NCO MegaCore Function

User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

UG-NCOCOMPILER-14.2

Feedback Subscribe

Contents

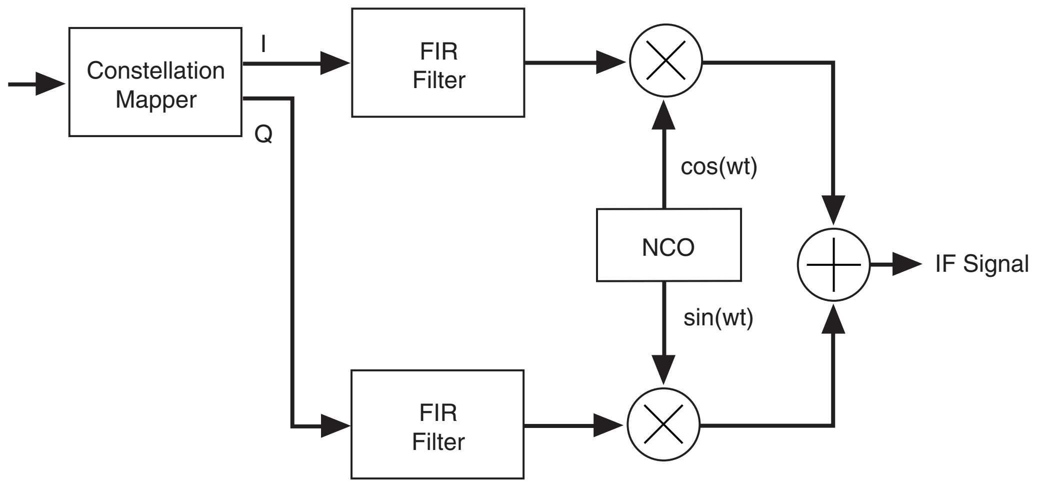
1. About This MegaCore Function

The Altera® NCO MegaCore® function generates numerically controlled oscillators (NCOs) customized for Altera devices. A numerically controlled oscillator (NCO) synthesizes a discrete-time, discrete-valued representation of a sinusoidal waveform.

Designers typically use NCOs in communication systems as quadrature carrier generators in I-Q mixers, in which baseband data is modulated onto the orthogonal carriers in one of a variety of ways.

Figure 1–1 shows an NCO in a simple modulator system.

Figure 1–1. Simple Modulator



Designers also use NCOs in all-digital phase-locked-loops (PLLs) for carrier synchronization in communications receivers, or as standalone frequency shift keying (FSK) or phase shift keying (PSK) modulators. In these applications, the phase or the frequency of the output waveform varies directly according to an input data stream.

You can implement a variety of NCO architectures, including ROM-based, CORDIC-based, and multiplier-based. The wizard also includes time and frequency domain graphs that dynamically display the functionality of the NCO, based on your parameter settings.

To decide which NCO implementation to use, consider several parameters, including the spectral purity, frequency resolution, performance, throughput, and required device resources. Also, consider the trade-offs between some or all of these parameters.

Features

The NCO MegaCore function supports the following features:

- 32-bit precision for angle and magnitude
- Source interface compatible with the *Avalon Interface Specification*

- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Multiple NCO architectures:
  - Multiplier-based implementation using DSP blocks or logic elements (LEs), (single cycle and multi-cycle)
  - Parallel or serial CORDIC-based implementation
  - ROM-based implementation using embedded array blocks (EABs), embedded system blocks (ESBs), or external ROM
- Single or dual outputs (sine/cosine)
- Variable width frequency modulation input
- Variable width phase modulation input
- User-defined frequency resolution, angular precision, and magnitude precision
- Frequency hopping
- Multichannel capability
- Simulation files and architecture-specific testbenches for VHDL, Verilog HDL and MATLAB
- Dual-output oscillator and quaternary frequency shift keying (QFSK) modulator example designs

Release Information

Table 1–1 provides information about this release of the Altera NCO MegaCore function.

Table 1–1. NCO MegaCore Function Release Information

Item	Description
Version	14.0 Arria 10 Edition
Release Date	August 2014
Ordering Code	IP-NCO
Product ID(s)	0014
Vendor ID(s)	6AF7

For more information about this release, refer to the *MegaCore IP Library Release Notes and Errata*.

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore® function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

Device Family Support

Altera offers the following device support levels for Altera IP cores:

- **Preliminary support**—Altera verifies the IP core with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. You can use it in production designs with caution.
- **Final support**—Altera verifies the IP core with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and you can use it in production designs.

Table 1–2 shows the level of support offered by the NCO MegaCore function to each of the Altera device families.

Table 1–2. Device Family Support

Device Family	Support
Arria® II GX	Final
Arria II GZ	Final
Arria V	Final
Arria 10	Final
Cyclone® IV	Final
MAX 10 FPGAs	Final
Stratix® IV GT	Final
Stratix IV GX/E	Final
Stratix V	Final
Other device families	No support

MegaCore Verification

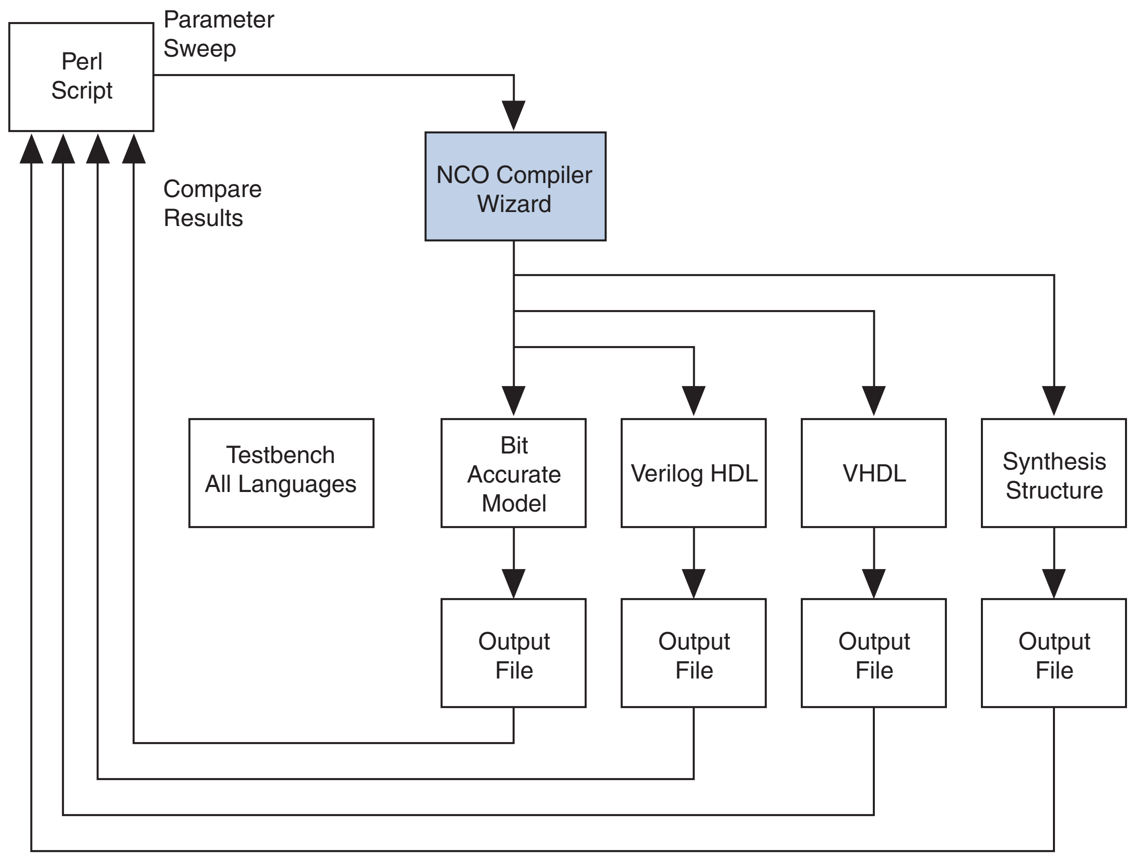
Before releasing a version of the NCO MegaCore function, Altera runs comprehensive regression tests to verify its quality and correctness.

First a custom variation of the NCO MegaCore function is created. Next, Verilog HDL and VHDL IP functional simulation models are exercised by their appropriate testbenches in ModelSim simulators and the results are compared to the output of a bit-accurate model.

The regression suite covers various parameters such as architecture options, frequency modulation, phase modulation, and precision.

Figure 1–2 shows the regression flow.

Figure 1–2. Regression Flow



Performance and Resource Utilization

Table 1–3 shows typical expected performance for a NCO IP core using the Quartus II software with the Arria V (5AGXFB3H4F40C4), Cyclone V (5CGXFC7D6F31C6), and Stratix V (5SGSMD4H2F35C2) devices:

Table 1–3. NCO IP Core Performance

Device	Parameters	ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
				M10K	M20K	Primary	Secondary	
Arria V	Cordic	838	0	1	--	1,879	8	340
Arria V	Large Rom	56	0	12	--	149	0	350
Arria V	Multiplier Based	92	2	2	--	244	2	310
Arria V	Small ROM	132	0	6	--	300	0	350
Cyclone V	Cordic	838	0	1	--	1,881	6	260
Cyclone V	Large Rom	56	0	12	--	149	0	275

Table 1–3. NCO IP Core Performance

Device	Parameters	ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
				M10K	M20K	Primary	Secondary	
Cyclone V	Multiplier Based	92	2	2	--	244	2	275
Cyclone V	Small ROM	120	0	6	--	300	0	275
Stratix V	Cordic	838	0	--	1	1,881	6	644
Stratix V	Large Rom	56	0	--	5	149	0	700
Stratix V	Multiplier Based	92	2	--	2	245	1	500
Stratix V	Small ROM	126	0	--	3	300	0	700

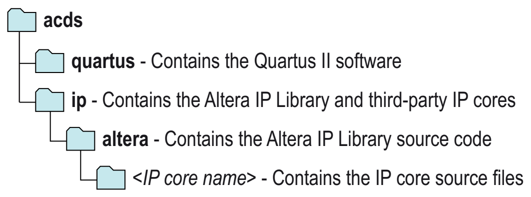
2. Getting Started

Installing and Licensing IP Cores

The Quartus II software includes the Altera IP Library. The library provides many useful IP core functions for production use without additional license. You can fully evaluate any licensed Altera IP core in simulation and in hardware until you are satisfied with its functionality and performance.

Some Altera IP cores, such as MegaCore® functions, require that you purchase a separate license for production use. After you purchase a license, visit the Self Service Licensing Center to obtain a license number for any Altera product. For additional information, refer to *Altera Software Installation and Licensing*.

Figure 2–1. IP core Installation Path


acds
quartus - Contains the Quartus II software
ip - Contains the Altera IP Library and third-party IP cores
altera - Contains the Altera IP Library source code
<IP core name> - Contains the IP core source files

The default installation directory on Windows is **<*drive*>:\altera\<*version number*>**; on Linux it is **<*home directory*>/altera/<*version number*>**.

OpenCore Plus Evaluation

With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP℠ megafunction) within your system.
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include megafunctions.
- Program a device and verify your design in hardware.

You only need to purchase a license for the NCO MegaCore function when you are completely satisfied with its functionality and performance, and want to take your design to production.

After you purchase a license, you can request a license file from the Altera website at **www.altera.com/licensing** and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

For more information about OpenCore Plus hardware evaluation, refer to *AN 320: OpenCore Plus Evaluation of Megafunctions*.

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following operation modes:

- *Untethered*—the design runs for a limited time.
- *Tethered*—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely.

All megafunctions in a device time-out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior might be masked by the time-out behavior of the other megafunctions.

The untethered time-out for the NCO MegaCore function is one hour; the tethered time-out value is indefinite.

The output of NCO MegaCore function is forced low by the internal hardware when the hardware evaluation time expires.

Specifying IP Core Parameters and Options

The parameter editor GUI allows you to quickly configure your custom IP variation. Use the following steps to specify IP core options and parameters in the Quartus II software. Refer to Specifying IP Core Parameters and Options (Legacy Parameter Editors) for configuration of IP cores using the legacy parameter editor.

1. In the IP Catalog (**Tools > IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name for your custom IP variation. The parameter editor saves the IP variation settings in a file named .<*your_ip*>**qsys**. Click **OK**.
3. Specify the parameters and options for your IP variation in the parameter editor, including one or more of the following. Refer to your IP core user guide for information about specific IP core parameters.
   - Optionally select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for processing the IP core files in other EDA tools.
4. Click **Generate HDL**, the **Generation** dialog box appears.
5. Specify output file generation options, and then click **Generate**. The IP variation files generate according to your specifications.
6. To generate a simulation testbench, click **Generate > Generate Testbench System**.
7. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate > HDL Example**.
8. Click **Finish**. The parameter editor adds the top-level **.qsys** file to the current project automatically. If you are prompted to manually add the **.qsys** file to the project, click **Project > Add/Remove Files in Project** to add the file.

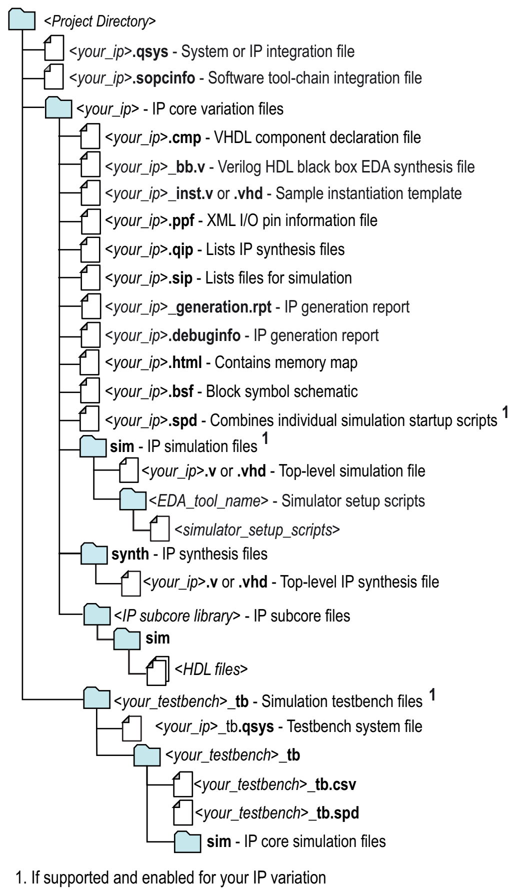
9. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

For information about using a legacy parameter editor, refer to “Specifying IP Core Parameters and Options (Legacy Parameter Editors)” in the *Introduction to Altera IP Cores*.

Files Generated for Altera IP Cores

The Quartus II software version 14.0 Arria 10 Edition and later generates the following output file structure for Altera IP cores:

Figure 2–2. IP Core Generated Files


<Project Directory>
<your_ip>.qsys - System or IP integration file
<your_ip>.sopcinfo - Software tool-chain integration file
<your_ip> - IP core variation files
<your_ip>.cmp - VHDL component declaration file
<your_ip>_bb.v - Verilog HDL black box EDA synthesis file
<your_ip>_inst.v or .vhd - Sample instantiation template
<your_ip>.ppf - XML I/O pin information file
<your_ip>.qip - Lists IP synthesis files
<your_ip>.sip - Lists files for simulation
<your_ip>_generation.rpt - IP generation report
<your_ip>.debuginfo - IP generation report
<your_ip>.html - Contains memory map
<your_ip>.bsf - Block symbol schematic
<your_ip>.spd - Combines individual simulation startup scripts 1
sim - IP simulation files 1
<your_ip>.v or .vhd - Top-level simulation file
<EDA_tool_name> - Simulator setup scripts
<simulator_setup_scripts>
synth - IP synthesis files
<your_ip>.v or .vhd - Top-level IP synthesis file
<IP subcore library> - IP subcore files
sim
<HDL files>
<your_testbench>_tb - Simulation testbench files 1
<your_ip>_tb.qsys - Testbench system file
<your_testbench>_tb
<your_testbench>_tb.csv
<your_testbench>_tb.spd
sim - IP core simulation files
1. If supported and enabled for your IP variation

Simulating IP Cores

The Quartus II software supports RTL- and gate-level design simulation of Altera IP cores in supported EDA simulators. Simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation.

You can use the functional simulation model and the testbench or example design generated with your IP core for simulation. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench. For a complete list of models or libraries required to simulate your IP core, refer to the scripts generated with the testbench. You can use the Quartus II NativeLink feature to automatically generate simulation files and scripts. NativeLink launches your preferred simulator from within the Quartus II software.

For more information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

Adding IP Cores to IP Catalog

The IP Catalog automatically displays Altera IP cores found in the project directory, in the Altera installation directory, and in the defined IP search path. The IP Catalog can include Altera-provided IP components, third-party IP components, custom IP components that you provide, and previously generated Qsys systems.

You can use the IP Search Path option (**Tools > Options**) to include custom and third-party IP components in the IP Catalog. The IP Catalog displays all IP cores in the IP search path. The Quartus II software searches the directories listed in the IP search path for the following IP core files:

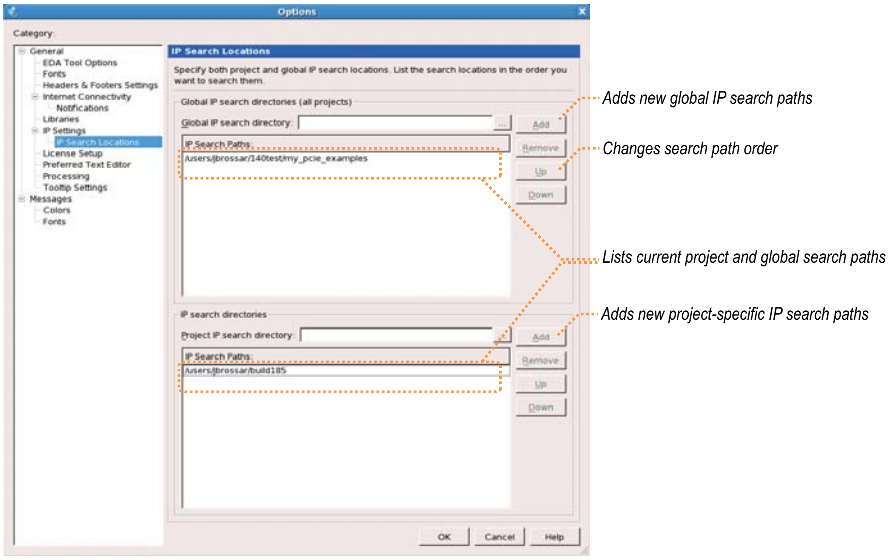
- Component Description File (**_hw.tcl**)—Defines a single IP core.
- IP Index File (**.ipx**)—Each **.ipx** file indexes a collection of available IP cores, or a reference to other directories to search. In general, **.ipx** files facilitate faster searches.

The Quartus II software searches some directories recursively and other directories only to a specific depth. When the search is recursive, the search stops at any directory that contains an **_hw.tcl** or **.ipx** file. In the following list of search locations, a recursive descent is annotated by **. A single * signifies any file.

Table 2–1. IP Search Locations

Location	Description
PROJECT_DIR/*	Finds IP components and index files in the Quartus II project directory.
PROJECT_DIR/ip//***	Finds IP components and index files in any subdirectory of the **/ip** subdirectory of the Quartus project directory.

Figure 2–3. Specifying IP Search Locations



Specify both project and global IP search locations. List the search locations in the order you want to search them.

Global IP search directories (all projects)

Global IP search directory:

IP Search Paths:
/users/jbrossar/140test/my_pcie_examples

IP search directories

Project IP search directory:

IP Search Paths:
/users/jbrossar/build185

If the Quartus II software recognizes two IP cores with the same name, the following search path precedence rules determine the resolution of files:

1. Project directory files.
2. Project database directory files.
3. Project libraries specified in **IP Search Locations**, or with the `SEARCH_PATH` assignment in the Quartus II Settings File (**.qsf**).
4. Global libraries specified in **IP Search Locations**, or with the `SEARCH_PATH` assignment in the Quartus II Settings File (**.qsf**).
5. Quartus II software libraries directory, such as <*Quartus II Installation*>**\libraries**.

If you add a component to the search path, you must refresh your system by clicking **File > Refresh** to update the IP Catalog.

Upgrading Outdated IP Cores

IP cores generated with a previous version of the Quartus II software may require upgrade before use in the current version of the Quartus II software. Click **Project > Upgrade IP Components** to identify and upgrade outdated IP cores.

The **Upgrade IP Components** dialog box provides instructions when IP upgrade is required, optional, or unsupported for specific IP cores in your design. Most Altera IP cores support one-click, automatic simultaneous upgrade. You can individually migrate IP cores unsupported by auto-upgrade.

The **Upgrade IP Components** dialog box also reports legacy Altera IP cores that support compilation-only (without modification), as well as IP cores that do not support migration. Replace unsupported IP cores in your project with an equivalent Altera IP core or design logic.Upgrading IP cores changes your original design files.

Before you begin

- Migrate your Quartus II project containing outdated IP cores to the latest version of the Quartus II software. In a previous version of the Quartus II software, click **Project > Archive Project** to save the project. This archive preserves your original design source and project files after migration. le paths in the archive must be relative to the project directory. File paths in the archive must reference the IP variation **.v** or **.vhd** file or **.qsys** file, not the **.qip** file.
- Restore the project in the latest version of the Quartus II software. Click **Project > Restore Archived Project**. Click **Ok** if prompted to change to a supported device or overwrite the project database.

To upgrade outdated IP cores, follow these steps:

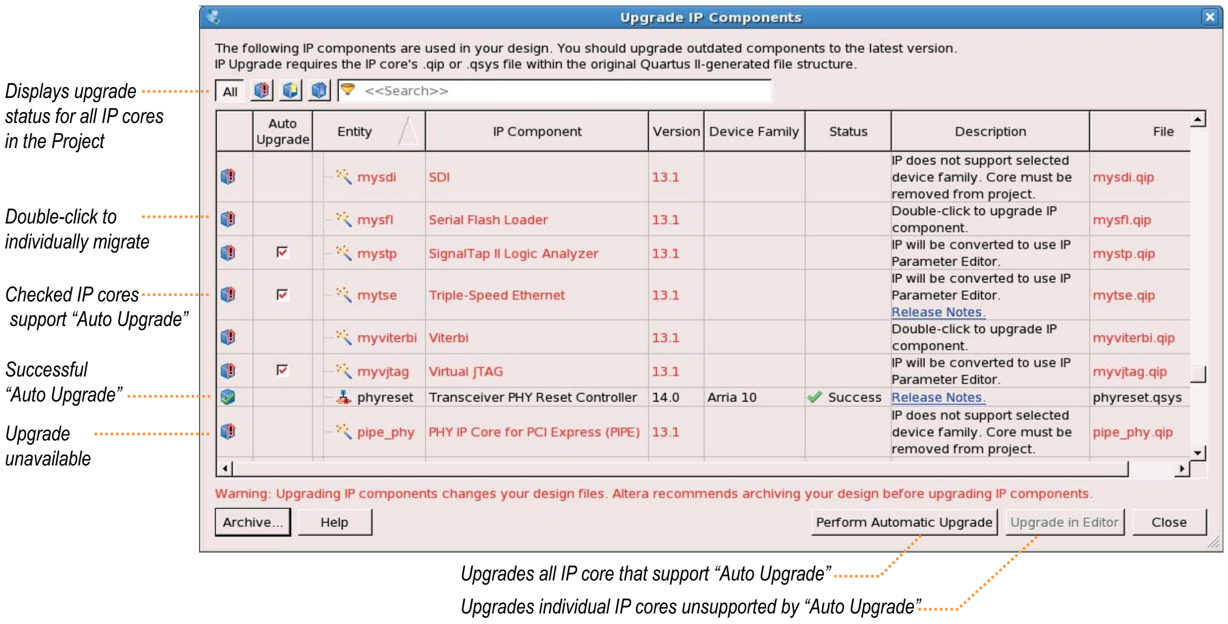
1. In the latest version of the Quartus II software, open the Quartus II project containing an outdated IP core variation.

   ☞ File paths in a restored project archive must be relative to the project directory and you must reference the IP variation **.v** or .**vhd** file or **.qsys** file, not the **.qip** file.

2. Click **Project > Upgrade IP Components**. The **Upgrade IP Components** dialog box displays all outdated IP cores in your project, along with basic instructions for upgrading each core.

3. To simultaneously upgrade all IP cores that support automatic upgrade, click **Perform Automatic Upgrade**. The IP cores upgrade to the latest version. The **Status** and **Version** columns reflect the update.

Figure 2–4. Upgrading IP Cores


Displays upgrade status for all IP cores in the Project
Double-click to individually migrate
Checked IP cores support “Auto Upgrade”
Successful “Auto Upgrade”
Upgrade unavailable
Upgrades all IP core that support “Auto Upgrade”
Upgrades individual IP cores unsupported by “Auto Upgrade”
Upgrade IP Components
All
<<Search>>
Archive...
Help
Perform Automatic Upgrade
Upgrade in Editor
Close

The following IP components are used in your design. You should upgrade outdated components to the latest version.
IP Upgrade requires the IP core's .qip or .qsys file within the original Quartus II-generated file structure.

	Auto Upgrade	Entity	IP Component	Version	Device Family	Status	Description	File
		mysdi	SDI	13.1			IP does not support selected device family. Core must be removed from project.	mysdi.qip
		mysfl	Serial Flash Loader	13.1			Double-click to upgrade IP component.	mysfl.qip
	✓	mystp	SignalTap II Logic Analyzer	13.1			IP will be converted to use IP Parameter Editor.	mystp.qip
	✓	mytse	Triple-Speed Ethernet	13.1			IP will be converted to use IP Parameter Editor. Release Notes.	mytse.qip
		myviterbi	Viterbi	13.1			Double-click to upgrade IP component.	myviterbi.qip
	✓	myvjtag	Virtual JTAG	13.1			IP will be converted to use IP Parameter Editor.	myvjtag.qip
		phyreset	Transceiver PHY Reset Controller	14.0	Arria 10	✓ Success	Release Notes.	phyreset.qsys
		pipe_phy	PHY IP Core for PCI Express (PIPE)	13.1			IP does not support selected device family. Core must be removed from project.	pipe_phy.qip

Warning: Upgrading IP components changes your design files. Altera recommends archiving your design before upgrading IP components.

Upgrading IP Cores at the Command Line

Alternatively, you can upgrade IP cores at the command line. To upgrade a single IP core, type the following command:

```
quartus_sh --ip_upgrade -variation_files <my_ip_path> <project>
```

To upgrade a list of IP cores, type the following command:

```
quartus_sh --ip_upgrade -variation_files
"<my_ip>.qsys;<my_ip>.<hdl>; <project>"
```

IP cores older than Quartus II software version 12.0 do not support upgrade. Altera verifies that the current version of the Quartus II software compiles the previous version of each IP core. The *MegaCore IP Library Release Notes* reports any verification exceptions for MegaCore IP. The *Quartus II Software and Device Support Release Notes* reports any verification exceptions for other IP cores. Altera does not verify compilation for IP cores older than the previous two releases.

DSP Builder Design Flow

DSP Builder shortens digital signal processing (DSP) design cycles by helping you create the hardware representation of a DSP design in an algorithm-friendly development environment.

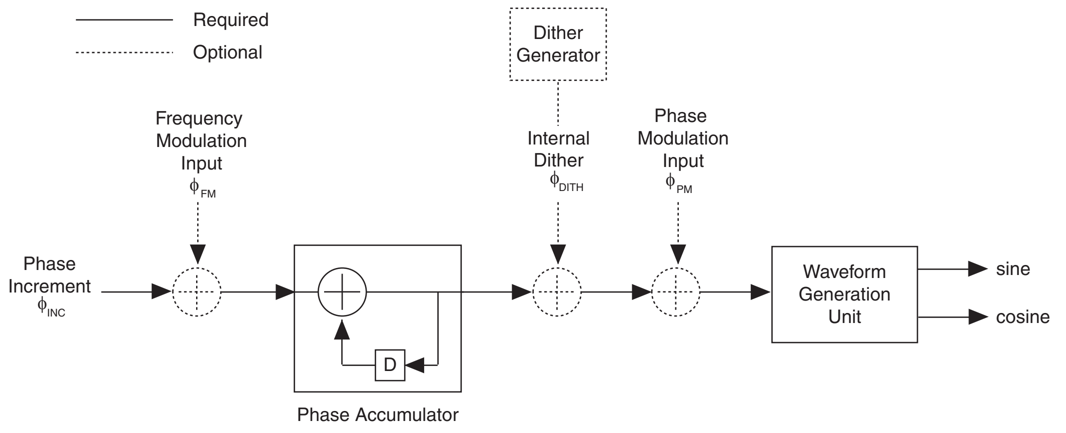
This IP core supports DSP Builder. Use the DSP Builder flow if you want to create a DSP Builder model that includes an IP core variation; use IP Catalog if you want to create an IP core variation that you can instantiate manually in your design.

For more information about the DSP Builder flow, refer to the *Using MegaCore Functions* chapter in the *DSP Builder Handbook*.

3. Functional Description

Figure 3–1 shows a block diagram of a generic NCO.

Figure 3–1. NCO Block Diagram



Required
Optional
Dither Generator
Frequency Modulation Input ϕ_{FM}
Internal Dither ϕ_{DITH}
Phase Modulation Input ϕ_{PM}
Phase Increment ϕ_{INC}
D
Phase Accumulator
Waveform Generation Unit
sine
cosine

The NCO MegaCore function allows you to generate a variety of NCO architectures. Your custom NCO includes both time- and frequency-domain analysis tools. The custom NCO outputs a sinusoidal waveform in two's complement representation.

The waveform for the generated sine wave is defined by the following equation:

$$s(nT) = A\sin\left[2\pi((f_O + f_{FM})nT + \phi_{PM} + \phi_{DITH})\right]$$

where:

- T is the operating clock period
- f_O is the unmodulated output frequency based on the input value ϕ_{INC}
- f_{FM} is a frequency modulating parameter based on the input value ϕ_{FM}
- ϕ_{PM} is derived from the phase modulation input value P and the number of bits (P_{width}) used for this value by the equation: $\phi_{PM} = \frac{P}{2^{P_{width}}}$
- ϕ_{DITH} is the internal dithering value
- A is $2^{N\text{-}1}$ where N is the magnitude precision (and N is an integer in the range 10–32)

The generated output frequency, f_o for a given phase increment, ϕ_{inc} is determined by the following equation:

$$f_o = \frac{\phi_{inc}\, f_{clk}}{2^M}\ \text{Hz}$$

where M is the *accumulator precision* and f_{clk} is the clock frequency

The minimum possible output frequency waveform is generated for the case where $\phi_{inc} = 1$. This case is also the smallest observable frequency at the output of the NCO, also known as the *frequency resolution* of the NCO, f_{res} given in Hz by the following equation:

$$f_{res} = \frac{f_{clk}}{2^M} \text{ Hz}$$

For example, if a 100 MHz clock drives an NCO with an accumulator precision of 32 bits, the frequency resolution of the oscillator is 0.0233 Hz. For an output frequency of 6.25 MHz from this oscillator, you should apply an input phase increment of:

$$\frac{6.25 \times 10^6}{100 \times 10^6} \times 2^{32} = 268435456$$

The NCO MegaCore function automatically calculates this value, using the specified parameters. IP Toolbench also sets the value of the phase increment in all testbenches and vector source files it generates.

Similarly, the generated output frequency, f_{FM} for a given frequency modulation increment, ϕ_{FM} is determined by the following equation:

$$f_{FM} = \frac{\phi_{FM}\, f_{clk}}{2^F} \text{ Hz}$$

where F is the *modulator resolution*

The *angular precision* of an NCO is the phase angle precision before the polar-to-cartesian transformation. The *magnitude precision* is the precision to which the sine and/or cosine of that phase angle can be represented. The effects of reduction or augmentation of the angular, magnitude, accumulator precision on the synthesized waveform vary across NCO architectures and for different f_o/f_{clk} ratios.

You can view these effects in the NCO time and frequency domain graphs as you change the NCO MegaCore function parameters.

Architectures

The NCO MegaCore function supports large ROM, small ROM, CORDIC, and multiplier-based architectures.

Large ROM Architecture

Use the large ROM architecture if your design requires very high speed sinusoidal waveforms and your design can use large quantities of internal memory.

In this architecture, the ROM stores the full 360 degrees of both the sine and cosine waveforms. The output of the phase accumulator addresses the ROM.

The internal memory holds all possible output values for a given angular and magnitude precision. The generated waveform has the highest spectral purity for that parameter set (assuming no dithering). The large ROM architecture also uses the fewest logic elements (LEs) for a given set of precision parameters.

Small ROM Architecture

To reduce LE usage and increase output frequency, use the small ROM architecture.

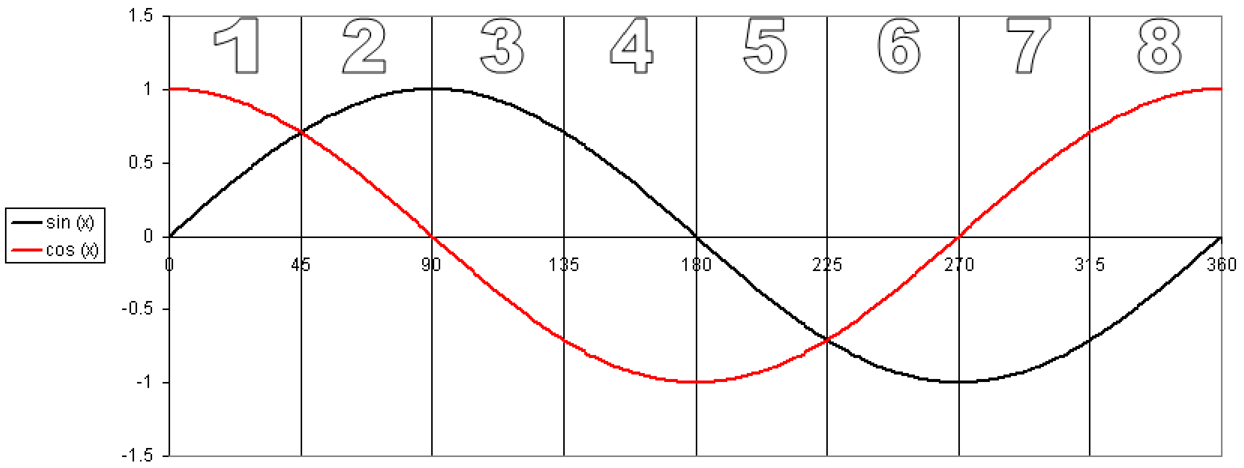
In a small ROM architecture, the device memory only stores 45 degrees of the sine and cosine waveforms. All other output values are derived from these values based on the position of the rotating phasor on the unit circle as shown in Table 3–1 and Figure 3–2.

Table 3–1. Derivation of Output Values

Position in Unit Circle	Range for Phase x	sin(x)	cos(x)
1	0 <= x < p/4	sin(x)	cos(x)
2	p/4 <= x < p/2	cos(p/4x)	sin(p/2-x)
3	p/2 <= x < 3p/4	cos(x-p/2)	-sin(x-p/2)
4	3p/4 <= x < p	sin(p-x)	-cos(p-x)
5	p <= x < 5p/4	-sin(x-p)	-cos(x-p)
6	5p/4 <= x < 3p/2	-cos(3p/2-x)	-sin(3p/2-x)
7	3p/2 <= x < 7p/4	-cos(x-3p/2)	sin(x-3p/2)
8	7p/4 <= x < 2p	-sin(2p-x)	cos(2p-x)

A small ROM implementation is more likely to have periodic value repetition, so the resulting waveform's SFDR is lower than that of the large ROM architecture. However, you can often mitigate this reduction in SFDR by using phase dithering. For information about this option, refer to "Phase Dithering" on page 3–6.

Figure 3–2. Derivation of output Values



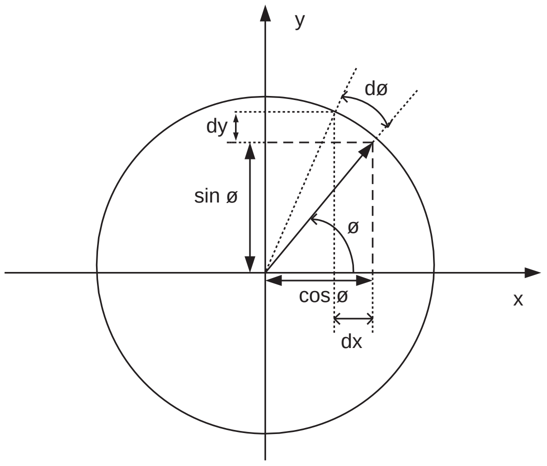
CORDIC Architecture

The CORDIC algorithm, which can calculate trigonometric functions such as sine and cosine, provides a high-performance solution for very-high precision oscillators in systems where internal memory is at a premium.

The CORDIC algorithm is based on the concept of complex phasor rotation by multiplication of the phase angle by successively smaller constants. In digital hardware, the multiplication is by powers of two only. Therefore, the algorithm can be implemented efficiently by a series of simple binary shift and additions/subtractions.

In an NCO, the CORDIC algorithm computes the sine and cosine of an input phase value by iteratively shifting the phase angle to approximate the cartesian coordinate values for the input angle. At the end of the CORDIC iteration, the *x* and *y* coordinates for a given angle represent the cosine and sine of that angle, respectively (Figure 3–3).

Figure 3–3. CORDIC Rotation for Sine & Cosine Calculation



With the NCO MegaCore function, you can select parallel (unrolled) or serial (iterative) CORDIC architectures:

- You an use the parallel CORDIC architecture to create a very high-performance, high-precision oscillator—implemented entirely in logic elements—with a throughput of one output sample per clock cycle. With this architecture, there is a new output value every clock cycle.
- The serial CORDIC architecture uses fewer resources than the parallel CORDIC architecture. However, its throughput is reduced by a factor equal to the magnitude precision. For example, if you select a magnitude precision of *N* bits in the NCO MegaCore function, the output sample rate and the Nyquist frequency is reduced by a factor of *N*. This architecture is implemented entirely in logic elements and is useful if your design requires low frequency, high precision waveforms. With this architecture, the adder stages are stored internally and a new output value is produced every *N* clock cycles.

Multiplier-Based Architecture

The multiplier-based architecture uses multipliers to reduce memory usage. You can choose to implement the multipliers in either:

- Logic elements (Cyclone series of devices) or combinational ALUTs (Stratix series of devices).
- Dedicated multiplier circuitry (for example, dedicated DSP blocks) in device families that support this feature (Stratix V, Stratix IV, Stratix III, Stratix II, Stratix GX, Stratix, or Arria GX devices).

When you specify a dual output multiplier-based NCO, the MegaCore function provides an option to output a sample every two clock cycles. This setting reduces the throughput by a factor of two and halves the resources required by the waveform generation unit.

Table 3–2 summarizes the advantages of each algorithm.

Table 3–2. Architecture Comparison

Architecture	Advantages
Large ROM	Good for high speed and when a large quantity of internal memory is available. Gives the highest spectral purity and uses the fewest logic elements for a given parameterization.
Small ROM	Good for high output frequencies with reduced internal memory usage when a lower SFDR is acceptable.
CORDIC	High performance solution when internal memory is at a premium. The serial CORDIC architecture uses fewer resources than parallel although the throughput is reduced.
Multiplier-Based	Reduced memory usage by implementing multipliers in logic elements or dedicated circuitry.

Multichannel NCOs

The NCO MegaCore function allows you to implement multichannel NCOs. You can generate multiple sinusoids of independent frequency and phase t at a very low cost in additional resources. The waveforms have an output sample-rate of f_{clk}/M where M is the number of channels. You can select 1 to 8 channels.

Multichannel implementations are available for all single-cycle generation algorithms. The input phase increment, frequency modulation value and phase modulation input are input sequentially to the NCO with the input values corresponding to channel 0 first and channel (M–1) last. The inputs to channel 0 should be input on the rising clock edge immediately following the de-assertion of the NCO reset.

On the output side, the first output sample for channel 0 is output concurrent with the assertion of `out_valid` and the remaining outputs for channels 1 to (M–1) are output sequentially. Refer to "Multi-Channel NCO Timing Diagram" on page 3–12 for details of how the data is provided to and received from a multi-channel NCO.

If you select a multichannel implementation, the NCO MegaCore function generates VHDL and Verilog HDL testbenches that time-division-multiplex the inputs into a single stream and demultiplex the output streams into their respective downsampled channelized outputs.

For an example of a multichannel NCO, refer to "Multichannel Design" on page A–1.

Frequency Hopping

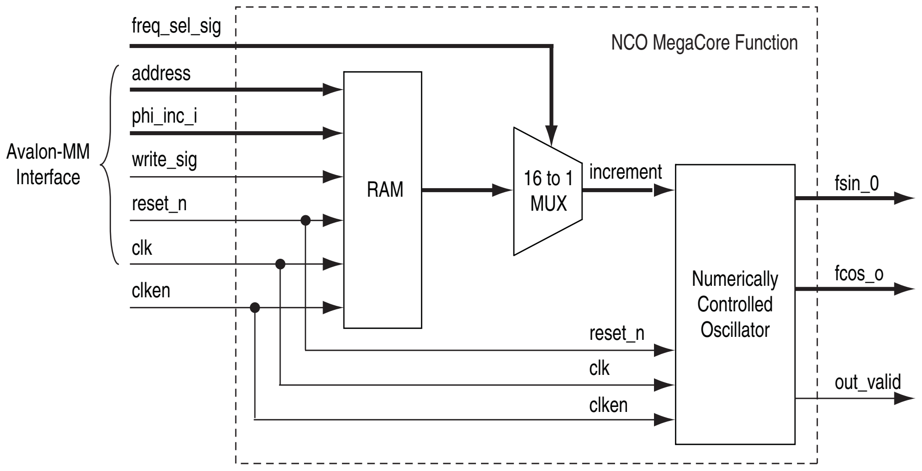
The NCO MegaCore function supports frequency hopping (except the serial CORDIC architecture). Frequency hopping allows control and configuration of the NCO MegaCore function at run time so that carriers with different frequencies can be generated and held for a specified period of time at specified slot intervals.

The MegaCore function supports multiple phase increment registers that you can load using an Avalon-MM bus. You select the phase increment register using an external hardware signal; changes on this signal take effect on the next clock cycle. The maximum number of phase increment registers is 16.

☞ During frequency hopping, the phase of the carrier should not experience discontinuous change. Discontinuous carrier phase changes may cause spectral emission problems.

Figure 3–4 shows the frequency hopping implementation.

Figure 3–4. Frequency Hopping Block Diagram



The RAM stores all hopping frequencies. The RAM size is <*width*>×<*depth*>, where <*width*> is the number of bits required to specify the phase accumulator value to the precision you select in the parameter editor, and <*depth*> is the number of bands you select in the parameter editor.

Phase Dithering

All digital sinusoidal synthesizers suffer from the effects of finite precision, which manifests itself as spurs in the spectral representation of the output sinusoid. Because of angular precision limitations, the derived phase of the oscillator tends to be periodic in time and contributes to the presence of spurious frequencies. You can reduce the noise at these frequencies by introducing a random signal of suitable variance into the derived phase, thereby reducing the likelihood of identical values over time. Adding noise into the data path raises the overall noise level within the oscillator, but tends to reduce the noise localization and can provide significant improvement in SFDR.

The extent to which you can reduce spur levels is dependent on many factors. The likelihood of repetition of derived phase values and resulting spurs, for a given angular precision, is closely linked to the ratio of the clock frequency to the desired output frequency. An integral ratio clearly results in high-level spurious frequencies, while an irrational relationship is less likely to result in highly correlated noise at harmonic frequencies.

The Altera NCO MegaCore function allows you to finely tune the variance of the dither sequence for your chosen algorithm, specified precision, and clock frequency to output frequency ratio, and dynamically view the effects on the output spectrum graphically.

For an example using phase dithering and its effect on the spectrum of the output signal, refer to the "Multichannel Design" on page A–1.

Frequency Modulation

In the NCO MegaCore function, you can add an optional frequency modulator to your custom NCO variation. You can use the frequency modulator to vary the oscillator output frequency about a center frequency set by the input phase increment. This option is useful for applications in which the output frequency is tuned relative to a free-running frequency, for example in all-digital phase-lock-loops.

You can also use the frequency modulation input to switch the output frequency directly.

You can set the frequency modulation resolution input in the NCO MegaCore function. The specified value must be less than or equal to the phase accumulator precision.

The NCO MegaCore function also provides an option to increase the modulator pipeline level; however, the effect of the increase on the performance of the NCO MegaCore function varies across NCO architectures and variations.

Phase Modulation

You can use the NCO MegaCore function to add an optional phase modulator to your MegaCore function variation, allowing dynamic phase shifting of the NCO output waveforms. This option is particularly useful if you want an initial phase offset in the output sinusoid.

You can also use the option to implement efficient phase shift keying (PSK) modulators in which the input to the phase modulator varies according to a data stream. You set the resolution and pipeline level of the phase modulator in the NCO MegaCore function. The input resolution must be greater than or equal to the specified angular precision.

Parameters

The wizard only allows you to select legal combinations of parameters, and warns you of any invalid configurations.

Architecture Parameters

Table 3–3 shows the architecture parameters.

Refer to "Architectures" on page 3–2 for more information about these parameter options.

Table 3–3. Architecture Parameters

Parameter	Value	Description
Generation Algorithm	Small ROM, Large ROM, CORDIC, Multiplier-Based	Select the required algorithm.
Outputs	Dual Output, Single Output	Select whether to use a dual or single output.
Device Family Target	—	Displays the target device family. The target device family is preselected by the value specified in the Quartus II or DSP Builder software. The HDL that is generated for your MegaCore function variation may be incorrect if you change the device family target in this wizard.
Number of Channels	1–8	Select the number of channels when you want to implement a multichannel NCO ("Multichannel NCOs" on page 3–5).
Number of Bands	1–16	Select a number of bands greater than 1 to enable frequency hopping ("Frequency Hopping" on page 3–5). Frequency hopping is not supported in the serial CORDIC architecture.
Use dedicated multipliers	On or off	When the multiplier-based algorithm is selected on the Parameters page, turn on to use dedicated multipliers and select the number of clock cycles per output, otherwise the design uses logic elements. This option is not available if you target the Cyclone device family.
CORDIC Implementation	Parallel, Serial	When you select the CORDIC generation algorithm, you can select a parallel (one output per clock cycle) or serial (one output per 18 clock cycles) implementation.
Clock Cycles Per Output	1, 2.	When the multiplier-based algorithm is selected on the Parameters page, you can select 1 or 2 clock cycles per output.

Frequency Parameters

Table 3–4 shows the frequency parameters.

Refer to "Frequency Modulation" on page 3–7 and "Phase Modulation" on page 3–7 for more information about these parameter options.

Table 3–4. Frequency Parameters (Part 1 of 2)

Parameter	Value	Description
Phase Accumulator Precision	4–64,	Select the required phase accumulator precision. (1)
Angular Resolution	4–24 or 32,	Select the required angular resolution. (2)
Magnitude Precision	10–32,	Select the required magnitude precision.
Implement Phase Dithering	On or Off	Turn on to implement phase dithering ("Phase Dithering" on page 3–6).
Dither Level	Min–Max	When phase dithering is enabled you can use the slider control to adjust the dither level between its minimum and maximum values,

Table 3–4. Frequency Parameters (Part 2 of 2)

Parameter	Value	Description
Clock Rate	1–999 MHz, kHz, Hz, mHz,	Select the clock rate using units of MegaHertz, kiloHertz, Hertz or milliHertz.
Desired Output Frequency	1–999 MHz, kHz, Hz, mHz,	Select the desired output frequency using units of MegaHertz, kiloHertz, Hertz or milliHertz.
Phase Increment Value	—	Displays the phase increment value calculated from the clock rate and desired output frequency.
Real Output Frequency	—	Displays the calculated value of the real output frequency.

Notes to Table 3–3:

(1) The phase accumulator precision must be greater than or equal to the specified angular resolution.

(2) The maximum value is 24 for small and large ROM algorithms; 32 for CORDIC and multiplier-based algorithms.

Optional Ports Parameters

Table 3–5 shows the optional ports parameters.

Table 3–5. Optional Ports Parameters

Parameter	Value	Description
Frequency Modulation input	On or Off	You can optionally enable the frequency modulation input ("Frequency Modulation" on page 3–7).
Modulator Resolution	4–64,	Select the modulator resolution for the frequency modulation input.
Modulator Pipeline Level	1, 2,	Select the modulator pipeline level for the frequency modulation input.
Phase Modulation Input	On or Off	You can optionally enable the phase modulation input ("Phase Modulation" on page 3–7).
Modulator Precision	4–32,	Select the modulator precision for the phase modulation input.
Modulator Pipeline Level	1, 2,	Select the modulator pipeline level for the phase modulation input.

Interfaces

The Avalon-ST interface defines a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface and simplifies the process of controlling the flow of data in a datapath.

Avalon-ST interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries. Such interfaces typically contain data, ready, and valid signals. The NCO MegaCore function is an Avalon-ST source and does not support backpressure.

The Avalon-MM interface provides a means to control the frequency hopping feature at run time.

For more information about the Avalon-MM and Avalon-ST interfaces including integration with other Avalon-ST components which may support backpressure, refer to the *Avalon Interface Specifications*.

Signals

Table 3–6 lists the input and output signals.

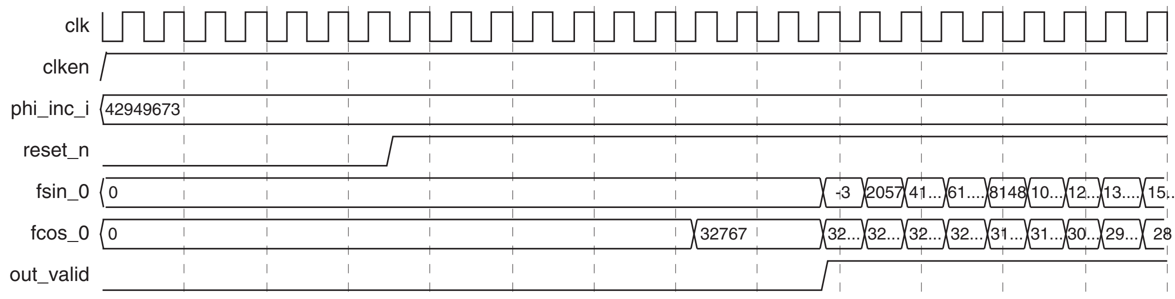
Table 3–6. Signals

Signal	Direction	Description
`address[2:0]`	Input	Address of the 16 phase increment registers when frequency hopping is enabled.
`clk`	Input	Clock.
`clken`	Input	Active-high clock enable.
`freq_mod_i [F-1:0]`	Input	Optional frequency modulation input. You can specify the modulator resolution *F* in IP Toolbench.
`freq_sel[`$\log_2$`N-1:0]`	input	Use to select one of the phase increment registers (that is to select the hopping frequencies), when frequency hopping is enabled. N is the depth.
`phase_mod_i [P-1:0]`	Input	Optional phase modulation input. You can specify the modulator precision *P* in IP Toolbench.
`phi_inc_i [A-1:0]`	Input	Input phase increment. You can specify the accumulator precision *A* in IP Toolbench.
`reset_n`	Input	Active-low asynchronous reset.
`write_sig`	Input	Active-high write signal when frequency hopping is enabled.
`in_data`	Output	In Qsys systems, this Avalon-ST-compliant data bus includes all the Avalon-ST input data signals.
`fcos_o [M-1:0]`	Output	Optional output cosine value (when dual output is selected). You can specify the magnitude precision *M* in IP Toolbench.
`fsin_o [M-1:0]`	Output	Output sine value. You can specify the magnitude precision *M* in IP Toolbench.
`out_valid`	Output	Data valid signal. Asserted by the MegaCore function when there is valid data to output.
`out_data`	Output	In Qsys systems, this Avalon-ST-compliant data bus includes all the Avalon-ST output data signals.

Timing Diagrams

Figure 3–5 shows the timing with a single clock cycle per output sample.

Figure 3–5. Single-Cycle Per Output Timing Diagram

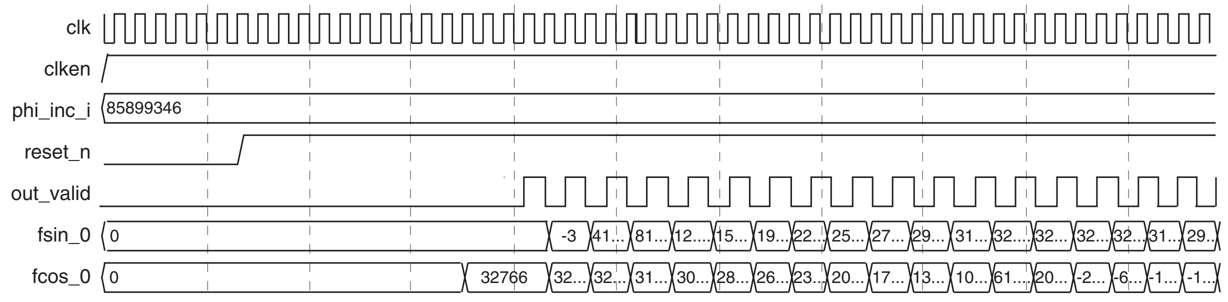


All NCO architectures—except for serial CORDIC and multi-cycle multiplier-based architectures—output a sample every clock cycle. After the clock enable is asserted, the oscillator outputs the sinusoidal samples at a rate of one sample per clock cycle, following an initial latency of *L* clock cycles. The exact value of *L* varies across architectures and parameterizations.

For the non-single-cycle per output architectures, the optional phase and frequency modulation inputs need to be valid at the same time as the corresponding phase increment value. The values should be sampled every 2 cycles for the two-cycle multiplier-based architecture and every *N* cycles for the serial CORDIC architecture, where N is the magnitude precision.

Figure 3–6 shows the timing diagram for a two-cycle multiplier-based NCO architecture.

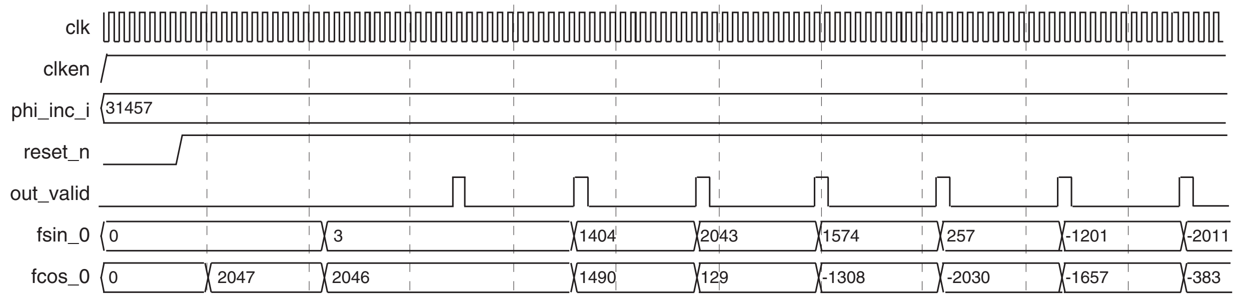
Figure 3–6. Two-Cycle Multiplier-Based Architecture Timing Diagram



After the clock enable is asserted, the oscillator outputs the sinusoidal samples at a rate of one sample for every two clock cycles, following an initial latency of *L* clock cycles. The exact value of *L* depends on the parameters that you set.

Figure 3–7 shows the timing diagram for a serial CORDIC NCO architecture.

Figure 3–7. Serial CORDIC Timing Diagram



The `fsin_0` and `fcos_0` values can change while `out_valid` is low.

After the clock enable is asserted, the oscillator outputs sinusoidal samples at a rate of one sample per *N* clock cycles, where *N* is the magnitude precision set in the NCO MegaCore function. Figure 3–7 shows the case where *N* = 8. The IP core has an initial latency of *L* clock cycles; the exact value of *L* depends on the parameters that you set.

Table 3–7 shows typical latency values for the different architectures.

Table 3–7. Latency Values

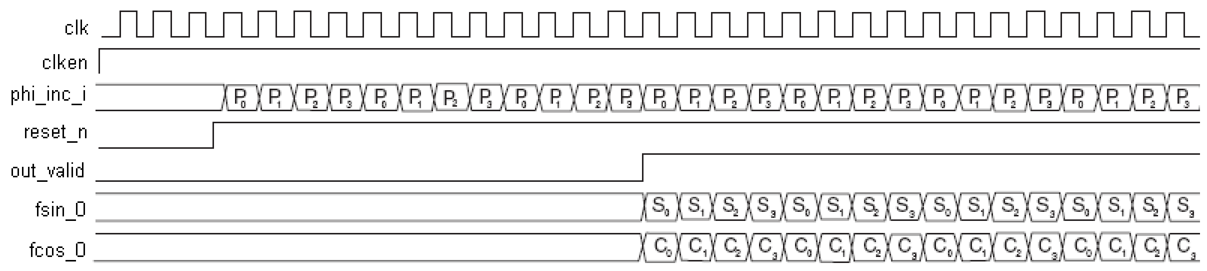
Architecture	Variation	Latency [2], [3]		
		Base	Minimum	Maximum
Small ROM	all	7	7	13
Large ROM	all	4	4	10
Multiplier-Based	Throughput = 1, Logic cells	11	11	17
Multiplier-Based	Throughput = 1, Dedicated, Special case [1]	8	8	14
Multiplier-Based	Throughput = 1, Dedicated, Not special case	10	10	16
Multiplier-Based	Throughput = 1/2	15	15	26
CORDIC	Parallel	2N + 4	20 [4]	74 [5]
CORDIC	Serial CORDIC	2N + 2	18 [4]	258 [5]

Notes for Table 3–7:

(1) Special case: (9 <= *N* <= 18 && WANT_SIN_AND_COS).
(2) Latency = base latency + dither latency+ frequency modulation pipeline + phase modulation pipeline (×*N* for serial CORDIC).
(3) Dither latency = 0 (dither disabled) or 2 (dither enabled).
(4) Minimum latency assumes *N* = 8.
(5) Maximum latency assumes *N* = 32

Figure 3–8 shows the timing diagram for a multi-channel NCO in the case where the number of channels, M is set to 4. The input phase increments for each channel, P_k are interleaved and loaded sequentially.

Figure 3–8. *Multi-Channel NCO Timing Diagram*



The phase increment for channel 0 is the first value read in on the rising edge of the clock following the de-assertion of `reset_n` (assuming `clken` is asserted) followed by the phase increments for the next (M-1) channels. The output signal `out_valid` is asserted when the first valid sine and cosine outputs for channel 0, S_0, C_0, respectively are available.

The output values S_k and C_k corresponding to channels 1 through (M-1) are output sequentially by the NCO. The outputs are interleaved so that a new output sample for channel k is available every M cycles.

A. Multichannel Design Example

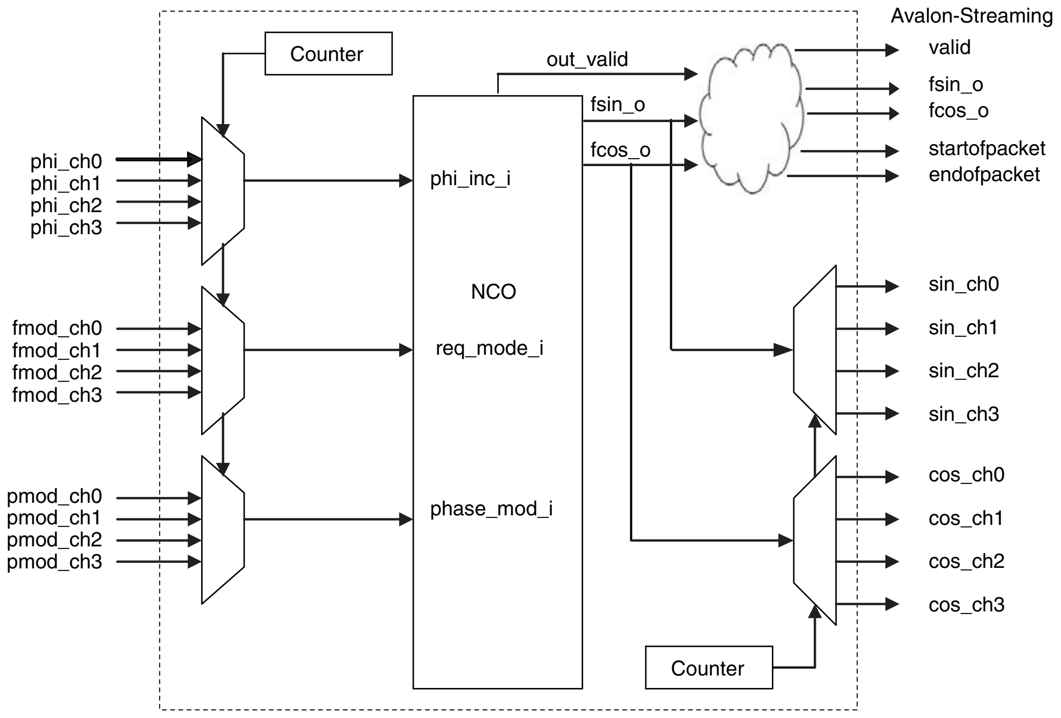
Multichannel Design

Often in a system where the clock frequency of the design is much higher than the sampling frequency, you can time share some of the hardware.

Consider a system with a clock frequency of 200 MHz and a sampling rate of 50 MSPS (Megasamples per second). You can generate four complex sinusoids using a single instance of the NCO MegaCore function.

Example design 3 generates four multiplexed and demultiplexed streams of complex sinusoids, which you can use in a digital up- or down-converter design (Figure A–1).

Figure A–1. Multichannel NCO Example Design



The design also generates five output signals (`valid`, `startofpacket`, `endfopacket`, `fsin_o` and `fcos_o`) that the Avalon-ST interface uses (Figure A–1.

The following directories contain separate top-level design files (named **multichannel_example.v** and **multichannel_example.vhd**) for Verilog HDL and VHDL in the directories:

***<IP install path>*\nco\example_designs\multi_channel\verilog**

***<IP install path>*\nco\example_designs\multi_channel\vhdl**

Opening the Multichannel Design Example

To open the multichannel example design:

1. Browse to the appropriate example design directory. Choose between VHDL and Verilog HDL files.
2. Create a new Quartus II project in the example design directory.
3. Add the Verilog HDL or VHDL files to the project and specify the top level entity to be `multichannel_example`.
4. On the Tools menu, click **MegaWizard Plug-In Manager**. In the **MegaWizard Plug-In Manager** dialog box, select **Edit an existing custom megafunction variation** and select the **nco.vhd** file with Megafunction name **NCO**.
5. Click **Next** to display IP Toolbench, Click **Parameterize** to review the parameters, then click **Generate**.
6. Open the ModelSim simulator, and change the directory to the appropriate multiple channel example design **verilog** or **vhdl** directory.
7. Select **TCL > Execute Macro** from the Tools menu in ModelSim. Select the **multichannel_example_ver_msim.tcl** script for the Verilog HDL design or the **multichannel_example_vhdl_msim.tcl** script for the VHDL design.
8. Observe the behavior of the design in the ModelSim Wave window.

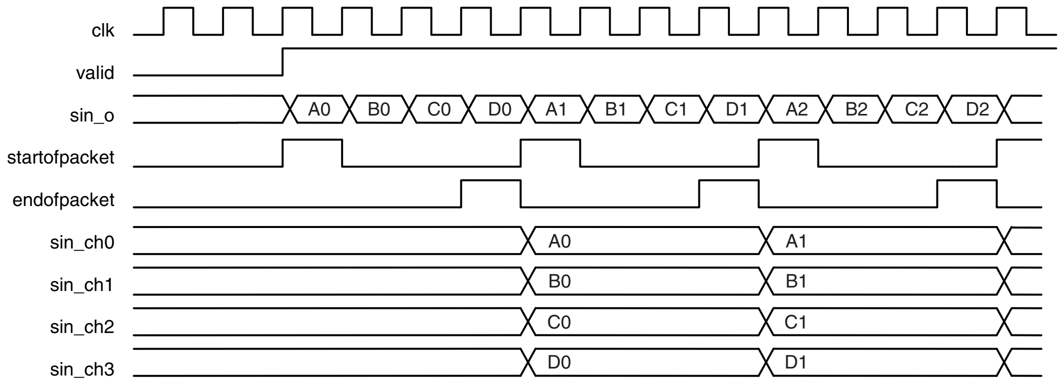
NCO Design Example Specification

The NCO meets the following specifications:

- SFDR: 110 dB
- Output Sample Rate: 200 MSPS (50 MSPS per channel)
- Output Frequency: 5MHz, 2MHz, 1MHz, 500KHz
- Output Phase: 0, $\pi/4$, $\pi/2$, π
- Frequency Resolution: 0.047 Hz
- Clock rate = 200MHz clock rate
- Number of channels = 4.
- Output sample-rate = $f_{\mathrm{clk}}/4$.
- Maximum output clock frequency = 50MHz.

The output signal has only one sample for a cycle. Figure A–2 shows the timing relationship between Avalon-ST signals, a generated multiplexed signal stream and demultiplexed signal streams.

Figure A–2. Multi-Channel NCO Output Signals

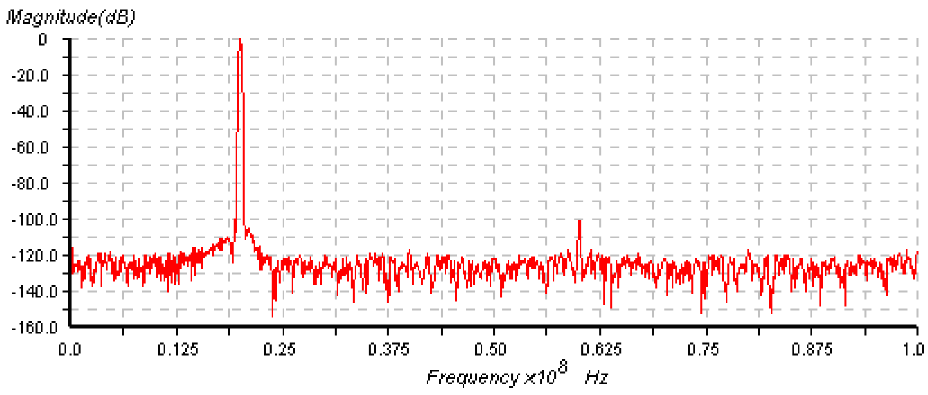


Design Example Parameters

To meet the specification, the design uses the following parameters:

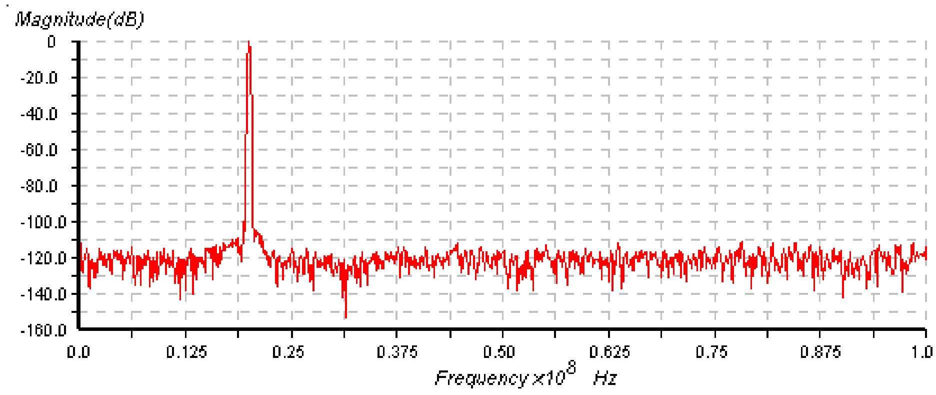
- Multiplier-based algorithm. By using the dedicated multiplier circuitry in Stratix devices, the NCO architectures that implement this algorithm can provide very high performance.
- Clock rate of 200 MHz and 32-bit phase accumulator precision to give a frequency resolution of 47 mHz.
- Angular and magnitude precision settings give an SFDR of approximately 100.05 dB to meet the SFDR requirement, while minimizing the required device resources. Figure A–3 shows the spectrum when you set the angular precision to 17 bits and the magnitude precision to 18 bits results in the spectrum.

Figure A–3. Spectrum After Setting Angular and Magnitude Precision



- Dither level to increase the variance of the dithering sequence until the design reaches the trade-off point between spur reduction and noise level augmentation. At a dithering level of 3, the SFDR is approximately 110.22 dB, which exceeds the specification (Figure A–4).

Figure A–4. Spectrum After the Addition of Dithering



- The frequency modulation input allows an external frequency for modulating the input signal. The modulator resolution is 32 bits and the modulator pipeline level is 1.
- A phase modulation input, which is necessary with 32 bits for modulator precision and the modulator pipeline level is 1.
- Dual output for generating both the sine and cosine outputs.
- Four multichannels.

Simulation Specification

The ModelSim simulation script generates signals with different frequencies and phases in four separate channels. Table A–1 lists the parameter settings to generate the required signals in four separate channels.

Table A–1. ModelSim Simulation Map

Channel	Generated Signal		Settings		
	Frequency (MHz)	Phase	f_0 (MHz)	f_{MOD} (MHz)	p_{MOD}
0	5	0	5	0	0
1	2	$\pi/4$	0.5	1.5	$\pi/4$
2	1	$\pi/2$	0.1	0.9	$\pi/2$
3	0.5	p	0.01	0.49	p

Additional Information

This chapter provides additional information about the document and Altera.

Revision History

The following table displays the revision history for this user guide.

Date	Version	Changes Made
December 2014	14.1	■ Added full support for Arria 10 and MAX 10 devices ■ Reordered parameters tables to match wizard
August 2014	14.0 Arria 10 Edition	■ Added support for Arria 10 devices. ■ Added new `in_data` and `out_data` bus descriptions. ■ Added Arria 10 generated files description. ■ Removed table with generated file descriptions.
June 2014	14.0	■ Removed device support for Cyclone III and Stratix III devices ■ Added support for MAX 10 FPGAs. ■ Added instructions for using IP Catalog
November 2013	13.1	■ Removed support for the following devices: ■ Arria ■ Cyclone II ■ HardCopy II, HardCopy III, and HardCopy IV ■ Stratix, Stratix II, Stratix GX, and Stratix II GX ■ Added full support for the following devices: ■ Arria V ■ Stratix V
November 2012	12.1	Added support for Arria V GZ devices.
May 2011	11.0	■ Updated support level to final support for Arria II GX, Arria II GZ, Cyclone III LS, and Cyclone IV GX devices. ■ Updated support level to HardCopy Compilation for HardCopy III, HardCopy IV E, and HardCopy IV GX devices.
December 2010	10.1	■ Added preliminary support for Arria II GZ devices. ■ Updated support level to final support for Stratix IV GT devices.
July 2010	10.0	■ Added preliminary support for Stratix V devices.
November 2009	9.1	■ Added parameter editor support for frequency hopping feature. ■ Removed frequency hopping design example. ■ Preliminary support for Cyclone III LS, Cyclone IV, and HardCopy IV GX devices.
March 2009	9.0	■ Preliminary support for Arria® II GX device family. ■ Added new frequency hopping design example.

Date	Version	Changes Made
November 2008	8.1	■ Full support for Stratix® III device family. ■ Replaced old design examples by new multichannel design. ■ Applied new documentation style. ■ Withdrawn support for UNIX.
May 2008	8.0	■ Separated the design flows and parameter setting sections. ■ Full support for Cyclone® III device family. ■ Preliminary support for Stratix IV device family.
October 2007	7.2	■ Updated NCO block diagram. ■ Added multi-channel description and timing diagram. ■ Added latency table. ■ Updated GUI screenshots. ■ Full support for Arria GX device family.
May 2007	7.1	■ Added 32-bit precision for angle & magnitude. ■ Preliminary support for Arria GX device family. ■ Full support for Stratix II GX and HardCopy II devices.
December 2006	7.0	■ Preliminary support for Cyclone III device family.
December 2006	6.1	■ Preliminary support for Stratix III device family. ■ Minor updates throughout the user guide.
April 2006	2.3.1	■ Maintenance release; updated screen shots and format
October 2005	2.3.0	■ Maintenance release; updated screen shots and format. ■ Preliminary support for HardCopy® II, HardCopy Stratix, and Stratix II GX device families. ■ Removed Mercury and Excalibur device support.
June 2004	2.2.0	■ Added Cyclone II support. ■ Updated functional description, tutorial instructions and screenshots.
February 2004	2.1.0	■ Enhancements include support for Stratix II devices; support for easy-to-use IP Toolbench; IP functional simulation models for use in Altera®-supported VHDL and Verilog HDL simulators; support for UNIX and Linux operating systems.
November 2002	2.0.2	■ Updated the screen shots; made some formatting and organization changes; minor wording changes to several sections.
July 2002	2.0.1	■ NCO MegaCore functions now display a single DSP Builder library for OpenCore and OpenCore Plus in the Simulink Library Browser.
May 2002	2.0.0	■ Updated functional description. Added DSP Builder, OpenCore Plus, and licensing information. Removed reference designs and replaced with example designs. Updated all screen shots. Made formatting and organization changes.
April 2000	1.0	■ Version 1.0 of this user guide.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact [1]	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, <*file name*> and <*project name*>**.pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
☞	The hand points to information that requires special attention.

Visual Cue	Meaning
❓	The question mark directs you to a software help system with related information.
👣	The feet direct you to another document or website with related information.
🎥	The multimedia icon directs you to a related multimedia presentation.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
⚠ WARNING	A warning calls attention to a condition or possible situation that can cause you injury.
✉	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.