LogiCORE IP CORDIC v4.0

DS249 March 1, 2011 **Product Specification**

Introduction

The Xilinx LogiCORE™ IP CORDIC core implements a generalized coordinate rotational digital computer (CORDIC) algorithm.

Features

- Drop-in module for Virtex®-7 and Kintex™-7, Virtex®-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3A/XA/AN/3A DSP and Spartan-3E/XA FPGAs
- Functional configurations
  - Vector rotation (polar to rectangular)
  - Vector translation (rectangular to polar)
  - Sin and Cos
  - Sinh and Cosh
  - Atan and Atanh
  - Square root
- Optional coarse rotation module to extend the range of CORDIC from the first quadrant (+Pi/4 to - Pi/4 Radians) to the full circle
- Optional amplitude compensation scaling module to compensate for the CORDIC algorithm's output amplitude scale factor
- Output rounding modes: Truncation, Round to Pos Infinity, Round to Pos/Neg Infinity, and Round to Nearest Even
- Word serial architectural configuration for small area
- Parallel architectural configuration for high throughput
- Control of the internal add-sub precision
- Control of the number of add-sub iterations
- Optional input and output registers
- Optional control signals: CE, ND, SCLR, RFD, and RDY
- X and Y data formats: Signed Fraction, Unsigned Fraction, and Unsigned Integer
- Phase data formats: Radian, Pi Radian
- Fully synchronous design using a single clock
- For use with Xilinx CORE Generator™ and Xilinx System Generator for DSP, v13.1.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex-7 and Kintex-7, Virtex-6, Virtex-5, Virtex-4, Spartan-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/XA/3AN/3A DSP
Supported User Interfaces	Not Applicable
Provided with Core	
Documentation	Product Specification
Design Files	Netlist
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Not Provided
Simulation Model	Verilog and VHDL
Tested Design Tools	
Design Entry Tools	CORE Generator tool 13.1 System Generator for DSP 13.1
Simulation	Mentor Graphics ModelSim 6.6d Cadence Incisive Enterprise Simulator (IES) 10.2 Synopsys VCS and VCS MX 2010.06 ISIM 13.1
Synthesis Tools	N/A
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.

General Description

The CORDIC core implements a generalized coordinate rotational digital computer (CORDIC) algorithm, initially developed by Volder[1] to iteratively solve trigonometric equations, and later generalized by Walther[2] to solve a broader range of equations, including the hyperbolic and square root equations. The CORDIC core implements the following equation types:

- Rectangular <-> Polar Conversion
- Trigonometric
- Hyperbolic
- Square Root

Two architectural configurations are available for the CORDIC core:

- A fully parallel configuration with single-cycle data throughput at the expense of silicon area
- A word serial implementation with multiple-cycle throughput but occupying a small silicon area

A coarse rotation is performed to rotate the input sample from the full circle into the first quadrant. (The coarse rotation stage is required as the CORDIC algorithm is only valid over the first quadrant). An inverse coarse rotation stage rotates the output sample into the correct quadrant.

The CORDIC algorithm introduces a scale factor to the amplitude of the result, and the CORDIC core provides the option of automatically compensating for the CORDIC scale factor.

A block diagram of the CORDIC core is presented in Figure 1.

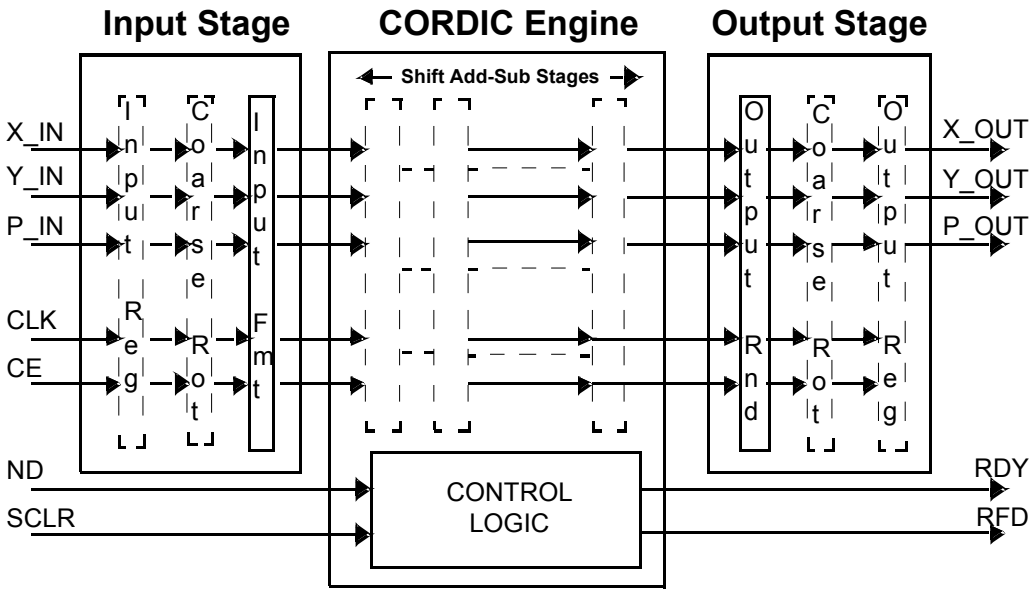

Figure 1: **CORDIC Symbol and Pinout**

Interface Pins

Table 1: **Core Pinout**

Port Name	Direction	Description
X_IN [Input_Width-1:0]	IN	X component of input sample. Required depending on Functional Configuration.
Y_IN [Input_Width-1:0]	IN	Y component of input sample. Required depending on Functional Configuration.
PHASE_IN [Input_Width-1:0]	IN	Phase component of input sample. Required depending on Functional Configuration.
X_OUT [Output_Width-1:0]	OUT	X component of output sample. Optional
Y_OUT [Output_Width-1:0]	OUT	Y component of output sample. Optional
PHASE_OUT [Output_Width-1:0]	OUT	Phase component of output sample. Optional
ND	IN	New sample on input ports. Active high.
RFD	OUT	Ready for new data sample. Active high.
RDY	OUT	New output data is ready. Active high.
CLK	IN	Clock. Active rising edge.
CE	IN	Clock enable. Active high.
SCLR	IN	Synchronous clear. Active high, SCLR has priority over CE

Data Inputs

X_IN, Y_IN and PHASE_IN are the data input ports for the CORDIC core. All data input ports are read simultaneously to form a single input sample. The width of the data input ports is configured using the GUI parameter Input Width. The data input ports are optionally registered. The set of data input ports required for a particular Functional Configuration are automatically determined by the GUI as shown in Table 2.

Data Outputs

X_OUT, Y_OUT and PHASE_OUT are the data output ports for the CORDIC core. The default settings for data output ports required for a particular Functional Configuration are automatically determined by the GUI as shown

in Table 2, but may be modified from the default settings by the user. The width of the CORDIC data output ports is set using the parameter Output Width. The data output ports are optionally registered.

Table 2: **Input/Output Pins vs. Functional Configuration(1)**

Function	XIN	YIN	PIN	XOUT	YOUT	POUT
Rotate	1	1	1	1	1	0
Translate	1	1	0	1	0	1
Sin and Cos	0	0	1	1	1	0
ArcTan	1	1	0	0	0	1
Sinh and Cosh	0	0	1	1	1	0
ArcTanh	1	1	0	0	0	1
Square Root	1	0	0	1	0	0

1. Grey shading indicates the port configurations that are fixed for the selected function.

The CORDIC Algorithm

The CORDIC algorithm was initially designed to perform a vector rotation, where the vector (X,Y) is rotated through the angle θ yielding a new vector (X',Y').

Vector Rotation Equation *Equation 1*

1a) $X' = (\cos(\theta) \times X - \sin(\theta) \times Y)$

1b) $Y' = (\cos(\theta) \times Y + \sin(\theta) \times X)$

1c) $\theta' = 0$

The CORDIC algorithm performs a vector rotation as a sequence of successively smaller rotations, each of angle atan(2^{-i}), known as *micro-rotations*. Equation 2 shows the expression for the i^{th} iteration where i is the iteration index from 0 to n.

Expression for the i^{th} microrotation *Equation 2*

2a) $x_{i+1} = x_i - \alpha_i \cdot y_i \cdot 2^{-i}$

2b) $y_{i+1} = y_i + \alpha_i \cdot x_i \cdot 2^{-i}$

2c) $\theta_{i+1} = \theta_i + \alpha_i \cdot \text{atan}(2^{-i}))$

α_i = (+ or -) 1, where α_i is the direction of rotation.

See Vector Rotation or Vector Translation for details on selecting α_i.

Each micro-rotation stage can be expressed as a simple shift and add/subtract operation.

Equation 3 shows the Vector rotation expression for the n^{th} iteration.

Vector rotation expressed as a series of 'n' micro-rotations *Equation 3*

3a) $$X' = \prod_{i=1}^{n} \cos(\text{atan}(2^{-i}))(X_i - \alpha_i Y_i 2^{-i})$$

3b) $$Y' = \prod_{i=1}^{n} \cos(\text{atan}(2^{-i}))(Y_i + \alpha_i X_i 2^{-i})$$

3c) $\theta' = \sum_{i=1}^{n} \theta - (\alpha_i \cdot \text{atan}(2^{-i}))$

α_i = (+ or -) 1.

The CORDIC algorithm can be used to generate either a vector rotation or a vector translation.

Vector Rotation

Vector rotation rotates the vector (X, Y) through the angle θ to yield a new vector (X′,Y′), as illustrated in Figure 2.

Vector rotation is performed by selecting α_i, such that θ' converges towards zero; that is, when $\theta_{i-1} >= 0$, α_i is set to -1 and when $\theta_{i-1} < 0$, α_i is set +1.

Vector Rotation Equations *Equation 4*

4a) $X' = Z_n \times (\cos(\theta) \times X - \sin(\theta) \times Y)$

4b) $Y' = Z_n \times (\cos(\theta) \times Y + \sin(\theta) \times X)$

4c) $\theta' = 0$

$$Z_n = \frac{1}{\prod_{i=1}^{n} \text{acos}(\text{atan}(2^{-i}))}$$

Vector Translation

Vector translation rotates the vector (X,Y) around the circle until the Y component equals zero as illustrated in Figure 3. The outputs from vector translation are the magnitude, X′, and phase, θ', of the input vector (X,Y).

Vector translation is performed by selecting α_i such that Y′ converges towards zero; that is, when $Y_{i-1} >= 0$, α_i is set to -1 and when $Y_{i-1} < 0$, α_i is set +1.

Vector Translation Equations *Equation 5*

5a) $X' = Z_n \times \sqrt{(X^2 + Y^2)}$

5b) $Y' = 0$

5c) $\theta' = \text{atan}\left(\frac{X}{Y}\right)$

$$Z_n = \frac{1}{\prod_{i=1}^{n} \text{acos}(\text{atan}(2^{-i}))}$$

The CORDIC Scale Factor

The outputs of the CORDIC algorithm, equations 4 and 5, are equivalent to a vector rotation or vector translation scaled by a constant Z_n. The constant Z_n is known as the CORDIC scale factor.

The CORDIC Scale Factor *Equation 6*

6a) $$Z_n = \frac{1}{\prod^{n} \text{acos}(\text{atan}(2^{-i}))}$$

The Taylor series expansion of acos (atan (2^{-i})) is $(1 + 2^{-2i})^{-1/2}$. Hence, the constant Z_n can be expressed as

6b) $$Z_n = \prod_{i=1}^{n} (1 + 2^{-2i})^{1/2}$$

The CORDIC scale factor, Z_n, is only dependent on the number of iterations, n. Only functional configurations Rotate, Translate, Rectangular to Polar, and Polar to Rectangular are affected by the CORDIC scale factor. When these functional configurations are selected, the CORDIC core provides the option of multiplying by 1 / Z_n to cancel out the scaling factor. See Advanced Configuration Parameters for detailed information.

Output Quantization Error

The Output Quantization Error can be split into two components; the Output Quantization Error due to the Input Quantization (OQEIQ) and the Output Quantization Error due to Internal Precision (OQEIP).

OQEIQ is due to the 1/2 lsb of quantization noise on the X,Y and Phase inputs. In a vector rotation this input quantization noise results in OQEIQ of 1/2 an lsb on both the X and Y outputs. In a vector translation this input quantization noise results in OQEIQ of 1/2 an lsb on the X output however OQEIQ on the phase output is dependant on the ratio (Y/ X). Thus for small X inputs the effect of input quantization noise on OQEIQ is greatly magnified.

OQEIP is due to the limited precision of internal calculations. In the CORDIC core the default internal precision is set such that the accumulated OQEIP is less than 1/2 the OQEIQ. The internal precision can be manually set to (input width + output width + $\log_2$(output_width)). This reduces OQEIP to 1/ 2 an lsb (the phase is calculated to full precision regardless of the magnitude input vector).

The Output Quantization Error, for a CORDIC core with default internal precision, is dominated by OQEIQ. OQEIQ can only be reduced by increasing the number of significant magnitude bits in the input vector (X,Y). Increasing the internal precision or zero padding X and Y inputs only affects OQEIP and has minimal effect on the total output quantization error.

The effect of input quantization and internal quantization on the CORDIC phase output quantization error is illustrated in the following examples.

Example 1a: The quantization error in phase output for a small input vector, (Xin_small, Yin_small).

Xin_small : “0000000001” => 1/256.

Yin_small : “0000000001” => 1/256.

Vector translation with no input quantization:

Xin_ideal : “0000000001” => 1/256.

Yin_ideal : “0000000001” => 1/256.

Pout_ideal : “0001100100” => 0.79.

Output quantization error due to the input quantization:

Xin_Quant = Xin_small - 1/2 lsb and Yin_Quant = Yin_small + 1/2 lsb.

Xin_Quant : “00000000001” => 1/512.

Yin_Quant : “00000000011” => 3/512.

Pout_Quant : “0010100000” =>1.25.

OQEIQ = abs(abs(Pout_Quant) - abs(Pout_Ideal)).

OQEIQ = "0000111100" => 0.47.

Output quantization error due to the internal precision:

Xin_cordic : “0000000001” => 1/256.

Yin_cordic : “0000000001” => 1/256.

Pout_cordic : “0001111010” => 0.95.

OQEIP = abs(abs(Pout_cordic) - abs(Pout_Ideal)).

OQEIP = "0000010110" => 0.17.

Example 1b: Quantization error in phase output for a large input vector, (Xin_large, Yin_large).

Xin_large : “0100000000” => 256/256.

Yin_large : “0100000000” => 256/256.

Vector translation with no input quantization:

Xin_ideal : “0100000000” => 256/256.

Yin_ideal : “0100000000” => 256/256.

Pout_ideal : “0001100100” => 0.79.

Output quantization error due to the input quantization:

Xin_Quant = Xin_large - 1/2 lsb and Yin_Quant = Yin_small + 1/2 lsb.

Xin_Quant : “00111111111” => 511/512.

Yin_Quant : “01000000001” => 513/512.

Pout_Quant : “0001100101” =>0.79.

OQEIQ = abs(abs(Pout_Quant) - abs(Pout_Ideal)).

OQEIQ = "0000000001" => 0.00.

Output quantization error due to the internal precision:

Xin_cordic : “0100000000” => 256/256.

Yin_cordic : “0100000000” => 256/256.

Pout_cordic : “0001100100” => 0.79.

OQEIP = abs(abs(Pout_cordic) - abs(Pout_Ideal)).

OQEIP = "0000000000" => 0.00

Functional Description

Vector Rotation

Polar to Rectangular Translation

When the vector rotation functional configuration is selected the input vector, (X, Y), is rotated by the input angle, θ , using the CORDIC algorithm. This generates the scaled output vector, Z_i * (X′, Y′), as shown in Figure 2.

The inputs, X_IN, Y_IN and PHASE_IN, are limited to the ranges given in Table 3. Inputs outside these ranges produce unpredictable results. See Input/Output Data Representation for detailed information regarding CORDIC binary data formats.

An optional coarse rotation module is provided to extend the range of the inputs, X, Y and Phase, to the full circle. For this functional configuration the coarse rotation module is selected by default but can be manually deselected by the user. See Advanced Configuration Parameters for detailed information.

An optional compensation scaling module is provided to compensate for the CORDIC scale factor Z_i. For this functional configuration the compensation scaling module is selected by default but can be manually deselected by the user. See Advanced Configuration Parameters for detailed information.

A Polar to Rectangular Translation can be implemented by setting the functional configuration to vector rotation, the input vector to (Mag, 0), and the rotation angle to θ , as shown in Figure 3.

Vector rotation is linear with respect to magnitude, thus the user can scale the input/output range; that is:
if (X, Y) rotated by angle θ = (X′, Y′) then

K*(X, Y) rotated by angle θ = K*(X′, Y′).

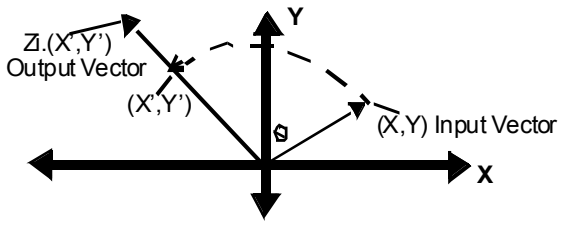

Figure 2: **Vector Rotation**

Table 3: **Vector Rotation I/O**

Signal	Range	Description
X_IN	-1 <= X_IN<=1	Input X Coordinate
Y_IN	-1 <= Y_IN<=1	Input Y Coordinate
PHASE_IN	-Pi <= PHASE_IN <= Pi	Input Rotation Angle
X_OUT	-Sqrt(2) <= X_OUT<= Sqrt(2)	Output X Coordinate * Z
Y_OUT	-Sqrt(2) <= Y_OUT<= Sqrt(2)	Output Y Coordinate * Z

Example 1: Vector Rotation

The input vector, (Xin, Yin), and the output vector, (Xout, Yout) are expressed as a pair of fixed-point 2's complement numbers with an integer width of 2 bits (1QN format). The input rotation angle, Pin radians, is also

expressed as a fixed-point 2's complement number but with an integer width of 3 bits (2QN format). Please refer to the Input/Output Data Representation section for further information on the CORDIC binary data formats.

In this example, the input/output width is set to 10 bits and the output vector (Xout, Yout) is scaled to compensate for the CORDIC scale factor.

Xin : "0010110101" => 00.10110101 => 0.707

Yin : "0001000000" => 00.01000000 => 0.25

Pin : "1100110111" => 110.0110111 => -Pi/2

Xout : "0001000001" => 00.01000001 => 0.25

Yout : "1101001011" => 11.01001011 => -0.707

Vector Translation

Rectangular to Polar Translation

When the vector translational functional configuration is selected, the input vector (X,Y) is rotated using the CORDIC algorithm until the Y component is zero. This generates the scaled output magnitude, $Z_i * Mag(X,Y)$, and the output phase, Atan(Y/X), as shown in Figure 3.

The inputs, X_IN and Y_IN, are limited to the ranges given in Table 4. Inputs outside these ranges produce unpredictable results. See Input/Output Data Representation for detailed information regarding CORDIC binary data formats.

An optional coarse rotation module is provided to extend the range of inputs, X and Y, to the full circle. For this functional configuration the coarse rotation module is selected by default but can be manually deselected by the user. See Advanced Configuration Parameters for detailed information.

An optional compensation scaling module is provided to compensate for the CORDIC scale factor Z_i. For this functional configuration the compensation scaling module is selected by default but can be manually deselected by the user. See Advanced Configuration Parameters for detailed information.

A rectangular to polar translation can be implemented by setting functional configuration to vector translation, and the input vector to (X,Y), as shown in Figure 3.

Vector translation is linear with respect to magnitude, thus the user can scale the input/output range; that is:

if vector (X, Y) is translated to (X', θ '), then
vector K*(X, Y) is translated to K*(X', θ ').

The phase angle of a zero length vector, (0,0), is indeterminate and the output phase angle generated by the core is unpredictable.

The accuracy of the phase output from the CORDIC vector translation algorithm is limited by the number of significant magnitude bits of the input vector (X, Y). See CORE Generator GUI and Parameters for detailed information.

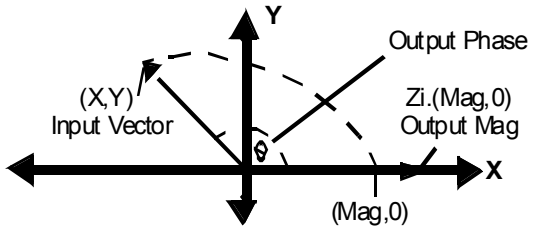

Figure 3: **Vector Translation (Polar to Rectangular)**

Example 2: Vector Translation

Table 4: **Vector Translation I/O**

Signal	Range	Description
X_IN	-1 <= X_IN <= 1	Input X Coordinate
Y_IN	-1 <= Y_IN <= 1	Input Y Coordinate
X_OUT	0 <= X_OUT <= Sqrt(2)	Output Magnitude * Z
PHASE_OUT	-Pi <= Phase Out <= Pi	Output Phase

The individual input vector elements, (Xin, Yin), and the output magnitude, Xout, are expressed as fixed-point 2's complement numbers with an integer width of 2 bits (1QN format). The output phase angle, Pout radians, is expressed as a fixed-point 2's complement number with an integer width of 3 bits (2QN format).

In this example the input/output width is set to 10 bits and the output Xout is scaled to compensate for the CORDIC scale factor.

Xin : "0010110101" => 00.10110101 => 0.707

Yin : "0001000000" => 00.01000000 => 0.25

Xout : "0011000000" => 00.11000000 => 0.75

Pout : "0000101011" => 000.0101011 => 0.336

Sin and Cos

When the Sin and Cos functional configuration is selected, the unit vector is rotated, using the CORDIC algorithm, by input angle, θ. This generates the output vector (Cos(θ), Sin(θ)).

The input, PHASE_IN, is limited to the range given in Table 5. Inputs outside this range produce unpredictable results. See Input/Output Data Representation for detailed information regarding CORDIC binary data formats.

An optional coarse rotation module is provided to extend the range of input angle, θ, to the full circle. For this functional configuration the coarse rotation module is selected by default but can be manually deselected by the user. See Advanced Configuration Parameters for detailed information.

The compensation scaling module is disabled for the Sin and Cos functional configuration as it is internally pre-scaled to compensate for the CORDIC scale factor.

Table 5: **Sin and Cos**

Signal	Range	Description
PHASE_IN	-Pi <= PHASE_IN <= Pi	Input Angle θ
X_OUT	-1 <= X_OUT <= 1	Output Cos(θ)
Y_OUT	-1 <= Y_OUT <= 1	Output Sin(θ)

Example 3: Sin and Cos

The input angle, Pin, is expressed as a fixed-point 2's complement number with an integer width of 3 bits (2QN format). The output vector, (Xout, Yout), is expressed as a pair of fixed-point 2's complement numbers with an integer width of 2 bits (1QN format).

In this example the input/output width is set to 10 bits.

Pin: "0001100100" => 000.1100100 => 0.781

Xout : "0010110110" => 00.10110110 => 0.711

Yout : "0010110100" => 00.10110100 => 0.703

Sinh and Cosh

When the Sinh Cosh functional configuration is selected, the CORDIC algorithm is used to move the vector (1,0) through hyperbolic *angle*, p, along the hyperbolic curve as shown in Figure 4. The hyperbolic angle represents the log of the area under the vector (X, Y) and is unrelated to a trigonometric angle. This generates the output vector (Cosh(p), Sinh(p)).

The input hyperbolic angle, PHASE_IN, is limited to the range given in Table 6. Inputs outside this range produce unpredictable results. See Input/Output Data Representation for detailed information regarding CORDIC binary data formats.

The coarse rotation module is disabled for the Sinh and Cosh functional configuration, as it does not apply to hyperbolic transformations.

The compensation scaling module is disabled for the Sinh and Cosh functional configuration, as it is internally pre-scaled to compensate for the CORDIC hyperbolic scale factor.

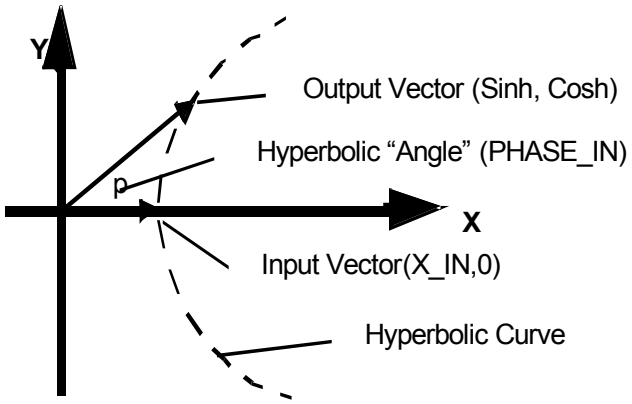

Figure 4: **Hyperbolic Sinh Cosh**

Table 6: **Sinh and Cosh**

Signal	Range	Description
PHASE_IN	-Pi/4 <= PHASE_IN <= Pi/4	Input Hyperbolic Angle
X_OUT	1 <= X_OUT < 2	Output Cosh
Y_OUT	-2 <= Y_OUT < 2	Output Sinh

Example 4: Sinh and Cosh

The input hyperbolic angle, Pin, is expressed as a fixed-point 2’s complement number with an integer width of 3 bits (2QN format). The output vector, (Xout, Yout), is expressed as a pair of fixed-point 2’s complement numbers with an integer width of 2 bits (1QN format).

In this example the input/output width is set to 10 bits.

Pin : “0001001110” => 000.1001110 => 0.781

Xout : “0100110001” => 01.00110001 => 1.191

Yout : “0010100110” => 00.10100110 => 0.648

ArcTan

When the ArcTan functional configuration is selected, the input vector (X,Y) is rotated (using the CORDIC algorithm) until the Y component is zero. This generates the output angle, Atan(Y/X).

The inputs, X_IN and Y_IN, are limited to the ranges given in Table 7. Inputs outside these ranges produce unpredictable outputs. See Input/Output Data Representation for detailed information regarding CORDIC binary data formats.

An optional coarse rotation module is provided to extend the range of inputs X and Y to the full circle. For this functional configuration the coarse rotation module is selected by default but can be manually deselected by the user. See Advanced Configuration Parameters for detailed information.

The compensation scaling module is disabled for the ArcTan functional configuration as no magnitude data is output.

The ArcTan of a zero length vector, (0,0), is indeterminate and the output angle generated by the core is undefined.

The accuracy of the output angle from the CORDIC vector translation algorithm is limited by the number of significant magnitude bits of the input vector (X, Y). See Output Quantization Error for detailed information.

Table 7: **ArcTan**

Signal	Range	Description
X_IN	-1 <= X_IN <=1	Input X Coordinate
Y_IN	-1 <= Y_IN <=1	Input Y Coordinate
PHASE_OUT	-Pi <= Phase Out <= Pi	Output Angle

Example 5: ArcTan

The input vector (Xin, Yin) is expressed as a pair of fixed-point 2's complement numbers with an integer width of 2 bits (1QN format). The output angle, Pout radians, is expressed as a fixed-point 2's complement number with an integer width of 3 bits (2QN format).

In this example, the input/output width is set to 10 bits.

Xin : "0010100000" => 00.10100000 => 0.625

Yin : "0010000000" => 00.10000000 => 0.500

Pout : "0001010110" => 000.1010110=> 0.672

ArcTanh

When the ArcTanh functional configuration is selected, the CORDIC algorithm is used to move the input vector (X,Y) along the hyperbolic curve (Figure 5) until the Y component reaches zero. This generates the hyperbolic "angle," Atanh(Y/X). The hyperbolic *angle* represents the log of the area under the vector (X,Y) and is unrelated to a trigonometric angle.

The inputs, X_IN and Y_IN, are limited to the ranges given in Table 8. Inputs outside these ranges produce unpredictable outputs. Additionally, Y_IN must be less than or equal to (4/5 * X_IN) or the CORDIC algorithm does not converge. See Input/Output Data Representation for detailed information about CORDIC binary data formats.

The coarse rotation module is disabled for the ArcTanh functional configuration, as it does not apply to hyperbolic transformations.

The compensation scaling module is disabled for the ArcTanh functional configuration as no output magnitude data is output.

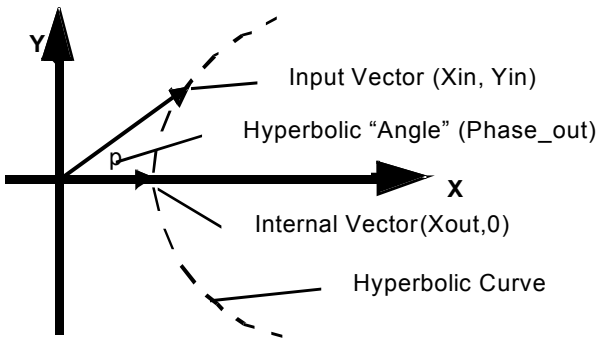

Figure 5: **Hyperbolic ArcTan**

Table 8: **ArcTanh**

Signal	Range	Description
X_IN	0 < X_IN <2	Input X Coordinate
Y_IN	-2 <= Y_IN < 2 -X_IN * 4/5 <= Y_IN <= X_IN * 4/5	Input Y Coordinate
PHASE_OUT	-Pi/2 <= Phase Out <= Pi/2	Output Hyperbolic Angle

Example 6: ArcTanh

The input vector, (Xin, Yin), is expressed as a pair of fixed-point 2's complement numbers with an integer width of 2 bits (1QN format). The output, Pout, is expressed as a fixed-point 2's complement number with an integer width of 3 bits (2QN format).

In this example, the input/output width is set to 10 bits.

Xin : "0001100101" => 00.01100101 => 0.395

Yin : "0001100101" => 00.01100101 => 0.395

Pout : "0001110001" => 000.1110001=> 0.883

Square Root

When the square root functional configuration is selected a simplified CORDIC algorithm is used to calculate the positive square root of the input.

The input, X_IN, and the output, X_OUT, are always positive and are both expressed as either unsigned fractions or unsigned integers.

When data format is set to Unsigned Fraction, X_IN is limited to the range: 0 <= X_IN < +2.

When data format is set to Unsigned Integer, X_IN is limited to the range: 0 <= X_IN < 2**Input Width, and the output width is determined automatically based on the input width.

See Input/Output Data Representation for detailed information regarding CORDIC binary data formats.

The coarse rotation module is disabled because coarse rotation is not required for the Square Root functional configuration.

The compensation scaling module is disabled because no output compensation is required for the Square Root functional configuration.

Table 9: **Square Root**

Signal	Range	Description
X_IN	Unsigned Fraction: 0 <= X_IN < +2 Unsigned Integer: 0 <= X_IN < 2**Input Width	Input X Value
X_OUT	Unsigned Fraction: 0 <= X_OUT < +2 Unsigned Integer: 0 <= X_OUT < 2**[int(Input Width/2)+1]	Output Square Root

Example 7a: Square Root - Unsigned Fraction

The input, Xin, and output, Xout, are expressed as an unsigned fixed-point number with an integer width of 1 bit.

In this example the input/output width is set to 10 bits.

Xin : "0000100000" => 0.000100000 => 1/16

Xout : "0010000000" => 0.010000000 => 1/4

Example 7b: Square Root - Unsigned Integer

The input, Xin, is expressed as an unsigned integer. The output, Xout, is expressed as an unsigned integer. In this example the input width is set to 10 bits so the output width is automatically set to 6 bits.

Xin : "0000100000" => 32

Xout : "000110" => 6

Architectural Configuration

Two architectural configurations are available for the CORDIC core: Parallel, with single-cycle data throughput and large silicon area, and Word Serial, with multiple-cycle throughput and a smaller silicon area.

Word Serial Architectural Configuration

The CORDIC algorithm requires approximately one shift-addsub operation for each bit of accuracy. A CORDIC core implemented with the word serial architectural configuration, implements these shift-addsub operations serially, using a single shift-addsub stage and feeding back the output.

A word serial CORDIC core with N bit output width has a latency of N cycles and produces a new output every N cycles. The implementation size of a word serial CORDIC core is directly proportional to the internal precision.

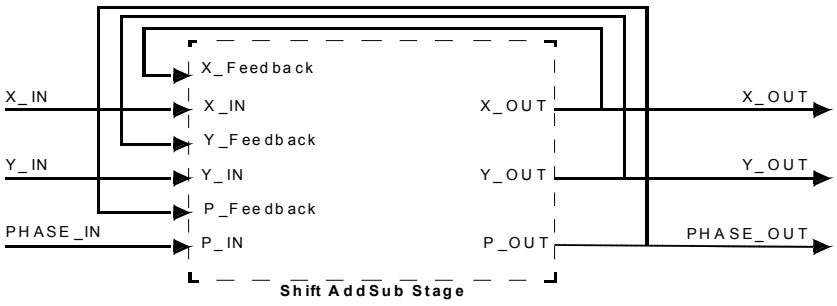

Figure 6: **Word Serial Architecture Configuration**

Parallel Architectural Configuration

The CORDIC algorithm requires approximately one shift-addsub operation for each bit of accuracy. A CORDIC core with a parallel architectural configuration implements these shift-addsub operations in parallel using an array of shift-addsub stages.

A parallel CORDIC core with N bit output width has a latency of N cycles and produces a new output every cycle. The implementation size of a parallel CORDIC core is directly proportional to the internal precision times the number of iterations.

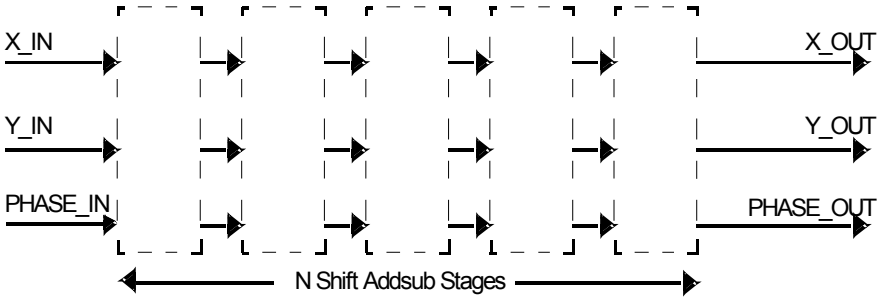

Figure 7: **Parallel Architectural Configuration**

Input/Output Data Representation

Data Signals

The Data Signals are: X_IN, Y_IN, X_OUT and Y_OUT.

For Functional Configurations Rotate, Translate, Sin, Cos and Atan the Data Signals are represented using fixed-point 2's complement numbers with an integer width of 2 bits. The integer width is fixed regardless of the word width; the remainder of the bits are used for the fractional portion of the number. Using the Q Numbers Format this representation is described as 1QN where N = word width - 2. It can also be described as Fix(N+2)_N using the System Generator Fix format.

Input data signals, X_IN and Y_IN, must be in the range: -1 <= input data signal <= 1. Input data outside this range produces undefined results.

Using a 10-bit word width, +1 and -1 are represented as:

"0100000000" => 01.00000000 => +1.0

"1100000000" => 11.00000000 => - 1.0

For the Square Root Functional Configuration, the Data Signals, X_IN and X_OUT, are both represented in either Unsigned Fractional or Unsigned Integer data format.

The input data signal, X_IN, must be in the range: 0 <= X_IN < +2 when data format is set to Unsigned Fraction or in the range 0 <= X_IN < 2**Input Width when data format is set to Unsigned Integer.

When Unsigned Fractional data format has been selected the Data Signals are represented using a unsigned fixed-point number with an integer with of 1 bit. The integer width is fixed and the remainder of the word is used to represent the fractional portion of the number. Using the System Generator Fix format this representation is described as UFix(N+1)_N, where is the number of fractional bits being used and is defined as N = word width -1. The Q Number format is used to represent signed 2's complement numbers and is therefore not suitable to describe the representation format used by the square root function.

Phase Signals

The Phase Signals are: PHASE_IN and PHASE_OUT. The phase signals are always represented using a fixed-point 2's complement number with an integer width of 3 bits. As with the data signals the integer width is fixed and any remaining bits are used for the fractional portion of the number. The Phase Signals require an increased integer width to accommodate the increased range of values they must represent when the Phase Format is set to Radians.

When Phase Format is set to Radians, PHASE_IN must be in the range: -Pi <= (PHASE_IN) <= Pi. PHASE_IN outside this range produce undefined results.

In 2Q7, or Fix10_7, format values, +Pi and -Pi are represented as:

"01100100100" => 011.00100100 => +3.14

"10011011100" => 100.11011100 => - 3.14

When Phase Format is set to Scaled Radians PHASE_IN must be in the range: -1 <= (PHASE_IN) <= +1. PHASE_IN outside this range produce undefined results.

In 2Q7, or Fix10_7 format values, +1 and -1 are represented as:

"0010000000" => 001.0000000 => +1.0

"1110000000" => 111.0000000 => - 1.0

Q Numbers Format

An XQN format number is an 1+X+N bit 2's complement binary number; a sign bit followed by X integer bits followed by an N bit mantissa (fraction). XQN format can be used to express numbers in the range (-2^X) to ($2^X - 2^{(-N)}$). An equivalent notation using the System Generator Fix format, defined as Fix*word_length_fractional_length*, would be Fix(1+X+N)_N.

A number using Q15 format is equivalent to a number using Fix16_15 representation, and a number in 1Q15 format is equivalent to a number using Fix17_15 representation.

Table 10 and Table 11 contain examples of XQN Format Numbers

Table 10: **1QN Format Data: Example of a 1Q7 (or Fix9_7) Format Number**

	(Sign) Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
+1	0	1	0	0	0	0	0	0	0
-1	1	1	0	0	0	0	0	0	0
+Pi/4	0	0	1	1	0	0	1	0	0
-Pi/4	1	1	0	0	1	1	0	1	1
			Fractional Bits						

Table 11: **2QN Format Phase: Example of a 2Q6 (or Fix9_6) Format Number**

	(Sign) Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	BIt 0
+1	0	0	1	0	0	0	0	0	0
-1	1	1	1	0	0	0	0	0	0
+Pi	0	1	1	0	0	1	0	0	1
-Pi	1	0	0	1	1	0	1	1	1
				Fractional Bits					

Mapping Different Data Formats

Rotate, Translate, Sin, Cos and Atan Functional Configurations

For Functional Configurations Rotate, Translate, Sin, Cos and Atan it is possible to map alternative Data Signal formats to the fixed integer width fractional number used by the CORDIC core.

When the input and output width differ, care must be taken to re-interpret the CORDIC output.

The following example develops Example 2 from the Vector Translation section to demonstrate a possible remapping.

Example 8a:

The Vector Translation function determines the magnitude and phase angle of a given input vector (X_IN, Y_IN). The input and output width is set to 10 bits. The standard CORDIC data representation is Fix10_8, the alternative format being mapped onto the CORDIC's input is Fix10_1.

X_IN value: "0010110101"

Table 12: **Example 8: Mapping an Alternative Data Format onto the X_IN input**

	Sign Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Decimal Value
Binary Value	0	0	1	0	1	1	0	1	0	1	
Fix10_8 weighting	-2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	0.707
Fix10_1 weighting	-2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	90.5

Y_IN value: "0001000000"

Table 13: **Example 8: Mapping an Alternative Data Format onto the Y_IN input**

	Sign Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Decimal Value
Binary Value	0	0	1	0	0	0	0	0	0	0	
Fix10_8 weighting	-2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	0.25
Fix10_1 weighting	-2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	32

Below, MATLAB® software is used to generate the expected results. Firstly the magnitude and phase angle for the standard CORDIC input format 1Q8, or Fix10_8 is generated:

>> a=0.707+0.25j

>> magnitude = abs(a)

magnitude = **0.7499**

>> phase_angle = angle(a)

phase_angle = **0.3399**

Secondly using the mapped input format, 9Q1 or Fix10_1:

>> b=90.5+32j

>> magnitude = abs(b)

magnitude = **95.9909**

>> phase_angle = angle(b)

phase_angle = **0.3399**

The CORDIC output is:

X_OUT value: "0011000000"

PHASE_OUT value: "0000101011"

Table 14 and Table 15 demonstrate the CORDIC's output value being interpreted using the two data representation formats.

Table 14: **Example 8: X_OUT Interpretation**

	Sign Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Decimal Value
Binary Value	0	0	1	1	0	0	0	0	0	0	
Fix10_8 weighting	-2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	0.75
Fix10_1 weighting	-2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	96

Table 15: **Example 8: PHASE_OUT Interpretation**

	Sign Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Decimal Value
Binary Value	0	0	0	0	1	0	1	0	1	1	
Fix10_7 weighting	-2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	0.336

Example 8b:

If the output width is less than the input width, the CORDIC reduces the fractional width of the result. When the data output, X_OUT, is being re-interpreted to an alternative data format, the value must be scaled appropriately.

The following table demonstrates how the resulting decimal value may change when the output width is reduced to 8 bits.

Table 16: **Example 8b: X_OUT Interpretation with Reduced Output Width**

	Sign Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Decimal Value
Binary Value	0	0	1	1	0	0	0	0	
Fix8_6 weighting	-2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	0.75
Fix8_0 weighting	-2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	48

A similar situation arises when the output width is greater than the input width. In this circumstance, the CORDIC increases the fractional width of the result. When the data output is being re-interpreted to a data format with no fractional bits this results in an increased magnitude. This output then needs to be scaled appropriately.

Square Root Functional Configuration

For the Square Root Functional Configuration it is also possible to map other data formats onto the CORDIC's data format but it may be necessary to re-interpret and scale the output.

The following example modifies Example 7a from the Square Root section.

Example 9:

X_IN value: "00001000"

Table 17: **Example 9: Mapping an Alternative Data Format onto the X_IN input**

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Decimal Value
Binary Value	0	0	0	0	1	0	0	0	
UFix8_7 weighting	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	0.0625
UFix8_1 weighting	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	4
UFix8_0 weighting	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	8

The expected output values for each input format are as follows:

UFix8_7 format: sqrt(0.0625) = **0.25**

UFix8_1 format: sqrt(4) = **2**

UFix8_0 format: sqrt(8) = **2.8284**

The CORDIC output is:

X_OUT value: "00100000"

Table 18 demonstrates the output value directly interpreted in each of the input formats.

Table 18: **X_OUT Direct Interpretation**

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Decimal Value
Binary Value	0	0	1	0	0	0	0	0	
UFix8_7 weighting	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	0.25
UFix8_1 weighting	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	16
UFix8_0 weighting	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	32

Table 18 shows that if the output value is directly interpreted in the alternative data format the wrong decimal value is determined. The output value must be scaled correctly.

The output scaling is determined as follows.

The CORDIC core calculates the square root of input values in the range 0 <= X_IN < 2.

$$Y = \sqrt{X}$$

The alternative data format represents values in the range 0<= X_IN < 2^{N+1} and we wish to calculate:

$$Y_{alt} = \sqrt{X_{alt}}$$

Interpreting X_{alt} using the standard CORDIC data format scales the input by 2^{-N}, as shown in Table 17.

$$Y = \sqrt{2^{-N} \cdot X_{alt}}$$

$$Y = 2^{(-N)/2} \cdot \sqrt{X_{alt}}$$

As Table 18 shows directly re-interpreting the CORDIC output in the alternative data formats results in an incorrect decimal value. This is due the scale factor introduced by the remapping of the input and the square root function. This scaling factor introduced is shown above, $2^{-N/2}$.

The corrected results are shown below:

UFix8_1 weighting: $16/2^{(6/2)}$ = **2**

UFix8_0 weighting: $32/2^{(7/2)}$ = **2.8284**

When N is even the scaling factor is an integer power of two. This can be applied by simply right shifting the CORDIC output, X_OUT, by N/2. The example using the UFix8_1 format demonstrates this with a scaling factor of $2^{-3} = 1/8$.

When N is odd the scaling factor is not an integer power of two. This introduces an additional output scaling factor of $\sqrt{2}$. The example using UFix8_0 demonstrates this with a scaling factor of $2^{-7/2} = 2^{-3.5}$.

This could be implemented by first scaling the output by a right shift of 4 and then multiplying by $\sqrt{2}$. A more efficient way would be to translate the $\sqrt{2}$ scaling to the input of the square root function.

This is demonstrated below where $2^{-N/2}=2^{-M-(1/2)}$.

$$Y = 2^{(-M-1/2)} \cdot \sqrt{X_{alt}}$$

$$Y = 2^{-M} \cdot \sqrt{2^{-1} \cdot X_{alt}}$$

The scaling becomes a simple divide by 2, or right shift, of the input, X_IN, before applying it to the square root function. Followed by scaling the output, X_OUT, by 2^{-M}.

An input value of 8 is used for the UFix8_0 formatting example. Divided by 2 this gives 4. Table 17 shows that 4 maps to 1/32 in the CORDIC input range.

sqrt(1/32) = 0.17678 = 0.0010110

Table 18 shows that the CORDIC output value, 0.0010110, maps to a decimal value of 22 in UFix8_0 formatting. Applying the output scaling of 2^{-3}, or 1/8, gives **2.75**. The loss in accuracy is due to representing sqrt(1/32) using only 8 bits. If the full accuracy result is used and then re-interpreted to the alternative data format (Fix8_0) and then scaled, the correct result is obtained; for example:

sqrt(1/32) * 2^7 * 2^{-3} = **2.8284**

CORE Generator GUI and Parameters

The CORDIC Graphical User Interface (GUI) contains three pages for configuring the core and two information tabs:

Tab 1 & 2: IP Symbol and Implementation Details

The IP Symbol tab illustrates the core pinout.

The Implementation Details tab displays the core latency and resource usage. The block RAM and Multiplier/XtremeDSP Slice resources are only utilized when Compensation Scaling is selected.

Page 1

Used to configure the functional selection and architecture of the CORDIC core.

- **Component Name**: Used as the base name of the output files generated for the core. Names must begin with a letter and be composed from the following characters: a to z, 0 to 9, and "_."
- **Functional Selection:** The following functional selections are available: Rotate, Sin and Cos, ArcTan, Square Root, Translate, Sinh and Cosh and ArcTanh. See the Functional Description section for detailed information on each of the supported functions. In general, X_IN, Y_IN, X_OUT and Y_OUT express signed binary numbers of 1QN format and PHASE_IN and PHASE_OUT express signed binary numbers of 2QN format. When Square Root is selected, two new data formats are available: Unsigned Integer and Unsigned Fraction. For details about CORDIC binary data formats see Input/Output Data Representation.
- **Architectural Configuration:** Two architectural configurations are available for the CORDIC core, Parallel and Word Serial. Refer to Architectural Configuration for more details.
- **Pipelining Mode:** The CORDIC core provides three pipelining modes: None, Optimal, and Maximum. The choice of pipelining mode is based on the selection of Functional Configuration and Architectural Configuration. Unavailable pipelining modes are greyed out in the GUI.
  - **None**: the CORDIC core is implemented without pipelining.
  - **Optimal**: the CORDIC core is implemented with as many stages of pipelining as possible without using any additional LUTs.
  - **Maximum**: the CORDIC core is implemented with a pipeline after every shift-add sub stage.

Page 2

Used to configure the phase and magnitude data formats, rounding mode and input-outputs.

- **Data Format**: The CORDIC core provides three formats for expressing the X and Y components of data samples:
  - **Signed Fraction**: Default setting. The X and Y inputs and outputs are expressed as fixed-point 2's complement numbers with an integer width of 2 bits. Example: "11100000" represents the value -0.5.
  - **Unsigned Fraction**: The X and Y inputs and outputs are expressed as unsigned fixed-point number with an integer with of 1 bit.
    Available only for Square Root functional configuration. Example: "11100000" represents the value +1.75.
  - **Unsigned Integer**: The X and Y inputs and outputs express unsigned integers.
    Available only for Square Root functional configuration. Example: "11100000" represents the value +224.
- **Phase Format:** The CORDIC core provides two Phase Format options:
  - **Radians**: The phase is expressed as a fixed-point 2's complement numbers with an integer width of 3 bits, in radian units.
    Example: "01100000" represents the value 3.0 radians.
  - **Scaled Radians**: The phase is expressed as fixed-point 2's complement numbers with an integer width of 3 bits, with pi-radian units. One scaled-radian equals Pi * 1 radians.
    Example: "11110000" represents the value -0.5 * Pi radians.

See Input/Output Data Representation for detailed information about CORDIC binary data formats.

- **Input / Output Options:** The CORDIC core provides four input / output common configuration options.
  - **Input Width**: Input Width controls the widths of the input ports, X_IN, Y_IN and PHASE_IN. The Input Width can be configured in the range 8 to 48 bits.
  - **Register Inputs**: Selects if the input signals X_IN, Y_IN, PHASE_IN are registered.
  - **Output Width**: Output Width controls the widths of the output ports, X_OUT, Y_OUT, PHASE_OUT. The Output Width can be configured in the range 8 to 48 bits.
  - **Register Outputs**: Selects if the output signals, X_OUT, Y_OUT, PHASE_OUT are registered.
- **Round Mode:** The CORDIC core provides four rounding modes. Table 19 illustrates the behavior of the different Rounding modes.
  - **Truncate**: The X_OUT, Y_OUT, and PHASE_OUT outputs are truncated.
  - **Positive Infinity**: The X_OUT, Y_OUT, and PHASE_OUT outputs are rounded such that 1/2 is rounded up (towards positive infinity). It is equivalent to the MATLAB function floor(x+0.5).
  - **Pos Neg Infinity**: The outputs X_OUT, Y_OUT, and PHASE_OUT are rounded such that 1/2 is rounded up (towards positive infinity) and -1/2 is rounded down (towards negative infinity). It is equivalent to the MATLAB function round(x).
  - **Nearest Even**: The X_OUT, Y_OUT, and PHASE_OUT outputs are rounded toward the nearest even number such that a 1/2 is rounded down and 3/2 is rounded up.

Table 19: **Rounding Modes**

	Truncate	Pos Neg Infinity	Positive Infinity	Nearest Even
1.50	1	2	2	2
1.00	1	1	1	1
0.50	0	1	1	0
0.25	0	0	0	0
0.00	0	0	0	0
- 0.25	-1	0	0	0
- 0.50	-1	-1	0	-1
- 0.75	-1	-1	-1	-1

Page 3

Provides options for advanced configuration parameters (Coarse Rotation, Iterations, Internal Precision, and Compensation Scaling) and optional control signals.

- **Advanced Configuration Parameters**
  - **Iterations:** Controls the number of internal add-sub iterations to perform.When Iterations is set to zero, the number of iterations performed is determined by the required accuracy of the output. By default, Iterations is set to zero, thus the number of iterations is automatically determined.
  - **Precision**: Configures the internal precision of the add-sub iterations. When Precision is set to zero, internal precision is determined automatically based on the required accuracy of the output and the number of internal iterations. By default, Precision is set to zero, thus the internal precision is automatically determined. When Precision is set to (input width + output width + $\log_2$(output_width)) the output phase is precise to the full output width regardless of input magnitude. However, the output phase accuracy is still limited by the OQEIQ component of Output Quantization Error and by the number of Iterations of the CORDIC Micro-Rotation block.

- **Coarse Rotation**: Controls the instantiation of the coarse rotation module. Instantiation of the coarse rotation module is the default for the following functional configurations: Vector rotation, Vector translation, Sin and Cos, and ArcTan. If Coarse Rotation is turned off for these functions then the input/output range is limited to the first quadrant (-Pi/4 to + Pi/4). Coarse rotation is not required for the Sinh and Cosh, ArcTanh, and Square Root configurations. The standard CORDIC algorithm operates over the first quadrant. Coarse Rotation extends the CORDIC operational range to the full circle by rotating the input sample into the first quadrant and inverse rotating the output sample back into the appropriate quadrant.
- **Compensation Scaling:** Controls the compensation scaling module used to compensate for CORDIC magnitude scaling. CORDIC magnitude scaling affects the Vector Rotation and Vector Translation functional configurations, and does *not* affect the Sin, Cos, Sinh, Cosh, ArcTan, ArcTanh and Square Root functional configurations. For the latter configurations, compensation scaling is set to No Scale Compensation. CORDIC magnitude scaling is a side effect of the CORDIC algorithm. The magnitude outputs, X and Y, are generated scaled by the CORDIC scale factor, Z_n. The compensation scaling module compensates for the effect of CORDIC magnitude scaling by scaling the outputs, X and Y, by $1/Z_n$.
  - **No Scale Compensation**: The outputs X and Y are not compensated and are generated, scaled by the ratio Z_n.
  - **LUT Based**: The outputs X and Y are compensated using a LUT-based Constant Coefficient Multiplier.
  - **BRAM:** The outputs X and Y are compensated using a block RAM-based Constant Coefficient Multiplier.
  - **Embedded Multiplier**: The outputs X and Y are compensated using the XtremeDSP™ Slice or embedded multiplier depending on the family of part chosen in the CORE Generator project options.

- **Optional Pin Selection**
  - **Control Signals:** ND, RDY, SCLR, and CE control signals are optional. The presence of the CLK and RFD control signals are determined based on the selected Architectural Configuration, Pipelining Mode, Register Inputs, and Register Outputs.
  - **Output Signals**: X_OUT, Y_OUT and PHASE_OUT are optional. The default states of these signals are determined based on the selected functional configuration but can be manually overridden by the user. Refer to Table 2 for more information.

System Generator GUI and Parameters

This section details the parameters that differ from the CORE Generator GUI. See CORE Generator GUI and Parameters for more detailed information about all other parameters.

Page 1

When Word Serial architectural configuration is selected, the System Generator block operates at the system sample period. The core's ND and RFD control signal should be used to determine and control when a new input sample is applied to the core. Refer to Control Signals and Timing for more details on the use of the core control signals.

Please refer to Page 1 of the CORE Generator GUI and Parameters section for details on each of the parameters.

Page 2

The input width parameter has been abstracted and its value is taken from the input ports of the System Generator block. The input and output registers are always enabled.

Please refer to Page 2 of the CORE Generator GUI and Parameters section for details on each of the parameters.

Page 3

The System Generator block renames CE to EN and SCLR to RST.

Please refer to Page 3 of the CORE Generator GUI and Parameters section for details on each of the parameters.

Implementation

Please refer to the System Generator documentation for information regarding the FPGA Area Estimation parameter.

Control Signals and Timing

The following section describes the control signals used by the CORDIC core. All control signals are synchronous to the rising edge of CLK.

A timing diagram for a CORDIC core with a Word Serial Architectural Configuration is shown in Figure 8.

A timing diagram for a CORDIC core with a Parallel Architectural Configuration is shown in Figure 9.

CLK

All core operations are synchronous with the rising edge of the CLK (Clock) input.

CLK is mandatory when the core is pipelined or Registered Inputs/Outputs have been selected. Otherwise, CLK is not present.

ND

When the ND (New Data) input is high, the input data is sampled on the same rising clock edge. ND is ignored if CE is low or if RFD is low.

ND is mandatory when Word Serial architecture has been selected. Otherwise, ND is optional.

RFD

RFD (Ready for Data) indicates that the core is ready to sample new input data. The RFD signal is set high upon startup or during reset.

RFD is mandatory when Word Serial architecture has been selected. Otherwise, RFD is not present.

RDY

The RDY (Ready) output signals that a new valid data sample is present on the Data Output Ports. RDY is pulsed high on the first clock cycle of valid data at the output. The RDY signal is set low upon startup or during reset.

RDY is mandatory when Word Serial architecture has been selected. Otherwise, RDY is optional.

SCLR

When SCLR is asserted (High), all the core flip-flops are synchronously initialized. The core remains in this state until SCLR is deasserted. SCLR is optional. SCLR has priority over CE.

CE

When CE (Clock Enable) is Low, all the synchronous inputs are ignored and the core remains in its current state. CE is optional.

Word Serial Timing

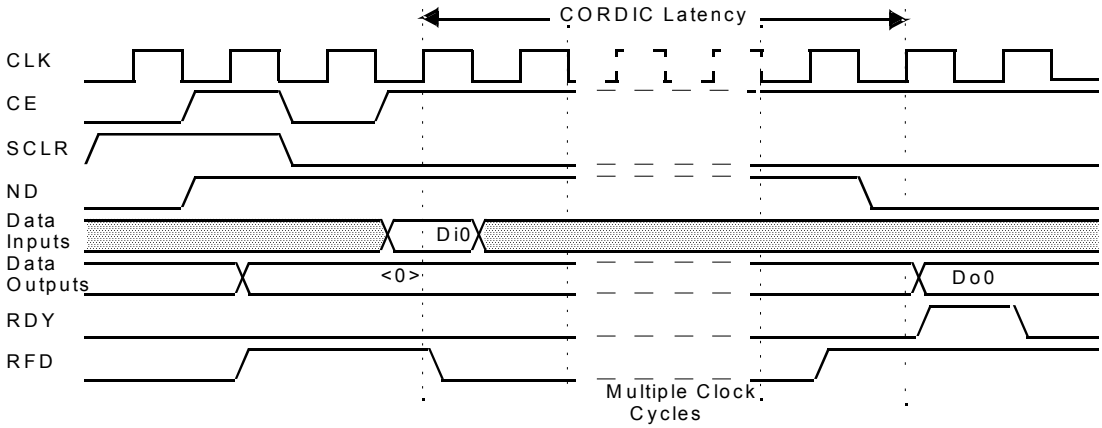

Figure 8: **Control Signal Timing Diagram (Word Serial Architecture)**

Parallel Architecture Timing

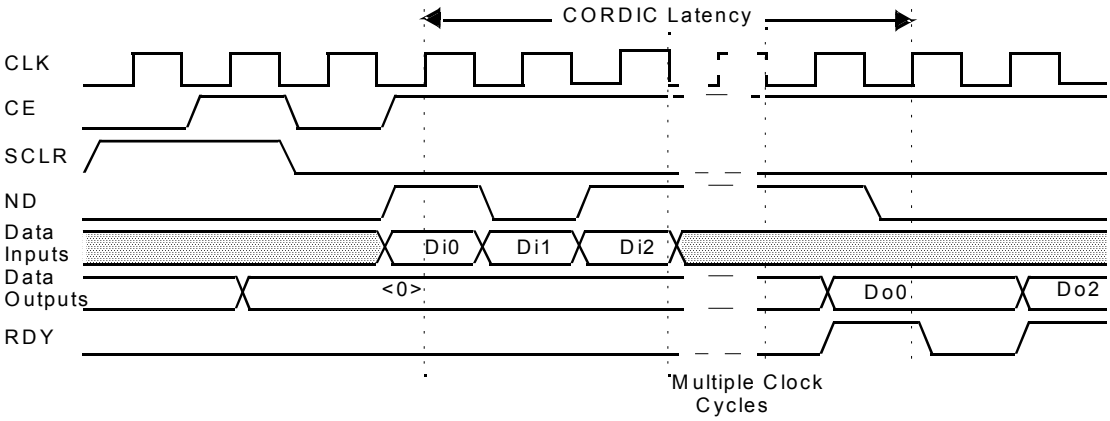

Figure 9: **Control Signal Timing Diagram (Parallel Architecture)**

Migrating to CORDIC v4.0 from CORDIC v3.0

The CORE Generator core update functionality may be used to update an existing XCO file from CORDIC v3.0 to CORDIC v4.0.

For more information on this feature, see the CORE Generator documentation.

Parameter changes

CORDIC v4.0 does not support ACLR, Synchronization Enable or Create RPM parameters.

The Compensation Scaling options have changed from:

No_Scale_Compensation, CCM_Scale_Compensation and Block_Multiplier

To:

No_Scale_Compensation, LUT_Based, BRAM_based and Embedded_Multiplier.

Where CCM_Scale_Compensation = LUT_Based and Block_Multiplier = Embedded_Multiplier and BRAM_based is a new option. All other parameters are unchanged.

Port changes

CORDIC v4.0 does not support an ACLR control port. All other port names and widths remain the same.

Performance and Resource Utilization

Table 20 through to Table 21 performance and resource usage information for a number of different core configurations.

The maximum clock frequency results were obtained by double-registering input and output ports to reduce dependence on I/O placement. The inner level of registers used a separate clock signal to measure the path from the input registers to the first output register through the core.

The resource usage results do not include the above "characterization" registers and represent the true logic used by the core. LUT counts include SRL16s or SRL32s (according to device family).

The map options used were: `"map -ol high"`

The par options used were: `"par -ol high"`

Table 20 contains characterization data for Virtex-5 using a XC5VSX35T-1FF665. The results have been generated with automatically determined Iterations and Precision, Coarse Rotation, no Compensation Scaling and Maximum Pipelining.

Table 20: **Virtex-5 Characterization Data**

Function	Architecture	Input/Output Width	Round Mode	LUT-FF pairs	Maximum Clock Frequency (Mhz)
Rotate	Word Serial	16	Truncate	625	220
		32	Nearest Even	1221	178
		48	Truncate	1863	156
	Parallel	16	Truncate	1179	339
		32	Nearest Even	4023	262
		48	Truncate	8524	227
Translate	Word Serial	16	Truncate	501	220
		32	Nearest Even	995	184
		48	Truncate	1490	156
	Parallel	16	Truncate	1179	339
		32	Nearest Even	4023	262
		48	Truncate	8416	213

Table 20: **Virtex-5 Characterization Data**

Function	Architecture	Input/Output Width	Round Mode	LUT-FF pairs	Maximum Clock Frequency (Mhz)
Cos and Sin	Word Serial	16	Truncate	481	235
		32	Nearest Even	983	184
		48	Truncate	1376	149
	Parallel	16	Truncate	1093	366
		32	Nearest Even	3812	296
		48	Truncate	8126	220
ArcTanh	Word Serial	16	Truncate	363	220
		32	Nearest Even	696	178
		48	Truncate	1000	149
	Parallel	16	Truncate	1185	339
		32	Nearest Even	3999	248
		48	Truncate	8274	192
Square Root	Parallel	16	Truncate	461	339
		32	Nearest Even	1550	283
		48	Truncate	2860	241

Table 21 contains characterization data for Spartan 3ADSP using a XC3SD1800A-4FG676. The results have been generated using the same default parameters as for Virtex-5.

Table 21: **Spartan-3A DSP Characterization Data**

Function	Architecture	Input/Output Width	Round Mode	Slices	Maximum Clock Frequency (Mhz)
Rotate	Word Serial	16	Truncate	403	102
		32	Nearest Even	796	78
		48	Truncate	1292	62
	Parallel	16	Truncate	649	180
		32	Nearest Even	2122	134
		48	Truncate	4352	110
Translate	Word Serial	16	Truncate	331	110
		32	Nearest Even	667	78
		48	Truncate	1090	62
	Parallel	16	Truncate	681	149
		32	Nearest Even	2189	110
		48	Truncate	4147	94

Table 21: **Spartan-3A DSP Characterization Data *(Cont'd)***

Function	Architecture	Input/Output Width	Round Mode	Slices	Maximum Clock Frequency (Mhz)
Cos and Sin	Word Serial	16	Truncate	336	102
		32	Nearest Even	653	86
		48	Truncate	1066	62
	Parallel	16	Truncate	575	180
		32	Nearest Even	1983	134
		48	Truncate	4147	110
ArcTanh	Word Serial	16	Truncate	291	94
		32	Nearest Even	501	78
		48	Truncate	786	62
	Parallel	16	Truncate	693	134
		32	Nearest Even	2178	102
		48	Truncate	4363	89
Square Root	Parallel	16	Truncate	255	172
		32	Nearest Even	992	134
		48	Truncate	1939	110

References

1. Volder, J., "*The CORDIC Trigonometric Computing Technique*" IRE Trans. Electronic Computing, Vol. EC-8, Sept. 1959, pp330-334
2. Walther, J.S., "*A Unified Algorithm for Elementary Functions*," Spring Joint computer conf., 1971, proc., pp379-385

Support

Xilinx provides technical support for this LogiCORE IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

Refer to the IP Release Notes Guide (XTP025) for further information on this core. There is a link to all the DSP IP and then to each core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for each core. The following information is listed for each version of the core:

- New Features
- Bug Fixes
- Known Issues

Ordering Information

This LogiCORE IP module is included at no additional cost with the Xilinx ISE Design Suite software and is provided under the terms of the Xilinx End User License Agreement. Use the CORE Generator software included with the ISE Design Suite to generate the core. For more information, please visit the core page.

Please contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE modules and software. Information about additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

Revision History

Date	Version	Revision
03/28/03	1.0	Revision History added to document.
03/28/03	1.1	Updated Hyperbolic Transformations and PiRadian format.
03/28/03	2.0	Improved parameterization, new rounding modes, and new data formats.
05/21/04	3.0	Added Virtex-4 support and update to v6.2i of Xilinx CORE Generator system.
04/28/05	3.1	Updated to indicate support for Spartan-3E and Xilinx ISE software v7.1i.
04/24/09	4.0	Updated to include support for Virtex-6, Virtex-5, Spartan-6 and Spartan-3A/3A DSP devices.
03/01/11	4.1	Updated to include support for Virtex-7 and Kintex-7 and Xilinx ISE Design Suite 13.1.

Notice of Disclaimer