Implementation of a Universal IP CORDIC Algorithm in Serial and Parallel Architectures

Igor A. Cintra[1], Guilherme P. Piedade[1], Luiz R. Pires[1], Natanael I. Souza[1], Rafael S. Macêdo[1], Marcelo P.P. Morais[1], Cícero L.A. Cunha[1], Letícia C. Souza[1], Elivander J.T. Pereira[1], Felipe G.F. Rocha[1], Francisco M. Portelinha Jr[1], Tales C. Pimenta[2]

[1] National Institute of Telecommunications, Santa Rita do Sapucaí, Brazil
[2] Federal University of Itajuba, Itajuba, Brazil
franciscoportelinha@inatel.br

Extended Abstract

Trigonometric and hyperbolic functions are widely used in scientific computing, energy, robotics, biomedicine, telecommunications, image processing, and neural networks, among others [1]. With the rapid development of those areas, speed and precision are essential requirements for the calculation of those mathematical functions, while the use of hardware resources must be minimized [2]. Unfortunately, their implementation suffers from the speed of convergence: generally, one iteration round can only increase by one effective digit, and more clock cycles are needed for the result to reach the required precision [1]. The basic algorithm of COrdinate Rotation DIgital Computer – CORDIC is an efficient iterative technique for computing vector rotations and elementary mathematical functions (addition, multiplication, division, square root, trigonometric functions, logarithms, and hyperbolic functions) [3][4]. The calculations of this algorithm can be performed by a sequence of additions and shifts. Therefore, CORDIC is quite useful in embedded systems that require real-time response and satisfactory accuracy.

The process of rotating a vector v_0 by an angle θ in order to obtain the final vector v_R.

CORDIC is modeled with three data paths (x, y e z). The x path corresponds to the real component of a complex value, y is the imaginary component, and z is the rotation angle. In rotation mode, the goal is to zero the z component after a number of rotations, ensuring that the input vector v_0 is rotated by a given angle θ, while in vectorization mode (Fig. 2.b), the approach seeks to minimize the value of y to zero, resulting in the final vector $v_R = (x_R, 0)$.

The implemented CORDIC algorithm is universal since it allows rotation and vectorization operation modes, it can use circular, linear, and hyperbolic modeling. It also offers serial and parallel architectures, as indicated in Fig. 4(a) and Fig. 4(b), respectively. Therefore, it can run various mathematical functions. The algorithm performs a sequence of partial pseudo-rotations using only addition and bit shift operations, combined with lookup operations, i.e., searches in Look-Up Tables – LUT. Several tests were also performed to validate the algorithm. Initially, it was defined the number of iterations as 16, and it was defined the signed Q16.16 data pattern for input and output. It was also adopted an error of 10E-3 [6]. Results

Tests were conducted using the ModelSim® simulation tool. The obtained results were compared with the expected ones from the program developed in Python®. Initially it was defined the number of iterations as 16 in order to compare the Q16.16 (32-bit) and Q16.32 (48-bit) standards.

Tests show that the results for the number of iterations and error are the same in both architectures. Nevertheless, the biggest difference refers to the test time. Considering 25 iterations and errors smaller than 5×10E-3, the test time for the parallel architecture is approximately 6.15 times shorter than the serial one.

The tests show that a larger number of iterations generally results in greater accuracy. However, this implies greater latency for sequential versions or more complex hardware with larger area for parallel versions. Consequently, in the implementation of the CORDIC algorithm, it was decided to use 25 iterations and to represent the numbers in fixed point, in the Q16.16 (32-bit) standard for input/output and the Q16.32 (48-bit) standard for internal processing.

The multiplication operation shows that the CORDIC algorithm in the serial architecture has errors smaller than 10E-1. The sine operation shows that the parallel architecture presented smaller errors, i.e., less than 10E-4. The parallel architecture is 23 times faster than the serial one.

Conclusions

It was implemented a universal CORDIC algorithm that can perform runs in rotation and vectorization modes, using circular, linear, and hyperbolic modeling. The serial and parallel implementations presented virtually identical results in calculations; however, the serial version exhibits significantly higher latency. The results also demonstrated good accuracy of the calculations, thus validating the algorithm efficiency.