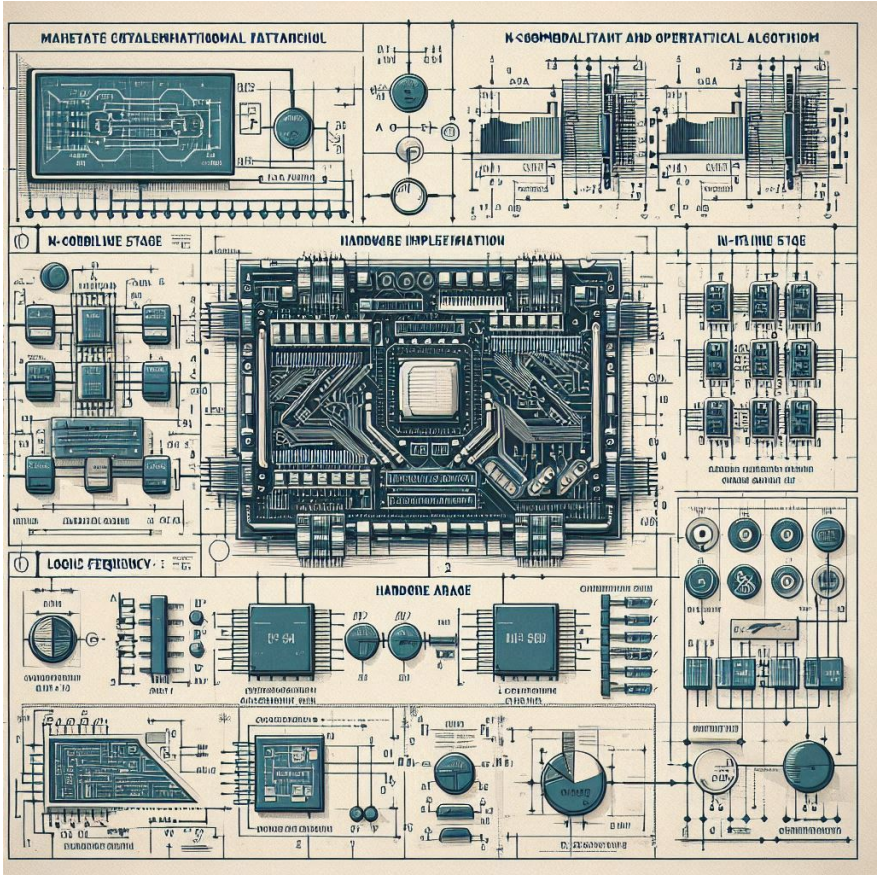
Comparison of 32-bit CORDIC Algorithm Implementations

Mohamed Niazy Hassaneen Ali Mohamed

September 2024

A Project Submitted as Part of Professional Development
Prepared by a Graduate of Electronics and Electrical Communications Engineering
Cairo University



Contact Information
Gmail: mohamedniazy972@gmail.com
LinkedIn: Mohamed Niazy

Contents

1 Introduction and CORDIC Operation **1**
1.1 Types of CORDIC operations 2
1.1.1 Vectoring CORDIC 2
1.1.2 Rotational CORDIC 3

2 Design Methodology **4**
2.1 13 Combinational Stage 4
2.2 13 Pipeline Stage 5
2.3 Single Stage 5

3 Comparison Metrics **6**

4 Results and Discussion **6**

5 Conclusion **7**

Abstract

This document presents a comparison of three hardware implementations of the CORDIC algorithm: an n-combinational stage, an n-pipeline stage, and a single stage. Each design is evaluated based on logic utilization, operating frequency, and hardware area. The CORDIC algorithm, known for efficiently calculating trigonometric and other functions, is introduced in detail in the following sections, where its mathematical formulation and operational principles are also outlined.

1 Introduction and CORDIC Operation

The CORDIC (COordinate Rotation DIgital Computer) algorithm is a hardware-efficient method used to compute trigonometric, hyperbolic, and other mathematical functions without using multipliers, dividers, or factorial operations. It takes the rotation matrix and manipulates its inputs to achieve a specific function:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} x_0\cos\theta + y_0\sin\theta \\ -x_0\sin\theta + y_0\cos\theta \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

We have the following system of rotation equations:

$$x = \cos\theta \cdot (x_0 + y_0 \cdot \tan\theta)$$

$$y = \cos\theta \cdot (y_0 - x_0 \cdot \tan\theta)$$

First, ignoring the cosine factor, instead of performing the rotation directly, we decompose it into several micro-rotations, as shown in Figure 1.

We use specific rotation values where $\tan(\theta_i) = 2^{-i}$, which replaces the multiplier with a shifter. This results in the following equations:

$$x_{i+1} = x_i + a_i \cdot y_i \cdot 2^{-i}$$
$$y_{i+1} = y_i - a_i \cdot x_i \cdot 2^{-i}$$
$$\theta_{i+1} = \theta_i + a_i \cdot \text{atan}(2^{-i})$$

Here, a_i denotes the direction of the micro-rotations. All we need are adders, subtractors, shifters, and a small lookup table to store the values of $\tan^{-1}(2^{-i})$. The number of micro-rotations required depends on the desired accuracy.

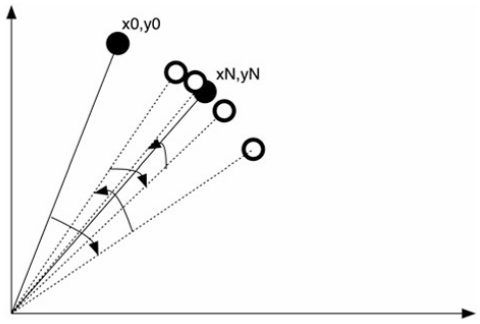


Figure 1: Dividing the rotation into micro-rotations

1.1 Types of CORDIC operations

1.1.1 Vectoring CORDIC

This method starts with a point in the plane (x_0, y_0) and ends with a point on the x-axis $(\sqrt{x_0^2 + y_0^2}, 0)$. During this process, the phase of the original point with the x-axis is computed as $\theta_N = \text{atan}\left(\frac{y_0}{x_0}\right)$.

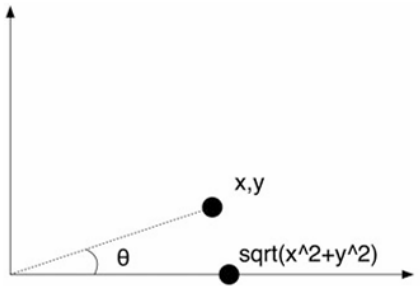


Figure 2: Vectoring CORDIC

For example, from [1], we expect the final value of y to be zero or close to zero because the vector lies on the x-axis, and θ will equal $\tan^{-1}\left(\frac{y_0}{x_0}\right)$.

The following table illustrates the evolution of the x, y, and θ registers over 13 iterations. This process iteratively rotates the vector, adjusting the values and refining the final result:

From the table, we note that:

1. 13 stages provide good accuracy.
2. The final value of x_n is scaled by a factor of 1.64676 times $\sqrt{x_0^2 + y_0^2}$ (this was expected since the cosine was ignored in the equations).

We can use vectoring CORDIC to compute the inverse tangent value from θ_n for any point in the first quadrant (x_c, y_c). The input values required are:

1. $x_i = x_c$
2. $y_i = y_c$
3. $\theta_i = 0$

The results will be:

1. $x_n = 1.64676 \cdot \sqrt{x_0^2 + y_0^2}$

Table 1: Evolution of the x, y, and θ registers with each step of the CORDIC algorithm

Step	x_i	y_i	a_i	θ	$\sqrt{x_i^2+y_i^2}$	$\frac{\sqrt{x_i^2+y_i^2}}{\sqrt{x_0^2+y_0^2}}$
Initialization	7	3	0	0	7.61577	1
0	10	-4	1	0.78540	10.77033	1.41421
1	12	1	-1	0.32175	12.04159	1.58114
2	125	-2	1	0.56673	12.41219	1.62980
3	12.5	-0.46875	-1	0.44237	12.50879	1.64248
4	12.52930	0.3125	-1	0.37996	12.53319	1.64569
5	12.53906	-0.07904	1	0.41120	12.53931	1.64649
6	12.54030	0.11688	-1	0.39557	12.54084	1.64669
7	12.54121	0.01891	1	0.40338	12.54122	1.64674
8	12.54128	-0.03008	1	0.40729	12.54132	1.64676
9	12.54134	-0.00558	-1	0.40534	12.54134	1.64676
10	12.54135	0.00666	-1	0.40436	12.54135	1.64676
11	12.54135	0.00054	1	0.40485	12.54135	1.64676
12	12.54135	-0.00252	1	0.40509	12.54135	1.64676
13	12.54135	-0.00099	-1	0.40497	12.54135	1.64676

2. $y_n = 0$
3. $\theta_n = \arctan\left(\frac{y_0}{x_0}\right)$

1.1.2 Rotational CORDIC

Rotational CORDIC: A rotation operation takes a point (x_0, y_0) and rotates it by a given angle θ_0 to a new coordinate point (x_n, y_n) where $\theta_n = 0$.

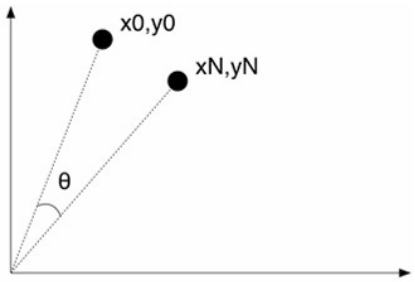


Figure 3: Rotational CORDIC

We can use rotational CORDIC to get the cosine and sine values from x_n and y_n for any angle θ_c in the first quadrant. The input values required are:

1. $x_0 = \frac{1}{1.64676}$
2. $y_0 = 0$
3. $\theta_i = \theta_c$

The results will be:

1. $x_n = \cos(\theta_0)$
2. $y_n = \sin(\theta_0)$

3. $\theta_n = 0$

But how did we get those results? From the previous input, we have:

$$\text{root} = \sqrt{x_0^2 + y_0^2} = \frac{1}{1.64676}$$

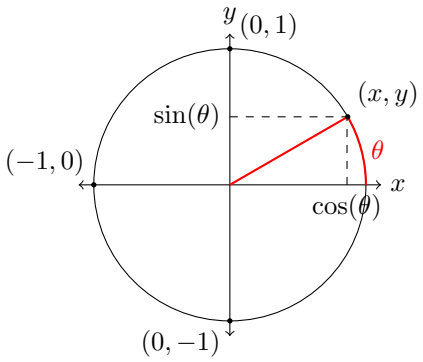
After applying the CORDIC algorithm, we get:

$$\text{root} = \sqrt{x_n^2 + y_n^2} = \sqrt{x_0^2 + y_0^2} \times 1.64676 = \frac{1}{1.64676} \times 1.64676 = 1$$

Do you remember the unit circle? It is a circle with certain properties:

1. Its radius is equal to one.
2. The x-coordinate represents the cosine value of its angle.
3. The y-coordinate represents the sine value of its angle.

Thus, we force the input to be on the unit circle in the end.



2 Design Methodology

The design consists of three parts. The first part handles the input based on the desired function (cosine, sine, or inverse tangent). The second part is the CORDIC stage, which represents the implementation of the rotation equation. The third part handles the sign of the output according to the input quadrant. Therefore, you will find combinational logic at both the beginning and the end of the design, corresponding to the first and third parts.

2.1 13 Combinational Stage

This version consists of multiple combinational stages operating in parallel. It requires fewer clock cycles but may consume more resources due to the parallel structure.

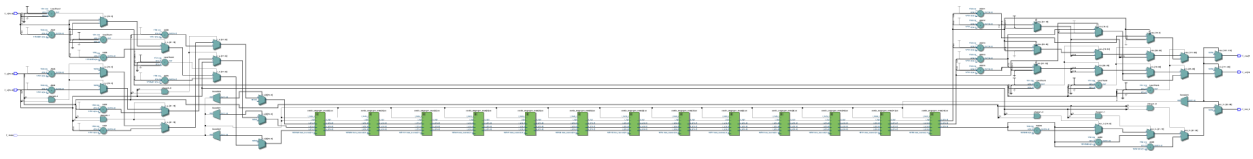


Figure 4: RTL view of the CORDIC algorithm (13-stage)

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name
1	16.37 MHz	1.27 MHz	clk_virtual

(a) Maximum frequency for the implementation (13-combinational-stage)

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	933
2		
3	Combinational ALUT usage for logic	1830
1	-- 7 input functions	25
2	-- 6 input functions	11
3	-- 5 input functions	863
4	-- 4 input functions	612
5	-- <=3 input functions	319
4		
5	Dedicated logic registers	0
6		
7	I/O pins	193
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	if_.mode~input
12	Maximum fan-out	1432
13	Total fan-out	9313
14	Average fan-out	4.20

(b) RTL resources for the CORDIC algorithm mapped to Cyclone V (13-combinational-stage)

Figure 5: Compilation results in Quartus for the 13-combinational-stage design

2.2 13 Pipeline Stage

In this design, each stage is pipelined to improve the clock speed at the cost of additional logic and area for registers between stages.

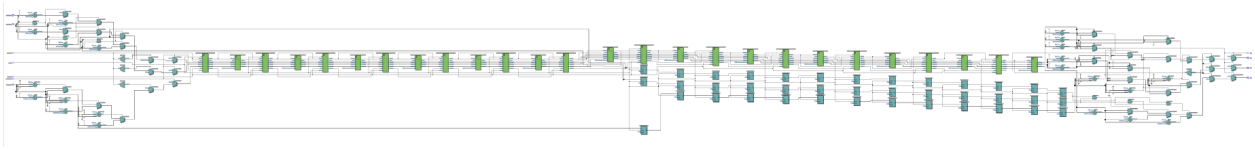


Figure 6: RTL view of the CORDIC algorithm (13 pipelined stages)

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name
1	46.7 MHz	46.7 MHz	clk_virtual

(a) Maximum frequency for the implementation (13-pipelined-stage)

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	978
2		
3	Combinational ALUT usage for logic	1818
1	-- 7 input functions	25
2	-- 6 input functions	50
3	-- 5 input functions	67
4	-- 4 input functions	69
5	-- <=3 input functions	1607
4		
5	Dedicated logic registers	1322
6		
7	I/O pins	197
8	Total MLAB memory bits	0
9	Total block memory bits	336
10		
11	Total DSP Blocks	0
12		
13	Maximum fan-out node	if_.clk~input
14	Maximum fan-out	1350
15	Total fan-out	11054
16	Average fan-out	3.10

(b) RTL resources for the CORDIC algorithm mapped to Cyclone V (13-pipelined-stage)

Figure 7: Compilation results in Quartus for the 13-pipelined-stage design

2.3 Single Stage

The single-stage design performs all iterations sequentially, reducing logic and area but increasing the number of clock cycles required to compute the result. It was expected to achieve the highest frequency among the

three implementations. However, in my design, the second part and third part are connected directly, resulting in a large combinational logic block (see Section 2). In future work, I plan to break this logic by introducing a register to reduce the critical path.

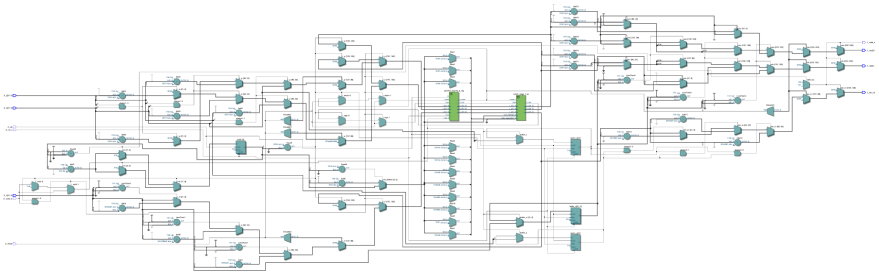


Figure 8: RTL view of the CORDIC algorithm (single-stage)

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name
1	40.57 MHz	40.57 MHz	clk_virtual

(a) Maximum frequency for the implementation (single-stage)

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	489
2		
3	Combinational ALUT usage for logic	799
1	-- 7 input functions	1
2	-- 6 input functions	173
3	-- 5 input functions	131
4	-- 4 input functions	123
5	-- <=3 input functions	371
4		
5	Dedicated logic registers	130
6		
7	I/O pins	197
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	if_clk~input
12	Maximum fan-out	130
13	Total fan-out	4033
14	Average fan-out	3.05

(b) RTL resources for the CORDIC algorithm mapped to Cyclone V (single-stage)

Figure 9: Compilation results in Quartus for the single-stage design

3 Comparison Metrics

The comparison is based on the following metrics:

- **Logic Utilization:** The amount of hardware resources used, such as ALUTs .
- **Operating Frequency:** The maximum clock frequency at which each design can operate.
- **Fan-Out:** The number of logic elements driven by a single output signal, which can impact timing and performance.

4 Results and Discussion

The n-combinational stage provides the fastest computation time but consumes the most logic and area, reducing the overall system frequency. The n-pipeline stage balances logic and speed by distributing computation across multiple clock cycles, while compensating for this with higher throughput. The single-stage design minimizes resource usage at the cost of the highest latency.

Design	Logic (ALUTs)	DSP block	Frequency (MHz)	Fan-out
13-Combinational Stage	1830	0	16.37	9313
13-Pipeline Stage	1818	0	46.7	11054
Single Stage	799	0	40.57	4033

Table 2: Comparison of CORDIC Implementations

5 Conclusion

Each design presents trade-offs between speed, resource usage, and area. The optimal choice of implementation depends on the specific application and hardware constraints.

References

References

[1] Karim Abbas. *From Algorithms to Hardware Architectures Using Digital Radios as a Design Example*. Springer, 2024.