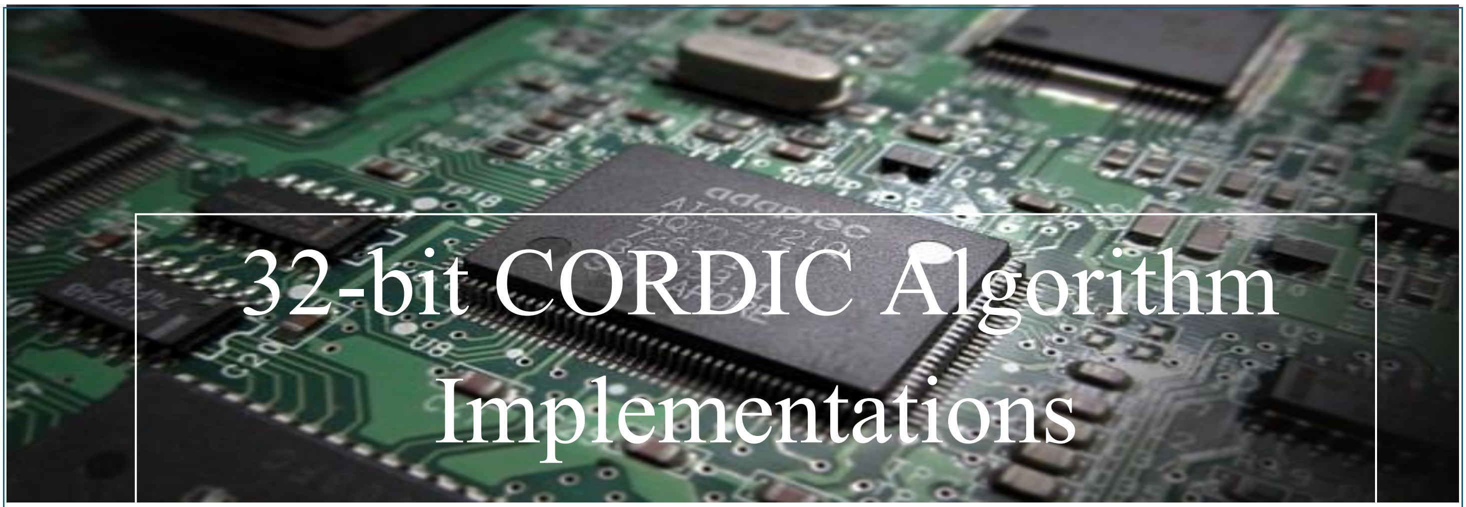


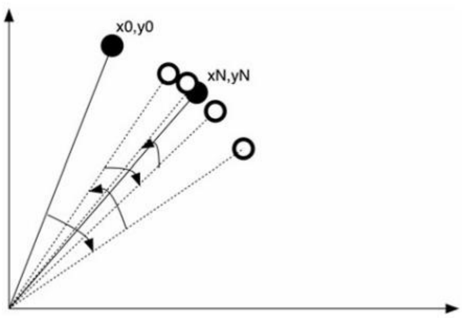
32-bit CORDIC Algorithm Implementations

1 Basic of the CORDIC Algorithm

CORDIC stands for Coordinate Rotation DIgital Computer. CORDIC is a processing unit that allows us to perform many transcendental functions efficiently. When applied to Givens rotations, it allows us to perform both vectoring and rotation operations using shifters and adders only.

CORDIC is used to compute:

- **Trigonometric functions:** sin, cos, tan, atan.
- **Vector operations:** magnitude, phase.
- **Hyperbolic functions:** sinh, cosh, tanh.
- **Other functions:** square roots, exponentials, logarithms (in extended forms).



To begin with, if we want to move from an initial point (X_0, Y_0) to a final point (X_n, Y_n), as illustrated in the figure below, we must follow a set of coordinate rotation equations that define this transformation:

$$(x_0, y_0) \rightarrow (x_N, y_N)$$
$$x_N = x_0 \cos\theta + y_0 \sin\theta = \cos\theta(x_0 + y_0 . \tan\theta)$$
$$y_N = y_0 \cos\theta - x_0 \sin\theta = \cos\theta(y_0 - x_0 . \tan\theta)$$

These equations can also be used to represent the microrotations because there is nothing about them that limits them to a certain rotation angle. When we use these equations to represent a single microrotation, in other words a single iteration, we use the indices i and i + 1 to indicate rotation by angle θi:

$$x_{i+1} = \cos\theta_i(x_i + y_i . \tan\theta_i)$$
$$y_{i+1} = \cos\theta_i(y_i - x_i . \tan\theta_i)$$

Instead of performing one large rotation, the CORDIC algorithm uses multiple smaller microrotations. By carefully choosing the rotation angles (θ_i) such that $\tan(\theta_i) = 2^{-i}$, each rotation step can be implemented using only bit shifts and additions—eliminating the need for complex multiplications. This simplification allows CORDIC to perform trigonometric operations efficiently using simple hardware components like adders, shifters, and a small lookup table.

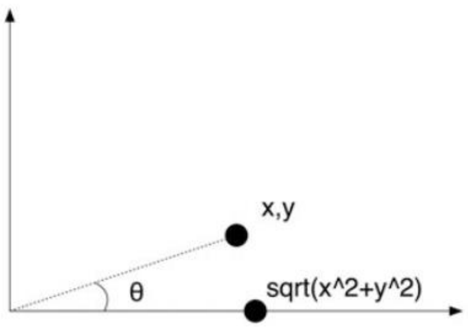
the values of x, y, and the phase are updated according to:

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

2 Types of CORDIC

2.1 Vectoring CORDIC

In the vectoring mode of the CORDIC algorithm, the operation begins with an initial point (x_0, y_0) on the plane and iteratively rotates it until the final point lies on the x-axis, i.e., (x_n, 0). During this process, the algorithm also computes the angle θ_n, which represents the phase between the original vector and the x-axis, given by .T$\theta_N = \text{atan}\left(\frac{y_0}{x_0}\right)$ primarily used to determine the magnitude and phase of a vector.



We start with three registers carrying the initial x and y values, as well as a null phase register. On every iteration i following this, the values of x, y, and the phase are updated according to:

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

The factor ai simply indicates the direction in which the current microrotation takes place So ai is either +1 or -1 causing the microrotation to be clockwise or counterclockwise. The factor ai should simply be the sign of the last value of y we calculated:

$$a_i = \text{sign}(y_i)$$

At the end of the iterations, we should end up with $x_N = \sqrt{x_0^2 + y_0^2}$ and $\theta_N = \text{atan}\left(\frac{y_0}{x_0}\right)$

For example if x=7 and y=3. The Evolution of the x, y, and θ registers with each step of the CORDIC algorithm shown in next table.

Step	X	Y	a	Theta	Magnitude	Ratio
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	12.25	−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

From the table, we note that:

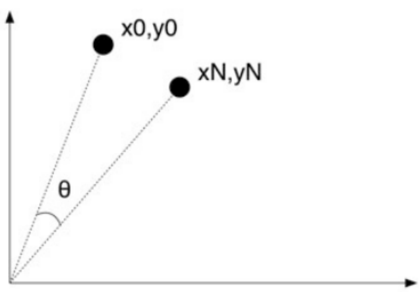
1. 13 stages provide good accuracy.
2. The final value of xn is scaled by a factor of 1.64676 times (this was expected since the cosine was ignored in the equations)

2.2 Rotation CORDIC

In the rotation mode, the CORDIC algorithm starts with an initial point (x_0, y_0) and rotates it by a specified angle θ_0 to obtain a new coordinate point (x_n, y_n). Unlike the vectoring mode, which calculates the angle as an output, the rotation mode uses the angle as an input. The resulting point after rotation does not generally lie on the x-axis, meaning the final y-value (y_n) is typically non-zero. This mode is mainly used to compute sine, cosine, and general vector rotation values.

In fact, both rotation and vectoring CORDIC operations are so fundamentally similar that they can be implemented using the same hardware architecture. The only difference between the two lies in how the direction of rotation — represented by the control signal a.

The factor ai in rotation cordic calculated by : $a_i = -\text{sign}(\theta_i)$



3 Limitation and Range Extension of the CORDIC Algorithm

One key limitation of the basic CORDIC algorithm—whether in rotation or vectoring mode—is that it operates correctly only within the right half-plane, corresponding to phase angles between –90° and +90°.

This restriction arises from the limited set of microrotations, where the largest available rotation angle is π/4 (45°). As a result, without modification, CORDIC cannot directly handle vectors or rotations outside this range. We can extend the range of the rotation CORDIC by doing the following simple initial modification:

$$
\begin{gathered}
\text{If } x_0 < 0 \\
d = -\text{sign}(y_0) \\
x_0 = -d.y_0 \\
y_0 = d.x_0 \\
\theta_0 = \theta_0 + d.\frac{\pi}{2}
\end{gathered}
$$

And for vectoring CORDIC:

$$
\begin{gathered}
\text{If } x_0 < 0 \\
d = \text{sign}(y_0) \\
x_0 = -x_0 \\
y_0 = y_0 \\
\theta_N = d.(\pi - d.\theta_N)
\end{gathered}
$$

This modification allows the CORDIC algorithm to correctly process inputs across the entire four quadrants of the coordinate plane.

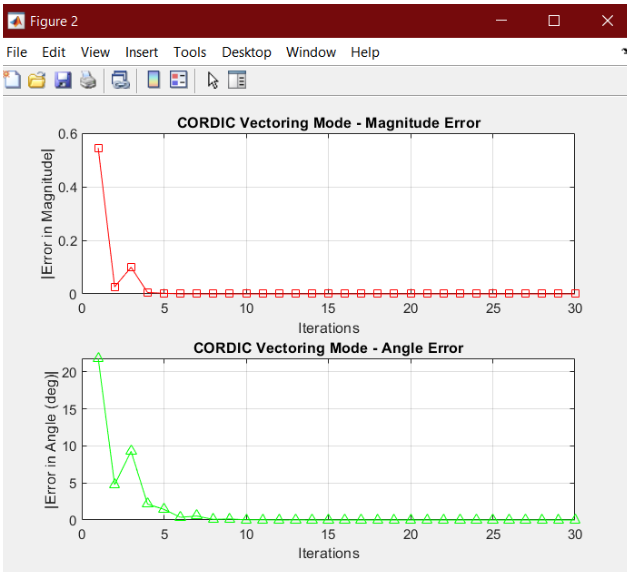
4 CORDIC implementation

4.1 Step 1: Determine the Optimal Number of Iterations

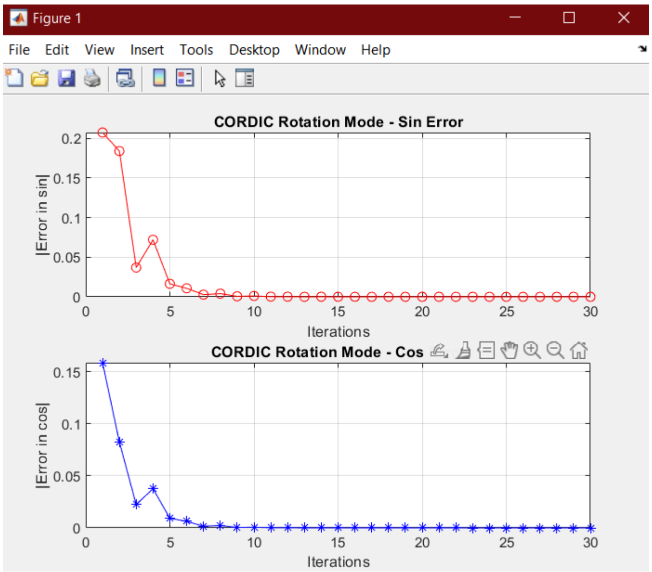
The more the iterations, the less the residue left. The number of iterations needed is a matter of trial and error but recall that all numbers in registers are fixed point. Thus, the number of iterations needed is only the number necessary to reduce the residue left in the registers below their resolutions.

After performing simulation tests in MATLAB using up to 30 iterations, the error was evaluated at each step. The iteration counts corresponding to an error less than the predefined tolerance= 1e-4 was selected as the optimal value.

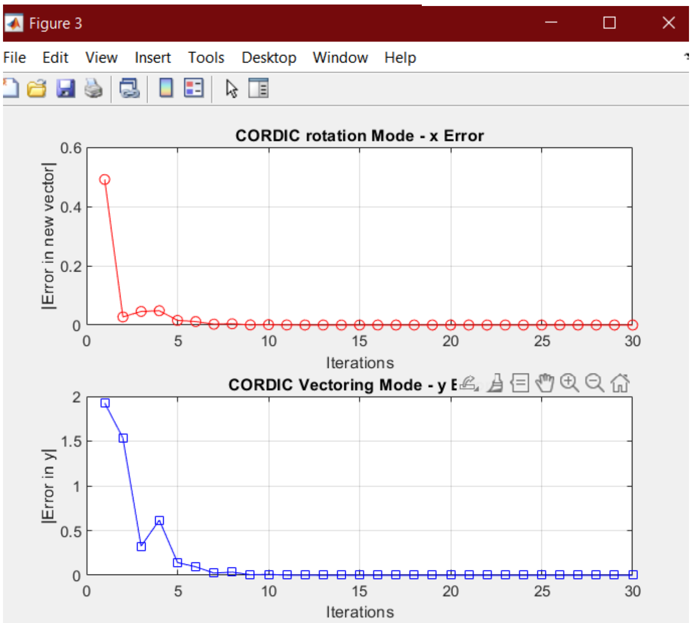
From the results, the best number of iterations was found to be 13, as illustrated in the following figure.



Determining the optimal number of iterations for magnitude and angle (atan) convergence



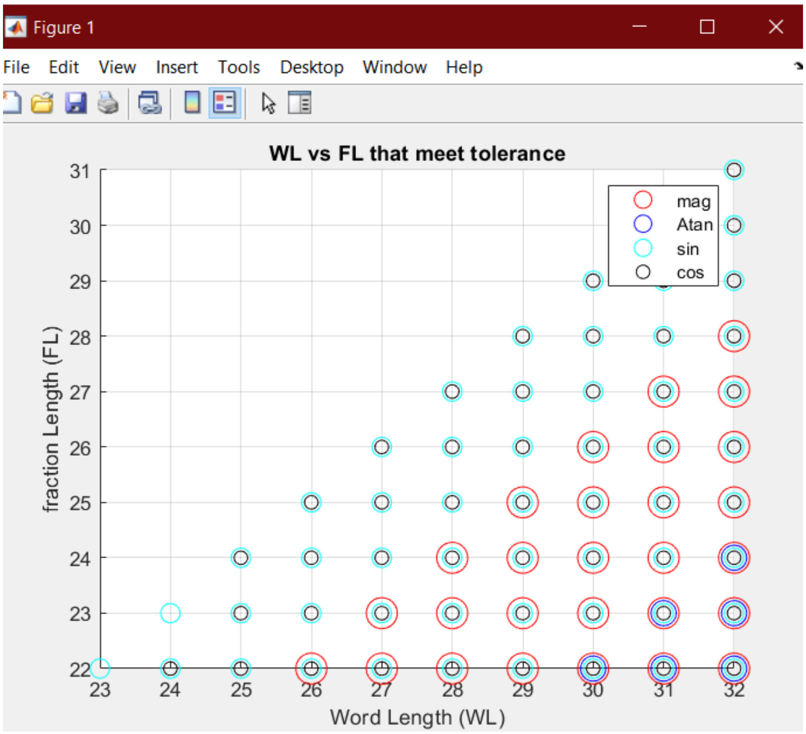
Determining the optimal number of iterations for sin and cose convergence



Determining the optimal number of iterations for new x and new y convergence

4.2 Step 2: Determining the Optimal Word Length (WL) and Fraction Length (FL)

The next step is to identify suitable fixed-point parameters — specifically the Word Length (WL) and Fraction Length (FL) — that ensure numerical precision while minimizing hardware cost.
To achieve this, several test vectors and rotation angles were evaluated in both vectoring (for magnitude and angle) and rotation (for sine and cosine) CORDIC modes. The algorithm iterates through a range of WL and FL combinations, quantizing each result and comparing it with the true floating-point values.
The acceptable configuration is selected when the maximum quantization error is less than the specified tolerance (1×10^{-7}) for all tested cases. Based on the MATLAB simulation results, the optimal configuration was found according to next figure be: Word Length=32, Fraction Length=22. Figure shows the relationship between the word length and fraction length values that satisfy the error tolerance criterion across all tested operations.



Determination of optimal Word Length (WL) and Fraction Length (FL) satisfying error tolerance (1×10^{-7}).

5 Hardware Implementation

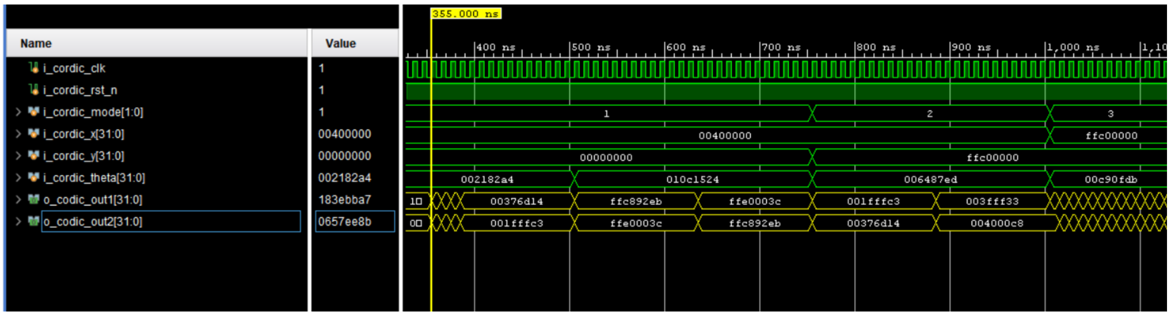
To achieve high performance and throughput, a pipelined architecture was adopted. The pipeline consists of 13 sequential stages, each corresponding to one iteration of the CORDIC algorithm.

- **Port Specification**

Port Name	Direction	Width	Type	Description
`i_cordic_clk`	Input	1	Clock (333.3MHZ)	Main system clock driving all sequential logic in the CORDIC pipeline.
`i_cordic_rst_n`	Input	1	Active-low Reset	Asynchronous reset signal to initialize all internal registers.
`i_cordic_mode`	Input	2	Control	Operation mode selector: • `00` → Vectoring mode (magnitude and phase) • `01` → Rotation mode (sin and cos) • `10` → Rotation mode (counterclockwise) • `11` → Rotation mode (clockwise)
`i_cordic_x`	Input	32	Signed Fixed-point	X-component of the input vector.
`i_cordic_y`	Input	32	Signed Fixed-point	Y-component of the input vector.
`i_cordic_theta`	Input	32	Signed Fixed-point	Input rotation angle (used only in rotation modes).
`o_cordic_out1`	Output	32	Signed Fixed-point	Output In vectoring mode → Magnitude Output In rotation mode → cos and X' (rotated X coordinate)
`o_cordic_out2`	Output	32	Signed Fixed-point	Output In vectoring mode → Angle (θ) Output In rotation mode → sine and Y' (rotated Y coordinate)

6 CORDIC Verification

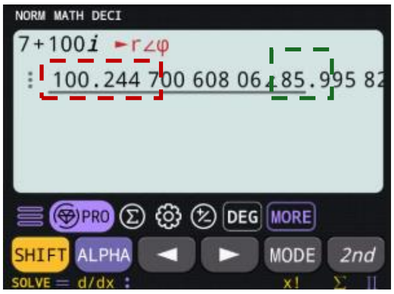
6.1 Waveform Snippets:



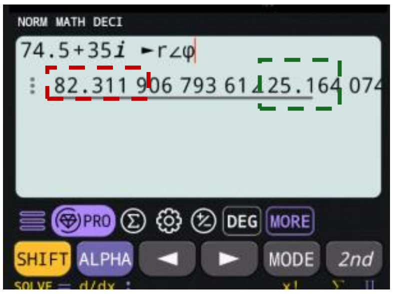
6.2 Transcript and self-checking with MATLAB

The testbench compares the results of the RTL implementation against the MATLAB reference model for different operating modes.
In **Mode 0**, both the **magnitude** and **arctangent (Atan)** values showed nearly identical results to the expected MATLAB values, with an absolute error below **0.0001%**, confirming excellent accuracy in vector magnitude and phase computations.

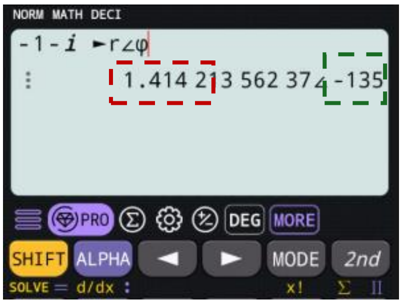
```
=================================================================
                        MODE    0
=================================================================
magntuide RTL       =  00011001000011111010100011011111
magnitude RTL       =  100.2447
magnitude expect    =  100.2447
magnitude ERR       =  0.000%
Atan RTL            =  00000000011000000001000100001110
Atan RTL            =  86.0033
Atan expect         =  86.0033
Atan ERR            =  0.000%
```




```
=================================================================
                        MODE    0
=================================================================
magntuide RTL       =  00010100100100111111011000001001
magnitude RTL       =  82.3119
magnitude expect    =  82.3119
magnitude ERR       =  0.000%
Atan RTL            =  00000000000111000001110100000000
Atan RTL            =  25.1683
Atan expect         =  25.1683
Atan ERR            =  0.000%
```

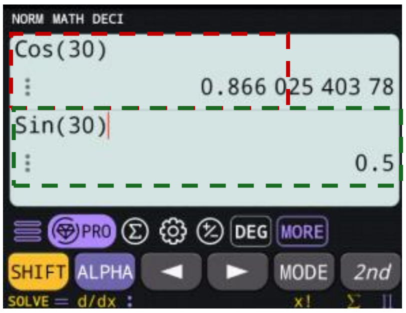



```
=================================================================
                        MODE    0
=================================================================
magntuide RTL       =  00000000010110101000001001110111
magnitude RTL       =  1.4142
magnitude expect    =  1.4142
magnitude ERR       =  0.000%
Atan RTL            =  11111111011010010011010101011011
Atan RTL            =  -134.9956
Atan expect         =  -134.9956
Atan ERR            =  0.000%
```

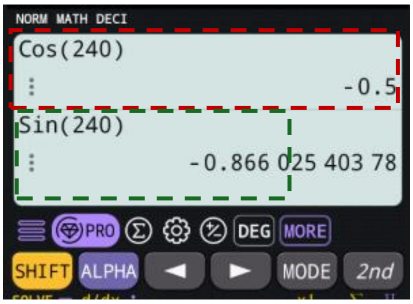


In **Mode 1**, which computes **sine** and **cosine** values for a given rotation angle, the RTL outputs exhibited outstanding agreement with MATLAB results across multiple test angles. The maximum observed deviation for sine and cosine was less than **0.001%**, indicating high precision in trigonometric function generation.

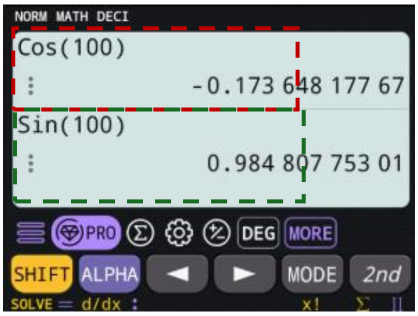
```
======================================================================
                          MODE    1
======================================================================
Cos RTL         =  00000000001101110110110100010100
Cose RTL        =  0.8660
Cose Expect     =  0.8660
Cose Err        =  0.000%
Sine RTL        =  00000000000111111111111111000011
Sine RTL        =  0.5000
Sine Expect     =  0.5000
Sine Err        =  0.000%
```




```
======================================================================
                          MODE    1
======================================================================
Cos RTL         =  11111111111000000000000000111100
Cose RTL        =  -0.5000
Cose Expect     =  -0.5000
Cose Err        =  0.000%
Sine RTL        =  11111111110010001001001011101011
Sine RTL        =  -0.8660
Sine Expect     =  -0.8660
Sine Err        =  0.000%
```



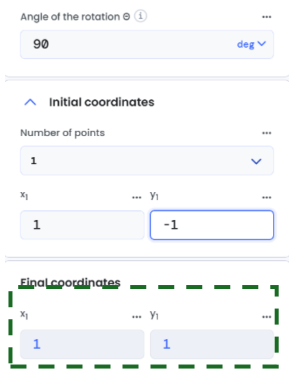

```
======================================================================
                          MODE    1
======================================================================
Cos RTL         =  11111111111101001110001000011100
Cose RTL        =  -0.1737
Cose Expect     =  -0.1737
Cose Err        =  0.000%
Sine RTL        =  00000000001111110000011011101101
Sine RTL        =  0.9848
Sine Expect     =  0.9848
Sine Err        =  0.000%
```



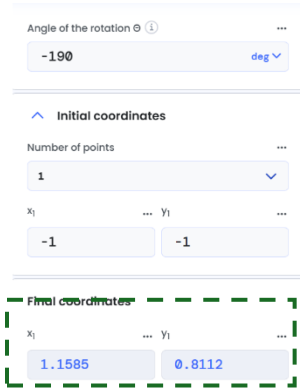

```
======================================================================
                          MODE    1
======================================================================
Cos RTL         =  00000000000001011110101010011010
Cose RTL        =  0.0924
Cose Expect     =  0.0924
Cose Err        =  0.001%
Sine RTL        =  11111111110000000100011000101011
Sine RTL        =  -0.9957
Sine Expect     =  -0.9957
Sine Err        =  0.000%
```

For **Modes 2 and 3**, which handle vector rotations in counterclockwise and clockwise directions respectively, the computed **X** and **Y** coordinates closely matched their expected values, with errors not exceeding **0.0003%**.

```
=================================================================
                      MODE   2
=================================================================
X RTL        =  00000000001111111111111100110011
X RTL        =  1.0000
X Expect     =  1.0000
X Err        =  0.000%
Y RTL        =  00000000010000000000000011001000
Y RTL        =  1.0000
Y Expext     =  1.0000
Y Err        =  0.000%
```

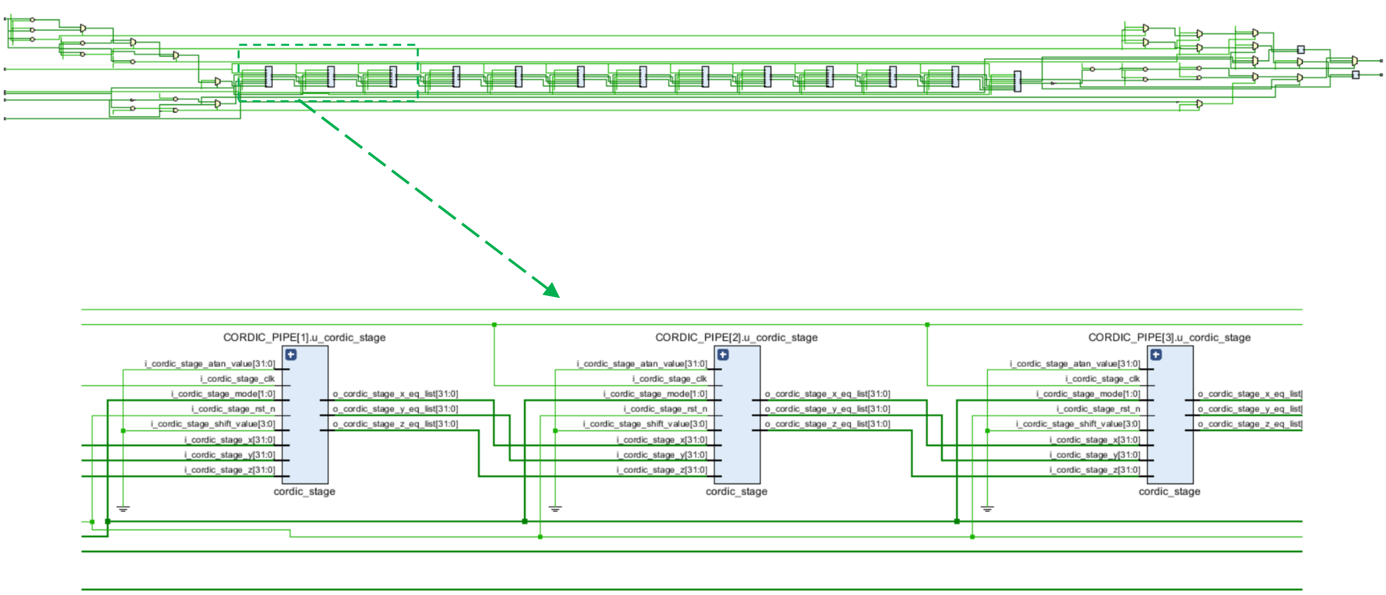



```
=================================================================
                      MODE   3
=================================================================
X RTL        =  00000000010010100010010011011000
X RTL        =  1.1585
X Expect     =  1.1585
X Err        =  0.000%
Y RTL        =  00000000001100111110100100001010
Y RTL        =  0.8111
Y Expext     =  0.8111
Y Err        =  0.000%
```



Overall, the CORDIC hardware design demonstrated excellent numerical stability and consistency across all modes of operation. The negligible error margins confirm that the chosen 13-stage pipelined architecture and the 32-bit word length with 22-bit fractional precision are well-optimized, achieving MATLAB-equivalent results in hardware implementation.

Elaborated Design



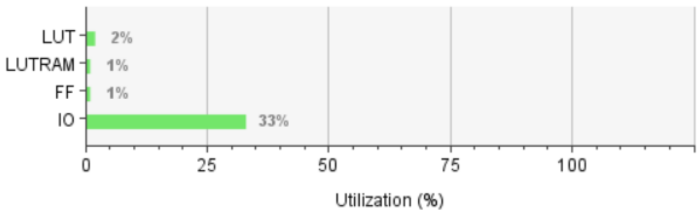
Constraints

```
#### This file is a general .xdc for the Genesys 2 Rev. H
#### To use it in a project:

##----------- device : xc7k325tffg900-2


## Clock Signal
set_property -dict { PACKAGE_PIN AD12  IOSTANDARD LVDS     } [get_ports { i_cordic_clk }]; #IO_L12N_T1_MRCC_33 Sch=sysclk_n
#set_property -dict { PACKAGE_PIN AD11  IOSTANDARD LVDS     } [get_ports { sysclk_p }]; #IO_L12P_T1_MRCC_33 Sch=sysclk_p


#-----------------------------constrains-------------------------------------
create_clock -add -name sys_clk -period 4 -waveform {0 2} [get_ports i_cordic_clk]


## Buttons
set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS12 } [get_ports { i_cordic_rst_n }]; #IO_25_17 Sch=btnc
#set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS12 } [get_ports { btnd }]; #IO_0_15 Sch=btnd
#set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS12 } [get_ports { btnl }]; #IO_L6P_T0_15 Sch=btnl
#set_property -dict { PACKAGE_PIN C19   IOSTANDARD LVCMOS12 } [get_ports { btnr }]; #IO_L24P_T3_17 Sch=btnr
#set_property -dict { PACKAGE_PIN B19   IOSTANDARD LVCMOS12 } [get_ports { btnu }]; #IO_L24N_T3_17 Sch=btnu
#set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports { cpu_resetn }]; #IO_0_14 Sch=cpu_resetn

## LEDs
set_property -dict { PACKAGE_PIN T28   IOSTANDARD LVCMOS33 } [get_ports { o_codic_out1[31] }]; #IO_L11N_T1_SRCC_14 Sch=led[0]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports { o_codic_out1[0] }]; #IO_L19P_T3_A10_D26_14 Sch=led[1]
set_property -dict { PACKAGE_PIN U30   IOSTANDARD LVCMOS33 } [get_ports { o_codic_out1[1] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[2]
set_property -dict { PACKAGE_PIN U29   IOSTANDARD LVCMOS33 } [get_ports { o_codic_out1[2] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=led[3]
set_property -dict { PACKAGE_PIN V20   IOSTANDARD LVCMOS33 } [get_ports { o_codic_out2[31] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V26   IOSTANDARD LVCMOS33 } [get_ports { o_codic_out2[0] }]; #IO_L16P_T2_CSI_B_14 Sch=led[5]
set_property -dict { PACKAGE_PIN W24   IOSTANDARD LVCMOS33 } [get_ports { o_codic_out2[1] }]; #IO_L20N_T3_A07_D23_14 Sch=led[6]
set_property -dict { PACKAGE_PIN W23   IOSTANDARD LVCMOS33 } [get_ports { o_codic_out2[2] }]; #IO_L20P_T3_A08_D24_14 Sch=led[7]

## Switches
set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS12 } [get_ports { i_cordic_mode[0] }]; #IO_0_17 Sch=sw[0]
set_property -dict { PACKAGE_PIN G25   IOSTANDARD LVCMOS12 } [get_ports { i_cordic_mode[1] }]; #IO_25_16 Sch=sw[1]
set_property -dict { PACKAGE_PIN H24   IOSTANDARD LVCMOS12 } [get_ports { i_cordic_x[22] }]; #IO_L19P_T3_16 Sch=sw[2]
set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS12 } [get_ports { i_cordic_x[21] }]; #IO_L6P_T0_17 Sch=sw[3]
set_property -dict { PACKAGE_PIN N19   IOSTANDARD LVCMOS12 } [get_ports { i_cordic_y[22] }]; #IO_L19P_T3_A22_15 Sch=sw[4]
set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS12 } [get_ports { i_cordic_y[21] }]; #IO_25_15 Sch=sw[5]
```

7 Synthesis

7.1 Utilization report after Synthesis

Resource	Utilization	Available	Utilization %
LUT	3738	203800	1.83
LUTRAM	31	64000	0.05
FF	919	407600	0.23
IO	164	500	32.80



7.2 Timing Summary after Synthesis on 250 MHZ

Setup

Worst Negative Slack (WNS):	1.085 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	853

Hold

Worst Hold Slack (WHS):	0.066 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	853

Pulse Width

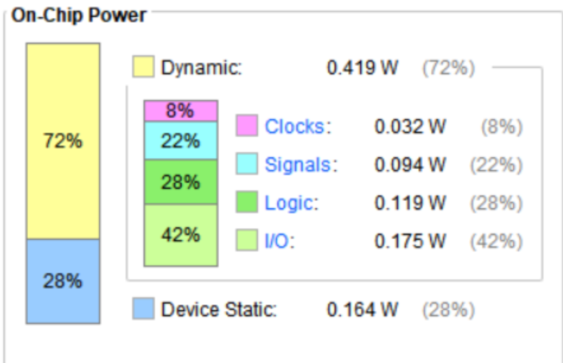
Worst Pulse Width Slack (WPWS):	1.358 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	951

All user specified timing constraints are met.

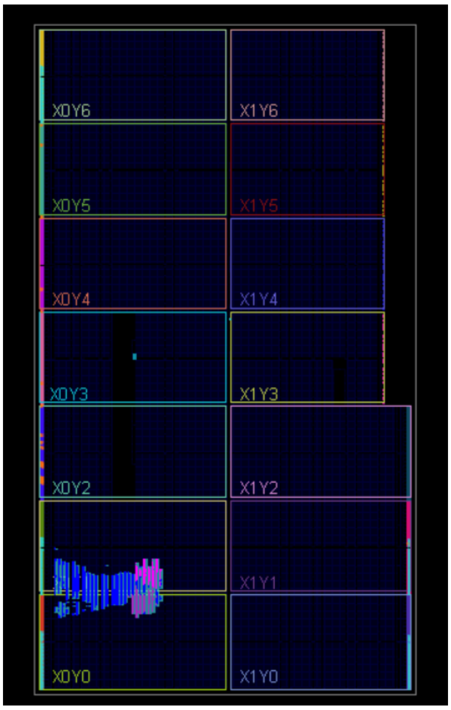
7.3 Power report after Synthesis

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.583 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.0°C**
Thermal Margin:	59.0°C (32.4 W)
Effective ϑJA:	1.8°C/W

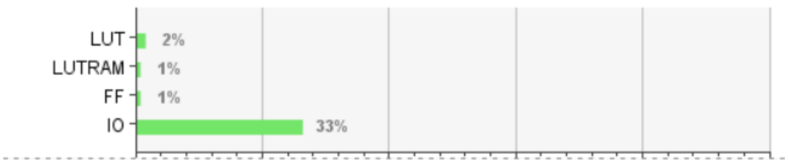


8 Implementation



8.1 Utilization report after Implementation

Resource	Utilization	Available	Utilization %
LUT	3731	203800	1.83
LUTRAM	31	64000	0.05
FF	919	407600	0.23
IO	164	500	32.80



8.2 Timing Summary after Implementation on 250 MHZ

Setup

Worst Negative Slack (WNS):	0.564 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	853

Hold

Worst Hold Slack (WHS):	0.064 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	853

Pulse Width

Worst Pulse Width Slack (WPWS):	1.358 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	951

All user specified timing constraints are met.

8.3 Power report after Implementation

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.601 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.1°C**
Thermal Margin:	58.9°C (32.4 W)
Effective ϑJA:	1.8°C/W
Power supplied to off-chip devices:	0 W