CORDIC Reference Design

June 2005, ver. 1.4 Application Note 263

Introduction

The co-ordinate rotation digital computer (CORDIC) reference design implements the CORDIC algorithm, which converts cartesian to polar coordinates and vice versa and also allows vectors to be rotated through a given angle.

CORDIC is an iterative process, using a series of shifts and adds. Hence it is often a hardware efficient solution over using multiplications, division and square roots.

The reference design comprises a MATLAB bit accurate model of the CORDIC algorithm implemented in hardware and a GUI test environment. The test environment allows you to configure the CORDIC reference design and test conditions and to compare the results with the theoretical results.

Altera provides separately the encrypted Verilog HDL source code, precompiled simulation files for the ModelSim simulator, and a testbench for the hardware implementation of CORDIC.

Background

CORDIC provides an iterative solution to performing vector rotations by arbitrary angles using only shifts and adds. The CORDIC algorithm can operate in either vectoring or rotation mode.

Vectoring mode performs cartesian to polar conversion. An input vector is rotated until it is on the x axis; the final x value is equal to the magnitude of the input vector. While rotating the vector, the total angle traversed is also recorded, which provides the phase of the input vector.

Rotation mode performs polar to cartesian conversion. The input vector (x value equals magnitude and y value equals zero) is rotated by the specified angle. The final vector is the cartesian equivalent of the input polar values.

Vector rotation mode rotates an input vector by a specified angle. The output gives the cartesian coordinates of the rotated vector.

The final x and y values in rotation, vector rotation, and vectoring modes arescaled by the CORDIC processing gain. This processing gain varies with the number of iterations performed. It approaches 1.6476 as the number of iterations goes to infinity.

At some point in the system this gain may have to be compensated for. If so, in vectoring mode, the resultant magnitude (x value) needs to be divided by the processing gain (or multiplied by the reciprocal of the processing gain); in rotation mode either the input polar magnitude (input x value) or the output x and y values can be divided by the processing gain.

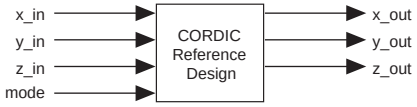
Functional Description

Figure 1 shows the top-level block diagram of the CORDIC reference design.

The x, y, and z inputs and outputs are twos complement signed numbers. The bit widths and number of iterations are parameterizable.

The range of z is given by $-\pi \leq z < +\pi$, where $-\pi = -2^{(z\_bits - 1)}$.

Figure 1. Block Diagram



Cartesian to Polar Conversion

The cartesian to polar conversion has the following attributes:

- `x_in` and `y_in` are the input cartesian values
- `z_in` must be zero
- `mode` = 0, to enforce vectoring mode
- `x_out` is the polar magnitude (scaled by processing gain)
- `z_out` is the polar phase

Polar to Cartesian Conversion

The polar to cartesian conversion has the following attributes:

- `x_in` is the input polar magnitude.
- `y_in` must be zero.
- `z_in` is the input polar phase.
- `mode` = 1 to enforce rotation mode
- `x_out`, `y_out` is the cartesian values (scaled by processing gain)

Vector Rotation

The vector rotation has the following attributes:

- `x_in` and `y_in` are the input cartesian values
- `z_in` is the angle to rotate the input vector by
- `mode = 1` to enforce rotation mode
- `x_out` and `y_out` are the cartesian values

Vector Width

The vector width of x and y should be increased by two over the required width, because of the processing gain of the CORDIC block. For example, if 16 bits represent the magnitude, you should select 18 bits as the vector width for x and y inputs and outputs. Any required sign extension of the inputs must be performed, prior to presenting the x and y inputs to the CORDIC reference design.

You can select input and output vector widths of up to 32 bits for x, y and z.

Iterations

The accuracy of the CORDIC algorithm improves with each iteration, until the number of iterations equals the bit width of the inputs.

The CORDIC reference design is pipelined at every iteration. The number of clock cycles required to perform a conversion is equal to the number of iterations plus 3 (this extra number required for additional registering).

The number of iterations can be from 5 to 32, but must not exceed the number of bits of the x, y, and z input values.

Processing Gain Compensation

The CORDIC reference design does not compensate for the processing gain. However, Altera supply source code for a gain compensation block with the encrypted source code for the CORDIC reference design,. This block is synthesizable and is provided as sample code (it is not necessarily optimized).

The gain compensation block has two implementations, which are selectable via parameters.

In the first implementation the gain compensation is a series of shifts and adds. It has a latency of 6 clock cycles. For 32 bit vector lengths this achieves performance of over 285 MHz and uses approximately 950 LEs.

The second implementation uses a Stratix DSP block and has a latency of two clock cycles. For 32 bit vector lengths this achieves performance of over 190 MHz using one DSP block.

Testbench

The Verilog HDL testbench optionally connects the processing gain compensation block either before (if testing in rotational mode and removing gain) and/or after the CORDIC reference design (if testing in vectoring mode and removing gain). Input data is read from text files, and the output from the system is read and compared to the expected data which is also stored in text files.

Any error messages are logged to the screen and also logged to an output file.

The MATLAB environment auto-generates the input data and expected output data text files or you can generate them manually.

Altera provides scripts for simulation with ModelSim simulators.

Design Flow

The MATLAB GUI environment configures the CORDIC reference design (i.e., vector widths, number of iterations, gain compensation) and the test conditions (i.e., rotation or vectoring modes, number of input samples, random or sequential input data, restrictions on input values).

You can plot graphs to compare the performance of the MATLAB bit accurate model of the CORDIC with the theoretical results. Once you are satisfied with the behavior of the CORDIC (i.e. accuracy), the GUI allows the input and output data to be written to text files and Verilog HDL parameters to configure the hardware CORDIC, gain compensation blocks, and testbench.

The Verilog HDL testbench confirms that the Verilog HDL behaves identically to your MATLAB CORDIC model that you required.

The Verilog HDL testbench is shipped with the CORDIC reference design encrypted source code.

Getting Started

This section involves the following steps:

- Software Requirements
- Install the Design
- Design Walkthrough

Software Requirements

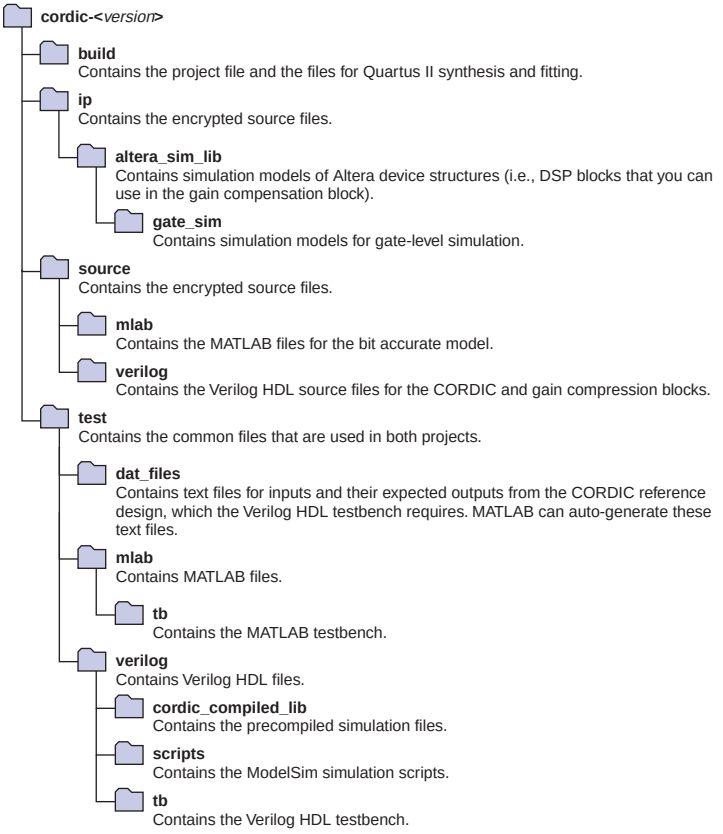
The design requires the following software:

- Altera® Quartus® II software version 2.2 SP2
- Mathworks MATLAB version 6.5
- ModelSim version 5.7a

Install the Design

To install the reference design, run the **.exe**. and follow the installation instructions. Figure 2 shows the directory structure.

Figure 2. Directory Structure



Tables 1 to 8 describe the files in each directory.

Table 1. sourcebuild Directory Files

File Name	Description
cordic.psf	Quartus II project settings file.
cordic.quartus	Quartus II project file.

Table 2. ip Directory Files

File Name	Description
altera_sim_lib/altera_mf.v	Verilog functional simulation models for structures in Altera devices (e.g., DSP blocks).
altera_sim_lib/gate_sim/stratix_atoms.v	Verilog simulation models required for gate-level simulation.

Table 3. source/mlab Directory Files

File Name	Description
cordic.m	Bit accurate MATLAB model of CORDIC and gain compensation blocks.
neg2pos.m	MATLAB function calculates the absolute number that represents negative numbers in twos complement.

Table 4. source/verilog Directory Files

File Name	Description
cordic.v	Top level CORDIC module.
cordic_core.v	module containing algorithm.
ip_quad_info.v	Determines quadrant information for CORDIC inputs.
ip_quad_adj.v	Adjusts inputs to algorithm.
op_quad_adj.v	Adjusts outputs from algorithm.
cordic_inc.v	Contains the static parameters.
cordic_inc_p2.v	Contains the parameters that change according to CORDIC configuration; can be auto-generated by MATLAB.
cordic_gain_corr.v	Gain compensation block.

Table 5. test/dat_files Directory Files (required for the Verilog HDL testbench)	
File Name	**Description**
ip_mode.txt	CORDIC mode of each input (rotation or vectoring); can be auto-generated by MATLAB.
ip_xy.txt	X and Y inputs to the CORDIC system.
ip_z.txt	Z inputs to CORDIC system.
op_exp_xy.txt	Expected X and Y outputs from CORDIC system.
op_exp_z.txt	Expected Z output from CORDIC system.
op_err.txt *(1)*	Output error log from simulation run.

Note:
(1) MATLAB can auto-generate all of the files except for **op_err.txt**.

Table 6. test/mlab/tb Directory Files	
File Name	**Description**
cordic_gui.fig	MATLAB file for GUI.
cordic_gui.m	MATLAB file for GUI.
cordic_test.m	MATLAB script that runs the CORDIC test.
comb_mlab_files	Pearl script that combines data results from two different sets of files containing the input and expected output data.

Table 7. test/verilog/scripts Directory Files	
File Name	**Description**
encrypt_msim_com.bat	PC batch file for using ModelSim to simulate design in command mode.
encrypt_msim_gui.bat	PC batch file for using ModelSim in GUI mode to simulate design.

Table 8. test/verilog/tb Directory Files	
File Name	**Description**
cordic_tb.v	Verilog HDL testbench.
modelsim_wave.do	Waveform file for ModelSim.

Design Walkthrough

The walkthrough involves the following steps:

- Start the MATLAB GUI
- Parameterize the Design
- Auto-generate Verilog Simulation Files from MATLAB
- Simulation

Start the MATLAB GUI

To start the MATLAB GUI, perform the following steps:

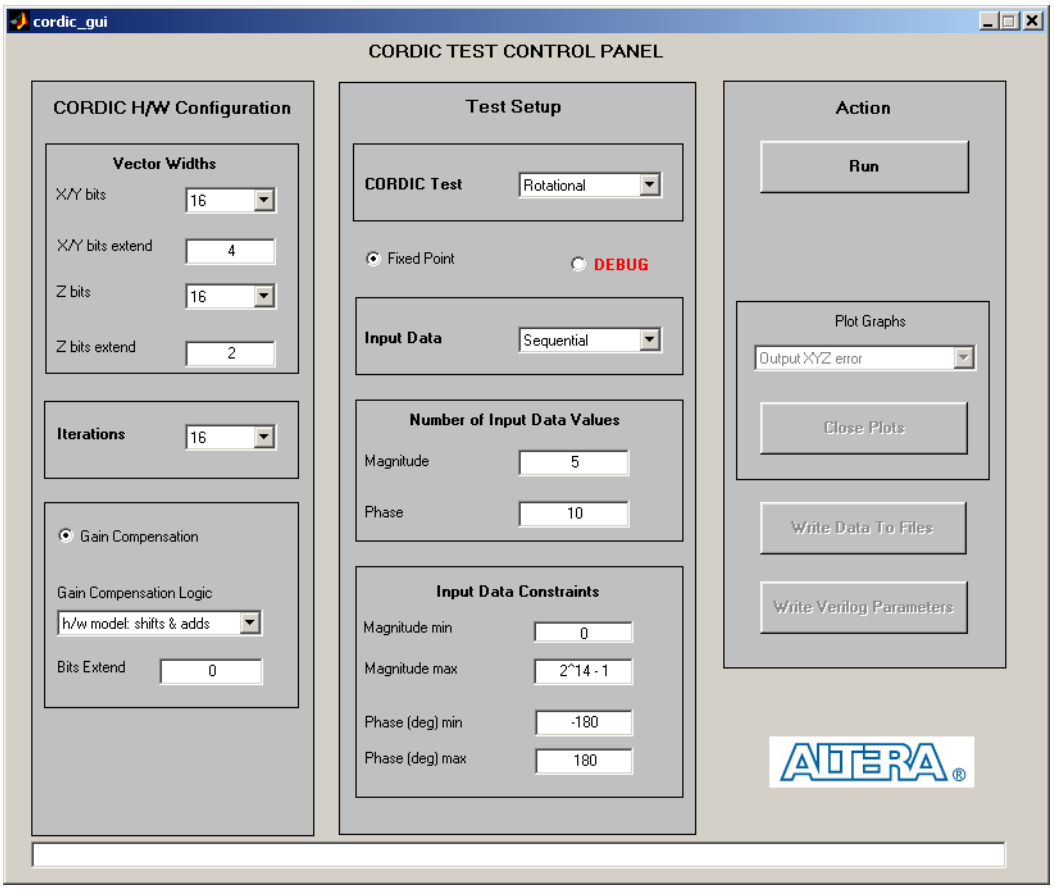
1. Start MATLAB.
2. Change the working directory to **cordic-**<*version*>**/test/mlab/tb**.
3. Type the following command:

```
cordic_gui
```

Output messages are still output to the MATLAB command window.

The CORDIC test control window opens (see Figure 3).

Figure 3. CORDIC Test Control Window



Parameterize the Design

To parameterize your design, perform the following steps:

1. In the CORDIC H/W Configuration part of the CORDIC test control panel, perform the following steps:

    a. Choose the vector widths: x/y and z bits, and x/y and z bits extend. Where the number of bits specifies the number of bits going into and out of the CORDIC. Bits extends specifies the number of extra bits the CORDIC works with, to increase accuracy.

b. Choose the number of iterations.

c. If you want gain compensation, select **Gain Compensation**, and choose the gain compensation logic. If you choose **h/w logic shifts & adds**, enter a value for **Bits Extend**, to increase accuracy.

2. In the Test Set-up part of the CORDIC test control panel, perform the following steps:

a. Choose **rotational**, or **vectoring**, or **vector rotation** mode.

b. If you require fixed point test, select **Fixed Point**.

Selecting **Fixed Point** overrides all settings for your x, y, and z bit widths.

c. Choose either **Sequential** or **Random** input data.

d. In rotational mode, enter the number of input data values for magnitude and phase.

or

In vectoring mode, enter the number of input data values for the X and Y values.

or

In vector rotation mode, enter the different number of rotation angles in the Phase box, enter the different number of input vectors in the Magnitude box (each different input magnitude has a different starting phase, resulting in the different input vector; the starting phases are distributed evenly throughout the 360 degree space). Each different input vector is rotated through all the different rotation angles.

e. In rotational mode, enter the **Input Data Constraints**: minimum magnitude, maximum magnitude, minimum phase, maximum phase.

or

In vectoring mode, enter the **Input Data Constraints**: minimum X, maximum X, minimum Y, and maximum Y.

3. Click **Run**.

4. Select the type of graph that you want MATLAB to plot. Figure 4 to Figure 6 show various graphs that you can select, for various CORDIC test control panel settings.

Figure 4. Input Values Graph

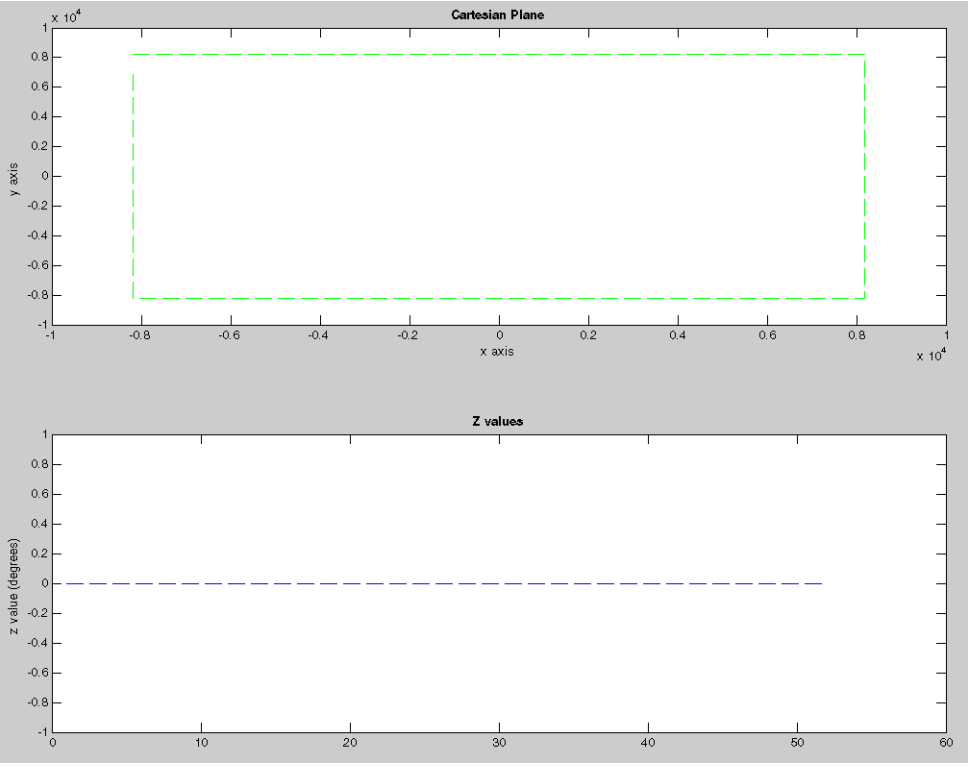


Figure 5. Output Polar Errors Graph

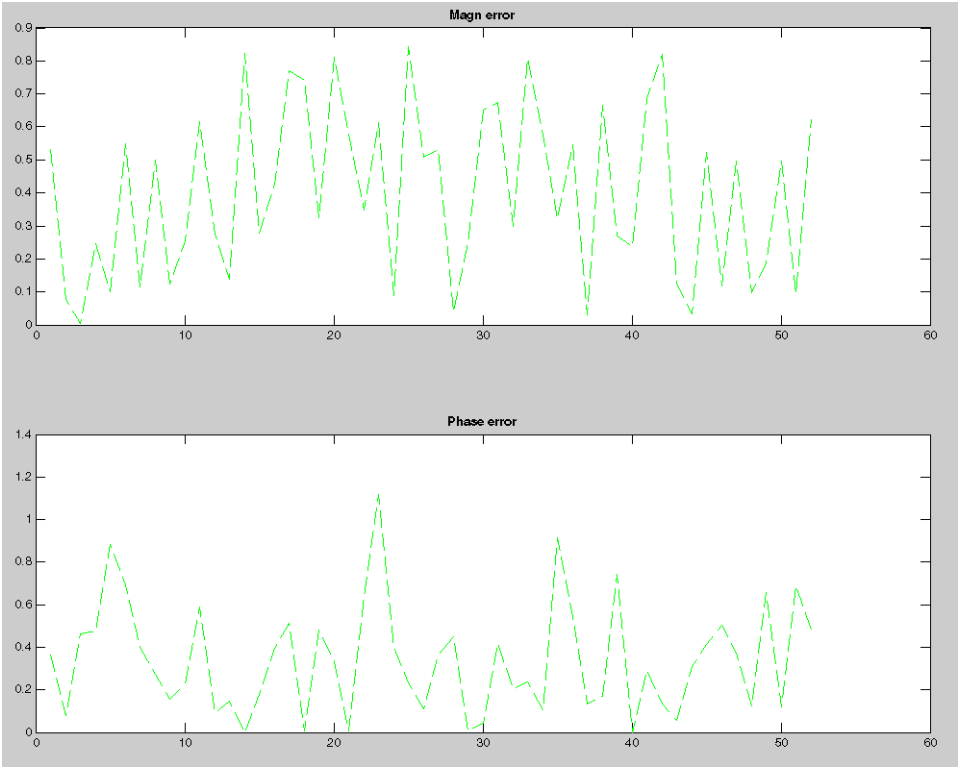
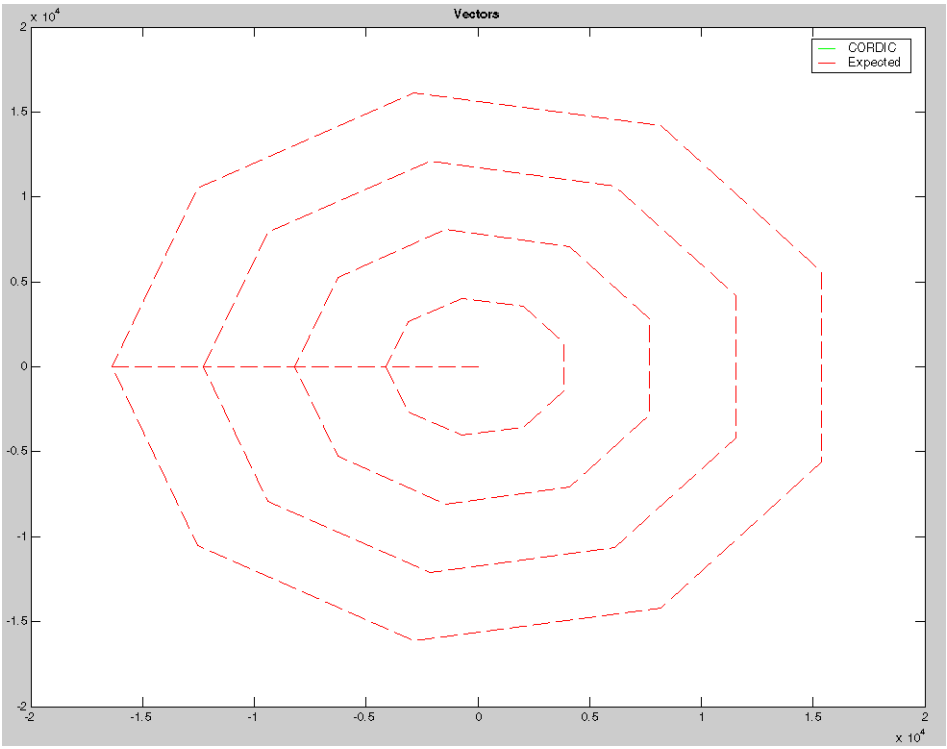


Figure 6. Output X Y on a Cartesian Plane Graph



5. When you have viewed the graph, click **Close Plots**.

Auto-generate Verilog Simulation Files from MATLAB

Once MATLAB testing reaches a conclusion with the configuration of a suitable CORDIC reference design to auto-generate Verilog simulation files from MATLAB, perform the following steps:

1. Click **Write Data to Files** to write the input and expected output data to **cordic-**<*version*>**/test/dat_files**.

2. Click **Write Verilog Parameters**, to write the required Verilog HDL parameters to **cordic-**<*version*>**/source/verilog/cordic_inc_p2.vh** and **cordic-**<*version*>**/test/verilog/tb/cordic_tb_inc.vh**.

☞ These files contain parameters to configure the CORDIC reference design, the gain compensation block, and the testbench.

Simulation

To use the ModelSim simulation files, perform the following steps:

1. Open the ModelSim simulator.
2. Change the working directory to **cordic-<*version*>/test/verilog/script**.
3. Ensure that the required data files are in the **cordic-<*version*>/test/dat_files** directory.
4. If you want to run the GUI run the **cordic-<*version*>/test/verilog/scripts/encrypt_msim_gui.bat** batch file otherwise run the **cordic-<*version*>/test/verilog/scripts/encrypt_msim_com.bat** batch file.
5. The output log file, which is generated from simulation run, is **cordic-<*version*>/test/dat_files/op_err.txt**.

If you subsquently change the parameter values or the data used in the test and still have the ModelSim simulator open from the previous run, you can type `cr` in the ModelSim window, which recompiles the testbench and reruns the simulation.

Performance

Table 9 shows the Quartus II push-button performance for the CORDIC reference design on Stratix™ devices.

Table 9. Example Performance in Stratix Devices

x, y, & z Bit Widths	Number of Iterations	Logic Elements (LEs)	% LEs in Device *Note (1)*	f_{MAX} (MHz)
8	8	292	2	269
16	16	965	9	221
24	24	2,021	19	202
32	32	3,462	32	185
40	40	5,287	50	179

Note:
(1) EP1S10F484C5 device.

Table 9 shows the Quartus II push-button performance the CORDIC reference design on Cyclone™ devices.

Table 10. Example Performance in Cyclone Devices

x, y, & z Bit Widths	Number of Iterations	Logic Elements (LEs)	% LEs in Device	f_{MAX} (MHz)
8	8	293	10 *(1)*	233
16	16	963	24 *(2)*	219
24	24	2,022	50 *(2)*	181
32	32	3,463	28 *(3)*	173
40	40	5,287	43 *(3)*	166

Note:
(1) EP1C 3T144C6 device.
(2) EP1C 4F324C6 device.
(3) EP1C 12F324C6 device.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com