CoreCORDIC CORDIC RTL Generator

DirectCore

Product Summary

Intended Use

- COordinate Rotation DIgital Computer (CORDIC) Rotator Function for Actel FPGAs

Key Features

- Vector Rotation – Conversion of Polar Coordinates to Rectangular Coordinates
- Vector Translation – Conversion of Rectangular Coordinates to Polar Coordinates
- Sine and Cosine Calculation
- Vector (*X*, *Y*) Magnitude $\sqrt{X^2 + Y^2}$ and Phase (arctan[*X*/*Y*]) Calculation
- 8-Bit to 48-Bit Configurable Word Size
- 8 to 48 Configurable Number of Iterations
- Parallel Pipelined Architecture for the Fastest Calculation
- Bit-Serial Architecture for the Smallest Area
- Word-Serial Architecture for Moderate Speed and Area
- Word Parallel Data I/Os

Supported Families

- Fusion
- ProASIC®3/E
- ProASIC^PLUS ®
- Axcelerator®
- RTAX-S
- SX-A
- RTSX-S

Core Deliverables

- Full Version
  - CoreCORDIC RTL Generator. Generates User-Defined CORDIC Model and Test Harness. Fully Supported in the Actel Libero® Integrated Design Environment (IDE)
- Evaluation Version
  - Supports CORDIC Engine and Test Harness Generation with Limited Parameters. Fully Supported in Libero IDE.

Synthesis and Simulation Support

- Libero IDE
- Synthesis: Synplicity®, Synopsys® (Design Compiler/FPGA Compiler), Exemplar™
- Simulation: OVI-Compliant Verilog Simulators and Vital-Compliant VHDL Simulators.

Table of Contents

General Description

CoreCORDIC is an RTL generator that produces an Actel FPGA–optimized CORDIC engine. The CORDIC algorithm by J. Volder provides an iterative method of performing vector rotations using shifts and adds only. The articles listed in "References" on page 12 present a detailed description of the algorithm.

Depending on the configuration defined by the user, the resulting module implements pipelined parallel, word-serial, or bit-serial architecture in one of two major modes: rotation or vectoring. In rotation mode, the CORDIC rotates a vector by a specified angle. This mode is used to convert polar coordinates to Cartesian

coordinates, for general vector rotation, and also to calculate sine and cosine functions (see Figure 1).

"Appendix I" on page 14 presents mathematical coordinate conversion formulae, and "Appendix II" on page 15 describes examples of a few of the most used CORDIC modes.

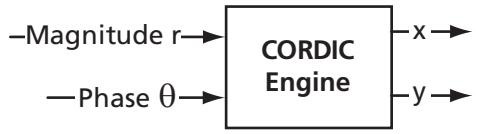

Figure 1 • **CORDIC Engine in Rotation Mode**

In vectoring mode, the CORDIC rotates the input vector towards the *x* axis while accumulating a rotation angle. Vectoring mode is used to convert Cartesian vector coordinates to polar coordinates; i.e., to calculate the magnitude and phase of the input vector (Figure 2).

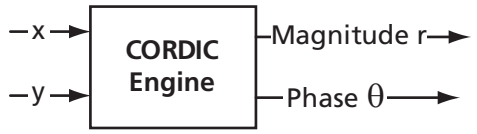

Figure 2 • **CORDIC Engine in Vectoring Mode**

The CORDIC results, such as *x, y,* and *r,* are scaled by the inherent processing gain, *K,* which depends on number of iterations and converges to about 1.647 after a few iterations. The gain is constant for a given number of iterations. When performing Cartesian/polar coordinate conversion, the CORDIC computes the results shown in EQ 1 and EQ 2 in rotation mode.

$$X = K \cdot r \cdot \cos x\theta \quad \text{EQ 1}$$

$$Y = K \cdot r \cdot \sin\theta \quad \text{EQ 2}$$

EQ 3 and EQ 4 show the CORDIC results in vectoring mode.

$$r = K \cdot \sqrt{X^2 + Y^2} \quad \text{EQ 3}$$

$$\theta = \arctan(Y/X) \quad \text{EQ 4}$$

The gain can be compensated for elsewhere in many applications when the system includes the CORDIC engine. To assist a user in doing so, the CoreCORDIC software computes the precise value of the gain and displays it on a screen. In the cases when only relative magnitude is of importance—for example, spectrum analysis and AM demodulation—the constant gain can be neglected. When calculating sine/cosine, the CORDIC gets initialized with a constant reciprocal value of the processing gain $r = 1/K$.

EQ 1 and EQ 2 become

$$X = \cos\theta$$

$$Y = \sin\theta$$

Thus, the gain does not impact the sine/cosine results or the phase output.

To perform the conversions, the CORDIC processor implements the iterative CORDIC equations EQ 5 through EQ 7.

$$x_{i+1} = x_i - y_i \times d_i \times 2^{-i} \quad \text{EQ 5}$$

$$y_{i+1} = y_i + x_i \times d_i \times 2^{-i} \quad \text{EQ 6}$$

$$a_{i+1} = a_i - d_i \times \arctan(2^{-i}) \quad \text{EQ 7}$$

The sign-controlling function d_i takes the values shown in EQ 8 and EQ 9:

- In rotation mode

$d_i = -1$ if $a_i < 0$, otherwise $d_i = 1$ EQ 8

- In vectoring mode

$d_i = 1$ if $y_i < 0$, otherwise $d_i = -1$ EQ 9

The input and output data is represented as n-bit words, where n is a user-defined number in the range from 8 to 48. The number of iterations is also defined by a user in the same range. The CORDIC result accuracy improves when the number of iterations is increased, as long as the number of iterations does not exceed data bit width. In other words, the bit width limits the number of meaningful iterations.

A system that utilizes the CORDIC engine (Figure 3 on page 3) consists of the following:

- A data source generating the vector data to be converted by the CORDIC
- The CORDIC module configured to work in either rotation or vectoring mode
- A data receiver accepting the newly converted vector data

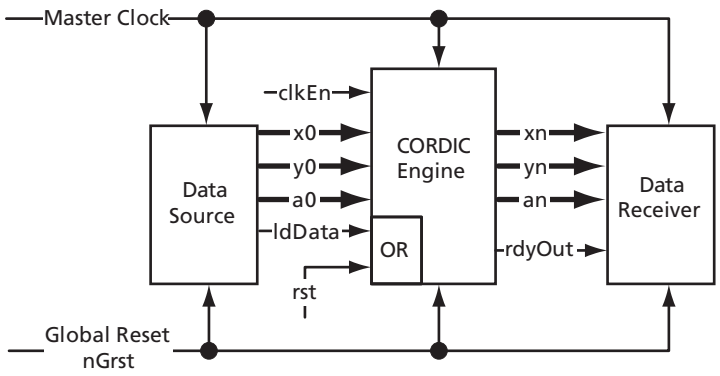

Figure 3 • **CORDIC-Based System**

The negative *nGrst* signal resets the CORDIC engine and, optionally, the entire system. After the reset (input *nGrst* taken high), the CORDIC module is ready to receive data samples to be processed. The module synchronous reset input *rst* can be used to bring the CORDIC unit to the ready state at any time after the initial global reset.

Note: The CORDIC module will lose half-processed data when *rst* is taken high by the system.

The data source supplies the CORDIC engine with the data to be converted. Depending on the mode (rotation or vectoring), the system uses different CORDIC inputs and outputs to enter and obtain the data. Table 1 shows the input/output signals used in each mode.

Table 1 • **CORDIC Connection to the System**

Input Data	CORDIC Input	Output Data	CORDIC Output
Common Rotation Modes			
Input vector magnitude	x0	Output vector coordinate X	xn
Constant 0	y0	Output vector coordinate Y	yn
Input vector phase	a0	N/A	an
Rotation Mode: Sine/Cosine Table Generator			
Constant reciprocal value of the processing gain r = 1/K	x0	sin(θ)	xn
Constant 0	y0	cos(θ)	yn
Sine/cosine argument θ	a0	N/A	an
Vectoring Mode			
Input vector coordinate X	x0	Output vector magnitude r	xn
Input vector coordinate Y	y0	N/A	yn
Constant 0	a0	Output vector phase θ	an

The system accompanies every new pair of the input data samples with the one-bit *ldData* signal. Upon receiving the *ldData* bit, the module assumes the vector coordinates are present on input data busses. Once the CORDIC results are ready, the engine puts these out, accompanied by the one-bit *rdyOut* signal. Upon receiving the *rdyOut* bit, the system can supply a new pair of input data and generate another *ldData* signal.

CoreCORDIC can generate three different CORDIC core implementation architectures and an appropriate testbench:

- Parallel pipelined
- Word-serial
- Bit-serial

The parallel pipelined architecture provides the fastest speed, whereas the bit-serial architecture provides the smallest area. The word-serial architecture provides the trade-off of moderate speed and area.

CoreCORDIC Device Requirements

Table 2 provides typical utilization and performance data for CoreCORDIC, implemented in various Actel devices with the CORDIC engine bit resolution set to 24 bits and the number of iterations set to 24. Device utilization and performance will vary depending upon the architecture chosen and the configuration parameters used. Time-driven settings were used when synthesizing parallel architectures; area optimization settings were used in other cases.

The CORDIC core does not utilize on-chip RAM blocks.

Table 2 • **CoreCORDIC Device Utilization and Performance**

Device	Engine Architecture	Mode	Cells or Tiles			Utilization %	Clock Rate, MHz	Transform Time, nsec
			Comb	Seq	Total			
Fusion			**Speed Grade –2**					
AFS600	Bit-serial	Rotate	297	110	407	3%	88	6,568
		Vector	293	108	401	3%	87	6644
AFS600	Word- serial	Rotate	668	103	771	6%	30	833
		Vector	660	101	761	6%	27	926
AFS600	Parallel	Rotate	11,810	1,884	13,694	99%	46	21.7
ProASIC3/E			**Speed Grade –2**					
A3P250	Bit-serial	Rotate	297	110	407	7%	83	6,964
		Vector	296	108	404	7%	93	6,215
A3P250	Word-serial	Rotate	664	103	767	12%	30	833
		Vector	658	101	759	12%	26	962
A3P1000	Parallel	Rotate	12,541	1,906	14,447	59%	46	21.7
		Vector	14,832	1,981	16,813	68%	62	16.1
ProASIC^{PLUS}			**Speed Grade STD**					
APA150	Bit-serial	Rotate	393	108	501	8%	61	9,475
		Vector	394	107	501	8%	63	9,175
APA150	Word-serial	Rotate	824	114	938	15%	20	1,250
		Vector	822	114	936	15%	19	1,316
APA1000	Parallel	Rotate	14,301	1,889	16,190	29%	32	31.3
		Vector	16,594	1,936	18,530	33%	37	27.0
Axcelerator			**Speed Grade –2**					
AX125	Bit-serial	Rotate	196	106	302	15%	113	5,115
		Vector	185	105	290	14%	115	5,026
AX125	Word-serial	Rotate	413	124	537	27%	103	243
		Vector	405	133	538	27%	109	229
AX500	Parallel	Rotate	4,633	1,832	6,465	80%	130	7.7
		Vector	4,617	1,835	6,452	80%	124	8.1
RTAX-S			**Speed Grade –1**					
RTAX250S	Bit-serial	Rotate	196	106	302	8%	92	6,283
		Vector	185	105	290	7%	100	5,780
RTAX250S	Word-serial	Rotate	413	124	537	14%	74	338
		Vector	405	133	538	14%	75	333
RTAX1000S	Parallel	Rotate	4,633	1,832	6,465	36%	89	11.2
		Vector	4,617	1,835	6,452	36%	81	12.3

Note: The above data were obtained by typical synthesis and place-and-route methods. Other core parameter settings can result in different utilization and performance values.

Table 2 • **CoreCORDIC Device Utilization and Performance (Continued)**

Device	Engine Architecture	Mode	Cells or Tiles			Utilization %	Clock Rate, MHz	Transform Time, nsec
			Comb	Seq	Total			
54SX-A			**Speed Grade –2**					
54SX72A	Bit-serial	Rotate	190	105	295	5%	67	8,627
		Vector	195	105	300	5%	71	8,141
54SX72A	Word-serial	Rotate	656	132	788	13%	55	455
		Vector	643	124	767	13%	50	500
RT54SX-S			**Speed Grade –1**					
RT54SX72S	Bit-serial	Rotate	189	104	293	5%	55	10,509
		Vector	190	104	294	5%	55	10,509
RT54SX72S	Word-serial	Rotate	677	132	809	13%	33	758
		Vector	664	125	789	13%	34	735

Note: The above data were obtained by typical synthesis and place-and-route methods. Other core parameter settings can result in different utilization and performance values.

Architectures

Word-Serial Architecture

Direct implementation of the CORDIC iterative equations (see "References" on page 12) yields the block diagram shown in Figure 4. The vector coordinates to be converted, or initial values, are loaded via multiplexers into registers RegX, RegY, and RegA. RegA, along with an adjacent adder/subtractor, multiplexer, and a small arctan LUT, is often called an angle accumulator. Then on each of the following clock cycles, the registered values are passed through adders/subtractors and shifters. The results described by EQ 5 through EQ 7 on page 2 are loaded back to the same registers. Every iteration takes one clock cycle, so that in *n* clock cycles, *n* iterations are performed and the converted coordinates are stored in the registers.

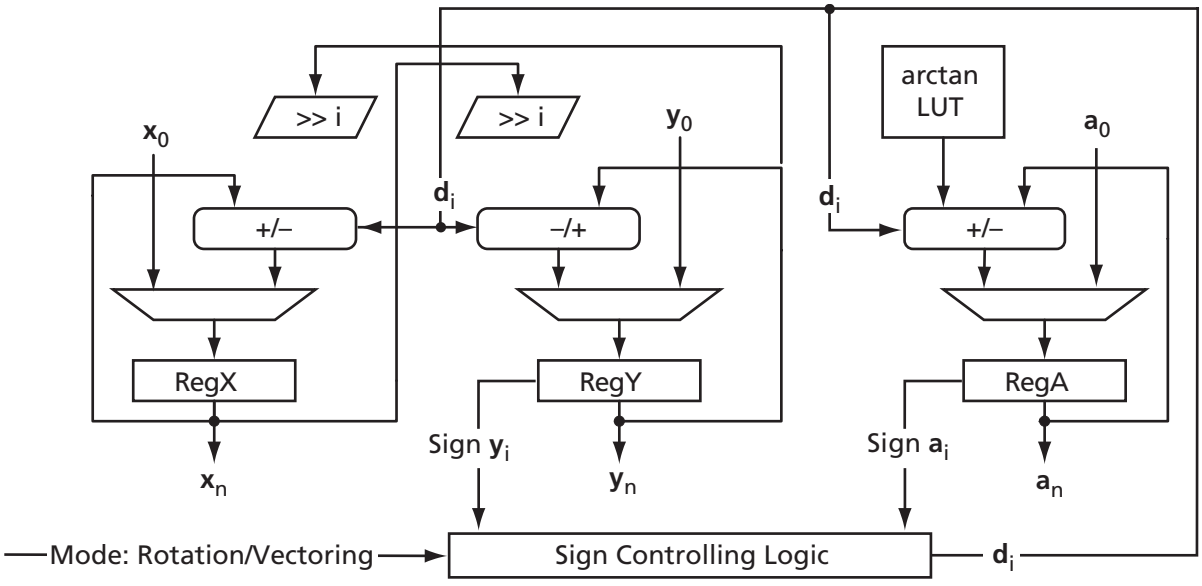

Figure 4 • **Word-Serial CORDIC Block Diagram**

Depending on the CORDIC mode (rotation or vectoring), the sign-controlling logic watches either the RegY or the RegA sign bit. Based on EQ 8 and EQ 9 on page 2, it decides what type of operation (addition or subtraction) needs to be performed at every iteration. The arctan LUT keeps a pre-computed table of the *arctan(2^{-i})* values. The number of entries in the arctan LUT equals the desirable number of iterations, *n*.

The word-serial CORDIC engine takes *n + 1* clock cycles to complete a single vector coordinate conversion.

Parallel Pipelined Architecture

This architecture presents an unrolled version of the sequential CORDIC algorithm above. Instead of reusing the same hardware for all iteration stages, the parallel architecture has a separate hardware processor for every CORDIC iteration. An example of the parallel CORDIC architecture configured for rotation mode is shown in Figure 5.

Each of the *n* processors performs a specific iteration, and a particular processor always performs the same iteration. This leads to a simplification of the hardware. All the shifters perform the fixed shift, which means these can be implemented in the FPGA wiring. Every processor utilizes a particular arctan value that can also be hardwired to the input of every angle accumulator. Yet another simplification is an absence of a state machine.

The parallel architecture is obviously faster than the sequential architecture described in the "Word-Serial Architecture" section on page 5. It accepts new input data and puts out the results at every clock cycle. The architecture introduces a latency of *n* clock cycles.

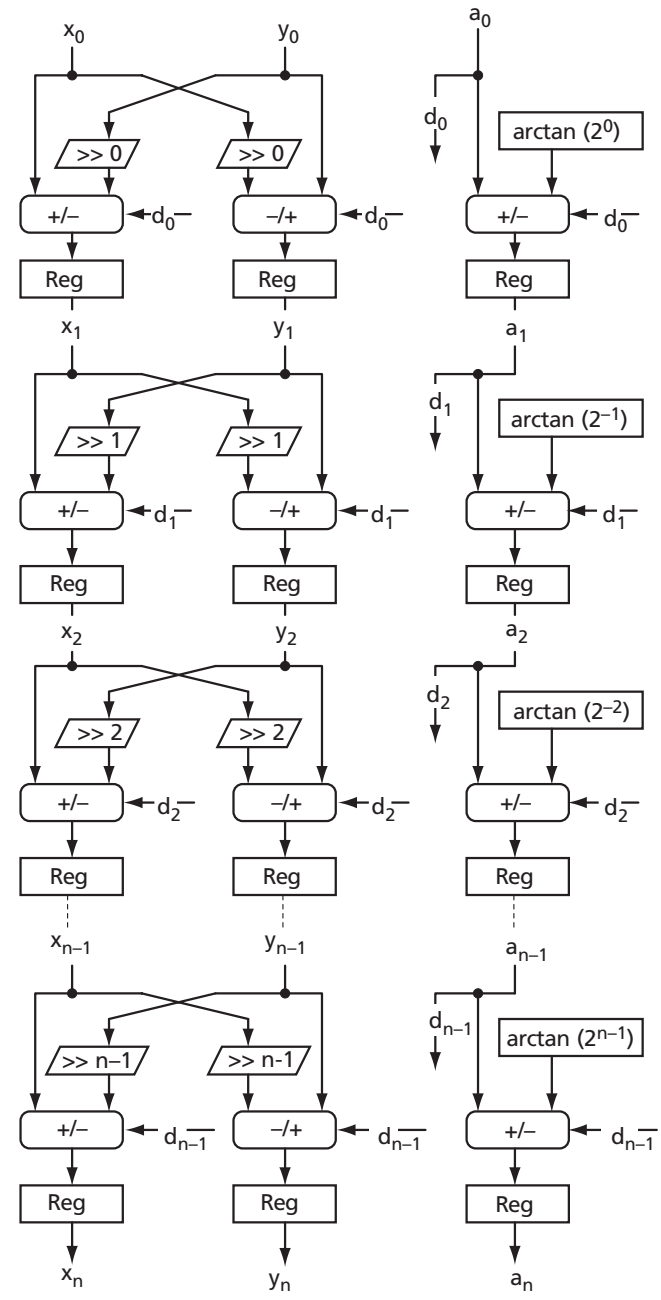

Figure 5 • **Parallel CORDIC Architecture**

Bit-Serial Architecture

Whenever the CORDIC conversion speed is not an issue, this architecture provides the smallest FPGA implementation. For example, in order to initialize a Sine/Cosine LUT, the bit-serial CORDIC is the solution. Figure 6 depicts the simplified block diagram of the bit-serial architecture. The shift registers get loaded with initial data presented in bit-parallel form, i.e., all bits at once. The data then shifts to the right, before arriving the serial adders/subtractors. Every iteration takes *m* clock cycles, where *m* is the CORDIC bit resolution. Serial shifters are implemented by properly tapping the bits of the shift registers. The control circuitry (not shown in Figure 6) provides sign-padding of the shifted serial data to realize its correct sign extension. The results from the serial adders return back to the shift registers, so that in *m* clock cycles the results of another iteration are stored in the shift registers.

A single full CORDIC conversion takes $n \times m + 2$ clock cycles.

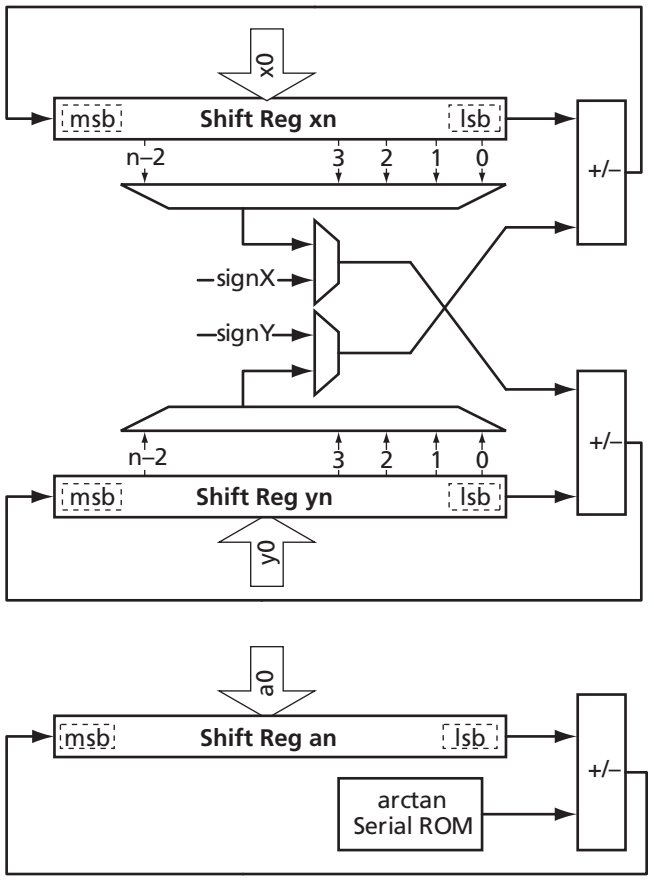

Figure 6 • **Bit-Serial CORDIC Architecture**

I/O Formats

Q Format Fixed-Point Numbers

CoreCORDIC, as virtually any FPGA DSP core does, utilizes fixed-point arithmetic. In particular, the numbers the core operates with are presented as two's complement signed fractional numbers. To identify the position of a binary point separating the integer and fractional portions of the number, the Q format is commonly used.

An mQn format number is an (n+1)-bit signed two's complement fixed-point number: a sign bit followed by n significant bits with the binary point placed immediately to the right of the m most significant bits. The m MSBs represent the integer part, and (n–m) LSBs represent the fractional part of the number, called the mantissa. Table 3 depicts an example of a 1Qn format number.

Table 3 • **1Qn Format Number**

Bit 2^n	Bit 2^{n-1}	Position of the Binary Point	Bits $[2^{n-2} : 2^0]$
Sign	Integer bit		Mantissa

Table 4 • **Qn Format Number**

Bit 2^n	Position of the Binary Point	Bits $[2^{n-1} : 2^0]$
Sign		Mantissa

The following sections explain in detail the formats of the input and output signals. The linear and angular values are explained separately. The linear signals include Cartesian coordinates and a vector magnitude. These come to the CORDIC engine inputs *x0* and *y0*, or appear on its outputs *xn* and *yn*. Since the sine and cosine functions the CORDIC calculates are essentially the Cartesian coordinates of the vector, the angular signals include the vector phase that comes to the CORDIC engine input *a0*, or appears on its output *an*. Both linear and angular signals utilize mQn formats and appropriate conversion rules from floating-point to the mQn formats.

I/O Linear Format

The CoreCORDIC engine utilizes the 1Qn format shown in Table 3. Though the 1Qn format numbers are capable of expressing fixed-point numbers in the range from (-2^n) to $(2^n - 2^{m-n})$, the input linear data must be limited to fit the smaller range from (-2^{n-1}) to (2^{n-1}). In terms of floating-point numbers, the input must fit the range from –1.0 to +1.0. For example, the 1Q9 format input data range is limited by the following 10-bit numbers:

Max input negative number of –1.0:

1100000000 ⇔ 11.00000000

Max input positive number of +1.0:

0100000000 ⇔ 01.00000000

This precaution is taken to prevent the data overflow that otherwise could occur as a result of the CORDIC inherent processing gain. The output data obviously do not have to fit the limited range.

To convert floating-point linear input data to the 1Qn format, follow the simple rule in EQ 10:

$$1Qn\ Fixed\text{-}Point\ Data = 2^{n-1} \times Floating\text{-}Point\ Data \quad EQ\ 10$$

Here it is assumed the floating-point data are presented in the range from –1.0 to 1.0. The product on the right-hand side of EQ 10 contains integer and fractional parts. The fractional part has to be truncated or rounded. Table 5 shows a few examples of converting the floating-point numbers to the 1Q9 format.

To convert the 1Qn format back to the floating-point format, use EQ 11.

$$Floating\text{-}Point\ Data = 1Qn\ Fixed\text{-}Point\ Data/2^{n-1} \quad EQ\ 11$$

Table 5 • **Floating-Point to 1Q9 Format Conversion**

Floating-Point Number X	$P = X \times 2^{(n-1)}$	P Rounded	Common Binary Format	1Q9 Format
1.00	256	256	0100000000	01.00000000
0.678915	173.80224	174	0010101101	00.10101101
0.047216	12.087296	12	0000001100	00.00001100
–1.00	–256	–256	1100000000	11.00000000
–0.678915	–173.80224	–174	1101010011	11.01010011
–0.047216	–12.087296	–12	1111110100	11.11110100

I/O Angular Format

The angle (phase) signals are *a0* and *an.* They are presented in Qn format, as shown in Table 4 on page 7. The relation between the floating-point angular value expressed in radians and the Qn format is shown in EQ 12. [1]

$$Qn\ Fixed\text{-}Point\ Angle = 2^{n-1} \times Floating\text{-}Point\ Angle/\pi \quad EQ\ 12$$

In EQ 12, the floating-point angle is measured in radians. The product on the right-hand side of EQ 10 contains integer and fractional parts. The fractional part must be truncated or rounded.

EQ 13 presents a rule for the conversion from the Qn format back to the floating-point radian measure.

$$Floating\text{-}Point\ Angle = Qn\ Fixed\text{-}Point\ Angle \times \pi/2^{n-1} \quad EQ\ 13$$

The conversion formulae (EQ 12 and EQ 13) support an important feature that greatly simplifies sine and cosine table calculations. Such tables usually have power of two entries (lines). At the same time, they often span angular values from $-\pi/2$ to $\pi/2$ radians. Therefore, it is beneficial to represent the angle of $\pi/2$ radians with the power of two fixed-point number. In particular, when having the CORDIC engine calculate the $\sin(\theta)$ and $\cos(\theta)$ table, it is sufficient to increment the fixed-point angular argument θ at each cycle.

The angular value range is from $-\pi/2$ to $\pi/2$, or in Q9 format:

Max input negative number of $-\pi/2$:

1100000000 ⇔ .1100000000

Max input positive number of $+\pi/2$:

0100000000 ⇔ .0100000000

Table 6 shows a few examples of converting floating-point numbers to Q9 format.

Table 6 • **Examples of Angular Value to Fixed-Point Conversion**

Floating-Point Angle A (rad)		$P = A \times 2^n$	Common Binary Format	Q9 Format (sign.mantissa)
$\pi/2$	1.5707963268	256	0100000000	0.100000000
$\pi/4$	0.7853981634	128	0010000000	0.010000000
$\pi/256$	0.0122718463	2	0000000010	0.000000010

1. This format means, literally, the angle of π radians is expressed as the floating-point value of 1.0.

Table 6 • **Examples of Angular Value to Fixed-Point Conversion**

–π/2	–1.5707963268	–256	1100000000	1.100000000
–π/4	–0.7853981634	–128	1110000000	1.110000000
–π/256	–0.0122718463	–2	1111111110	1.111111110

CoreCORDIC Configuration Parameters

CoreCORDIC generates the CORDIC engine RTL code based on parameters set by the user when generating the module. The core generator supports the variations specified in Table 7.

Table 7 • **Core Generator Parameters**

Parameter Name	Description	Values
module_name	Name of the generated RTL code module	–
architecture	Bit-serial, word-serial, or word parallel architecture	0 (bit-serial), 1 (word-serial), 2 (parallel). Default value = 0.
mode	Vector rotation (polar to rectangular coordinate conversion and sine/cosine calculation) or vector translation (rectangular to polar conversion)	0 (vector rotation), 1 (vector translation). Default value = 0.
bit_width	I/O data bit width	8–48. Default value = 16.
iterations	Number of iterations	8–48. Default value = bit_width.*
fpga_family	Family of the Actel FPGA device	ax (Axcelerator), apa (ProASIC^{PLUS}), pa3 (ProASIC3), sx (SX-A), af (Fusion)
lang	RTL code language	vhdl, verilog

Note: *A warning is issued if the number of iterations is set greater than the bit width.

I/O Signal Description

Figure 7 shows the CoreCORDIC module pinout.

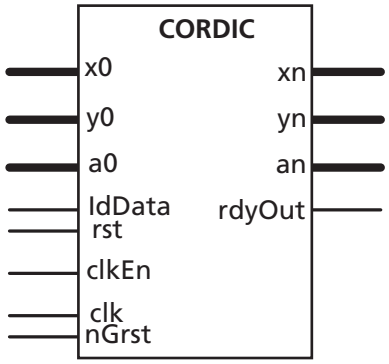

Figure 7 • **CoreCORDIC I/O Signals**

The CoreCORDIC module I/O signal functionality is listed in Table 8.

Table 8 • **I/O Signal Descriptions**

Signal Name	Direction	Description
x0 [bit_width – 1 : 0]	Input	Input data bus x0. The abscissa of the input vector in the vectoring mode or the magnitude of the input vector in rotation mode should be placed on this bus. Bit [bit_width – 1] is the MSB. Data are assumed to be presented in two's complement format. The other vector coordinates are to be supplied simultaneously.
y0 [bit_width – 1 : 0]	Input	Input data bus y0. The ordinate of the input vector in the vectoring mode should be placed on this bus. In rotation mode, the bus should be grounded or left idle. Bit [bit_width – 1] is the MSB. Data are assumed to be presented in two's complement format. The other vector coordinates are to be supplied simultaneously.
a0 [bit_width – 1 : 0]	Input	Input angle data bus a0. The phase of the input vector in the rotation mode should be placed on this bus. In vectoring mode, the bus should be grounded or left idle. Bit [bit_width – 1] is the MSB. Data are assumed to be presented in two's complement format. The other vector coordinates are to be supplied simultaneously.
clk	Input	System clock. Active rising edge.
nGrst	Input	System asynchronous reset. Active low.
rst	Input	System/module synchronous reset. Active high. Valid in parallel architecture only. Resets all registers of the core.
clkEn	Input	Clock enable signal. Active high. Valid in word-serial and bit-serial architectures.
ldData	Input	Load input data. Indicates that input vector coordinates are ready for the CORDIC engine to be processed. Active high. Valid in word-serial and bit-serial architectures.
rdyOut	Output	Output data (vector coordinates or sine/cosine values) are ready for the data receiver to read. Active high. Valid in word-serial and bit-serial architectures.
xn [bit_width-1 : 0]	Output	Output data bus xn. The abscissa of the output vector in rotation mode or the magnitude of the output vector in the vectoring mode appears on this bus. Bit [bit_width – 1] is the MSB. Data are presented in two's complement format. The other vector coordinates emerge on their respective output busses simultaneously.
yn [bit_width-1 : 0]	Output	Output data bus yn. The ordinate of the output vector in rotation mode. Bit [bit_width – 1] is the MSB. Data are presented in two's complement format. The other vector coordinates emerge on their respective output busses simultaneously.
an [bit_width-1 : 0]	Output	Output data bus an. The phase of the output vector in vectoring mode. Bit [bit_width – 1] is the MSB. Data are presented in two's complement format. The other vector coordinates emerge on their respective output busses simultaneously.

I/O Interface and Timing

Upon reset, the CORDIC core returns to its initial state. Signal *nGrst* asynchronously resets any architecture. Other I/O interfaces and timing depend on core architecture.

Bit-Serial Architecture Interface and Timing

Figure 8 depicts a typical timing diagram for the bit-serial architecture. Signal *ldData* resets the bit-serial CORDIC module and loads a set of data present on the *a0, x0,* and *y0* input busses. The set of input data is shown in Figure 8 as In0. Normally, a next *ldData* signal has to come after the end of a current CORDIC cycle; i.e., after the *rdyOut* signal appears on the module output. In the case that the next *ldData* signal is issued prior to the end of the current cycle, the CORDIC engine starts a new computation cycle and discards the incomplete results of the interrupted cycle.

Once the CORDIC engine completes calculating the result, it generates *rdyOut* signal one clock period in width. The result on the output busses (*an*, *xn*, and *yn*) is valid while the *rdyOut* signal is active. The next *ldData* signal can coincide with the *rdyOut* signal. Obviously a valid, fresh set of input data, shown as In1 in Figure 8, must be ready by then.

One cycle of CORDIC computation = (bit_width × iterations + 2) clock cycles.

Signal *clkEn* can be manipulated as desired. While this signal is low, the CORDIC engine retains all the data it has collected or processed so far. Normally, the bit-serial CORDIC engine is used to fill up the LUT on a power-on event. Once the CORDIC fulfills this function, a high-level state machine may disable the *clkEn* signal.

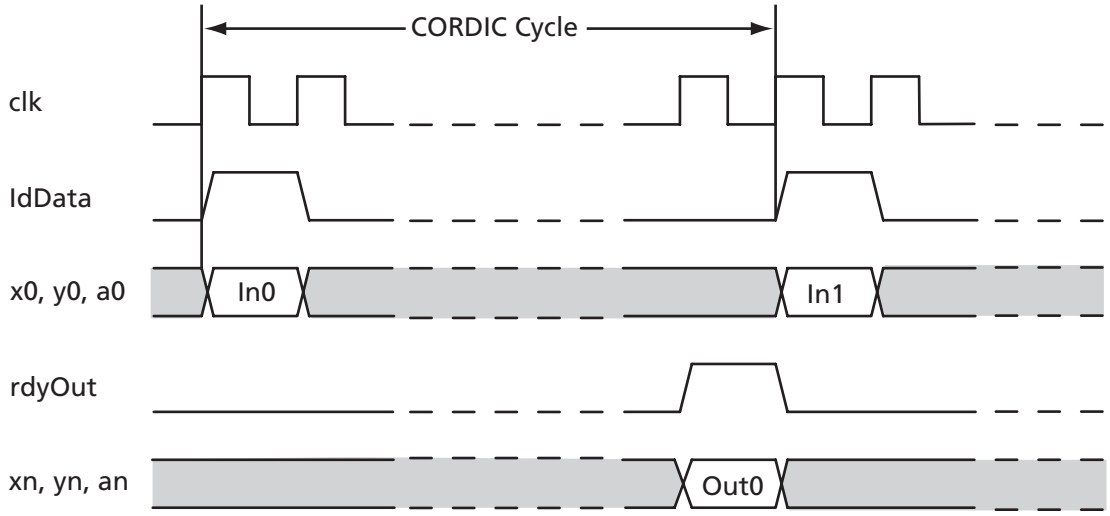

Figure 8 • **Bit-Serial Architecture Timing Diagram**

Word-Serial Architecture Interface and Timing

Figure 9 on page 12 depicts a timing diagram for the word-serial architecture. It is very similar to the bit-serial timing diagram. Signal *ldData* resets the word-serial CORDIC module and loads the set of data present on the *a0, x0,* and *y0* input busses. The set of input data is shown in Figure 9 on page 12 as In0. Normally the next *ldData* signal must come after the end of the current CORDIC cycle; i.e., after the *rdyOut* signal appears on the module output. In the case that the next *ldData* signal is issued prior to the end of a current cycle, the CORDIC engine starts a new computation cycle and discards the incomplete results of the interrupted cycle.

Once the CORDIC engine completes calculating the result, it generates a *rdyOut* signal one clock period in width. The result on the output busses (*an*, *xn*, and *yn*) is valid while the *rdyOut* signal is active. The next *ldData* signal can immediately follow the *rdyOut* signal. Obviously a valid, fresh set of input data, shown as In1, must be ready by then.

One cycle of CORDIC computation = (iterations + 1) clock cycles.

Signal *clkEn* can be manipulated as desired. While this signal is low, the CORDIC engine retains all the data it has collected or processed so far. As an example, the word-serial CORDIC engine is used to fill up the LUT on a power-on event. Once the CORDIC completes the task, a high-level state machine may disable the *clkEn* signal.

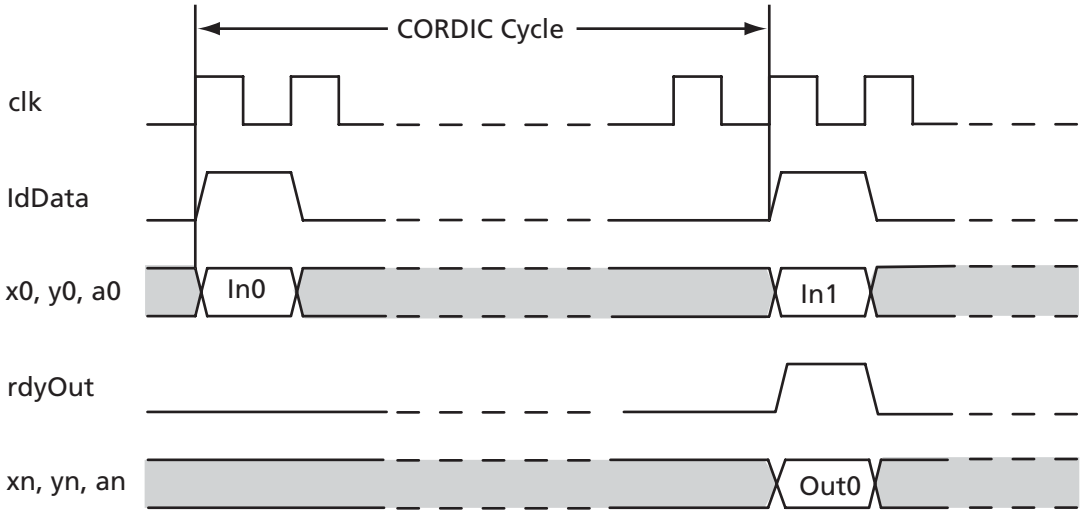

Figure 9 • **Word-Serial Architecture Timing Diagram**

Parallel Architecture Interface and Timing

Figure 10 depicts a timing diagram for the parallel architecture. At the beginning of every clock cycle, a fresh set of input arguments *a0*, *x0*, and *y0* enters the CORDIC engine. No control signals accompany the input data. The CORDIC engine puts out the results at the beginning of every clock cycle with the latency of *iterations* clock cycles.

Signal *rst* synchronously resets the parallel architecture; i.e., resets all the registers of the parallel engine.

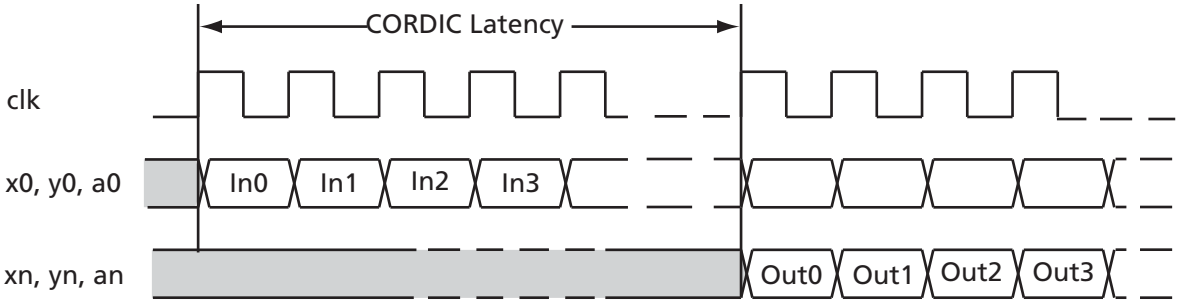

Figure 10 • **Parallel Architecture Timing Diagram**

References

J.E. Volder. 1959. "The CORDIC Trigonometric Computing Technique." *IRE Transaction on Electronic Computers*, EC-8:330-334. http://lap.epfl.ch/courses/comparith/Papers/3-Volder_CORDIC.pdf

Ray Andraka, "A Survey of CORDIC Algorithms for FPGA Based Computers," http://www.fpga-guru.com/files/crdcsrvy.pdf, 1998.

Norbert Lindlbauer, "The CORDIC-Algorithm for Computing a Sine," http://www.cnmat.berkeley.edu/~norbert/cordic/node4.html, 2000.

Grant R. Griffin, "CORDIC FAQ," http://www.dspguru.com/info/faqs/cordic.htm.

A Sample Configuration File

The following is an example of the configuration file:

module_name	Cordic_test
architecture	0
mode	0
bit_width	16
iterations	16
fpga_family	pa3
lang	verilog

Ordering Information

Order CoreCORDIC through your local Actel sales representative. Use the following numbering convention when ordering: CoreCORDIC-XX, where XX is listed in Table 9.

Table 9 • **Ordering Codes**

XX	Description
EV	Evaluation version
AR	RTL for unlimited use on Actel devices
UR	RTL for unlimited use and not restricted to Actel devices

Datasheet Categories

In order to provide the latest information to designers, some datasheets are published before data has been fully characterized. Datasheets are designated as "Product Brief," "Advanced," and "Production." The definitions of these categories are as follows:

Product Brief

The product brief is a summarized version of an advanced or production datasheet containing general product information. This brief summarizes specific device and family information for unreleased products.

Advanced

This datasheet version contains initial estimated information based on simulation, other products, devices, or speed grades. This information can be used as estimates, but not for production.

Unmarked (production)

This datasheet version contains information that is considered to be final.

Appendix I

Polar and Rectangular Coordinate Relations

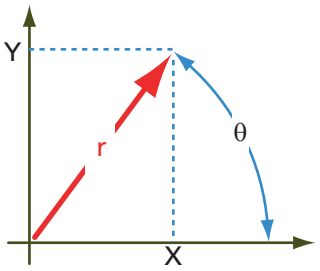

Figure 11 • **Cartesian Coordinate Definition**

The Cartesian coordinates (*X*, *Y*) are defined in terms of the polar coordinates *r* (vector magnitude, or radial coordinate) and θ (vector phase, or polar angle), as given in EQ 14 and EQ 15.

$$X = r\cos\theta$$

EQ 14

$$Y = r\sin\theta$$

EQ 15

In terms of Cartesian coordinates, the polar coordinates are expressed as given in EQ 16 and EQ 17.

$$r = \sqrt{X^2 + Y^2}$$

EQ 16

$$\theta = \arctan(Y/X)$$

EQ 17

Appendix II

Examples of CORDIC Modes

Polar to Cartesian Coordinate Conversion

The CORDIC engine is in rotation mode. Input data represent magnitude *r* and phase θ of the vector whose polar coordinates are to be converted to Cartesian coordinates. The CORDIC engine puts out a pair of Cartesian coordinates (*X*K*, *Y*K*) scaled by processing gain *K* (Figure 12).

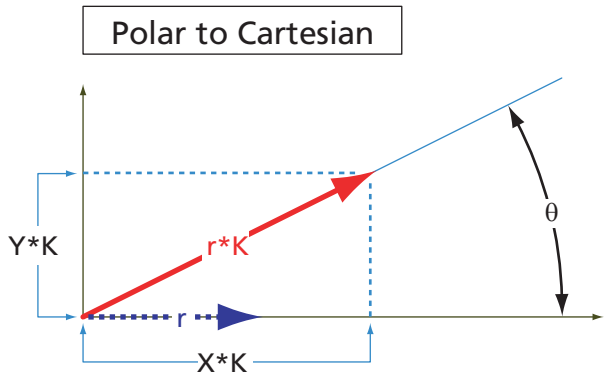

Figure 12 • **Polar to Cartesian Vector Conversion**

General Rotation

The CORDIC engine is in rotation mode. Input data (X_0, Y_0, *Angle*) represent initial vector Cartesian coordinates, as well as an angle to rotate the vector. The CORDIC engine puts out a pair of Cartesian coordinates (*X*K*, *Y*K*) of the resulting rotated vector scaled by processing gain *K* (Figure 13).

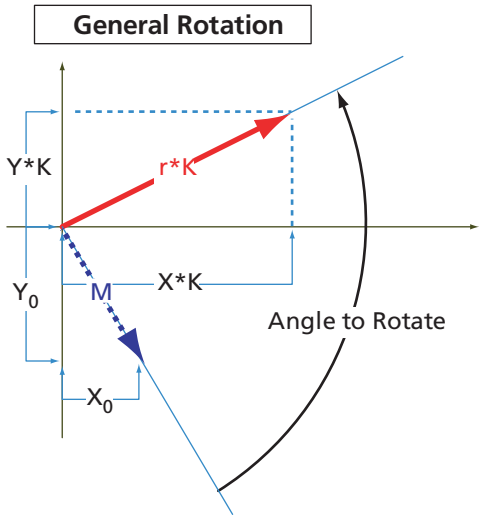

Figure 13 • **CORDIC General Vector Rotation**

Sine and Cosine CORDIC Calculator

The CORDIC engine is in rotation mode. Input data $r = 1/K$ and phase θ represent initial vector polar coordinates. The CORDIC engine puts out a pair of Cartesian coordinates equal to ($\cos\theta$, $\sin\theta$), as shown in Figure 14.

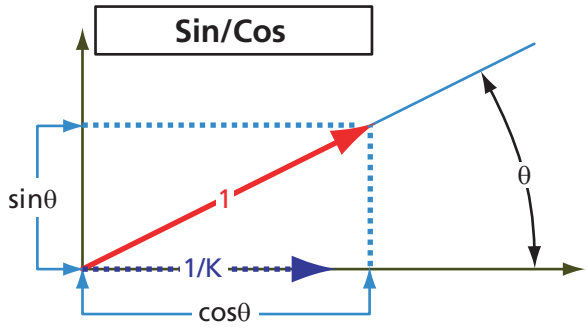

Figure 14 • **Sine and Cosine CORDIC Computation**

Cartesian to Polar Coordinate Conversion

The CORDIC engine is in vectoring mode. Input data represent Cartesian coordinates (X_0, Y_0) of the input vector. The CORDIC engine puts out a pair of polar coordinates: magnitude $r*K$ and phase θ of the input vector (Figure 15).

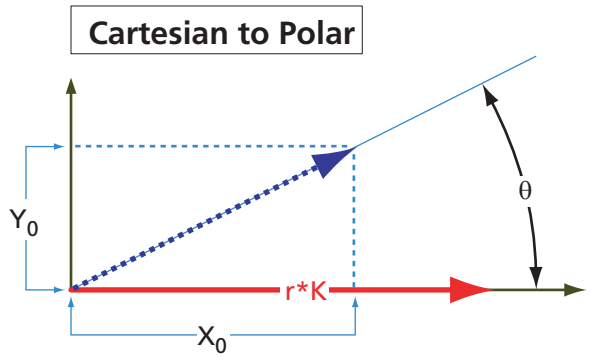

Figure 15 • **Cartesian to Polar Coordinate Conversion**

CORDIC Square Root Calculator

The CORDIC engine is in vectoring mode. Input data represent Cartesian coordinates (X_0, Y_0) of the input vector. The CORDIC engine puts out a pair of polar coordinates: magnitude $r = K\sqrt{x_0^2 + y_0^2}$ and phase θ of the input vector (Figure 16).

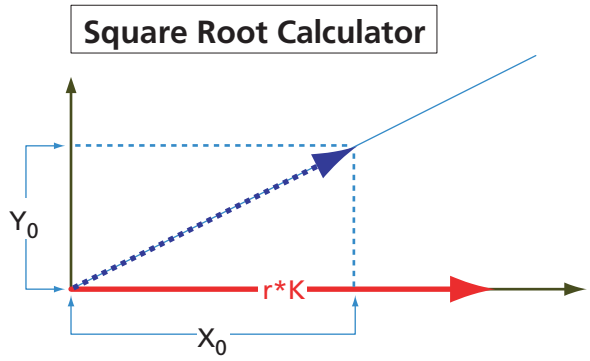

Figure 16 • **CORDIC Square Root Calculator**

CORDIC Arctan Calculator

The CORDIC engine is in vectoring mode. Input data represent Cartesian coordinates (X_0, Y_0) of the input vector. The CORDIC engine puts out a pair of polar coordinates: magnitude r and phase $\theta = \arctan(Y_0 / X_0)$ of the input vector.

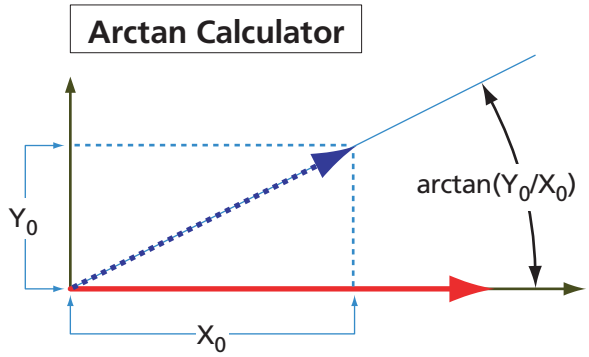

Figure 17 • **CORDIC Arctan Phase Calculator**

www.actel.com

Actel Corporation

2061 Stierlin Court
Mountain View, CA
94043-4655 USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.

Dunlop House, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Phone +44 (0) 1276 401 450
Fax +44 (0) 1276 401 490

Actel Japan
www.jp.actel.com

EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668

Actel Hong Kong
www.actel.com.cn

Suite 2114, Two Pacific Place
88 Queensway, Admiralty
Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488

51700064-0/3.06