Low Latency Recoding CORDIC Algorithm for FPGA Implementation

Pawel Poczekajlo[2][0000−0003−4742−7872], Leonid Moroz[4][0000−0003−4131−309X], Ewa Deelman[1][0000−0001−5106−503X], Michela Taufer[3][0000−0002−0031−6377], Pawel Gepner[4][0000−0003−0004−1729], and Jerzy Krawiec[4][0000−0001−5535−1850]

[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] University of Tennessee, Knoxville,TN, USA
Department of Electrical Engineering and Computer and Science
mtaufer@utk.edu
[4] Warsaw University of Technology, Warsaw, Poland
Faculty of Mechanical and Industrial Engineering
{leonid.moroz, pawel.gepner,jerzy.krawiec}@pw.edu.pl

Abstract. The Coordinate Rotation Digital Computer (CORDIC) algorithm is widely recognized for its fast real-time processing capabilities, making it highly suitable for hardware implementations in diverse applications such as signal processing, high-performance computing, and edge computing devices. Despite its advantages, the traditional CORDIC algorithm's iterative computational method introduces significant challenges, including a complex structure and high hardware resource consumption, which can limit its efficiency and scalability in certain applications.
In this article, we introduce an innovative and efficient variation of the CORDIC algorithm designed to address these challenges. Our proposed algorithm significantly reduces the number of required operations while maintaining computational accuracy, thereby optimizing performance. Furthermore, we demonstrate that this streamlined algorithm can be effectively implemented on Field-Programmable Gate Arrays (FPGAs), leveraging their reconfigurable hardware to achieve enhanced processing speeds and reduced resource utilization. This advancement not only improves the feasibility of using CORDIC in resource-constrained environments but also expands its applicability in modern computing contexts.

Keywords: CORDIC · algorithm · recoded · FPGA · latency.

1 Introduction

The main drawback of the classical CORDIC method [17,18,5] is its low speed due to its linear convergence (only one correct bit of the result per iteration) and

the hardware complexity associated with the need to implement three simultaneous iteration equations for $x_{i+1}, y_{i+1}, z_{i+1}$ when applying a pipeline structure to the computation:

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

that needs $m+1$ elementary rotations.

To reduce the number of iterations, hybrid structures have been developed that use three steps sequentially: table-based + CORDIC + piecewise-linear multiplication (linear approximation) [19,16,1]. To simplify the hardware complexity, a CORDIC method with angle recoding has been proposed [14,13,11,10,9,15], which reduces the system CORDIC iterations (1) to only two equations for x_{i+1}, y_{i+1}. The simplest in terms of hardware implementation is the CORDIC angle recoding method [14,3]. Recently, there have been several new publications on the topic of angle transcoding in CORDIC, one of them being [15]. The theory described therein does not allow for flexible separation of iterations between memory table size and CORDIC. Therefore, it cannot lead to a significant reduction in latency and an increase in the operating frequency of the CORDIC. The algorithm in [14,3] exhibits a drawback as large size of memory (type LUT) is required for large values of m (a table of size not less than $2^{(m/3)} \times m$ bits is needed without the possibility of its reduction). Additionally, the output multipliers are implemented in the {-1,1} basis, preventing the use of multipliers that are part of the DSP blocks in modern FPGA devices.

This work addresses the challenges described above by introducing a new approach to angle recoding that allows flexible adjustment of the memory table size and the number of CORDIC iterations. The performance issue is the main motivation of the work (primarily occupancy of the hardware structure and computation time).

2 Description of the Proposed Method

2.1 Known approaches to angle recoding

Let's consider an arbitrary angle θ given in radians ($\theta \in \{0, 2 - 2^{-m}\}$), represented as

$$\theta = \sum_{i=0}^{m} a_i 2^{-i} \tag{2}$$

where a binary basis of coefficients $a_i \in \{0, 1\}$ is used with the weight of the corresponding digit 2^{-i}. In the CORDIC method, this angle is presented through the arctangent set of constant angles

$$\theta = \sum_{i=0}^{m} \sigma_i \arctan(2^{-i}) \tag{3}$$

where a signed basis of rotation direction $\sigma_i \in \{-1, 1\}$ is used with the weight of the corresponding angle $\arctan(2^{-i})$. This set acts as a new arctangent system when iteratively rotating the unit vector

$$\{x_0, y_0\} = \{1, 0\} \tag{4}$$

around the origin in the Cartesian coordinate system by an angle θ [19].

One of the drawbacks of the traditional CORDIC method is the sequential execution of all consecutive iterations, which results in its high latency. The latency has two main causes. First, it is necessary to maintain a constant value of the scaling factor P:

$$P = \prod_{i=0}^{m} \frac{1}{\sqrt{1 + 2^{-2i}}} \tag{5}$$

All iterations must be carried out; skipping them is not allowed.

Second, there is some ambiguity in the subsequent direction σ_i of vector rotation. To find the actual value of σ_i for any i, all previous iterations must be calculated before estimating σ_i.

Let's consider an approach that can eliminate the second reason, which is the bottleneck of the CORDIC method. In this case, the first reason remains valid: the constancy of the value P. To achieve this, instead of the binary basis of coefficients a_i, a signed basis $b_i \in \{-1, 1\}$ similar to σ_i can be applied, for example, $b_i = 2a_i - 1$ (other options for b_i are provided below). This substitution leads to a clear choice of the vector rotation direction because $b_i = -1$ when $a_i = 0$ and $b_i = 1$ when $a_i = 1$. Thus, the ambiguity in estimating σ_i in the next rotation step is eliminated, simplifying the method's structure by avoiding the calculation of this estimate. This technique is known as angle recoding θ [14,13,11,10,15]. It is known that during the recoding of the angle θ into θ_r, any of the three formulas can be used [14,13,15]:

$$\theta_r = \sum_{i=1}^{m+1} b_i 2^{-i} \tag{6}$$

where $b_i = 2a_{i-1} - 1$ [14];

$$\theta_r = \sum_{i=0}^{m} b_i 2^{-i-1} \tag{7}$$

where $b_i = 2a_i - 1$ [13];

$$\theta_r = \sum_{i=0}^{m} b_i 2^{-i} \tag{8}$$

where $b_i = 2a_i - 1$ [15].

In this work, we use formula (5). If we perform the recoding θ_r in the case of the binary basis of coefficients a_i, we get

$$\theta_r = \sum_{i=0}^{m} b_i 2^{-i-1} = \sum_{i=0}^{m} (2a_i - 1) 2^{-i-1} \tag{9}$$

$$= -(1/2)^{m+1} + \sum_{i=0}^{m} a_i 2^{-i}. \tag{10}$$

The relationship between the angles θ and θ_r is as follows:

$$\theta = \theta_r - \theta_c \tag{11}$$

where

$$\theta_c = -(1/2)^{m+1} \tag{12}$$

2.2 Our approach recoding CORDIC

If we want to apply formula (5) for the CORDIC method, we need to encode (5) as an angle θ_{rr} in the arctangent system. Thus, we will have

$$\theta_{rr} = \sum_{i=0}^{m} b_i \arctan(2^{-i-1}) = \sum_{i=0}^{m} (2a_i - 1) \arctan(2^{-i-1})$$

$$= \sum_{i=0}^{m} 2a_i \arctan(2^{-i-1}) - \sum_{i=0}^{m} \arctan(2^{-i-1}) \tag{13}$$

or

$$\theta_{rr} = \theta_{rv} - \theta_{rc}; \tag{14}$$

where

$$\theta_{rv} = \sum_{i=0}^{m} 2a_i \arctan(2^{-i-1}); \tag{15}$$

$$\theta_{rc} = -\sum_{i=0}^{m} \arctan(2^{-i-1}). \tag{16}$$

Then the approximation of the angle θ (similar to (7)) for recoding CORDIC will be:

$$\theta_{rv} = \theta_{rr} - \theta_{rc} \tag{17}$$

To implement formula (12), the CORDIC method with the following iterative equations can be applied:

$$\begin{gathered} x_{i+1} = x_i - b_i y_i \cdot 2^{-i-1}; \\ y_{i+1} = y_i + b_i x_i \cdot 2^{-i-1}; \\ i = (0, 1, 2, ..., m); \\ x_0 = P \cos(-\theta_{rc}); \\ y_0 = P \sin(-\theta_{rc}); \\ b_i = 2a_i - 1; \end{gathered} \tag{18}$$

$$P = \frac{1}{\prod_{i=0}^{m} \sqrt{1 + 2^{-2i-2}}}.$$

As a result, we get $y_{m+1} = \sin(\theta_{rv})$ and $x_{m+1} = \cos(\theta_{rv})$. Obviously, the angle θ_{rv} will be smaller than the angle θ by the value *del* (lag angle)

$$del = \theta - \theta_{rv} = \sum_{i=0}^{m} a_i d_i =$$

$$= \sum_{i=0}^{m} a_i [2^{-i} - 2\arctan(2^{-i-1})]. \tag{19}$$

To correctly determine the sine and cosine of the given θ angle, it is necessary to additionally rotate the vector $\{ x_{m+1}, y_{m+1}\}$ by the lag angle *del*.
Let's estimate the number of iterations within which we should consider the value *del*. For this, let's set the condition:

$$d_i \le 2^{-m}, \tag{20}$$

then the inequality must hold:

$$d_i = 2^{-i} - 2\arctan(2^{-i-1}) \le 2^{-m}. \tag{21}$$

Using the Taylor series expansion for $\arctan(2^{-i-1})$, we obtain:

$$2^{-i} - 2\left(2^{-i-1} - \frac{2^{-3i-3}}{3}\right) \le 2^{-m}, \tag{22}$$

or

$$i = m_a = \lceil (m - 2 - \log_2 3)/3 \rceil \tag{23}$$

- for this value of i, the condition $d_i \le 2^{-m}$ will be satisfied.

Starting from this value $i = m_a$, the approximate equality holds (with accuracy to m bits):

$$2^{-m_a} - 2\arctan(2^{m_a - 1}) \le 2^{-m}. \tag{24}$$

For example if $m = 16$, then $m_a = 5$.

Hence, the value *del* for this m should be computed within only these limits of i:

$$del = \sum_{i=0}^{m_a} a_i d_i. \tag{25}$$

The use of the recoding approach allows excluding one of the three components of CORDIC - determining the next direction of micro-rotations σ_i (approximately one-third of hardware resources are saved in FPGA implementation), and also eliminates the need to continuously store the micro-rotation angles $\arctan(2^{-i-1})$.

Let's divide the input angle θ (1) into three angles:

$$\theta = \theta_1 + \theta_2 + \theta_3 \tag{26}$$

We process angle θ_1 to which we add a constant angle θ_{rc} using the look-up table method (where $m_1 + 1$ is the number of most significant bits of angle θ fed into the LUT, for example, $m_1 = 1...m_2$ for $m = 16$), angle θ_2 using the recoding CORDIC method (bits from $m_1 + 1$ to m_2, for example, $m_2 = 8...m$ for $m = 16$), and angle θ_3 using the method of output multiplication (bits from $m_2 + 1$ to m), where:

$$\begin{aligned} \theta_1 &= \sum_{i=0}^{m_1} a_i 2^{-i}; \\ \theta_2 &= \sum_{i=m_1+1}^{m_2} a_i 2^{-i}; \\ \theta_3 &= \sum_{i=m_2+1}^{m} a_i 2^{-i}; \end{aligned} \tag{27}$$

and $a_i \in \{0, 1\}$.

Applying the table method involves reading precomputed sine and cosine values of the angle $\theta_1 - \theta_{rc}$ from LUT. Here, the angle θ_{rc} is necessary to consider the recoding. These values are obtained using the formulas:

$$\begin{aligned} x_{m_1} &= P\cos(\theta_1 - \theta_{rc}); \\ y_{m_1} &= P\sin(\theta_1 - \theta_{rc}); \\ P &= \prod_{i=m_1+1}^{m_2} \frac{1}{\sqrt{1 + 2^{-2i-2}}}; \\ \theta_{rc} &= -\sum_{i=m_1+1}^{m_2} \arctan(2^{-i-1}). \end{aligned} \tag{28}$$

For angle θ_2, we use the iterative equations of our recoding for CORDIC:

$$\begin{aligned} x_{i+1} &= x_i - b_i y_i \cdot 2^{-i-1}; \\ y_{i+1} &= y_i + b_i x_i \cdot 2^{-i-1}; \\ i &= (m_1 + 1, ..., m_2); \end{aligned} \tag{29}$$

In the final stage, we use the method of output multiplication with the angle z,

$$\begin{aligned} x_{m+1} &= x_{m_2+1} - z y_{m_2+1}; \\ y_{m+1} &= y_{m_2+1} + z x_{m_2+1}. \end{aligned} \tag{30}$$

The formula defines the angle z:

$$z = \theta_3 + del, \tag{31}$$

where $del = \sum_{i=m_1+1}^{m_a} a_i[2^{-i} - 2\arctan(2^{-i-1})]$.

The uniqueness of our approach to angle conversion in the CORDIC method compared to existing ones lies in the following modifications:

– we propose to organise the computation of the sine and cosine of the input Theta angle with a scheme of LUT+ CORDIC+ linear interpolation, which gives the flexibility to choose the number of older bits of the angle (m- exponential of the Theta argument represented in binary code) that determine the size of the LUT type array and reduce the number of CORDIC iterations. This flexibility gives the possibility to dispense with a rigidly defined LUT size up to m/3 as in [14];
– this flexibility of our approach additionally gives the possibility to organise the implementation of the computation scheme: CORDIC+ linear interpolation [15], LUT + linear interpolation (the fastest computation scheme in the approach) according to the user's needs;
– we propose a new conversion approach taking into account the deformation of the input theta angle (an absolutely unique approach that did not exist before) when using a LUT, which gives the possibility to reduce the error significantly (which cannot be done in [14]).

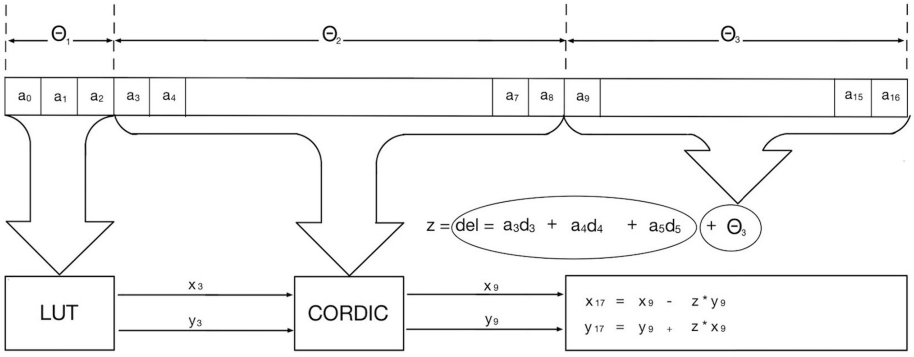

Fig. 1. Schematic view of our CORDIC algorithm

Our modified, recoded CORDIC algorithm is presented in the schematic view in Figure 1 and its FPGA implementation block scheme in Figure 2, where it is easy to observe the flow and relationships for signals and variables. This approach allows for the efficient implementation of an FPGA architecture aimed at achieving high computational throughput. The physical realization of the algorithm consists of three basic blocks: A lookup table (LUT), CORDIC, and Multiplier. The presented scheme is directly linked to the pseudo-code of the Algorithm 1. Also shown is the full cycle of calculating the output values for an example angle in Algorithm 2.

The LUT block has been implemented with a 3-bit input and two 20-bit outputs. The output of the LUT gives the values $x3$, $y3$ according to formula (11), which serve as input parameters for the CORDIC block. In our approach,

Algorithm 1 Pseudocode of the proposed algorithm.

Input a in binary format
$a \leftarrow \{0,1,1,1,1,0,1,1,1,1,0,0,1,0,0,1,1\}$
$xx \leftarrow \{1037958, 973957, 849399, 672030,$
$452877, 205567, -54525, -311226\}$
$yy \leftarrow \{128268, 381076, 610190, 801365, 942715,$
$1025452, 1044432, 998473\}$
$x \leftarrow \{0,0,0,0,0,0,0,0,0,0\}$
$y \leftarrow \{0,0,0,0,0,0,0,0,0,0\}$
$j \leftarrow a[0] \times 4 + a[1] \times 2 + a[2]$
$x[3] \leftarrow xx[j]$
$y[3] \leftarrow yy[j]$
for $i \leftarrow 3$ **to** 8 **do**
  if $a[i] == 1$ **then**
    $x[i+1] \leftarrow x[i] - (y[i] >> (i+1))$
    $y[i+1] \leftarrow y[i] + (x[i] >> (i+1))$
  else
    $x[i+1] \leftarrow x[i] + (y[i] >> (i+1))$
    $y[i+1] \leftarrow y[i] - (x[i] >> (i+1))$
  end if
end for
$d3 \leftarrow 170$
$d4 \leftarrow 21$
$d5 \leftarrow 3$
$del \leftarrow a[3] \times d3 + a[4] \times d4 + a[5] \times d5$
$Theta3 \leftarrow a[9] \times 2048 + a[10] \times 1024 + a[11] \times 512 + a[12] \times 256 + a[13] \times 128 + a[14] \times 64 + a[15] \times 32 + a[16] \times 16$
$z \leftarrow del + Theta3$
$x17 \leftarrow x[9] - ((z \times y[9]) >> 20)$
$y17 \leftarrow y[9] + ((z \times x[9]) >> 20)$
return $x17, y17$

we can arbitrarily choose the number of the most significant bits of the angle θ (the argument) in the binary code, which determines the size of the LUT and reduces the number of iterations. The table size can vary from 1 to m/2 at the designer's request (note that with m/2, there are no CORDIC iterations, and the algorithm's structure takes the form of LUT + linear interpolation).

In the CORDIC block, each iteration of the algorithm from $x3$ to $x9$ is implemented as a pipeline stage by logic elements, and it has two 20-bit outputs. This block has a pipelined structure consisting, in this case, of 5 stages. Each stage has two adders with an output register for each adder.

The Multipliers block performs hardware multiplication by the residual angle z, as described in theory and Algorithm 1.

The outputs of the Multipliers block are 20 bits, and they are implemented by dedicated blocks on the FPGA. The efficiency of the proposed approach is achieved by implementing a table-based computation method for the most significant bits of the input angle. It reduces the number of iterations of the CORDIC

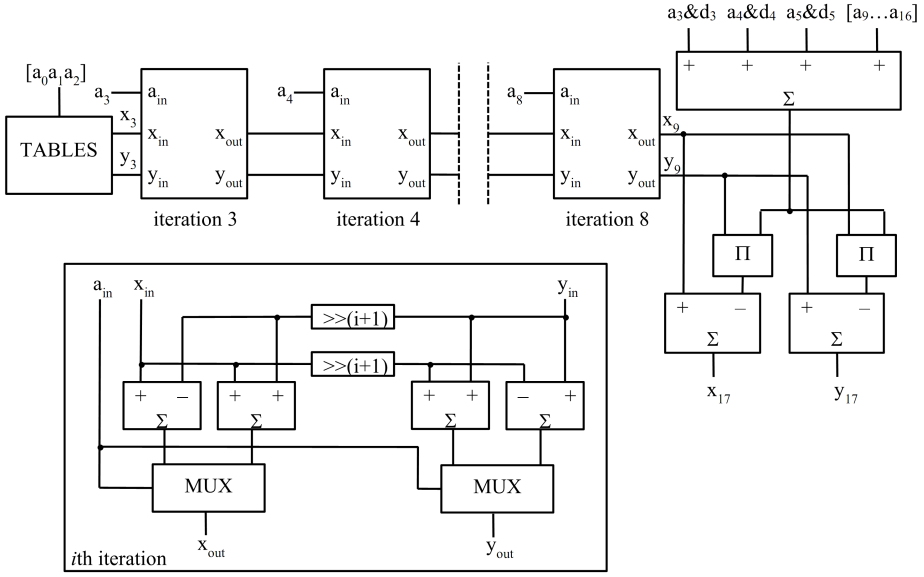

Fig. 2. Implementation block scheme of the algorithm in FPGA chip

block. In addition, our approach allows for flexible resizing of the table (number of high-angle bits as an input LUT parameter). However, we have used only the three oldest bits for this purpose. In the best-known FPGA implementation of the conventional CORDIC algorithm for high throughput, all iterations from x1 to x16 are unrolled, and each iteration is implemented as a pipelined stage. Such an implementation method, therefore, requires high latency and many logic elements. In contrast, our implementation of the CORDIC requires fewer logic elements. Moreover, it causes shorter latency due to fewer pipeline stages. It is worth mentioning that computing m-bit sin(x) and cos(x) in a pipeline, fashion requires 3(m+g) additions of size m+g where g is a number of guard bits ensuring last-bit accuracy [4].

The remainder of this article examines the practical implementation of the proposed method on an FPGA circuit and compares it with the well-known built-in CORDIC implementation in the Altera library and other CORDIC implementations presented in the literature. A comprehensive analysis of the current advancement in CORDIC algorithms is presented in [15]. We compared our approach to the best implementation of the CORDIC algorithms presented in this article.

3 FPGA Implementation and Results

The hardware implementation is done in Verilog language in the Quartus Prime environment for the Cyclone EP4CE115 FPGA chip. The implementation of

the algorithm in the FPGA chip is made of basic logic elements (gates, registers, flip-flops) - without DSP blocks (to have a fair comparison with other implementations). Figure 2 shows a block diagram of the implementation of the algorithm in an FPGA.

The function calculating $sin()$ and $cos()$ values from Altera IP Core libraries (ALTERA_CORDIC) are also implemented for comparison. For the ALTERA_CORDIC libraries, implementations with outputs with 16 bits of fractional precision (fixed-point) are made. Furthermore, the implementations of the ALTERA_CORDIC library are made only on logic elements (without DSP blocks).

The Altera library supports fixed-point calculations with a computational core controlled by latency or frequency. Code for the VHDL or Verilog HDL description language can be generated. The returned results are rounded to one of the two closest numbers that can be represented in the selected format.

The ALTERA_CORDIC library supports several functions:

- determining trigonometric sine and cosine functions for a given angle (SinCos Function);
- determining the value for 2-argument arctangent (Atan2 Function);
- determining the angle and magnitude of the input vector for the given input coordinates (Vector Translate Function);
- determining the coordinates of a point after rotation by a given angle (Vector Rotate Function).

For the purposes of the conducted research, the focus was only on SinCos Function. This function can be configured in two ways, depending on the input angle range:

- for the angle range $[-\pi, +\pi]$ (signed), the values of the sine and cosine functions take the full range $[-1, +1]$ (this is important because some systems do not return the boundary values -1 and $+1$);
- for the angle range $[0, +\pi/2]$ (unsigned), the output sine and cosine values take the full range $[0, +1]$.

When configuring the library, it is possible to set additional parameters (format) of inputs/outputs. The following can be entered: number of fraction bits (1 to 64, but there must be no more for the output than for the input); width of fixed-point data and the sign of the fixed-point data (signed or unsigned).

Figure 3 shows the implementation model of both versions of the CORDIC algorithm.

3.1 Testing scenario and evaluation methodology

The performance evaluation of CORDIC algorithms on FPGA devices differs from their evaluation on general-purpose computing platforms like CPUs and GPUs. While similar metrics such as speed, precision, and resource usage are considered, the architectural differences between FPGAs and these platforms lead to distinct evaluation criteria.

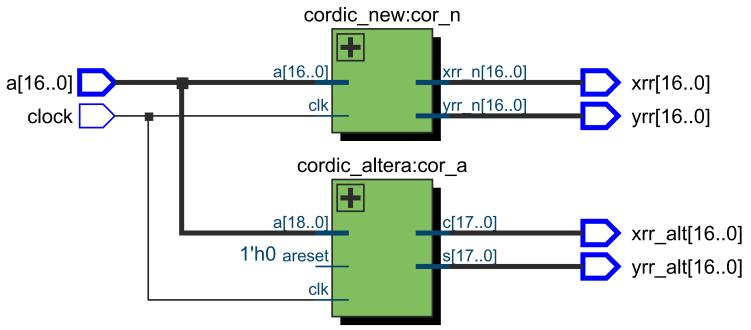

Fig. 3. Implementation model of both versions of the CORDIC algorithm (from RTL tool in Quartus Prime)

For CPUs and GPUs, performance is often measured in terms of execution time, computational accuracy, and resource efficiency (e.g., cycles, bandwidth, and cache use). Energy efficiency is particularly critical in high-performance and mobile computing. The performance evaluation of CPUs and GPUs and their key metrics are well described in various studies[7,2,12,6,8].

FPGA-based evaluations emphasize the unique flexibility and hardware configurability of these devices. FPGAs excel in implementing repetitive computations, such as those required by CORDIC algorithms, through parallelism and pipelining. Key metrics in this context include:

- **Throughput/Latency**: While both are critical, FPGA designs often optimize throughput—completing multiple operations concurrently—while maintaining acceptable latency levels.
- **Precision/Accuracy**: FPGAs enable customization of data bit-widths, optimizing computational precision for specific applications. This reduces resource consumption and power usage, offering a balance between accuracy and efficiency.
- **Logic Occupation**: Efficient use of FPGA resources, such as Look-Up Tables (LUTs) and Flip-Flops, is vital for achieving high performance without excessive costs.

For testing purposes, 104 input samples (θ angle) are prepared. They are determined with a fit to the bit representation (register lengths).

We measure the following metrics:

- Occupancy of the FPGA chip - number of LUTs (Lookup Tables) - values returned from the report after compilation in the Quartus Prime environment;
- Maximum and minimum absolute error - determined according to:

$$\begin{aligned} &\max\{q(j) - q_{fp}(j)\} \\ &\min\{q(j) - q_{fp}(j)\} \end{aligned} \tag{32}$$

for $j = 1, 2, ..., 104$, where $q(j)$ is the value obtained from a given implementation, $q_{fp}(j)$ is the full precision value for a given angle (determined in the SciLab computer environment);
– Time required to return the result (latency) - determined as the minimum period (=1/clock frequency) for which all 104 input samples (given one after another) returned the expected result.

We compare [15] and aggregate the results in tables 1, 2 and 3.

Table 1. Results of FPGA occupancy.

CORDIC Realisation	LUTs
Presented CORDIC algorithm	1597
ALTERA_CORDIC with 16-bits fractional outputs	2866
CORDIC algorithm [15]	1438
BBR-CORDIC [34] from [15]	1643
CORDIC II [38] from [15]	1433

Table 2. Results of response time (latency).

CORDIC Realisation	Time (ns)
Presented CORDIC algorithm	56
ALTERA_CORDIC with 16-bits fractional outputs	94
CORDIC algorithm [15]	60
BBR-CORDIC [34] from [15]	60
CORDIC II [38] from [15]	90

Table 3. Results of maximum and minimum absolute errors.

CORDIC Realisation	Max error	Min error
Presented CORDIC algorithm	4.78e-06 (sin) 4.84e-06 (cos)	-1.12e-05 (sin) -1.27e-05 (cos)
ALTERA CORDIC with 16-bit fractional outputs	1.33e-05 (sin) 1.37e-05 (cos)	-1.21e-05 (sin) -1.14e-05 (cos)
CORDIC algorithm [15]	3.04e-05 (sin) 3.04e-05 (cos)	-3.04e-05 (sin) -3.04e-05 (cos)

In addition, it is worth noting that the ALTERA_CORDIC library used does not get the configured parameters. Declaratively, the implemented version of ALTERA_CORDIC should return a result in less than 53ns. The results from ALTERA_CORDIC for the 104 samples used were unsatisfactory - most of the

results were incorrect. As the clocking was lowered, the results improved. The ALTERA_CORDIC library returned correct results only when feeding samples with a period of 94ns - much more than the declared value. At the same time, ALTERA_CORDIC has the highest occupancy, which is due to the poor optimization of the ready-made library 1. The only advantage of this library is the accuracy (precision of calculations), which is comparable with other implementations 3. Presented CORDIC algorithm has occupancy (LUTs) at a very good level - only slightly higher than the lowest results 1. The advantage of the presented algorithm is the latency time (56ns) which allows higher sampling (and processing) frequencies 2. The aspect of smaller errors is also important (even an order lower than for other algorithms in the case of max errors) 3. Other errors are at similar levels (about 1.2e-05). Numerical accuracy is preserved while providing faster processing. CORDIC algorithm [15] has by far the largest errors - which may be related to almost the lowest hardware occupancy (LUTs). Typically, lower occupancy is obtained by reducing fractional bits, and this results in larger errors.

4 Conclusion

The use of our approach has allowed high performance and efficiency improvements of the CORDIC method, with up to 40% power consumption reduction and up to 21% delay reduction, offering overall occupancy reduction of up to 44% (compared to the presented implementation of ALTERA_CORDIC). The approximate CORDIC has been characterized by its absolute arithmetic error, which shows negligible average and maximal errors, i.e., RMS relative errors being only 0.0014%.

In this work, we give the basics of the recoding theory and formalize it in detail using algorithms suitable for hardware implementation on the FPGA platform. We propose a flexible algorithm for freely choosing the number of most significant bits of the angle θ that sets the size of the LUT and reduces the number of iterations. In addition, the difference from [3] is that the residual angle, by which the values from the CORDIC output are multiplied, is always positive. This simplifies the structure of the output multipliers (unsigned multiplication can be used). The proposed approach simplifies the angle recoding method's software and hardware implementation and gives it an advantage over the built-in library functions offered by FPGA developers.

Acknowledgments. 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. Antelo, E., Bruguera, J., Zapata, E.: Unfolded redundant cordic vlsi architectures with reduced area and power consumption. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **31**(5), 872–880 (May 2023)

2. Ciznicki, M., Kopta, P., Kulczewski, M., Kurowski, K., Gepner, P.: Elliptic solver performance evaluation on modern hardware architectures. Parallel Processing and Applied Mathematics **8384** (2014). https://doi.org/10.1007/978-3-642-55224-3_16
3. Curticapean, F., Niittylahti, J.: A hardware efficient direct digital frequency synthesizer. IEEE Journal of Solid-State Circuits **58**(4), 876–884 (Apr 2023)
4. Dinechin, F., Istoan, M., Sergent, G.: Fixed-point trigonometric functions on fpgas. ACM SIGARCH Computer Architecture News **41**, 83–88 (06 2014). https://doi.org/10.1145/2641361.2641375
5. Doe, J., Smith, J., Johnson, A.: Implementation of floating point cordic algorithm using 45 nm technology. IEEE Transactions on Computers **72**(3), 450–458 (Mar 2023)
6. Gepner, P.: Using avx2 instruction set to increase performance of high performance computing code. Computing and Informatics **36**(5), 1001–1018 (2017)
7. Gepner, P., Fraser, D.L., Kowalik, M.F.: Second generation quad-core intel xeon processors bring 45 nm technology and a new level of performance to hpc applications. Computational Science – ICCS 2008, ICCS 2008 **5101** (2008)
8. Gepner, P., Gamayunov, V., Fraser, D.L.: Effective implementation of dgemm on modern multicore cpu. Procedia Computer Science **9**, 126–135 (2012). https://doi.org/10.1016/j.procs.2012.04.014
9. Hu, Y.H., Naganathan, S.: An angle recoding method for cordic algorithm implementation. IEEE Transactions on Computers **42**(1), 74–79 (Jan 1993)
10. Juang, T.: Low latency angle recoding methods for the higher bit-width parallel cordic rotator implementations. IEEE Transactions on Circuits and Systems II: Express Briefs **55**(11), 1139–1143 (Nov 2008)
11. Juang, T.B., Hsiao, S.F., Tsai, M.Y.: Para-cordic: Parallel cordic rotation algorithm. IEEE Transactions on Circuits and Systems I: Regular Papers **51**(8), 1515–1524 (Aug 2004)
12. Kopta, P., Kulczewski, M., Kurowski, K., Piontek, T., Gepner, P., Puchalski, M., Komasa, J.: Parallel application benchmarks and performance evaluation of the intel xeon 7500 family processors. Procedia Computer Science **4**, 372–381 (2011). https://doi.org/10.1016/j.procs.2011.04.039
13. Kuhlmann, M., Parhi, K.K.: P-cordic: A precomputation based rotation. EURASIP Journal on Applied Signal Processing **2002**(1), 936–943 (2002). https://doi.org/10.1155/S1110865702205028
14. Madisetti, A., Kwentus, A.Y., Willson, A.N.: A 100 mhz, 16-b, direct digital frequency synthesizer with 100-dbc spurious-free dynamic range. IEEE Journal of Solid-State Circuits **34**(8), 1034–1043 (1999)
15. Qin, M., Liu, T., Hou, B., Gao, Y., Yao, Y., Sun, H.: A low-latency rdp-cordic algorithm for real-time signal processing of edge computing devices in smart grid cyber-physical systems. Sensors **22**(19) (2022). https://doi.org/10.3390/s22197489
16. Timmermann, D., Hahn, H., Hosticka, B.: Low latency time cordic algorithms. IEEE Transactions on Computers **41**, 1010–1015 (1992)
17. Volder, J.E.: The cordic trigonometric computing technique. IEEE Transactions on Electronic Computers **EC-8**(3), 330–334 (Sep 1959)
18. Walther, J.S.: A unified algorithm for elementary functions. In: Proc. AFIPS Conf. vol. 38, pp. 385–389 (1971)
19. Wang, S., Piuri, V., Swartzlander, E.E.: Hybrid cordic algorithms. IEEE Transactions on Computers **46**(11), 1260–1263 (Nov 1997)

Algorithm 2 Example of calculation of values according to the presented algorithm for a selected angle (step by step).

The vectors xx, yy, x, y a values d3, d4, d5 are completed as in Algorithm 1.
$Theta = 0.967086792(rad) \times 2^{16} = 63379 = (01111011110010011)\text{BCD}$
$a = \{0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1\}$
$j = 3$
$x[3] = 672030$
$y[3] = 801365$
for $i = 3$ **to** 8 (consecutive iterations) **do**
  for $i = 3$ and $a = 1$
    $x[4] = 621945$
    $y[4] = 843366$
  for $i = 4$ and $a = 1$
    $x[5] = 595590$
    $y[5] = 862801$
  for $i = 5$ and $a = 0$
    $x[6] = 609071$
    $y[6] = 853495$
  for $i = 6$ and $a = 1$
    $x[7] = 602404$
    $y[7] = 858253$
  for $i = 7$ and $a = 1$
    $x[8] = 599052$
    $y[8] = 860606$
  for $i = 8$ and $a = 1$
    $x[9] = 597372$
    $y[9] = 861776$
end for
$del = 191$
$Theta3 = 2352$
$z = 2543$
$x17 = 595283$
$y17 = 863224$
$x17/2^{20} = 0.567706108(\cos)$
$y17/2^{20} = 0.823234558(\sin)$