correctness of $S(x)$ and $T(x)$. Then, we explore the accuracy distribution of computing $S(x)$ and $T(x)$ through changing the iterations of VLC and RHC.

In order to evaluate the accuracy of the proposed architecture, we characterize it with root mean square error (RMSE) and it is defined as:

$$RMSE = \sqrt{\frac{\Sigma(V_{ori} - V_{pro})^2}{NUM}}, \tag{7}$$

where V_{ori} stands for the value of original functions, V_{pro} represents the value of proposed architecture, and NUM means the total test points.

Next, we use the control variable method to explore the precision distribution in three cases. All cases set NUM to be 100,000 and the input x is generated by random number.

Case1: We set $m = 0$, $n = 16$, p from 15 to 20, and then explore the accuracy distribution of computing $S(x)$ and $T(x)$ under different iterations (p) of VLC, respectively. The result is shown in Fig. 2 and the precision is 10^{-5}.

Case2: We set $m = 0$, $p = 18$, n from 15 to 20, and then explore the accuracy distribution of computing $S(x)$ and $T(x)$ under different iterations (n) of RHC, respectively. The result is shown in Fig. 3 and the precision is 10^{-6}.

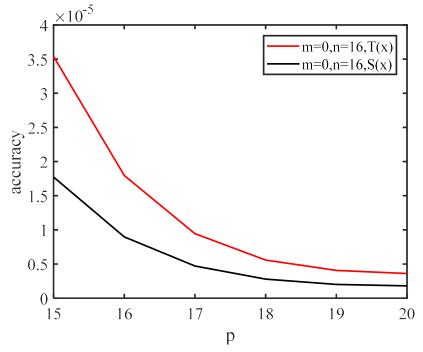

Fig. 2. Accuracy distribution corresponding to different p

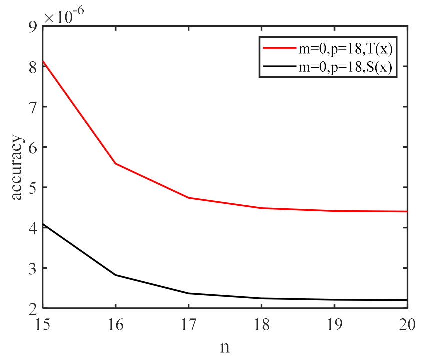

Fig. 3. Accuracy distribution corresponding to different n

Case3: We set $n = 15$, $p = 18$, m from 0 to 4, and then explore the accuracy distribution of computing $S(x)$ and $T(x)$ under different iterations (m) of RHC, respectively. The result is shown in Fig. 4 and the precision is 10^{-6}.

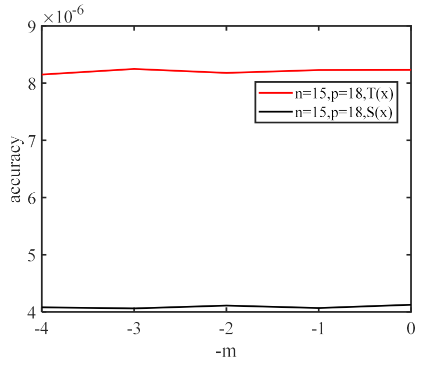

Fig. 4. Accuracy distribution corresponding to different m

From the above cases, it is obvious that our proposed method is feasible and has flexible precisions in computing $sigmoid$ and $tanh$ functions.

IV. HARDWARE IMPLEMENTATION

Firstly, we introduce a general top-level architecture to compute $S(x)$ and $T(x)$. Then, we design a special case $ARCH.(1, 15, 18)$ in Verilog HDL and evaluate it under the

TSMC 40nm CMOS technology. Finally, we analyze the case from different aspects.

A. General Top-Level Architecture

The general top-level architecture is shown in Fig. 5, which mainly includes RHC module, VLC module and adder. When the input x and t are valid, if $t = 0$, the sign bit of x will flip, otherwise, x will be shifted left for one bit as the initial input z_0 of RHC module. After a fixed number of iterations, the output of RHC module is valid, x_n and y_n of RHC will be added up by adder and used as the initial input x_0 of VLC module. Similarly, when the output of VLC module is valid, if $t = 1$, z_p will convert its sign bit and then be shifted one bit to the left, the result is added up by 1 as the final result of $T(x)$. But if $t = 0$, z_p will be the final result of $S(x)$.

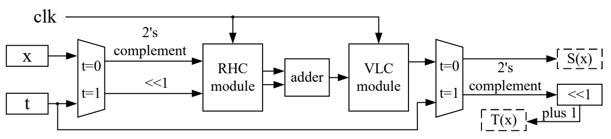

Fig. 5. The general top-level architecture of computing $T(x)$ and $S(x)$.

Among them, in order to obtain high throughput and high computing speed, we adopt a pipelined structure for RHC module and VLC module. Fig. 6 shows that the inputs and outputs of each VLC are cascaded back and forth. The kth stage also includes the shift operation, which needs to shift x_k by k bits to the right. The constants contained in the look-up table are calculated by 2^{-k}. In the same way, Fig. 7 describes a pipelined architecture of RHC module. Unlike the VLC module, there are two kinds of architecture in the RHC module. One is the RHC of positive iterative number (P-RHC), another is that of nonpositive iterative number (NP-RHC). For the P-RHC, the kth stage needs to shift x_k and y_k by k bits to the right and the constants in look-up table are computed by $tanh^{-1}(2^{-k})$. However, the kth stage of NP-RHC should shift x_k and y_k by 2^{1-k} bits to the right and the elements in look-up table are calculated by $tanh^{-1}(1 - 2^{-2^{-k+1}})$.

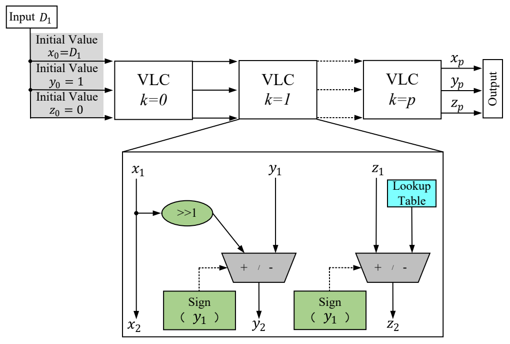

Fig. 6. The pipelined architecture of VLC module.

B. Evaluation of a Specific Case

We design a specific case $ARCH.(1, 15, 18)$ in Verilog HDL and synthesize it under the TSMC 40nm CMOS technology.

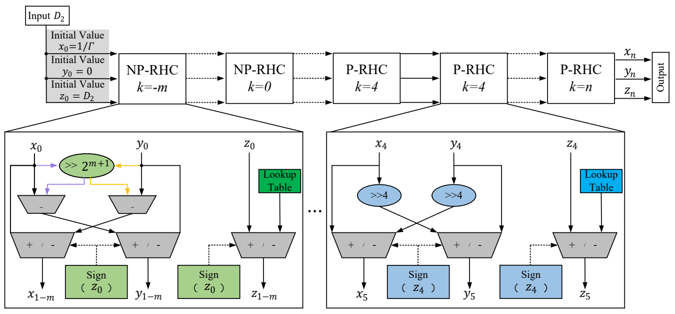

Fig. 7. The pipelined architecture of RHC module.

The report shows that the area consumes 36512.78 μm^2 and the power costs 12.35mW at the frequency of 1GHz (period: 1ns). The maximum work frequency can reach 1.5GHz (period: 0.67ns), which is better than the state-of-the-art methods such as [12] (period: $tanh$ is 1.72ns, $sigmoid$ is 1.73ns) and [13] (period: 1.74ns).

Next, we discuss the extensibility of the input scope. In this case, we know that the input range of RHC is [-3.745, 3.745] from Table II. In other words, the computation range of $S(x)$ is [-3.745, 3.745] and that of $T(x)$ is [-1.872, 1.872]. In the artificial neural networks, the range is appropriate enough for practical application requirements. Not only that, we can also easily expand their ranges by changing the iteration parameter m of RHC. Compared with [12] and [13], their ranges are relatively fixed ([-1, 1]) and difficult to be scaled because of the increased complexity and instability of hardware implementations.

Finally, we make an error analysis of the case. The accuracy of the specific case is evaluated by comparing the outputs from Vivado with MATLAB's original functions. After the comparison, we no doubt find that the order of magnitude of RMSE keeps the same for both hardware implementation and software validation. The reason is that in order to achieve the precision (From Section II.B, we know that the order of RMSE's magnitude for $ARCH.(1, 15, 18)$ is -6), we set the fractional part of x_k, y_k, z_k) in RHC or VLC to be 25 bits, which is enough to make this end.

V. Conclusion

A novel method is proposed to compute $sigmoid$ and $tanh$ functions, which utilizes the same blocks in a unified architecture. The key idea is to use RHC for natural exponential operation and VLC for division operation. The proposed architecture has the advantages of adjustable high-precision and flexible scalability, which are both better than the existing methods. In MATLAB, we explore their accuracy distributions under different iterations of CORDIC (RHC and VLC) with the control variable method. Finally, under the TSMC 40nm CMOS technology, we synthesize a specific design and analyze it from different aspects. Compared with the latest methods, the proposed architecture can also work at a higher frequency.

References

[1] A. H. Namin *et al.*, “Efficient hardware implementation of the hyperbolic tangent sigmoid function,” in *Proc. ISCAS*, May 2009, pp. 2117–2120.

[2] A. Armato *et al.*, “Low-error digital hardware implementation of artificial neuron activation functions and their derivative,” *Microprocess. Microsyst.*, vol. 35, no. 5, pp. 557–567, 2011.

[3] B. Zamanlooy and M. Mirhassani, “Efficient VLSI implementation of neural networks with hyperbolic tangent activation function,” *IEEE Trans. VLSI Syst.*, vol. 22, no. 1, pp. 39–48, Jan 2014.

[4] Ming Zhang, S. Vassiliadis, and J. G. Delgado-Frias, “Sigmoid generators for neural computing using piecewise approximations,” *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1045–1049, Sep. 1996.

[5] Z. Xie, “A non-linear approximation of the sigmoid function based on FPGA,” in *Proc. ICACI*, Oct 2012, pp. 221–223.

[6] K. Leboeuf *et al.*, “High speed VLSI implementation of the hyperbolic tangent sigmoid function,” in *Proc. ICCIT*, vol. 1, Nov 2008, pp. 1070–1073.

[7] P. Kumar Meher, “An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks,” in *Proc. Int. Conf. VLSISoC*, Sep. 2010, pp. 91–95.

[8] Y. Luo *et al.*, “CORDIC-based architecture for computing Nth root and its implementation,” *IEEE Trans. Circuits Syst. I*, vol. 65, no. 12, pp. 4183–4195, Dec 2018.

[9] J. E. Volder, “The CORDIC trigonometric computing technique,” *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[10] J. S. Walther, “The story of unified CORDIC,” *J. VLSI Signal Process.*, vol. 25, no. 2, pp. 107–112, Jun. 2000.

[11] X. Hu, R. G. Harber, and S. C. Bass, “Expanding the range of convergence of the CORDIC algorithm,” *IEEE Trans. Comput.*, vol. 40, no. 1, pp. 13–21, Jan 1991.

[12] V. Nguyen *et al.*, “An efficient hardware implementation of activation functions using stochastic computing for deep neural networks,” in *Proc. Int. Symp. Embedded MCSoC*, Sep. 2018, pp. 233–236.

[13] K. K. Parhi and Y. Liu, “Computing arithmetic functions using stochastic logic by series expansion,” *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 44–59, Jan 2019.