Modified CORDIC Algorithm for Givens Rotator

Pawel Poczekajlo[2][0000−0003−4742−7872], Leonid Moroz[3][0000−0003−4131−309X], Ewa Deelman[1][0000−0001−5106−503X], and Pawel Gepner[3][0000−0003−0004−1729]

[1] University of Southern California, Marina del Rey CA 90292, USA
USC Information Sciences Institute
deelman@isi.edu
[2] Koszalin University of Technology, Koszalin, Poland
Faculty of Electronics and Computer Science
pawel.poczekajlo@tu.koszalin.pl
[3] Warsaw University of Technology, Warsaw, Poland
Faculty of Mechanical and Industrial Engineering
{leonid.moroz, pawel.gepner}@pw.edu.pl

Abstract. The article presents a modified CORDIC algorithm for implementing a Givens rotator. The CORDIC algorithm is an iterative method for computing trigonometric functions and rotating vectors without using complex calculations. The authors propose two modifications for improving the classical CORDIC algorithm: completing iterations with one-directional rotation of the vector at the final stages and choosing a scaling factor value that can be implemented with low-cost dedicated hardware utilising canonical signed digits representation. The modified algorithm is implemented in a pipeline approach using Verilog language in an Altera Cyclone V System-on-Chip FPGA. The results show that the proposed algorithm achieves higher accuracy and lower latency than the classic CORDIC algorithm.

Keywords: CORDIC algorithm · Givens rotator · FPGA.

1 Introduction

With their pivotal role in numerical linear algebra, Givens rotators serve as indispensable tools across various mathematics and computer science domains. Primarily renowned for their efficacy in the QR decomposition (QRD) of matrices, where the strategic zeroing of matrix elements is a common practice [1–4], Givens rotators exhibit versatility that extends beyond traditional linear algebra applications. These rotators form the cornerstone for implementing point rotation within 2D coordinate systems [5–7], as well as in the intricate realms of 3D space, showcasing adaptability in various variants [8].

Beyond mathematics, Givens rotators find practical utility in computer vision and image processing applications. A notable example includes their integral role in estimating head direction (orientation/position) within images or frames [9]. This feature holds significant implications for tracking positions or gazes, particularly in the dynamic landscapes of virtual reality (VR), augmented reality

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

(AR), and mixed-reality (XR) systems. As technology advances, implementing Givens rotators emerges as a crucial element in enhancing the precision and efficacy of position tracking in these immersive environments.

While Givens rotators focus on matrix manipulations and factorisations, the coordinate rotation digital computer (CORDIC) algorithm is more geared towards trigonometric function calculations, emphasising simplicity and hardware efficiency. They are complementary in specific applications, where Givens rotations are used for matrix transformations, and CORDIC is employed for efficient angle computations, such as coordinate rotations. Currently, CORDIC remains one of the main approaches to rotation [10–14].

In this article, we propose CORDIC's modified algorithm and its embedded implementation in an FPGA. We compare it with an earlier implementation that was our reference and the starting point for our considerations.

2 Review of algorithms and solutions

The critical component of a Givens rotator is an orthogonal rotation matrix, and in the basic form, it is a 2x2 matrix [5, 6]. The elements of this matrix assume values defined by trigonometric functions, and the orthogonality of the matrix arises from the Pythagorean trigonometric identity. This orthogonality property also allows using Givens rotators to implement orthogonal filters (orthogonal state equations) in a pipelined structure [15].

An important aspect when applying Givens rotators is their realisation to minimise computation complexity. The basic approach of software implementation of this is based on using multiplication and addition operations [3]. However, more commonly encountered implementations utilise an iterative CORDIC algorithm [16], often using an FPGA/CPLD chip [17].

In general, the CORDIC algorithm appears in two fundamental variants [18]:

- For determining the coordinates of a point after rotation according to the values of the rotation matrix (values of trigonometric functions sin() and cos()). This is a typical implementation for Givens rotators [4, 19].
- For determining the values of trigonometric functions for a specified angle, usually associated with QR decomposition [1–3, 20].

The mathematical basis of the CORDIC algorithm involves conditional addition/subtraction operations and bit-shifts (as division/multiplication by 2). CORDIC was invented (and is commonly used) as an algorithm based on simple fixed-point operations. But, some variants utilise floating-point arithmetic [19]. The CORDIC algorithm and its individual iterations have been modified and improved for many years [18]. The motivations for improving CORDIC are:

- Increased precision with fewer iterations.
- Reduced iteration complexity (fewer arithmetic/logical operations).
- Lower utilisation of computational unit resources (e.g., fewer ALUs elements in FPGA/CPLD).

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

– Higher operating frequency (higher sampling rate for iterations).

An alternative to the CORDIC algorithm is the BKM algorithm, which can also be used for implementing Givens rotators [21]. However, BKM is more complex and, therefore, rarely employed.

3 CORDIC algorithm and its modified version

The classical CORDIC algorithm takes the input vector $[x_{in}, y_{in}]^T$ and the rotation angle $\theta \in \{-Pi/2, +Pi/2\}$, then the output vector $[x_{out}, y_{out}]^T$ can be found from

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \quad (1)$$

In the CORDIC method, angle θ is presented through the arctangent set of constant angles

$$\theta = \sum_{i=0}^{m} \sigma_i \arctan(2^{-i}) \quad (2)$$

where a signed basis of bi-directional rotation $\sigma_i \in \{-1, 1\}$ is used with the weight of the corresponding angle $\arctan(2^{-i})$. The main idea of the classical CORDIC method is the linear convergence of the method (only one correct bit of the result per iteration) associated with the need to implement three simultaneous iteration equations (for $x_{i+1}, y_{i+1}, z_{i+1}$) in the case of applying a pipeline structure of the computation.

$$x_{i+1} = x_i - \sigma_i 2^{-i} y_i;$$
$$y_{i+1} = y_i + \sigma_i 2^{-i} x_i;$$
$$z_{i+1} = z_i - \sigma_i \arctan(2^{-i});$$
$$\sigma_i = \text{sign}(z_i), i = (0, .., m)$$

needs $m + 1$ elementary bi-directional rotation.
Initial values:

$$z_0 = \theta,$$
$$x_0 = x_{in},$$
$$y_0 = y_{in}$$

Rotations, in this case, are called pseudo-rotations and have gain factor

$$K = \prod_{i=0}^{m} \sqrt{1 + 2^{-2i}} \quad (3)$$

and scaling factor

$$P = \frac{1}{K}$$

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

Rotations equations take the matrix form

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (4)$$

If we complete all $m+1$ iterations according to a formula (4), we obtain:

$$\begin{bmatrix} x_{m+1} \\ y_{m+1} \end{bmatrix} = (\prod_{i=0}^{m} \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix}) \begin{bmatrix} x_i \\ y_i \end{bmatrix} = K * \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \quad (5)$$

or

$$\begin{bmatrix} x_{m+1} \\ y_{m+1} \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} K * x_{in} \\ K * y_{in} \end{bmatrix} \quad (6)$$

It follows from the last equation that the rotated vector will be modified in the Cordic procedure, so it is necessary to perform scaling operations and obtain the corrected values of the components of the output vector. To do this, we must scale the components of the output vector as follows (then the connection between vectors $[x_{m+1}, y_{m+1}]^T$ and $[x_{out}, y_{out}]^T$ is):

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = P * \begin{bmatrix} x_{m+1} \\ y_{m+1} \end{bmatrix} = \begin{bmatrix} P * x_{m+1} \\ P * y_{m+1} \end{bmatrix} \quad (7)$$

The Cordic Rotation Algorithm approximates rotation through iterative steps without relying on complex trigonometric computations. This makes it particularly suitable for hardware implementations or applications with generalizable computational efficiency. Algorithm 1 presents a generalised pseudocode version of the classic CORDIC rotation algorithm for eleven iterations.

Our proposed approach for improving the classic CORDIC algorithm is based on two principles:

- Completion of iterations with the help of one-directional rotation of the vector at the final stages (only the first half of iterations has a significant effect on the value of the gain factor [22], and the second half starting with $i = m/2 + 1$ - can be marginalised, in our paper $m = 16$).
- Calculating the scaling factor value for the first half of the iteration by selectively choosing the iteration steps used to calculate the gain factor in a way that can be implemented in inexpensive dedicated hardware, with a canonical signed digits approach. Of course, such a choice of iteration steps determines an adequate selection of the value $arctg(2^{-i})$.

The first 11 iterations of classical CORDIC are performed according to equation (4), but the individual iteration steps are chosen according to the following scheme $i = (1, 1, 2, 2, 4, 4, 4, 5, 6, 7, 8)$. In fact, iteration steps are selected in such a way and with full premeditation that the scaling factor P is represented as

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

Algorithm 1: Cordic Rotation Algorithm

Input: $x, y, angle$
Output: x, y

$theta\_table = [\arctan(2^{-i})$ for i in 1 to 11$]$
$P = 0.6072530315291345$
$\theta = 0.0$
$P2i = 1$
for $current\_angle$ ***in*** $theta\_table$ **do**
    $\sigma = 1$
    if $\theta \geq angle$ **then**
        $\sigma = -1$
    $\theta = \theta + \sigma \cdot \text{current\_angle}$
    $x\_temp = x - \sigma \cdot y \cdot P2i$
    $y = \sigma \cdot P2i \cdot x + y$
    $x = x\_temp$
    $P2i = P2i/2$
$x = x \cdot P$
$y = y \cdot P$
return x, y

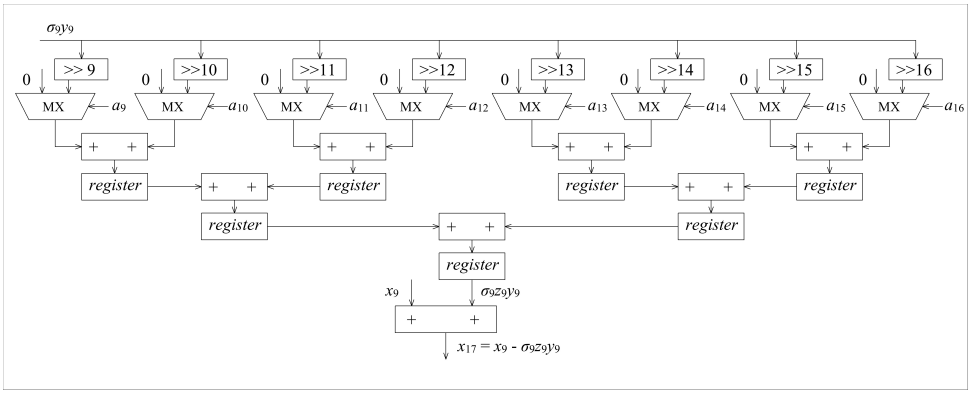


Fig. 1: The output multiplier is built on the shift-add principle

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

the simplest possible number in the signed canonical digit system (shifts and summations). After 11 iterations, we have the following computation results:

$$x_9, y_9, \sigma_9, z_9.$$

The next stage is multiplication x_9 and y_9 by scaling factor P. In this article, we use $P = 0.748065769 \approx 1 - 2^{-2} - 2^{-9} + 2^{-16}$ (approximation error 0.000004).

In the last stage, next two variables ($x9$ and $y9$) are multiplied on the residual angle $z9$. The multiplications are performed according to the principle of addition and summations (see Figure 1) :

$$\begin{gathered} x_{17} = x_9 - \sigma_9 z_9 y_9; \\ y_{17} = y_9 + \sigma_9 z_9 x_9; \\ z_9 = a_9 * 2^{-9} + a_{10} * 2^{-10} + a_{11} * 2^{-11} + a_{12} * 2^{-12} + \\ + a_{13} * 2^{-13} + a_{14} * 2^{-14} + a_{15} * 2^{-15} + a_{16} * 2^{-16}; \\ a_i \in \{0, 1\}, i = 9, ..., 16 \end{gathered}$$

A detailed hardware implementation is given in the next section.

3.1 Realisationation

This improvement algorithm was implemented for the realisation of a rotator in Altera Cyclone V System-on-Chip (SoC) FPGA (5CSXFC6D6F31C6N). An FPGA programmable chip is an ideal hardware system for implementing iterative algorithms in a pipeline approach. This allows each iteration to do calculations with subsequent input data simultaneously.

All operations involve basic arithmetic and logical operations (e.g. addition, subtraction, bit shifts). Using simple operations makes it possible to use higher clock frequencies for data inputs than when using complex operations (e.g. multiplication). Based on the presented CORDIC algorithm, the rotator is implemented using the Verilog language [23] in the development environment Quartus Prime [24].

All elements are implemented with the use of a fixed-point representation. The precision of registers and variables is Q8.12 U2 (8 integer bits and 12 fractional bits) format for coordinates (i.e. x and y) and Q2.18 U2 (2 integer bits and 18 fractional bits) for other values (θ). All values are quantised via round (round a number to the nearest in a given representation, if it is required). The pseudocode of one iteration is presented in Algorithm 2.

The variable E is a number from the sequence, which determines the variable $arctgE$. Value of the $arctgE$ can be calculated from the equation $arctgE = arctg(2^{-E})$. Sequence for E is constant for any rotator (any angle θ). Table 1 shows values of E and $arctgE$ for each iteration.

Algorithm 3 describes a full rotator using pseudocode. The one iteration (scheme) of the presented CORDIC algorithm is also shown in Figure 2. The

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

Algorithm 2: Pseudocode of one iteration of the presented CORDIC algorithm

Input: X, Y, T, E
Output: Xi, Yi, Ti
do in parallel
  $arctgE = arctg[E]$ *//values for each iteration are tabulated*
do in parallel if T≥0
  $Xi = X\text{-}(Y >> E)$
  $Yi = Y + (X >> E)$
  $Ti = T\text{-}arctgE$
do in parallel if T<0
  $Xi = X + (Y >> E)$
  $Yi = Y\text{-}(X >> E)$
  $Ti = T + arctgE$

Table 1: Values of sequence E and $arctgE$ for each iteration

Number of iteration	Values of E	Values of $arctgE$ after quantisation to fixed-point format
1	1	0.463645935
2	1	0.463645935
3	2	0.244979858
4	2	0.244979858
5	4	0.062419891
6	4	0.062419891
7	4	0.062419891
8	5	0.031238556
9	6	0.015625
10	7	0.0078125
11	8	0.00390625

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

scheme is generated with the use the RTL Viewer tool. Also, to test proper operation, simulations were performed in the Intel ModelSim environment (a dedicated simulator for Intel FPGA programming technology). Figure 3 shows the waveforms for the inputs of subsequent iterations and output data of the selected input data case. The values are displayed with limited precision due to the lack of space on the graph. Clock signal period is written to 20ns.

Algorithm 3: Pseudocode of the presented CORDIC algorithm

```
   Input: Xin, Yin, Tin
   Output: Xout, Yout
 1 do in parallel
 2     X1 = Xin
 3     Y1 = Yin
 4     T1 = Tin //angle θ
 5     [Xi1,Yi1,Ti1] = iteration1[X1,Y1,T1, E1]  //E1=1
 6     [X2,Y2,T2] = [Xi1,Yi1,Ti1]
 7     [Xi2,Yi2,Ti2] = iteration2[X2,Y2,T2, E2]  //E2=1
 8     [X3,Y3,T3] = [Xi2,Yi2,Ti2]
 9     [Xi3,Yi3,Ti3] = iteration3[X3,Y3,T3, E3]  //E3=2
10     [X4,Y4,T4] = [Xi3,Yi3,Ti3]
11     [Xi4,Yi4,Ti4] = iteration4[X4,Y4,T4, E4]  //E4=2
12     [X5,Y5,T5] = [Xi4,Yi4,Ti4]
13     [Xi5,Yi5,Ti5] = iteration5[X5,Y5,T5, E5]  //E5=4
14     [X6,Y6,T6] = [Xi5,Yi5,Ti5]
15     [Xi6,Yi6,Ti6] = iteration6[X6,Y6,T6, E6]  //E6=4
16     [X7,Y7,T7] = [Xi6,Yi6,Ti6]
17     [Xi7,Yi7,Ti7] = iteration7[X7,Y7,T7, E7]  //E7=4
18     [X8,Y8,T8] = [Xi7,Yi7,Ti7]
19     [Xi8,Yi8,Ti8] = iteration8[X8,Y8,T8, E8]  //E8=5
20     [X9,Y9,T9] = [Xi8,Yi8,Ti8]
21     [Xi9,Yi9,Ti9] = iteration9[X9,Y9,T9, E9]  //E9=6
22     [X10,Y10,T10] = [Xi9,Yi9,Ti9]
23     [Xi10,Yi10,Ti10] = iteration10[X10,Y10,T10, E10]  //E10=7
24     [X11,Y11,T11] = [Xi10,Yi10,Ti10]
25     [Xi11,Yi11,Ti11] = iteration11[X11,Y11,T11, E11]  //E11=8
26     Xtemp = Xi11−(Xi11»2)−(Xi11»9)+(Xi11»16)
27     Ytemp = Yi11−(Yi11»2)−(Yi11»9)+(Yi11»16)
28     Xtemp2 = Xtemp*Ti11  //implementation by multiple shifts and
         summations
29     Ytemp2 = Ytemp*Ti11  //implementation by multiple shifts and
         summations
30     Xout = Xtemp−Ytemp2
31     Yout = Ytemp+Xtemp2
```

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

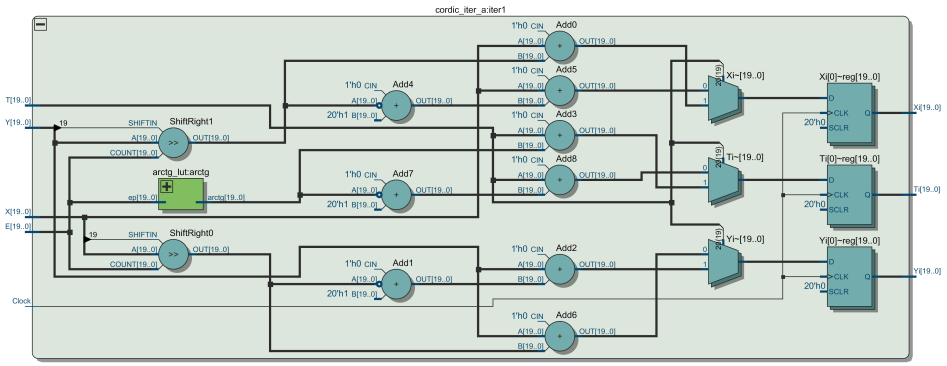


Fig. 2: The scheme of the first iteration of the presented CORDIC algorithm (from RTL Viewer of Quartus Prime), where: X[•], Y[•] - input coordinates of the rotated point; T[•] - input value of the angle θ; E[•] - input value from the sequence; Xi[•], Yi[•] - output coordinates of the rotated point; Ti[•] - output value of the angle θ.

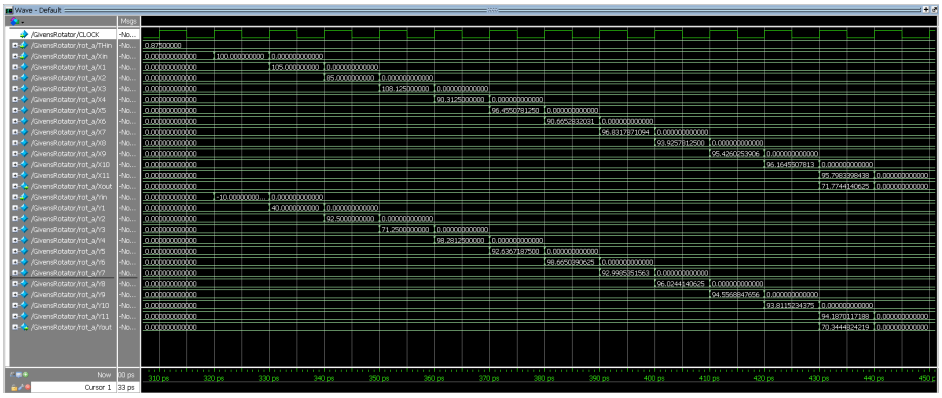


Fig. 3: Input waveforms of successive iterations and output values for the selected input data case: $Xin = 100$, $Yin = -10$, $Tin = 0.875$.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

4 Measurements

The presented CORDIC algorithm is implemented in an FPGA structure, and a measurements are made for selected input values. 100 different values each for x, y and θ are randomly selected.Table 3 and Fig. 5 illustrate the selected values. In this way, $100^3 = 1000000$ different combinations of input data are obtained. The input data is processed by the system on finite precision calculations (like a hardware structure in an FPGA chip). For comparison, the second realisations of CORDIC algorithm from [25] is implemented.

The algorithm [25] is based on the values of the trigonometric functions $sin()$ and $cos()$ for the angle of rotation θ. In successive iterations, the values of these functions approach zero (by summing/subtracting successive values of 2^{-i}). The input is also the sum and difference of the x and y coordinates. The output coordinates are also obtained by summing/subtracting the values of $(x+y)^{-i}$ and $(x-y)^{-i}$ (depending on the sign of $sin()$ and $cos()$ in next iterations). Thus, successive approximations of the coordinates of a point after rotation coincide with zeroing the values of the trigonometric functions of the angle of rotation.

In both realisations, the precision of the registers is the same (Q8.12 for x and y and Q2.18 for θ, $sin(\theta)$ and $cos(\theta)$). Values $sin(\theta)$ and $cos(\theta)$ are quantised via round to suitable format. To determine errors, a dedicated script is written in Scilab which for the same input data determined the output at with full precision. The scheme of the error determination for the implemented systems on Figure 4 is presented.

Accordingly, $\{x_a(n), y_a(n)\}$ realisation are output data from our new proposed realisation of CORDIC and $\{x_b(n), y_b(n)\}$ realisation are output from referenced realisation of CORDIC from [25].

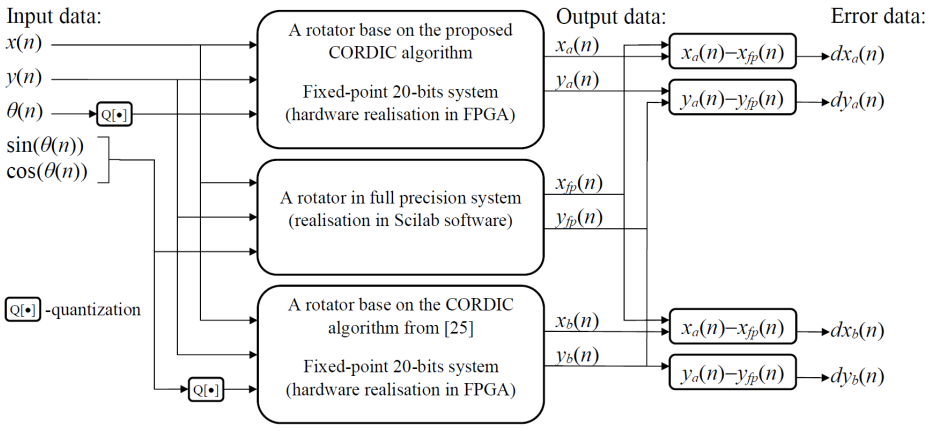


Fig. 4: The scheme of error determination.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

For easier comparison, from the output data of both systems, mean, max, and mean square errors are respectively determined:

$$mean_{dx} = \frac{\sum_{n=1}^{N} |dx_{\bullet}(n)|}{n}$$

$$mean_{dy} = \frac{\sum_{n=1}^{N} |dy_{\bullet}(n)|}{n}$$

$$max_{dx} = \text{MAX}\{|dx_{\bullet}(n)|\} \text{ for } n=1,2,...,N$$

$$max_{dy} = \text{MAX}\{|dy_{\bullet}(n)|\} \text{ for } n=1,2,...,N$$

$$ms_{dx} = \sqrt{\frac{\sum_{i=1}^{n} (dx_{\bullet}(n))^2}{N}}$$

$$ms_{dy} = \sqrt{\frac{\sum_{i=1}^{n} (dy_{\bullet}(n))^2}{N}}$$

where:
$dx_{\bullet}(n) = x_{\bullet}(n) - x_{fp}(n)$ - error of x from selected CORDIC realisation of a rotator with finite precision;
$dy_{\bullet}(n) = y_{\bullet}(n) - y_{fp}(n)$ - error of y from selected CORDIC realisation of a rotator with finite precision;
$x_{\bullet}(n) = \{x_a(n), x_b(n)\}$;
$y_{\bullet}(n) = \{y_a(n), y_b(n)\}$;
$x_{fp}(n), y_{fp}(n)$ - output data from full precision system;
$N = 100^3$ - numbrealisation samples (sets);
MAX{$\bullet$} - a max function.

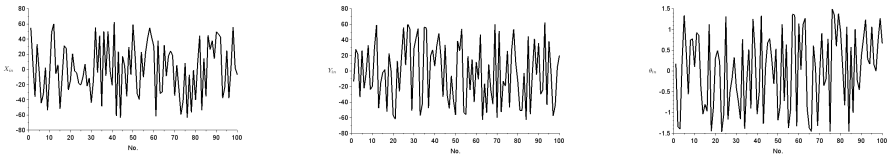
Table 2 shows the described statistical values for obtained results. Completed measurements and research highlight that the presented CORDIC realisation consistently yields lower errors compared to the version from [25]. The Table 2 clearly shows that the statistical parameters obtained with our algorithm are less than half of those achievable with the CORDIC from [25]. The test on a large number of input data also confirms the correct operation of the algorithm in the full data range.

At the same time, the indicated results were obtained with a smaller number of iterations (the presented algorithm requires 11, while the implementation in accordance with [25] required 16). This also means lower consumption of FPGA processor resources and lower latency (delay between the first input sample and the first output sample). All the more reason to see significant progress in the presented solution.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

Table 2: Result for both realisation of rotator with CORDIC algorithm for 100^3 different inputs

Statistical parameters	For the presented realisation of CORDIC algorithm	For CORDIC algorithm from [25]
$mean_{dx_\bullet}$	0.000404	0.000871
$mean_{dy_\bullet}$	0.000344	0.000873
$max_{dx_\bullet}$	0.002187	0.005113
$max_{dy_\bullet}$	0.001809	0.004646
$ms_{dx_\bullet}$	0.000500	0.001113
$ms_{dy_\bullet}$	0.000430	0.001099



(a) Input values X_{in} (b) Input values Y_{in} (c) Input values θ_{in}

Fig. 5: Graphs of the distribution of randomly selected input values X_{in},Y_{in}, θ_{in}

5 Conclusion

In summary, the implemented CORDIC algorithm within an FPGA structure underwent a comprehensive evaluation, contrasting its performance with the standard CORDIC algorithm presented in [25]. The assessment involved processing 100 varied values for x, y, and θ, generating a total of 1000000 diverse input combinations. Both FPGA based implementations employed finite precision calculations, utilizing Q8.12 precision for x and y, and Q2.18 precision for θ, $sin(\theta)$, and $cos(\theta)$.

To gauge the accuracy of the systems, a dedicated Scilab script computed output data at full precision for identical input sets. The error assessment procedure, depicted in Figure 4, compared output data from the newly proposed CORDIC realization $(x_a(n), y_a(n))$ with the referenced implementation from [25] $(x_b(n), y_b(n))$.

For comparative analysis, mean, max, and mean square errors for output x and y components are determined (following the formulas for $mean_{dx}$, $mean_{dy}$, max_{dx}, max_{dy}, ms_{dx}, and ms_{dy}). Detailed data are presented in Table 2.

The superior accuracy demonstrated by the FPGA implementation of the CORDIC algorithm positions it as a compelling choice for real-time applications requiring precise trigonometric calculations. Beyond its performance in traditional CORDIC applications, the enhanced accuracy may particularly benefit Givens rotator implementations. The refined precision could contribute to more

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

Table 3: Values of input data X, Y and θ

No.	X_{in}	Y_{in}	θ_{in}	No.	X_{in}	Y_{in}	θ_{in}
1	54.686768	-13.691650	0.1776543	51	18.411133	38.604980	-0.8566322
2	7.2453613	27.958984	-1.3416557	52	-31.682617	25.765869	1.1263771
3	-35.985840	21.395508	-1.4022789	53	-39.565430	54.284912	-0.7227516
4	33.136475	-33.696045	0.2222099	54	23.140869	-53.439209	0.6395493
5	-2.8244629	26.485596	1.3384476	55	-10.260498	-55.881348	-1.3748894
6	-44.203369	-22.261475	0.4598045	56	21.527344	19.163818	-0.9397888
7	-32.554443	-2.5603027	-0.5633698	57	41.037842	-24.477783	1.3701820
8	2.25	33.317139	0.7503815	58	54.882324	14.881836	1.3425331
9	-53.773682	-23.946289	0.7742653	59	41.742920	-40.162109	-1.3309402
10	-8.0004883	-19.677246	0.0965271	60	30.294678	11.783936	1.1404228
11	49.181641	30.155518	0.9295235	61	-61.528076	-10.580811	0.0332298
12	60.213623	59.224609	0.8558731	62	37.803711	46.683594	1.1209145
13	-5.7312012	-47.788330	-0.3750763	63	-31.557373	-63.023438	1.2699356
14	5.9826660	-15.373535	-1.0399971	64	-29.861084	-16.056152	-0.9068604
15	-51.795898	-1.9924316	-0.8022614	65	31.992920	-53.655518	-1.3718338
16	-10.700928	1.9377441	-0.9709969	66	-8.7856445	8.2387695	-1.4570351
17	31.282471	-52.482910	1.1290512	67	17.609863	-24.588379	0.6099663
18	27.670410	22.790527	-1.4500771	68	24.364990	-42.585693	0.0637779
19	-27.226562	-0.2170410	-0.8927536	69	18.544678	60.291016	-1.3251991
20	-14.371338	-57.136475	0.3126755	70	-34.749512	-59.960205	-0.0350609
21	20.920166	-61.917969	0.4943581	71	5.9055176	51.533203	0.9105721
22	-2.0537109	12.845215	0.2130623	72	-25.404785	-52.716797	-0.3540306
23	-4.4880371	-26.485840	-1.4655914	73	-59.287842	-13.067871	-0.2065315
24	-18.754395	17.931396	-1.0358696	74	-42.360596	-19.529541	0.7719536
25	-20.282471	56.011963	1.3166656	75	8.3732910	25.858398	-1.4639931
26	-9.9829102	7.6108398	-1.1705666	76	-63.537598	-46.740723	1.4983902
27	6.9682617	60.286621	-0.4913292	77	-6.9572754	25.703125	1.3526115
28	-22.028320	54.686035	-0.1704292	78	-56.674561	53.873535	0.5973282
29	-11.200928	-51.004639	0.3290825	79	-1.0581055	10.558350	1.3801918
30	-43.733154	28.613281	-0.3980331	80	-40.010498	-13.555908	0.9557190
31	-14.684570	42.085938	-0.7365456	81	7.6435547	-50.685059	0.1027145
32	55.576416	54.473877	-1.1620560	82	44.739990	-50.711426	-0.8232880
33	-4.0180664	-57.550293	0.7624626	83	-53.906494	-2.4414062	1.0680122
34	44.377441	-42.698486	-1.3940582	84	14.691650	-62.891113	-1.4600182
35	-48.661621	56.344482	-0.2734604	85	-35.538818	44.604736	0.5777702
36	50.292236	55.258545	0.5201035	86	44.992676	-55.353760	-1.0067711
37	-8.3117676	-47.870117	-0.9015388	87	26.354736	-2.6320801	1.0023117
38	49.995850	19.341797	1.2544518	88	37.930664	41.502930	-0.2214317
39	-1.6289062	27.339600	0.4810982	89	14.147217	-4.8681641	-0.4543991
40	-20.791748	6.0939941	-1.0403938	90	49.769043	35.700439	0.3698921
41	62.570068	32.607666	-0.5035858	91	46.766357	-29.708008	0.7878532
42	-61.264404	48.277344	1.3361053	92	42.550049	-24.773682	1.2350807
43	23.336426	17.103027	-1.2712784	93	-38.079834	62.423096	0.2882652
44	-63.429443	-33.214844	0.0814323	94	-23.784912	-43.524414	0.1655083
45	17.252686	33.524170	0.6752968	95	24.986572	38.187012	1.0634651
46	4.7851562	-30.674072	0.7812843	96	-37.671631	-1.8007812	0.1593971
47	-35.454834	-48.004150	0.2856369	97	2.3264160	-57.724609	-0.0062294
48	29.522949	-6.2275391	-0.3152084	98	55.588623	-44.486084	0.7303886
49	-6.8962402	-36.974854	0.5320129	99	2.1872559	4.4125977	1.2736359
50	58.912354	-56.394775	-1.1868744	100	-6.7250977	19.757568	0.6630516

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

accurate transformations and improved overall performance in applications relying on Givens rotators, potentially leading to advancements in areas such as signal processing, numerical linear algebra, and applications like virtual reality.

Acknowledgements This paper and the research behind it would not have been possible without the exceptional support of Graphcore Customer Engineering and Software Engineering team. We would like to express our very great appreciation to Hubert Chrzaniuk, Krzysztof Góreczny and Grzegorz Andrejczuk for their valuable and constructive suggestions connected to testing our algorithms and developing this research work. This research was partly supported by PLGrid Infrastructure at ACK Cyfronet AGH, Krakow, Poland. This work was also partly supported by the National Science Foundation under grant #2331153.

References

1. Aslan, S., Niu, S., Saniie, J.: FPGA implementation of fast QR decomposition based on givens rotation. Midwest Symposium on Circuits and Systems, 470-473 (2012). https://doi.org/10.1109/MWSCAS.2012.6292059
2. Zhang, J., Chow, P., Liu, H.:An efficient FPGA implementation of QR decomposition using a novel systolic array architecture based on enhanced vectoring CORDIC. 2014 International Conference on Field-Programmable Technology (FPT), pp. 123-130, Shanghai, China (2014). https://doi.org/doi: 10.1109/FPT.2014.7082764
3. Omran, S., Abdul-Abbas, A.K.: Implementation QR decomposition based on triangular systolic array. International Journal of New Technologies in Science and Engineering, Vol 5, Issue 6, 150–161 (2018). ISSN 2349-0780
4. Kuan-Ting, C., Wei-Hsuan, M., Yin-Tsung, H., Kuan-Ying, C.: A Low Complexity, High Throughput DoA Estimation Chip Design for Adaptive Beamforming. Electronics, 9 (2020). https://doi.org/10.3390/electronics9040641
5. Dasgupta, B.:Applied Mathematical Methods, Pearson Education India (2006). ISBN 813177600X, 9788131776001.
6. Golub, G.H., Van Loan, C.F.: Matrix Computations. Fourth edition. The Johns Hopkins University Press, Baltimore (2013). ISBN 13: 978-1-4214-0794-4
7. Soler, T.: Active versus Passive Rotations. Journal of Surveying Engineering, 144, 9 (2018). https://doi.org/10.1061/(ASCE)SU.1943-5428.0000247
8. Wittenburg, J., Lilov, L.,: Decomposition of a Finite Rotation into Three Rotations about Given Axes. Multibody System Dynamics, 9, 353–375 (2003). https://doi.org/10.1023/A:1023389218547
9. Ranganathan, A., Yang, M.-H., Ho, J.:Online Sparse Gaussian Process Regression and Its Applications. Image Processing, IEEE Transactions, no 20, 391–404(2011). https://doi.org/10.1109/TIP.2010.2066984
10. Yen, M. H., Lu, H. Y., Lin, S. Y., Lu, K. H., and Chan, C. C. (2023). A Partial-Givens-Rotation-Based Symbol Detector for GSM MIMO Systems: Algorithm and VLSI Implementation. IEEE Systems Journal.
11. Wang, Y. P., Wen, C. C., Kao, C. C., Huang, C. J., Liu, D. Z., and Yang, C. H. (2020, December). Iterative Receiver with a Lattice-Reduction-Aided MIMO Detector for IEEE 802.11 ax. In GLOBECOM 2020-2020 IEEE Global Communications Conference (pp. 1-6). IEEE.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8

12. Chen, L., Xing, Z., Li, Y., and Qiu, S. (2020). Efficient MIMO preprocessor with sorting-relaxed QR decomposition and modified greedy LLL algorithm. IEEE Access, 8, 54085-54099.
13. Yen, M. H., Lu, H. Y., Lu, K. H., Lin, S. Y., and Chan, C. C. (2023). Algorithm and VLSI Architecture of a Near-Optimum Symbol Detector for QSM MIMO Systems. IEEE Access.
14. Hormigo-Aguilar, J., and Muñoz, S. (2020). Efficient floating-point givens rotation unit.
15. Poczekajło, P.: An Overview of tRealisationof Synthesis, Realization and Implementation of Orthogonal 3-D Rotation Filters and Possibilities of Further Research and Development. International Journal of Electronics and Telecommunications, Vol 67, No 2, 295–300 (2021). https://doi.org/10.24425/ijet.2021.135979
16. Volder, J.E.: The CORDIC Trignometric Computing Technique. IRE Trans, Electronics, Computer, Vol.8, 330–334 (1957).
17. Andraka, R.: A Survey of CORDIC Algorithm for FPGA Based Computers. Proc. of ACM/SIGDA Sixth International Symposium on FPGAs, 191–200, Monterrey CA (1998). https://doi.org/10.1145/275107.275139
18. Nagvajara, P., Lin, Z., Nwankpa, C., Johnson, J.: State Estimation Using Sparse Givens Rotation Field Programmable Gate Array. 2007 39th North American Power Symposium, NAPS, 421–427 (2007). https://doi.org/10.1109/NAPS.2007.4402344
19. Hormigo, J., Muñoz, S.: Efficient Floating-Point Givens Rotation Unit. Circuits, Systems, and Signal Processing, 40, 1–24 (2021). https://doi.org/10.1007/s00034-020-01580-x
20. Moroz, L.V., Nagayama S., Mykytiv, T., Kirenko, I.O., Boretskyy, T.: Simple Hybrid Scaling-Free CORDIC Solution for FPGAs. Int. J. Reconfigurable Comput, 615472:1-615472:4 (2014).
21. Bajard, J.C., Kla, S., Muller, J.M.: BKM: A New Hardware Algorithm for Complex Elementary Functions. IEEE Trans. Computers, 43, 955–963 (1994). https://doi.org/10.1109/ARITH.1993.378098
22. D. Timmermann, H. Hahn, B.J. Hosticka, and B. Rix: A New Addition Scheme and Fast Scaling Factor Compensation Methods for Cordic Algorithms, The VLSI J. Integration, vol. 11, no. 1, pp. 85-100, Mar. 1991.
23. IEEE Standard for Verilog Hardware Description Language, in IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001) , vol., no., pp.1-590, 7 April 2006, doi: 10.1109/IEEESTD.2006.99495.
24. Online. Intel Quartus Prime - FPGA Design Software. Accessed 15 January 2024. https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.htmll
25. Wawryn, K., Poczekajło, P., Wirski, R.: FPGA implementation of 3-D separable Gauss filter using pipeline rotation structures. 22nd International Conference Mixed Design of Integrated Circuits & Systems (MIXDES), Torun, Poland, 2015, 589–594. https://doi.org/10.1109/MIXDES.2015.7208592

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_8