Atan2 CORDIC Report

Table of Contents

[Matlab 3](#_Toc62486369)

[Simulink 3](#_Toc62486370)

[Hardware 5](#_Toc62486371)

[Timing 5](#_Toc62486372)

[Resources 5](#_Toc62486373)

[Accuracy 5](#_Toc62486374)

# Matlab

Table : Analysis Results of Baseline vs Atan2 CORDIC in Matlab

|  |  |  |
| --- | --- | --- |
| Category | Angle | Magnitude |
| MSR | 5.6225e-04 | 2.1103e-06 |
| Max Error | 9.7651e-04 | 2.4360e-06 |
| Min Error | 0 | 5.2876e-07 |

The table above illustrates the mean-squared-root (MSR) error between using the baseline functions, atan2 and sqrt(x2 + y2), and the CORDIC implementation. This was done on all possible inputs ranging from -2 to 2 in 0.01 increments. At this point, the CORDIC algorithm and the baseline functions were using double floating-point precision.

# Simulink

Table : Analysis Results of Baseline vs Atan2 CORDIC in Simulink

|  |  |  |
| --- | --- | --- |
| Category | Angle | Magnitude |
| MSR | 5.8047e-04 | 1.1050e-04 |
| Max Error | 1.1891e-03 | 5.0516e-02 |
| Min Error | 7.9911e-07 | 0 |
| Quantization | 1.2207e-04 | |

The table above illustrates similar statistics as the Matlab table, with a small indication of the quantization error (or minimum step between two different numbers). This was done comparing the double precision float-point input values going into the baseline functions vs the signed, Q16.13 values going into the CORDIC block. Since Simulink does not have the ability to iterate over two numbers easily, the verification process was split between the magnitude and the angle. The magnitude data was set by simply taking the magnitude of the same number. This was to see how the calculation would work over the entire range. The angle data was a bit more complex, taking a single input and computing both the sin and cos of that input as using those as the iY and iX values. Below are the differences plotted against time.

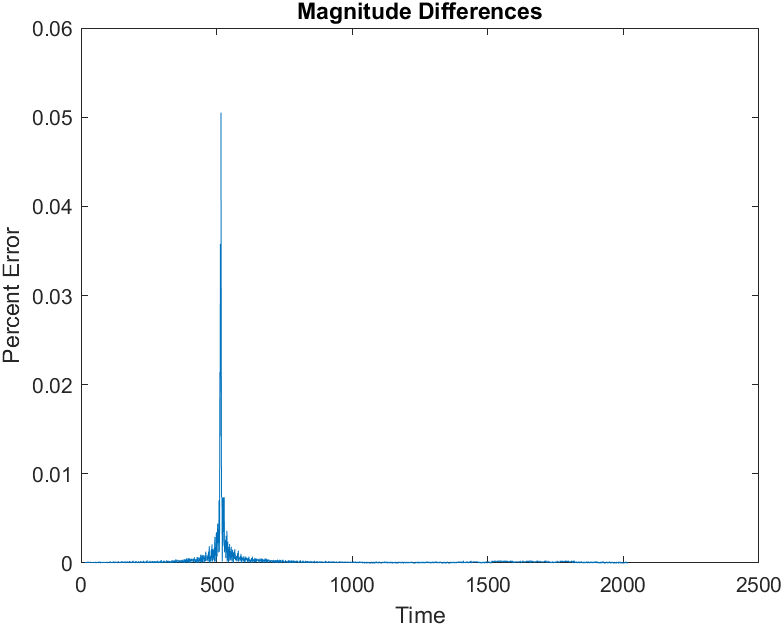


Figure : Magnitude Percent Error vs Time

Here, the magnitude had a slight spike in the error around the double value of 0.03. After some digging, it became apparent that 0.03 cannot be represented accurately as a signed Q16.13. The fixed-point approximation of 0.03 is 0.0299, thus creating that quantization error. Removing those outliers, most of the data remains below 1% error.

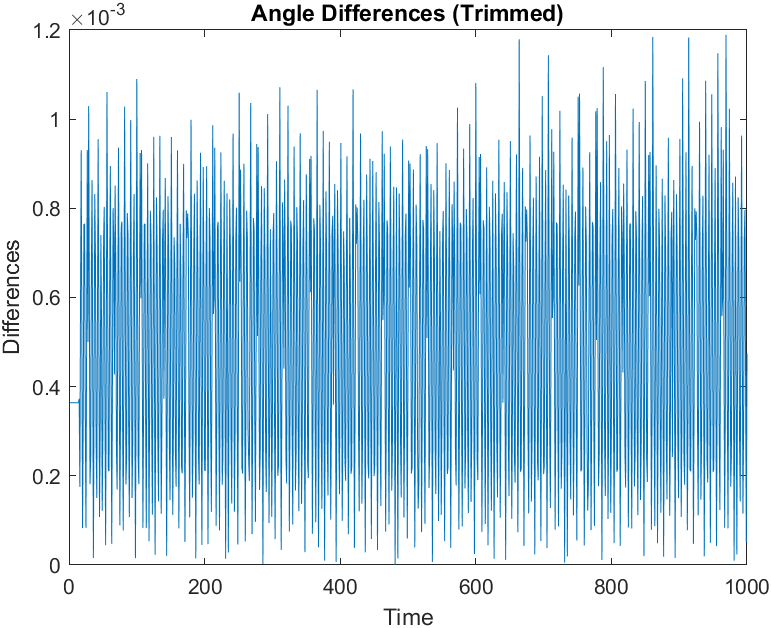


Figure : Angle Differences vs Time (Trimmed)

Here, the angle data also had a similar spike during the magnitude test but was omitted during the analysis of the angle data. As mentioned before, the angle data cannot be put through a percent error or any relative error algorithm since some of the accepted values can be zero. As a result, only the differences are illustrated in the figure above. Here, the difference between the baseline and the CORDIC implementation remains miniscule, hovering less than 9 times the Q16.13 epsilon value (roughly around 874% or 8.74).

# Hardware

## Timing

Table 3: Vivado Implementation Timing Report Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Target Frequency: 27 MHz | Setup (ns) | Hold (ns) | Pulse Width (ns) |
| Worst Negative Slack | 28.053 | 0.150 | 17.538 |
| Total Negative Slack | 0 | 0 | 0 |

The table illustrates the timing report summary compared against the minimum frequency of 27 MHz. This frequency was chosen as the baseline since the camera being used will operate with 27 MHz per pixel. Since this algorithm must work per pixel, it must operate at least 27 MHz or, ideally, faster.

## Resources

Table : Vivado Implementation Utilization Report Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Resource | Utilization | Available | Utilization % |
| LUT | 855 | 53200 | 1.61 |
| LUTRAM | 12 | 17400 | 0.07 |
| FF | 732 | 106400 | 0.69 |
| DSP | 4 | 220 | 1.82 |

The table illustrates the utilization report summary of the atan2 algorithm as synthesized by the HDL Coder output. There was a single minor tweak to the generated code but details of that tweak is outside the scope of this document. Overall, the CORDIC algorithm does not take a lot of resources at all but can certainly be optimized more within the Simulink portion of the design. One example would be to maintain the signed Q16.13 fixed point notation throughout the entire implementation which will drastically reduce the LUT and FF utilization. The downside of this is that its highly possible that more quantization errors can occur aside from 0.03 as depicted before.

## Accuracy

A post functional simulation was conducted at each step of the process. Only the post functional simulation of the implementation will be shown. The simulations were done with a 20 ns clock period (for simplicity sake) and the data were read in from files generated from the same inputs as the Simulink model.

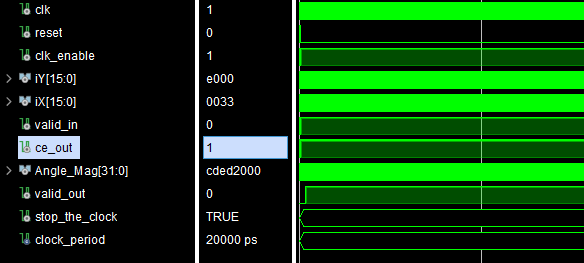


Figure : Vivado Testbench Waveforms

As depicted in the figure, this testbench was also exhaustive, starting from the minimum of a signed Q16.13 fixed point number, -4, to the upper bound of 3.99999. Here, the outputs were compared to the outputs in the Simulink model, reporting if any discrepancies occurred. From this testbench, there were no errors reported, signifying the hardware implementation matches the Simulink model completely.

Note, while the Simulink model was designed to have a pipeline of 14 stages, the HDL Coder increased it to 16 stages. This was most likely due to any pipeline matching or strange synthesized code. As a recommendation in the future, this design can most certainly be hand done. The CORDIC algorithm is not complex enough to require the HDL Coder. In this instance, it was used for convenience.