*Politechnika Gdańska*

*Wydział Elektrotechniki i Automatyki*

Gdańsk 2015

# FPGA COMPUTATION OF MAGNITUDE OF COMPLEX NUMBERS USING MODIFIED CORDIC ALGORITHM

**Maciej CZYŻAK[1], Robert SMYK[2]**

1. Politechnika Gdańska, Wydział Elektrotechniki i Automatyki
   tel.: 58 347-15-02 e-mail: maciej.czyzak@pg.gda.pl
2. Politechnika Gdańska, Wydział Elektrotechniki i Automatyki
   tel.: 58 347-13-32 e-mail: robert.smyk@pg.gda.pl

**Abstract:** In this work we present computation of the magnitude of complex numbers using a modified version of the CORDIC algorithm that uses only five iterations. The relationship between the computation error and the number of CORDIC iterations are presented for floating-point and integer arithmetics. The proposed modification of CORDIC for integer arithmetic relies upon the introduction of correction once basic computations are performed in order to reduce the maximum error. The correction value is derived using the coordinate and magnitude values obtained after the fifth iteration. The correction allows to reduce the maximum error by about 79%. The exemplary FPGA implementation of the modified algorithm is also presented.

**Keywords:** magnitude of complex number, CORDIC, FPGA.

## 1. INTRODUCTION

Computation of the magnitude of complex numbers is necessary in these algorithms of digital signal processing (DSP) which have the complex output signal. The important applications are fast Fourier transform (FFT) and complex finite impulse response (FIR) filters. Computation of the magnitude of complex samples at the output of an FFT processor requires squaring of real and imaginary parts with the successive square-rooting (SR). Generally, the square root belongs to the s.c. arithmetic standard functions. In general purpose computers for their calculation the relevant procedures from numerical libraries are used or arithmetic coprocessors that intercept arithmetic instructions from the code generated by compiler. However, in FFT processors and complex FIR filters such approaches does not allow to fulfill requirements with respect to the processing speed because of the required computation time. Moreover, the pipelining with frequencies on the order of hundreds of MHz required in certain applications, is not achievable.

The important factor that facilitates such tasks is a relatively small number range commonly limited to 12-16 bit. In this work we propose the computation of the magnitude of complex numbers using the CORDIC (Coordinate Rotation Digital Computer) with five iterations only and with the final correction. The exemplary FPGA architecture is also given.

In Section 2 we review selected methods of square rooting, in Section 3 the CORDIC algorithm is presented and in Section 4 results of numerical experiments are shown and in Section 5 the CORDIC FPGA architecture is described. This work is an extended and modified version of [15]. The modifications pertain mainly to the new FPGA architecture, number range and results of numerical simulation.

## 2. REVIEW OF MAGNITUDE COMPUTATION METHODS

For computation of the magnitude of complex numbers one of the SR algorithms can be utilized. There exists several SR algorithms such as those based on expansion in Taylor series [1] and rough estimation [2]. However, these algorithms are not used in hardware implementations because of the excessive number of arithmetic operations.

The review of SR algorithms used for hardware implementations was given in [3]. Usually, in such implementations modified variants of non-restoring techniques are applied [4,5]. These algorithms can be implemented in a pipelined form, however, this form may introduce a substantial delay because of the required number of computational steps. The introduced delay can be sometimes longer than the acceptable computation time. In hardware implementations the computation results with a given accuracy can be obtained within the fixed number of clock cycles as opposed to iteration methods. Nevertheless, the iteration methods such as Newton-Raphson algorithm [6] can be transformed to the non-iterative form by loop-unrolling, i.e., by the realization of the chosen number of iterations in the sequential stages of the circuit with each one performing computation for one iteration. The substantial obstacle in the hardware implementation of this algorithm is the necessity to perform multiplication and division which not only slows down the algorithm execution but also substantially increases the hardware complexity. Moreover, the computation error for the fixed number of stages depends on the selected starting point. Thus the accuracy can vary.

If the radicand is a sum of squares as it is the case when computing the magnitude of complex numbers the alpha max and beta min algorithm can be applied [7]. In its original version this algorithm does not require neither division nor iterations and uses one or two approximation regions. For one approximation region the maximum error does not exceed 3.95% and 1% for two regions. The extended version of this algorithm was presented in [8]. This

version allows for a theoretically unlimited number of approximation regions and thus the reduction of approximation error to the acceptable level.

The other class of SR algorithms are those based on the CORDIC [9,10,11]. The CORDIC has been widely applied for computation of various arithmetic functions. Its basic advantage is the lack of multiplications which are replaced by binary shifts if coefficients are negative powers of 2. The development summary of CORDIC was presented in [12]. Typically one CORDIC stage calls for 2 adders and the same number of shift registers. In its direct form CORDIC requires minimum eight stages. After execution the division by a constant is needed because vector pseudorotations are used that increase the vector length. When implementing CORDIC the important problem is the choice of arithmetic. For FPGAs there exist s.c. CORDIC cores [13] that include the choice of arithmetic, number range and the type of architecture. For example, Xilinx LogiCORE IP CORDIC can implement vector rotation, translation, calculation of functions such as sin, cos, sinh, cosh, arctan and square root. For magnitude computation this core uses the simplified form of CORDIC because the radicand is not negative. The input $X_{in}$ and output $X_{out}$ numbers (coordinates) are represented as unsigned fractions or integers. For unsigned fractions the input number range is limited to $0 \le X_{in} < 2$ and for unsigned integers to $0 \le X_{in} < 2^{l}$ .

## 3. CORDIC ALGORITHM

CORDIC performs vector rotations that with the suitable formulation allow to obtain the realization of the required function. Assume, that we have a vector $X=Re^{j\varphi}$ and we want to rotate it clockwise by the angle $\delta$ to obtain $X'=Re^{j\varphi+\delta}$. We receive the following relationship between the coordinates of these vectors

$$x' = x \cdot \cos\delta + y \cdot \sin\delta , \tag{1a}$$

$$y' = -x \cdot \sin\delta + y \cdot \cos\delta . \tag{1b}$$

For small $\delta$ we may adopt $\sin\delta \approx \delta$ and $\cos\delta \approx \delta$. We then obtain the modified rotations of the following form

$$x' = x + y \cdot \delta , \tag{2a}$$

$$y' = y - x \cdot \delta . \tag{2b}$$

This transformation changes the vector length $R$ and the rotation angle. We receive the new $R$' vector of the length

$$R' = R\sqrt{1+\delta^2} . \tag{3}$$

The coordinates of the new vector $R' = (x_w, y_w)$ are as follows

$$x_w = \frac{x}{\sqrt{1+\delta^2}} + \frac{y\delta}{\sqrt{1+\delta^2}} , \tag{4a}$$

$$y_w = \frac{y}{\sqrt{1+\delta^2}} + \frac{x\delta}{\sqrt{1+\delta^2}} . \tag{4b}$$

Now we can determine the vector rotation angle $\alpha$ as the result of (2) as

$$\sin\alpha = \frac{\delta}{\sqrt{1+\delta^2}} , \tag{5a}$$

$$\cos\alpha = \frac{1}{\sqrt{1+\delta^2}} . \tag{5b}$$

so $\tan\alpha \approx \delta$ and $\alpha = \arctan\ \delta$ .

Transformations as in (2) can be applied to compute $\sqrt{x^2+y^2}$ . This can be done by performing rotations in such a manner so as to bring down the $y$ coordinate to zero. For this purpose we can use the following relationships

$$x_{i+1} = x_i + y_i \cdot \delta_i , \tag{6a}$$

$$y_{i+1} = y_i - x_i \cdot \delta_i , \tag{6b}$$

and $\delta_i = \pm\ 0.5^i$, $i$ = 0, 1, 2, 3...,$n$. The choice of $\delta_i$ as a negative power of 2 allows to perform multiplication in (6), as the binary right shift.

In order to obtain the reduction of the absolute value of $y_i$ in each iteration step, the sign must be selected, in such a way that the product on the right-hand side had the opposite sign with respect to $y_i$. We then have for $y_i < 0$, $\delta_i = 0.5^i$, and for $y_i > 0$ $\delta_i = -0.5^I$. The final $y_i$ has to be divided by the cumulation factor $\sqrt{1+\delta^2}$ .

## 4. NUMERICAL RESULTS OF CORDIC ERROR ANALYSIS

The aim of this work was the FPGA CORDIC hardware implementation using the modified form of CORDIC. Such implementation requires several assumptions with respect to the type of arithmetic, input word length, maximum absolute error and related number of iterations as well as the form of the circuit. In this implementation the number of iterations directly translates to the number of stages of the circuit, which should be as small as possible because this gives the smaller delay. 12-bit unsigned representations have been assumed for the input signal and 15-bit as the internal wordlength. These representation lengths result from the prospective application of the circuit to compute the magnitude of the complex samples at the output of the pipelined FFT processor.

The maximum magnitude error at the CORDIC output results from the number of required iterations and the type of arithmetic. Generally, for practical purposes it is enough to assume the maximum magnitude error as the maximum difference between the output of the floating-point arithmetic *sqrt* standard function and the value obtained for the assumed $n$ CORDIC iterations for all possible pairs of integer arguments from the given number range. *sqrt* is usually computed using 64-bit floating-point double type and has 15-16 significant decimal digits.

The maximum absolute magnitude error, $e_{max}$ , can be defined as

$$e_{\max} = \max_{x,y\in[1,2^{m-1}-1]} \left| R_f(x,y) - R(x,y)_{CORDIC(n)} \right| , \tag{7}$$

where the $R_f(x,y)$ – the magnitude for *sqrt* and $R_{\mathrm{CORDIC}(n)}$ - the magnitude for $n$ CORDIC iterations.

The designed circuit, because of speed requirements, will use integer arithmetic. For this reason (6) is modified to the form

$$x_{i+1} = x_i + \lfloor y_i 2^{-i} \rfloor, \tag{8a}$$

$$y_{i+1} = y_i - \lfloor x_i 2^{-i} \rfloor. \tag{8b}$$

The binary right-shift in (8) represents the integer division with a truncation of the fractional part of the result. This truncation may cause considerable errors, hence it is necessary to analyze the numerical behavior of the integer version of the algorithm.

A simple simulator has been written that allows to determine the magnitude error for floating-point and integer point arithmetics and the variable number of iteration steps. The magnitude errors have been calculated by searching the space of possible arguments.

It may be conjectured for the CORDIC algorithm with truncation that the magnitude error depends in general on the magnitude value and also on $|y_i|$, that can be treated as a form of a remainder. It was stated as the effect of simulation that the magnitude error generally depends on $y_5$ and the value of the calculated magnitude. The calculated magnitude is evidently smaller than that for floating-point because of truncation and the resulting difference should be compensated.

Initially the corrections have been chosen experimentally with the following form: if $R_{e1} = R^{(5)} + 2$ and if $|y_5| > 256$ then $R_{e2} = R_{e1} + 4$ $R_{e2}$=$R_{e1}$ + 4 (v1 – correction). The results with and without correction are given in Table 1.

In order to reduce the maximum error an alternative correction method has been introduced that estimates the rotation angle $\alpha_{cor}$ and the additional rotation of $R$ vector by that angle is performed. The important fact is that the approximate calculation of this angle can be carried out with the ROM addressed with arguments of the reduced length. The performed simulation indicated that the correction should be halved that results in its final form as follows

$$x_{corr} = x[n] + y[n] \cdot \alpha_{corr} / 2, \tag{9}$$

where $\alpha_{corr} = \arctan(y[n]/x[n])$.

Table 1. Results of CORDIC simulations

| Algorithm version | No. of steps, *n* | Correction | Error max | $P_{emax}$ | $Q_{emax}$ |
|---|---|---|---|---|---|
| truncation | 5 | no | 11,.99 | 3371 | 4082 |
| truncation | 5 | yes (v1) | 5,99 | 3371 | 4082 |
| truncation | 6 | no | 4,56 | 3264 | 3961 |
| truncation | 6 | yes (v1) | 3,64 | 1203 | 1882 |
| double | 5 | no | 10,63 | 3382 | 4088 |
| double | 6 | yes (v1) | 2,57 | 3823 | 4075 |
| truncation | 5 | yes (v2) | 2,45 | 275 | 378 |
| truncation | 6 | yes (v2) | 2,84 | 663 | 814 |
| double | 5 | yes (v2) | 0,67 | 4057 | 4078 |
| double | 6 | yes (v2) | 1,17 | 4073 | 4091 |
| truncation | 5 | yes (v3) | 1,49 | 2587 | 4077 |
| truncation | 5 | yes (v4) | 2,49 | 82 | 444 |

v2 – angle correction using (9),

v3 – angle correction using 3 *msb* bits for *arctan* computation and multiplication by full-length (9 bit) $y[n]$ representation,

v4 - angle correction using 3 *msb* bits for *arctan* computation and multiplication by 3 bit *msb* bits of $y[n]$ representation.

For this correction $e_{max}$ has been reduced to 2.49.

## 5. HARDWARE FPGA ARCHITECTURE FOR MODIFIED CORDIC

The elaborated system performs the following algorithm

Algorithm 1. (10)

$x_{k+1} = x_k + y_k\, ls$ s ,
*if* $y_1 >= 0$
then $y_{k+1} = (y_1 - x_1)$ *shift-right*-arithmetic
if $y_1 < 0$
then $y_{k+1} = (x_1 - y_1)$ *shift-right-arithmetic*

for $k$=1,2,3,4. For $k$=0 there is no shift.

In Figure 1 the circuit architecture is depicted.

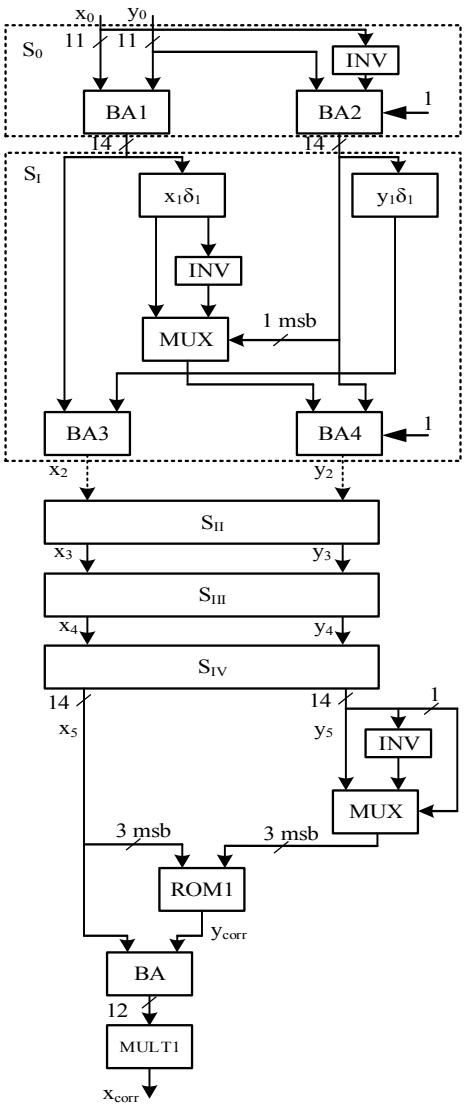


Fig. 1. Architecture of CORDIC magnitude calculator

For $k$=0 the addition in (10) is performed using 12-bit unsigned numbers by the BA1 adder. In order to implement subtraction, the two's complement representation of $-x_0$ is

formed by negation of $x_0$ and introduction of 1 as a carry to the *lsb* position of the BA2 adder. At the output of BA2 13-bit representation is extended by two 0-bits to obtain two's complement representation. The block denoted as "$x_1$ *ls* 1" shifts right $x_1$ representation by 1 bit and analogically the block "$y_1$ *ls* 1" shifts right arithmetically two's complement representation of $y_1$ (the sign bit is retained and the *msb* bit is set to 1 to preserve the value of the number).

At the output of INV1 one's complement representation of the shifted $x_1$ is formed. The MUX1 selects $x_1$ or $-x_1$ in dependence upon the sign of $y_1$. BA3 performs addition for $x_1$ nonnegative and $y_1$ negative and BA4 inversely. 1 needed to obtain two's complement representation from one's complement is introduced as a $2^0$ position of BA4 in order to form the proper $-x_1$ representation.

The final correction is implemented using ROM1 addressed by 3 *msb* bits of $x_5$ and $y_5$. The use of only three bits for both operands allows to apply 6-bit address for the LUT block. Denote $x\_3\_msb=\{x[n]/512\}\cdot 512$ and $y\_3\_msb=\{y[n]/64\}\cdot 64$, where $\{\ .\}$ denotes rounding-off. The ROM1 using $x\_3\_msb$ and $y\_3\_msb$ computes $y_{\text{corr}} = \arctan\left(\frac{y\_3\_msb}{x\_3\_msb}\right)\cdot y\_3\_msb$. Finally $y\_3\_msb$ is multiplied by the inverse of the cumulation factor $1/\sqrt{1+\delta^2} = 0.6076$ using MULT1. The architecture from Fig.1 has been implemented in the Xilinx FPGA. The elaborated FPGA implementation attains 100 MHz processing speed.

## 6. CONCLUSIONS

We have presented a modified CORDIC algorithm and its FPGA implementation. The algorithm uses unrolled five CORDIC iterations and a suitable correction after the fifth stage. It allows to limit the magnitude error to 2.47 for 12-bit integer range. The implementation uses only one multiplier by a constant in series. However, the use of the second multiplier would give the error reduction to 1.49.

## 7. REFERENCES


1. Kwon T., Sondeen J., Draper J.: Floating-point division and square root using a Taylor-series expansion algorithm, In 50th Midwest Symposium on Circuits and Systems, MWSCAS 2007, 2007, pp. 305 – 308.
2. Ercegovac M., D.: On Digit-by-Digit Methods for Computing Certain Functions, In Conference Record of the 41th Asilomar Conference on Signals, Systems and Computers, ACSSC 2007, 2007, pp. 338 – 342.
3. Montuschi P., Mezzalama M.: Survey of square rooting algorithms, Comput. Digit. Tech. IEE Proc. E, Jan. 1990, vol. 137, no. 1, pp. 31– 40.
4. Sutikno T.: An efficient implementation of the nonrestoring square root algorithm in gate level, Int. Journal Comput. Theory Eng., 2011, vol. 3, no. 1, pp. 46 – 51.
5. Sajid Ahmed M., Ziavras S. G.: Pipelined implementation of fixed point square root in FPGA using modified non-restoring algorithm, In 2010 2nd International Conference on Computer and Automation Engineering (ICCAE), 2010, vol. 3, pp. 226–230.
6. Kabuo H., Taniguchi T., Miyoshi A., Yamashita H., Urano M., Edamatsu H., Kuninobu S.: Accurate rounding scheme for the Newton-Raphson method using redundant binary representation, IEEE Trans. Comput., Jan. 1994, vol. 43, no. 1, pp. 43–51.
7. Filip A. E.: Linear approximations to sqrt($x^2$+$y^2$) having equiripple error characteristics, IEEE Trans. Audio Electroacoustics, Dec. 1973, vol. 21, no. 6, pp. 554–556.
8. Czyżak M., Smyk R.: FPGA realization of an improved alpha max plus beta min algorithm, Poznan University of Technology Academic Journals Electrical Engineering, 2014, vol. 80, pp.151 – 160.
9. Volder J.E.: The CORDIC Trigonometric Technique, IRE Transactions on Electronic Computers, Sept. 1959, pp. 330-334.
10. Walther J. S.: A unified algorithm for elementary functions, In Proc. of Sprint Joint Computer Conference, May 1971, pp. 379–385.
11. Ye M., Liu T., Ye Y., Xu G., Xu T.: FPGA Implementation of CORDIC-Based Square Root Operation for Parameter Extraction of Digital Pre-Distortion for Power Amplifiers, In 2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), 2010, pp. 1– 4.
12. Meher P., K., Vallis J., Tso-Bing Juang, Sridharan K., Maharanta K.: 50 Years of CORDIC: Algorithms, Architectures, and Applications, IEEE Trans. Circuits Syst. Regul. Pap., vol. 56, no. 9, pp. 1893 – 1907, Sept. 2009.
13. Xilinx: LogiCORE IP CORDIC v4.0. Product specification, www.xilinx.com, March 2011.
14. Xilinx: Virtex-6, ww.xilinx.com/products/silicon-devices/fpga/virtex-6.html, Feb. 2015.
15. Czyżak M., Smyk R.: Obliczanie modułu liczby zespolojnej w FPGA z użyciem algorytmu CORDIC, FPGA realization of an improved alpha max plus beta min algorithm. Poznan University of Technology Academic Journals Electrical Engineering, 2015, vol. 84, pp. 161 – 171.


## OBLICZANIE MODUŁU LICZB ZESPOLONYCH W FPGA PRZY ZASTOSOWANIU ALGORYTMU CORDIC


W pracy zaprezentowano obliczanie modułu liczb zespolonych przy zastosowaniu zmodyfikowanego algorytmu CORDIC, który wykorzystuje tylko pięć iteracji. Podano związek między błędem aproksymacji a liczbą iteracji dla arytmetyki zmiennoprzecinkowej i całkowitej. Zaproponowana modyfikacja algorytmu CORDIC dla arytmetyki całkowitej polega na wprowadzeniu korekcji po zakończeniu podstawowych obliczeń w celu zmniejszenia błędu maksymalnego. Korekcja jest wprowadzana na podstawie współrzędnych otrzymanych po piątym stopniu algorytmu. Pokazano także przykładową implementacje algorytmu w FPGA.



**Słowa kluczowe:** moduł liczby zespolonej, CORDIC, FPGA.