CORDIC algorithms

System On Chip Design Laboratory
Course number 384.178

Enhancing performances

Giacomo Giannetti 12006424
e12006424@student.tuwien.ac.at
Amer Ahmic 1225650
Benjamin Gräf 51832224
e51832224@student.tuwien.ac.at
Zifei Li 51813032
e51813032@student.tuwien.ac.at

23 February 2021

Contents

1 Introduction

In the recent years it has became more important the possibility to design algorithms characterized by a compromise between accuracy and resources [1]. This work focuses on the COordinate Rotation Digital Computer (CORDIC) algorithm that, in its circular version, allows to evaluate the trigonometric sine and cosine functions and that has been introduced by Volder [2]. During the years several possible implementations of such an algorithm have been proposed, someone performing exact computations and others approximated ones [3]. In the following, the basic CORDIC, the parallel CORDIC (Para-CORDIC) and the fully parallel approximated CORDIC (FPAX-CORDIC) are presented and compared in terms of accuracy, power, delay and area. The first two versions of the algorithm perform exact calculations, in the limit of the prescribed word length precision, while approximation is introduced in the last one by means of the *error tolerant parameter* p.

2 Algorithm descriptions

In this section the mathematical operations for the three algorithms under consideration are shown. In all cases, the input angle θ is supposed to lie between $-\pi/4$ and $+\pi/4$ in order to simplify the treatment and due to the fact that it is straightforward to convert into this bound a whatsoever input angle.

2.1 Basic CORDIC

The basic CORDIC algorithm is a sequential algorithm that performs the rotation of a whatsoever two-dimensional vector (x_0, y_0) of an angle θ, obtaining the final vector (x'_n, y'_n). This operation can be performed by a matrix-vector product

$$\begin{pmatrix} x'_n \\ y'_n \end{pmatrix} = R \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

where the matrix R is called rotation matrix. Gathering the cosine and exploiting the equality

$$\cos(\beta) = \frac{1}{\sqrt{1 + \tan^2(\beta)}}, \quad \beta \in [-\pi/2, +\pi/2]$$

it is possible to rewrite the rotation matrix R as

$$R = K \begin{bmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{bmatrix} \tag{1}$$

where $K = (1 + \tan^2(\theta))^{-1/2}$. To allow a hardware implementation, the basic CORDIC decomposes the input angles into a series of elementary rotations through predefined angles, α_i, as indicated in Fig. 1. The values α_i are chosen such that $\tan(\alpha_i) = 2^{-i}$: this permits to perform the multiplication of the tangent term in (1) by means of shift operation through i bit positions. Instead of performing a single rotation of amplitude θ, the basic CORDIC executes a number n of microrotations through angle α_i such that

$$\theta = \sum_{i=0}^{n-1} \sigma_i \alpha_i$$

where α_i is the magnitude of the i^{th} rotation angle and $\sigma_i = \pm 1$ determines the i^{th} rotation direction: counterclockwise if 1, clockwise if -1. The rotation matrix for the i^{th} iteration is then given by

$$R_i = K_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix}$$

where $K_i = (1 + 2^{-2i})^{-1/2}$ being the scale factor. Considering all the rotations together, the final vector is given by

$$\begin{pmatrix} x'_n \\ y'_n \end{pmatrix} = \left(\prod_{i=0}^{n-1} R_i \right) \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = K \left(\prod_{i=0}^{n-1} \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \right) \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

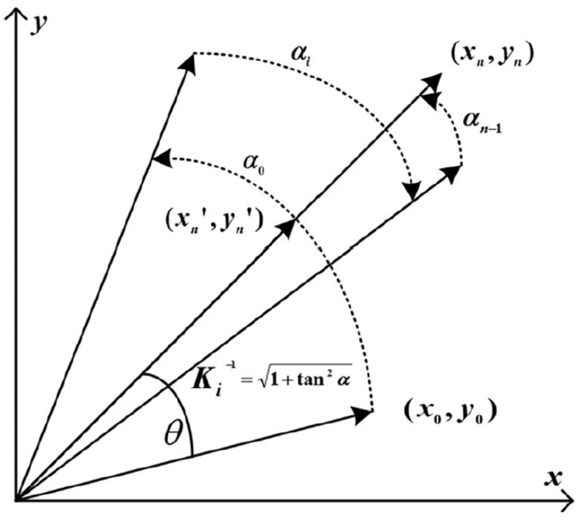


Figure 1: Two-dimensional vector rotation.

where $K = \prod_{i=0}^{n-1} K_i$ is a scaling factor not depending on the direction and amplitude of the rotation. Moreover, it is monotonically decreasing and $\lim_{i\to\infty} K_i \approx 0.60725$. Summarizing, the iterative equations of the CORDIC algorithm for radix-2 in circular coordinate systems are as follows:

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i y_i 2^{-i} \\ y_{i+1} &= \sigma_i x_i 2^{-i} + y_i \\ \theta_{i+1} &= \theta_i - \sigma_i \tan^{-1}(2^{-i}) \\ \sigma_i &\in \{-1, 1\} \\ i &= 0, 1, 2, \ldots, n-1. \end{aligned} \tag{2}$$

Finally, the vector $(x_n', y_n')^T$ is obtained multiplying $(x_n, y_n)^T$ by the scalar quantity K. Eq. (2) is often referred to as the CORDIC equation and can be use in either rotation mode and vectoring mode. In our case, the rotation mode has been considered, allowing the calculation of the trigonometric functions sine and cosine. The direction of the rotation in any iteration is determined using the sign of the residual angle θ_i found in the previous operation, that is, $\sigma_{i+1} = sign(\theta_{i+1})$. Let $x_0 = K$, $y_0 = 0$, after n iterations, the final outputs are the sine and the cosine functions, $x_n = \cos(\theta)$ and $y_n = \sin(\theta)$.

2.2 Para-CORDIC

The sequential termination of the i^{th} direction rotation limits the speed of the conventional CORDIC. At each iteration the value σ_i is know only after that the operation $\theta_i = \theta_{i-1} - \tan^{-1}(2^{-i})$ has been performed. As outlined in [4], a possible way to overcome this drawback is to predict the direction rotation directly from the binary representation of the input angle θ, resulting in a parallel algorithm. The input angle can be represented in two's complement binary representation in the following way

$$\theta = \theta^L + \theta^H = \underbrace{(-b_0) + \sum_{j=1}^{l-1} b_j 2^{-j}}_{\theta^L} + \underbrace{\sum_{j=l}^{N} b_j 2^{-j}}_{\theta^H} \tag{3}$$

where l is the smallest index value such that $2^{-l} - \tan^{-1} 2^{-l} < 2^{-N}$. As written in [5] this value is equal to $\lceil (N - \log_2(3)/3 \rceil$. Furthermore, $b_j \in \{0, 1\}$, $j = 0, 1, 2, \ldots N$.

As next step a Binary-to-Bipolar Recoding (BBR) is applied to the first $l-1$ bits of the input angle, i.e., θ^L, in order to retrieve the corresponding rotation directions, from σ_1 to σ_l. The binary values can be converted to the corresponding bipolar representation $r_k \in \{-1, 1\}$ in the following way:

$$\theta = (-b_0) + \sum_{j=1}^{N} b_j 2^{-j} \quad = (-b_0) + \sum_{j=1}^{N} \left[2^{-j-1} + (2b_j - 1)2^{-j-1}\right] = \sum_{k=1}^{N+1} r_k 2^{-k} - 2^{-N-1} \tag{4}$$

where $r_1 = 1 - 2b_0$ and $r_k = (2b_{k-1} - 1)$, $k = 2, 3, \ldots, N$. Applying (4) to the angle θ^L, we obtain

$$\theta^L = (-b_0) + \sum_{j=1}^{l-1} b_j 2^{-j} = \sum_{j=1}^{l} r_j 2^{-j} - 2^{-l} \tag{5}$$

with $r_0 = 1 - 2b_0$ and $r_i = 2b_{i-1} - 1$, $i = 2, 3, \ldots, l$. The bipolar values r_i can be interpreted as the rotation directions σ_i. To show this, a generic power of two 2^{-i} must be expanded as sum of arctangent of $2^{-s_i^j}$. To do so, the Microrotation Angle Recoding (MAR), introduced in [4], is applied. It shows that

$$2^{-i} = \sum_{j=1}^{n(i)} \tan^{-1}\left(2^{-s_i^j}\right) + e_i, \quad s_i^1 = i \tag{6}$$

for $i = 1, 2, \ldots, l-1$. The term e_i is a positive error term accounting for the finite number of term $\tan^{-1}\left(2^{-s_i^j}\right)$ that are considered. Substituting (6) into (5) leads to the equation

$$\theta^L = \sum_{i=1}^{l} \sigma_i \left\{ \sum_{j=1}^{n(i)} \tan^{-1}\left(2^{-s_i^j}\right) + e_i \right\} - 2^{-l} \tag{7}$$

In (7) each term of the type $\tan^{-1}\left(2^{-s_i^j}\right)$ is associated with a rotation whose angle is $2^{-s_i^j}$ great and whose direction is determined by σ_i. It follows that the angle θ^L is decomposed in $\sum_{i=1}^{l} n(i)$ rotations. Furthermore, (7) tells us the different approach adopted by the Para-CORDIC with respect to the basic CORDIC. Indeed, while in the latter is the tangent of the rotation angle to be written as power of two, in the former is the rotation angle itself to be written as power. This implies that, on one hand, the rotation angles must be computed apart and stored in memory, while on the other the error e_i are found and saved in memory. This is the key difference between the two algorithms. Considering again (7), each index i is associated with $n(i)$ rotations in direction σ_i and shift sequence s_i^j. Taking into account Eqs. (3), (7) and (6), the corrected remaining rotation angle $\hat{\theta}^H$ can be derived as

$$\hat{\theta}^H = \theta^H + \sum_{i=1}^{l-1} \sigma_i e_i - 2^{-l} \tag{8}$$

In order to approximate within accuracy of N fractional bits each binary weighting of 2^{-i} in the remaining angle by $\tan^{-1}(2^{-i})$, $|\hat{\theta}^H|$ must satisfied the constraint:

$$\left|\hat{\theta}^H\right| < 2^{-(l-1)} \tag{9}$$

In order to be true, [4] proves that the errors e_i must fulfil the requirement

$$\sum_{i=1}^{l-1} e_i < 2^{-l} \tag{10}$$

Eq. (10) suggests a practical way to find the errors e_i and the sequences s_i^j of the MAR algorithm. In particular, we implemented in Matlab the algorithm proposed in [4].

Once that the corrected higher part $\hat{\theta}^H$ has been found, it is treated in the same way of the lower part. This means that a two's complement format is considered and then a BBR is applied to find the last $N - l + 2$ rotation directions. In particular, we have

$$\begin{aligned} \hat{\theta^H} &= \theta^H + \sum_{i=1}^{l-1} \sigma_i e_i - 2^{-l} \\ &= (-\hat{b}_{l-1}) 2^{-l+1} + \sum_{k=l}^{N} \hat{b}_k 2^{-k} \\ &= \sum_{i=l}^{N+1} \hat{r}_i 2^{-i} - 2^{N-1} \end{aligned} \tag{11}$$

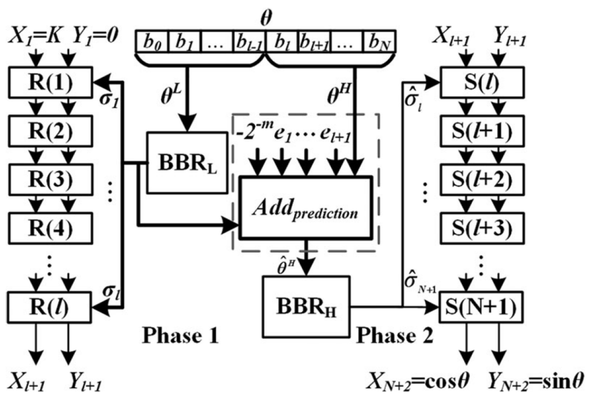


Figure 2: Para-CORDIC scheme

where $\hat{b}_k$ $k = l, l+1, \ldots, N$ are binary values and $\hat{r}_i$ $i = l, l+1, \ldots, N+1$ bipolar values: $\hat{r}_l = 1 - 2\hat{b}_{l-1}$ and $\hat{r}_i = (2\hat{b}_{i-1} - 1)$, $i = l+1, l+2, \ldots, N+1$. The remaining $N + 2 - l$ rotation directions $\hat{\sigma}_l$ to $\hat{\sigma}_{N+1}$ can be obtained directly from the recoding results $\hat{r}_l$ to $\hat{r}_{N+1}$ in a way similar to (7) but in this case $2^{-i} - \tan^{-1}(2^{-i}) < 2^{-N}$.

The presented Para-CORDIC scheme is shown in Fig. 2. There are two BBRs required in the Para-CORDIC rotation and they have been denoted by BBR_L and BBR_H and represent the operations of Eqs. (5) and (11). The operation of 8 is called $Add_{prediction}$. Then, there are two phases of microrotations. Phase 1 contains $(l-1)$ blocks of $R(i)$ plus one block of $S(l)$ while phase 2 has $(N-l+2)$ blocks of $S(i)$, $i = l, \ldots, N+1$. A block $R(i)$ is composed of $n(i)$ cascaded sub-blocks of $S(s_i^j)$, $j = 1, 2, \ldots, n(i)$. A block $S(i)$ corresponds to a single rotation of a 2^{-i} angle. Due to the extra microrotations, the constant scaling factor is

$$K' = \left[\prod_{i=1}^{l-1}\left(\prod_{j=1}^{n(i)}\left(1 + 2^{-2s_i^j}\right)^{-1/2}\right)\right](1 + 2^{-2l})^{-1/2}\prod_{i=l}^{N+1}(1 + 2^{-2i})^{-1/2}.$$

2.3 FPAX-CORDIC

Although Para-CORDIC parallelizes the computation of σ_i in two phases, it is still present a dependency between Phase 1 and Phase 2. Thus, the generation of the rotation direction σ_i is still not fully parallel. The connection between the two phases is realized by the block $Add_{prediction}$: it takes into account the errors made in the expansion of 2^{-i} by means of arctangent terms. Fur a fully parallel execution either the error compensation must be performed differently, or the error should be tolerated. We followed the second possibility since interested in an approximate computing technique. Eliminating the block $Add_{prediction}$ the algorithm is fully parallel and there is no need to store in memory the error compensation terms e_i. The result of this operation is the fully parallel approximate CORDIC (FPAX-CORDIC), introduced in [6]. It is now important to study the error introduced by the generation of σ_i and how it can be controlled and recovered in the second rotation phase. In particular, the main function of the $Add_{prediction}$ block is to compensate the error introduce by BBR_L. In MAR the value of the corrected angle $\hat{\theta}^H$ must fulfil the following requirement in (9) in order to approximate the binary weighting of 2^{-i} in the remaining angle by $\tan^{-1}(2^{-i})$ within the accuracy allowed by the N fractional bits. This is true when the condition (10) is satisfied. The complete elimination of the $Add_{prediction}$ block requires that the $\sum_{i=1}^{l-1} e_i = 0$ and implying that

$$\hat{\theta}^H = \theta^H - 2^{-l} \tag{12}$$

and that error compensation is not necessary, eliminating the need for the $Add_{prediction}$ block. Generally speaking, $\sum_{i=1}^{l-1} \sigma_i e_i$ is not zero but it can be controlled so to have it as near to zero as one wishes. Let $\theta^E = \sum_{i=1}^{l-1} \sigma_i e_i$, where e_i is not negative and σ_i assumes only the values ± 1; then $\left|\theta^E\right| = \left|\sum_{i=1}^{l-1} \sigma_i e_i\right| \leq \sum_{i=1}^{l-1} e_i$. So, if $\sum_{i=1}^{l-1} e_i \to 0$, hence $\theta^E \to 0$. For an input angle θ with N-bit precision, let $\sum_{i=1}^{l-1} e_i < 2^{-N}$ so that the error θ^E can be ignored for N-bit precision. Nevertheless, we are not so much interested in an error-free algorithm, that would require more resources in terms of area and power; rather we are

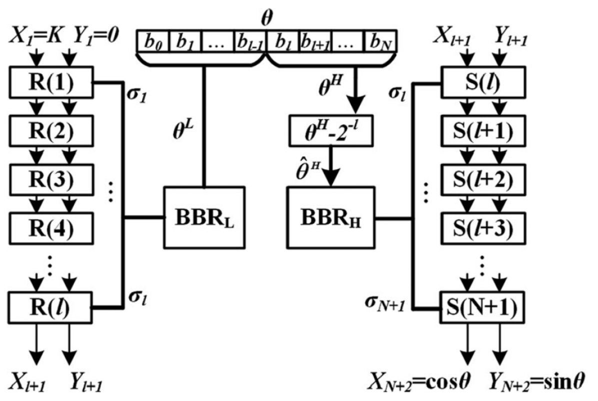


Figure 3: FPAX-CORDIC scheme proposed in [6].

interested in the possibility of introducing a controllable error, loosing the requirements on the resources. Indeed, the lower $\sum_{i=1}^{l-1} e_i < 2^{-N}$, the higher the number of microrotation stages $n(i)$. When the error term $\left|\theta^E\right| < 2^{-p}$ is tolerated for a specific application, then $\left|\theta^E\right| \leq \sum_{i=1}^{l-1} e_i < 2^{-p}$, where $p \in [l, \ldots, N]$ is the so-called *error tolerant parameter*. The architecture of the proposed FPAX-CORDIC is shown in Fig. 3.

3 Hardware implementation

The three algorithm versions have been implemented on the FPGA. The model at our disposal is a Xilinx Artix-7 (xc7a100tcs324) on a Nexys 4 DDR board. The model has DSP slices and Block RAM, which could be used to enhance the performance of the algorithm. The maximum internal clock frequency is 450 MHz. The current implementation of CORDIC is clocked and sequential while that of Para-CORDIC and FPAX-CORDIC is pipelined without clock. The clock frequency is chosen depending on the computation delay.

For testing the function of the algorithm on the hardware the input angle is sent to the board over UART and the result is sent back.

3.1 Basic CORDIC

As for all algorithms the hardware description language VHDL is used.

Unlike the Para-CORDIC and the FPAX-CORDIC the basic cordic algorithm does not have any modules in common with these two algorithms. It is a completely independent implementation.

The functionality is realised with a state machine. Where in the initial state, the two outputs which are the values for the sine and cosine of the desired input angle are set to 0.60725 and 0 respectively. After starting the algorithm by a high signal on the start input line, the algorithm will perform one rotation for each rising of the clock signal. These will be done until the set bit-width is reached and the algorithm stops with its current values for the sine and cosine at the output.

As the angles are presented in radiant with a range from -1 to 1, there is the need for sub decimal representation in VHDL. Fixed point shift is used, because of two main reasons. The first one being that it is much resource efficient compared to a floating point representation cornering the calculations. The second reason is the very small range of the used numbers.

The needed predefined values for the results of the rotations which are needed by the algorithm are stored in a look up table.

3.2 Para-CORDIC and FPAX-CORDIC

Observing the architectures in Figs. 2 and 3, one can notice that both the Para-CORDIC and the FPAX-CORDIC can be seen as composed of the following blocks:

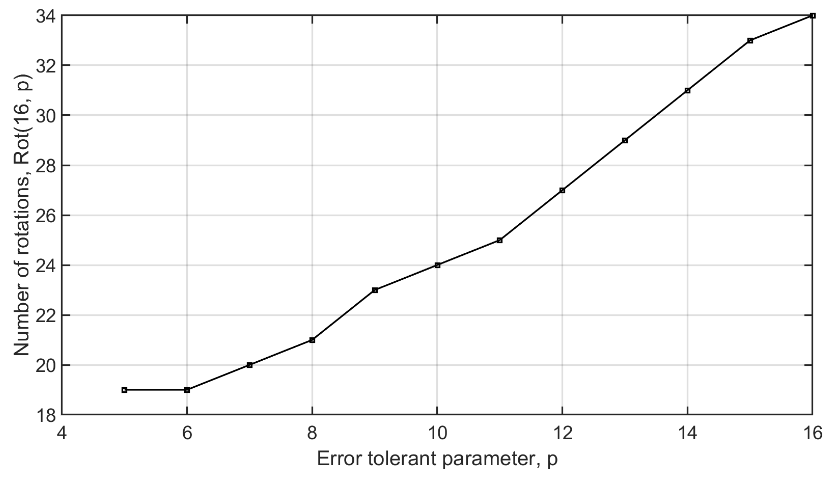
- *X/Y data path*, that is, the rotations performed on the initial vector $(x_0, y_0)^T$ up to achieve the final result $(x_n', y_n')^T$; in turns this can be split in *Phase 1*, composed by the blocks $R(\cdot)$, and *Phase 2*, composed by the blocks $S(\cdot)$;

- *Binary-to-Bipolar Recoding blocks*, replacing the rotation decision scheme of the basic CORDIC;
- θ *path*, that is, the operations required to generate the corrected angle $\hat{\theta}^H$.

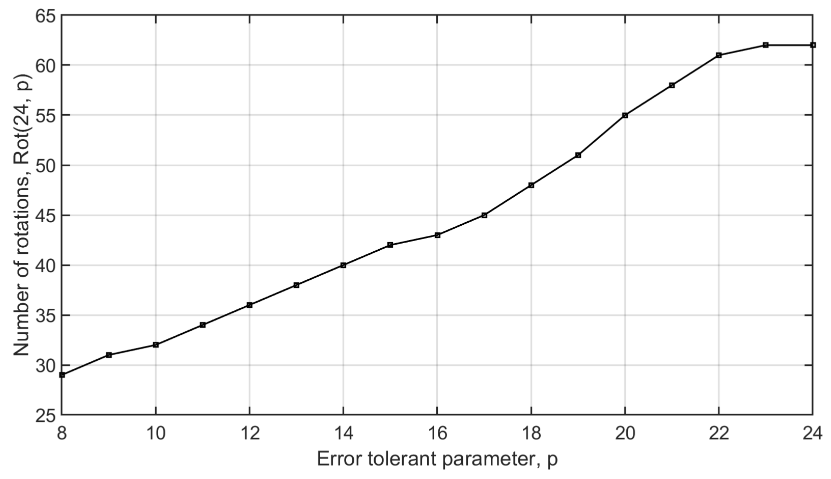
The total number of rotations performed by the *X/Y data path* in the FPAX-CORDIC scheme is $Rot(N,p) = \sum_{i=1}^{l-1} n(i) + N - l + 3$, where the p-dependence is hidden inside the term $n(i)$; instead, regarding the Para-CORDIC, the number of rotations is $\bar{Rot}(N) = Rot(N,l)$. In both cases, $\sum_{i=1}^{l-1} n(i) + 1$ rotations refer to *Phase 1* while the remaining $N - l + 2$ to *Phase 2*. In Fig. 4 the number of rotations $Rot(N,p)$ is displayed for word length N equal to 16, 24 and 32.

Concerning the *Binary-to-Bipolar Recoding blocks*, they cover a control role, since they determine the rotation directions. In particular, they produce as output logical signals that are used to control whether the adders of the rotation blocks need to perform a clockwise or a counterclockwise rotation, that is, according to (2) sums or subtractions.

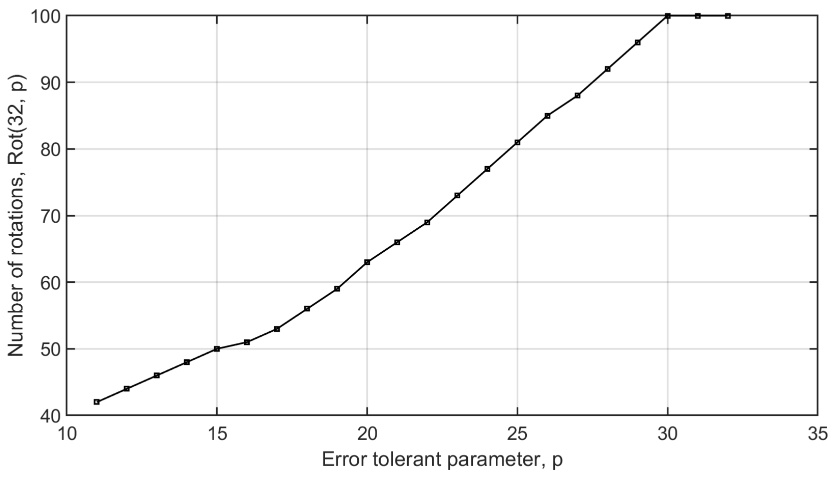
Finally, the θ *path* produces the necessary corrected $\hat{\theta}^H$ that is used to determine the further rotations realized by *Phase 2*. The θ *path* of the Para-CORDIC is more complicated than that of the FPAX-CORDIC since it performs a more accurate correction taking into account the errors that have been committed in approximating a power of two by means of arctangent terms. Its precise architecture realizes (8) and is shown in Fig. 2. The analogous scheme for the FPAX-CORDIC is shown in Fig. 3 and realizes (12).



(a) $N = 16$



(b) $N = 24$



(c) $N = 32$

Figure 4: The figures display the number of microrotations that are performed in the FPAX-CORDIC for a word length N equal to 16, 24 and 32.

$$\begin{array}{rl}
\theta^H & = 0.\overbrace{00\ldots0}^{(m-1)-bits}\, b_m \ldots b_B \\
|\sigma_1 e_1| & = 0.00\ldots0 e_1^m \ldots e_1^B \\
|\sigma_2 e_2| & = 0.00\ldots0 e_2^m \ldots e_2^B \\
 & \bullet \\
 & \bullet \\
|\sigma_{m-1} e_{m-1}| & = 0.00\ldots0 e_{m-1}^m \ldots e_{m-1}^B \\
-\,2^{-m} & = -0.00\ldots0100\ldots0 \\
\hline
\left|\hat{\theta}^H\right| & = 0.\underbrace{00\ldots0}_{(m-1)-bits}\, \hat{b}_m \hat{b}_{m+1} \cdots \hat{b}_B
\end{array}$$

Figure 5: θ *path* for the Para-CORDIC performing (8).

$$\begin{array}{rl}
\theta^H & = 0.\overbrace{00\ldots0}^{(l-1)\,bits} b_l \ldots b_N \\
-\,2^{-l} & = -0.0\ldots0100\ldots0 \\
\hline
\left|\hat{\theta}^H\right| & = 0.\underbrace{00\ldots0}_{(l-1)\,bits} \hat{b}_l \hat{b}_{l+1} \ldots \hat{b}_N
\end{array}$$

Figure 6: θ *path* for the FPAX-CORDIC performing (12).

4 Comparisons

In the following section the three algorithms under study are compared in terms of accuracy, delay, power and area. According to the means utilized in finding the results, delay, power and area comparisons are considered together, while accuracy is considered apart.

4.1 Accuracy

In order to achieve comparable results, all the algorithms are tested with the same input angles, that is, one hundred evenly spaced points comprised between $-\pi/4$ to $\pi/4$. For each version and for each word length in the set $16, 24, 32$, the following graphs are provided:

- difference between the cosine *Matlab* function, $\cos_M$, and the cosine simulation results, $\cos_s$;
- difference between the sine *Matlab* function, $\sin_M$, and the sine simulation results, $\sin_s$.

In order to furnish a more direct idea, the goodness of the three different implementations is then assessed through the variance σ^2

$$\sigma^2 = \frac{1}{N_p} \sum_{i=1}^{N_p} \left[(\cos_M - \cos_s)^2 + (\sin_M - \sin_s)^2 \right] \tag{13}$$

where N_p is the number of points considered in the calculation, that is, one hundred in our case.

It is worth to notice that the errors for both the cosine and the sine and for the variance converge to a limit value when either the number of iterations in the basic CORDIC increases, Figs. 7, 8 and 9, or when the error tolerant parameter p increases, Figs. 10, 11, 12 and 13. Regarding the Para-CORDIC and the Fpax-CORDIC results in it is interesting to observe that the errors for both the cosine and the sine present the same symmetry of the function, that is, even for the cosine and odd for the sine. Finally, Fig. 14 displays how the variance defined in (13) varies for the FPAX-CORDIC design when the error tolerant

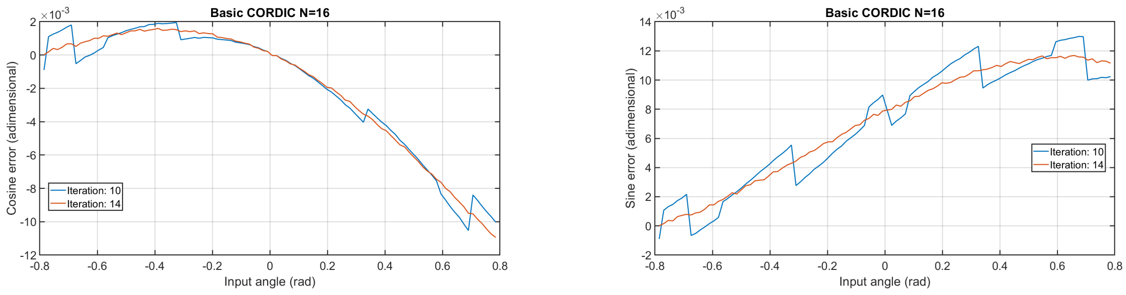


Figure 7: Basic CORDIC, $N = 16$.

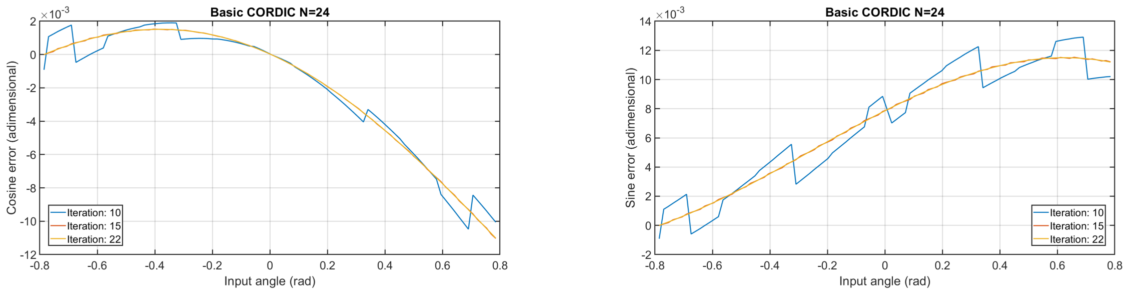


Figure 8: Basic CORDIC, $N = 24$.

parameter p increases; the variance for the basic CORDIC and the Para-CORDIC is, instead, displayed in Tabs. 1 and 2, respectively.

4.2 Delay, power and area

The results shown in this section have been obtained through a functional simulation in the Vivado environment. Furthermore, only the results for the Para-CORDIC and the FPAX-CORDIC are given since they are combinatorial logic while the basic CORDIC is a sequential circuit and therefore the delay is equal to the number of iterations times the clock period and power and area are those requested for a single rotation. The results for the Para-CORDIC are shown in Tab. 3, while those for the FPAX-CORDIC in Tabs. 4, 5 and 6. Focusing on the delay, it increases increasing the word width N since more rotations are requested before reaching the precision of 2^{-N}. Furthermore, differently from what expected, the delay for the FPAX-CORDIC with $N = 16$ decreases increasing the error tolerant parameter p; on the other hand, the figures for $N = 24$ and $N = 32$ respect this trend, that is, the delay increases for increasing p. Analogous considerations can be made also for power: its value increases increasing the word width and it is almost constant varying the error tolerant parameter. Regarding the area, the number of adders and MUXes required has been considered as an index for the occupied area. Observing the tables, the required resources in terms of area increase while increasing the error tolerant parameter.

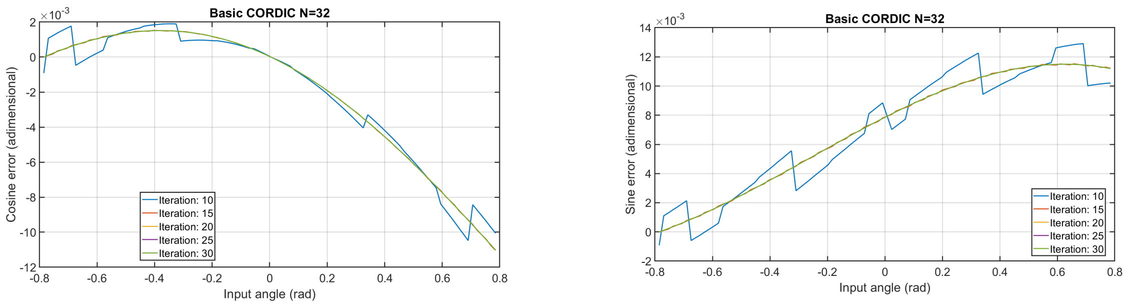


Figure 9: Basic CORDIC, $N = 32$.

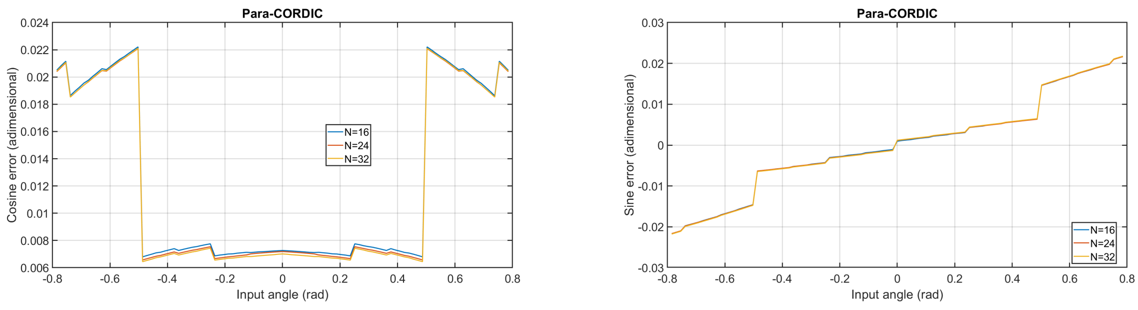


Figure 10: Para-CORDIC.

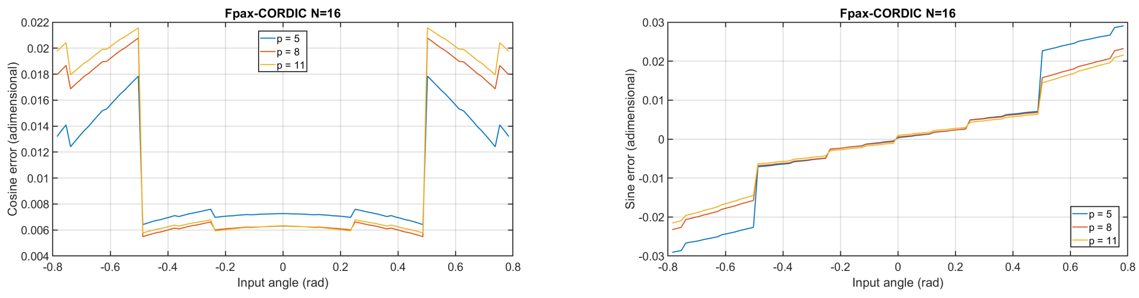


Figure 11: Fpax-CORDIC, $N = 16$.

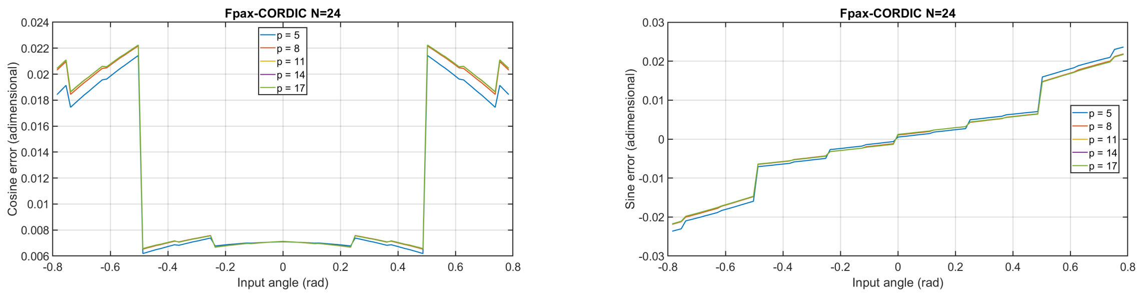


Figure 12: Fpax-CORDIC, $N = 24$.

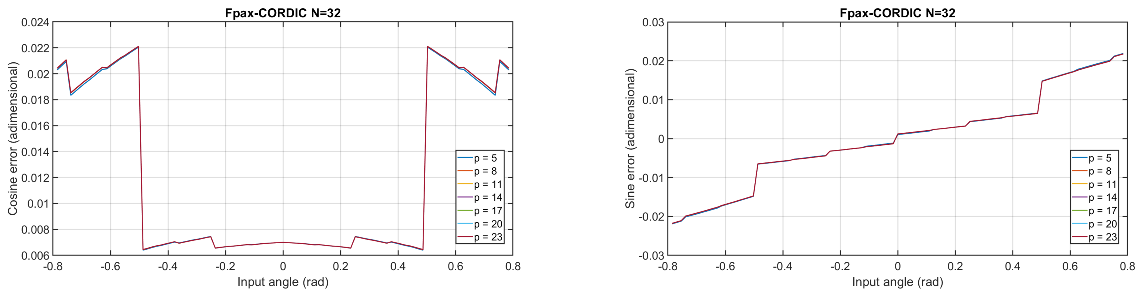


Figure 13: Fpax-CORDIC, $N = 32$.

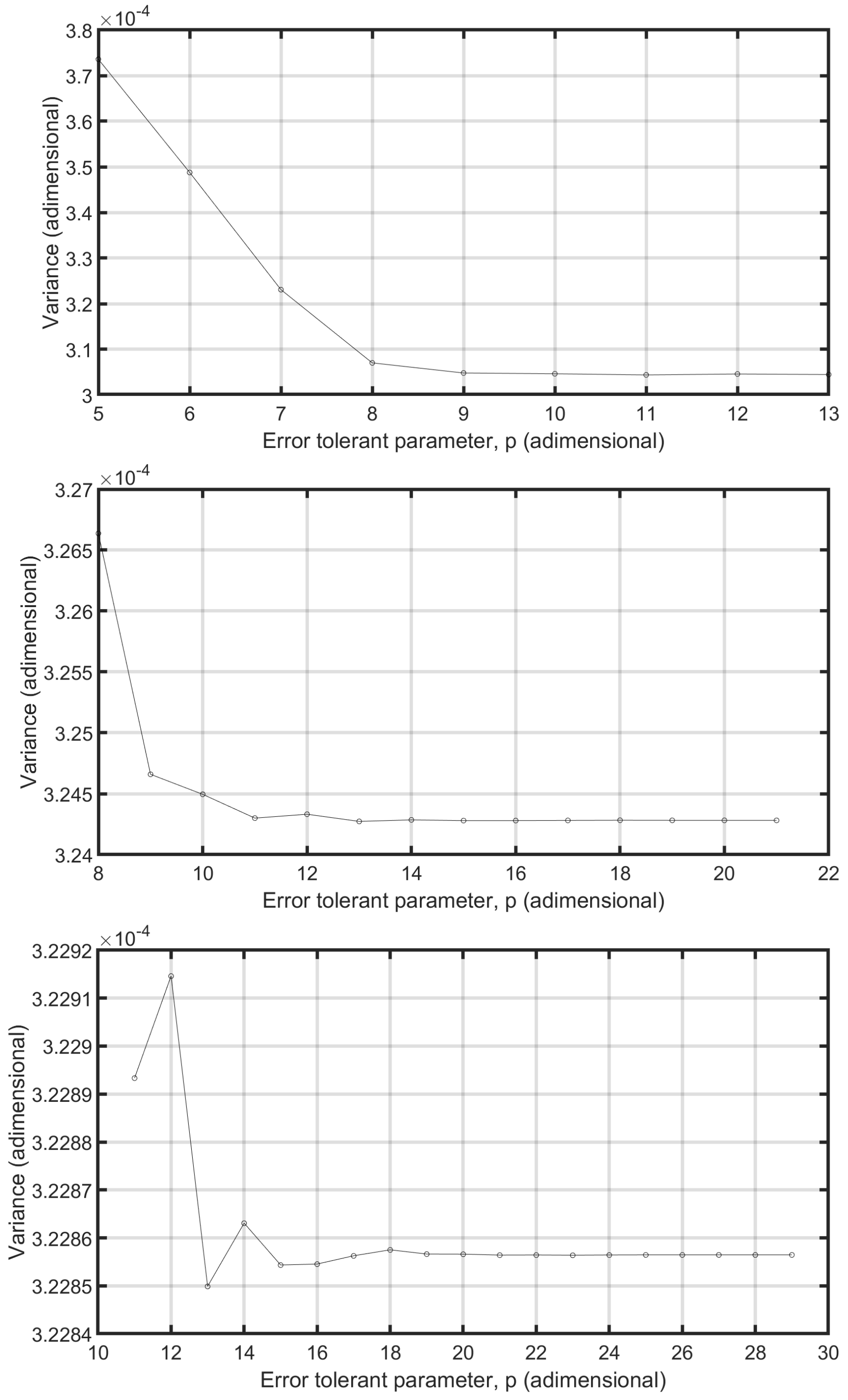


Figure 14: Variance as defined in (13) for the FPAX-CORDIC algorithm as a function of the input tolerant parameter.

Table 1: Variance as defined in (13) for the basic CORDIC algorithm.

		Number of iterations						
	N	10	14	15	20	22	25	30
$\sigma^2 \cdot 10^{-5}$ (adim)	16	8.4417	8.3098					
	24	8.4208		8.2658		8.2662		
	32	8.4207		8.2658	8.2663		8.2661	8.2661

Table 2: Variance as defined in (13) for the Para-CORDIC algorithm.

	N		
	16	24	32
$\sigma^2 \cdot 10^{-4}$ (adim)	3.2466	3.2154	3.2258

Table 3: Comparisons for the Para-CORDIC algorithm

	N		
	16	24	32
delay (ns)	32.0	53.7	68.8
dynamic power (W)	0.060	0.125	0.196
Number of adders	35	55	81
Number of MUXes	102	162	240

Table 4: Comparisons for the FPAX-CORDIC algorithm with $N = 16$

	p			
	5	8	11	14
delay (ns)	36.1	33.6	33.5	31.6
dynamic power (W)	0.056	0.052	0.050	0.051
Number of adders	35	39	47	59
Number of MUXes	102	114	138	174

Table 5: Comparisons for the FPAX-CORDIC algorithm with $N = 24$

	p				
	8	11	14	17	20
delay (ns)	51.5	52.9	50.8	52.1	52.1
dynamic power (W)	0.109	.106	0.104	0.102	0.102
Number of adders	55	65	79	87	107
Number of MUXes	162	192	228	258	318

Table 6: Comparisons for the FPAX-CORDIC algorithm with $N = 32$

	p					
	11	14	17	20	23	26
delay (ns)	69.4	65.3	69.2	70.1	73.2	73.2
dynamic power (W)	0.168	0.162	0.172	0.170	0.167	0.167
Number of adders	81	93	103	123	143	167
Number of MUXes	240	276	306	366	426	498

5 Conclusion

In this report three possible CORDIC results have been reviewed and implemented in Hardware Description Language. Then, they have been compared in terms of accuracy, delay, power and area in order to put in evidence the key idea of approximate computing, that is, to introduce errors in order to loose resource requirements. The results obtained by us confirm this aspect only for what concerns the accuracy and the area consumption. The results for delay and area should be taken carefully.

Finally, all the material that has been produced during this project is stored in the GitHub repository available at the following link: *https://github.com/Gianne97/Enhancing_performances*.

References

[1] L. Sekanina, "Introduction to approximate computing: Embedded tutorial," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2016, pp. 1–6.

[2] J. Volder, "The cordic computing technique," in *Papers presented at the the March 3-5, 1959, western joint computer conference*, 1959, pp. 257–261.

[3] R. Andraka, "A survey of cordic algorithms for fpga based computers," in *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, 1998, pp. 191–200.

[4] Tso-Bing Juang, Shen-Fu Hsiao, and Ming-Yu Tsai, "Para-cordic: parallel cordic rotation algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 8, pp. 1515–1524, 2004.

[5] S. Wang, V. Piuri, and E. Wartzlander, "Hybrid cordic algorithms," *IEEE Transactions on Computers*, vol. 46, no. 11, pp. 1202–1207, 1997.

[6] L. Chen, J. Han, W. Liu, and F. Lombardi, "Algorithm and design of a fully parallel approximate coordinate rotation digital computer (cordic)," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 3, no. 3, pp. 139–151, 2017.