Contents

About this User Guide
Revision History .............................................................................................................. v
How to Contact Altera ............................................................................................................................... v
Typographic Conventions....................................................................................................................... vi

Chapter 1. About this Compiler
Release Information....................................................................................................................... 1-1
Device Family Support ....................................................................................................................... 1-1
Introduction ............................................................................................................................................. 1-2
New in Version 2.1.0 ....................................................................................................................... 1-2
Features .................................................................................................................................................... 1-2
General Description....................................................................................................................... 1-3
    OpenCore Plus Evaluation............................................................................................................... 1-4
Performance.............................................................................................................................................. 1-4

Chapter 2. Getting Started
System Requirements....................................................................................................................... 2-1
Design Flow .............................................................................................................................................. 2-1
Download & Install the NCO Compiler.............................................................................................. 2-2
    Download the NCO Compiler MegaCore Function ........................................................................ 2-2
    Install the NCO Compiler MegaCore Function Files ....................................................................... 2-3
        Windows ....................................................................................................................................... 2-3
        Solaris ............................................................................................................................................. 2-3
        Linux ............................................................................................................................................ 2-3
Directory Structure............................................................................................................................ 2-4
NCO Compiler Tutorial....................................................................................................................... 2-4
    Create a New Quartus II Project ..................................................................................................... 2-5
    Launch IP Toolbench ....................................................................................................................... 2-6
    Step 1: Parameterize ....................................................................................................................... 2-7
    Step 2: Set Up Simulation ................................................................................................................ 2-13
    Step 3: Generate ............................................................................................................................... 2-15
Simulate the Design........................................................................................................................... 2-18
    Simulating the Design in ModelSim ................................................................................................ 2-19
    Compile the Design ............................................................................................................................ 2-19
Program a Device ............................................................................................................................... 2-19
    Set Up Licensing ............................................................................................................................... 2-20
Append the License to Your License File .......................................................................................... 2-20
Specify the License File in the Quartus II Software ............................................................................. 2-21
## Chapter 3. Specifications

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to NCOs</td>
<td>3-1</td>
</tr>
<tr>
<td>Spectral Purity</td>
<td>3-1</td>
</tr>
<tr>
<td>Maximum Output Frequency</td>
<td>3-2</td>
</tr>
<tr>
<td>Functional Description</td>
<td>3-3</td>
</tr>
<tr>
<td>Architectures</td>
<td>3-5</td>
</tr>
<tr>
<td>Large ROM Architecture</td>
<td>3-5</td>
</tr>
<tr>
<td>Small ROM Architecture</td>
<td>3-5</td>
</tr>
<tr>
<td>CORDIC Architecture</td>
<td>3-5</td>
</tr>
<tr>
<td>Multiplier-Based Architecture</td>
<td>3-7</td>
</tr>
<tr>
<td>Frequency Modulation</td>
<td>3-7</td>
</tr>
<tr>
<td>Phase Modulation</td>
<td>3-7</td>
</tr>
<tr>
<td>Phase Dithering</td>
<td>3-8</td>
</tr>
<tr>
<td>Timing Diagrams</td>
<td>3-8</td>
</tr>
<tr>
<td>OpenCore Plus Time-Out Behavior</td>
<td>3-9</td>
</tr>
<tr>
<td>Signals</td>
<td>3-10</td>
</tr>
<tr>
<td>MegaCore Verification</td>
<td>3-10</td>
</tr>
</tbody>
</table>

## Appendix A. Example Designs

| Example Design 1                                                        | A-1  |
| Example Design 2                                                        | A-2  |
Revision History

The table below displays the revision history for this User Guide.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 2004</td>
<td>2.1.0</td>
<td>● New document for product version 2.1.0. Enhancements to the product 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 Solaris and Linux operating systems.</td>
</tr>
<tr>
<td>November 2002</td>
<td>2.0.2</td>
<td>● Updated the screen shots; made some formatting and organization changes; minor wording changes to several sections.</td>
</tr>
<tr>
<td>July 2002</td>
<td>2.0.1</td>
<td>● Minor modifications for product version 2.0.1. MegaCore now displays a single DSP Builder library for OpenCore® and OpenCore Plus in the Simulink Library Browser.</td>
</tr>
<tr>
<td>May 2002</td>
<td>2.0.0</td>
<td>● 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.</td>
</tr>
<tr>
<td>April 2000</td>
<td>1.0</td>
<td>● Version 1.0 of the user guide.</td>
</tr>
</tbody>
</table>

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

<table>
<thead>
<tr>
<th>Information Type</th>
<th>USA &amp; Canada</th>
<th>All Other Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)</td>
<td>(408) 544-7000 (1) (7:00 a.m. to 5:00 p.m. Pacific Time)</td>
</tr>
<tr>
<td>Product literature</td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
</tr>
<tr>
<td>Altera literature services</td>
<td><a href="mailto:lit_req@altera.com">lit_req@altera.com</a> (1)</td>
<td><a href="mailto:lit_req@altera.com">lit_req@altera.com</a> (1)</td>
</tr>
<tr>
<td>Non-technical customer service</td>
<td>(800) 767-3753 (7:00 a.m. to 5:00 p.m. Pacific Time)</td>
<td>(408) 544-7000 (7:00 a.m. to 5:00 p.m. Pacific Time)</td>
</tr>
<tr>
<td>FTP site</td>
<td>ftp.altera.com</td>
<td>ftp.altera.com</td>
</tr>
</tbody>
</table>

Note to table:
(1) You can also contact your local Altera sales office or sales representative.
This document uses the typographic conventions shown below.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold Type with Initial Capital Letters</strong></td>
<td>Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: <strong>Save As</strong> dialog box.</td>
</tr>
<tr>
<td><strong>bold type</strong></td>
<td>External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: \textit{f}_{MAX}, \texttt{\qdesigns} directory, \texttt{d:} drive, \texttt{chiptrip.pdf} file.</td>
</tr>
<tr>
<td><em>Italic Type with Initial Capital Letters</em></td>
<td>Document titles are shown in italic type with initial capital letters. Example: <em>AN 75: High-Speed Board Design.</em></td>
</tr>
<tr>
<td><em>Italic type</em></td>
<td>Internal timing parameters and variables are shown in italic type. Examples: \textit{t}_{PIA}, \textit{n} + 1.</td>
</tr>
<tr>
<td></td>
<td>Variable names are enclosed in angle brackets (&lt; &gt;) and shown in italic type. Example: \texttt{&lt;file name&gt;}, \texttt{&lt;project name&gt;.pof} file.</td>
</tr>
<tr>
<td>Initial Capital Letters</td>
<td>Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.</td>
</tr>
<tr>
<td>“Subheading Title”</td>
<td>References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”</td>
</tr>
<tr>
<td>Courier type</td>
<td>Signal and port names are shown in lowercase Courier type. Examples: \texttt{data1}, \texttt{tdi, input}. Active-low signals are denoted by suffix \texttt{n}, e.g., \texttt{resetn}.</td>
</tr>
<tr>
<td></td>
<td>Anything that must be typed exactly as it appears is shown in Courier type. For example: \texttt{c:\qdesigns\tutorial\chiptrip.pdf}. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword \texttt{SUBDESIGN}), as well as logic function names (e.g., \texttt{TRI}) are shown in Courier.</td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., etc.</td>
<td>Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.</td>
</tr>
<tr>
<td>• •</td>
<td>Bullets are used in a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td>✓</td>
<td>The checkmark indicates a procedure that consists of one step only.</td>
</tr>
<tr>
<td>□</td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td>⚠</td>
<td>The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.</td>
</tr>
<tr>
<td>⚠</td>
<td>The warning indicates information that should be read prior to starting or continuing the procedure or processes</td>
</tr>
<tr>
<td>⇐</td>
<td>The angled arrow indicates you should press the Enter key.</td>
</tr>
<tr>
<td>Ⓒ</td>
<td>The feet direct you to more information on a particular topic.</td>
</tr>
</tbody>
</table>
Chapter 1. About this Compiler

Release Information

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2.1.0</td>
</tr>
<tr>
<td>Release Date</td>
<td>February 2004</td>
</tr>
<tr>
<td>Ordering Code</td>
<td>IP-NCO</td>
</tr>
<tr>
<td>Product ID(s)</td>
<td>0014</td>
</tr>
<tr>
<td>Vendor ID(s)</td>
<td>6AF7</td>
</tr>
</tbody>
</table>

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera device families, as described below:

- **Full support** means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs.
- **Preliminary support** means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

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

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix™ II</td>
<td>Full</td>
</tr>
<tr>
<td>Stratix GX</td>
<td>Full</td>
</tr>
<tr>
<td>Stratix</td>
<td>Full</td>
</tr>
<tr>
<td>Cyclone™</td>
<td>Full</td>
</tr>
<tr>
<td>Mercury™</td>
<td>Full</td>
</tr>
<tr>
<td>Excalibur™</td>
<td>Full</td>
</tr>
</tbody>
</table>
Introduction

The Altera NCO Compiler MegaCore function generates numerically controlled oscillators (NCOs) customized for Altera devices. You can use the IP Toolbench interface to implement a variety of NCO architectures, including ROM-based, CORDIC-based, and multiplier-based. IP Toolbench also includes time and frequency domain graphs that dynamically display the functionality of the NCO based on your parameter settings.

New in Version 2.1.0

- Support for Stratix II devices
- Support for easy-to-use IP Toolbench interface
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for Windows, Solaris, and Linux operating systems

Features

- Support for MATLAB version 6.5
- Supports multiple NCO architectures:
  - Multiplier-based implementation using Stratix II, Stratix GX, and Stratix DSP blocks or logic elements (LEs), (single cycle and multi-cycle)
  - Parallel/serial CORDIC-based implementation using LEs with multiple pipeline levels
  - ROM-based implementation using device embedded array blocks (EABs), embedded system blocks (ESBs), or external ROM
- Supports single or dual outputs (sine/cosine)
- Allows variable width frequency modulation input
- Allows variable-width phase modulation input
- User-defined frequency resolution, angular precision, and magnitude precision
- Generates simulation files and architecture-specific testbenches
  - VHDL testbench
  - Verilog HDL testbench
  - MATLAB model and testbench
  - Quartus® II Vector Files

Table 1–2. Device Family Support (Part 2 of 2)

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACEX® 1K</td>
<td>Full</td>
</tr>
<tr>
<td>APEX™ II</td>
<td>Full</td>
</tr>
<tr>
<td>APEX 20KE &amp; APEX 20KC</td>
<td>Full</td>
</tr>
<tr>
<td>APEX 20K</td>
<td>Full</td>
</tr>
<tr>
<td>Other device families</td>
<td>No support</td>
</tr>
</tbody>
</table>

Table 1–2. Device Family Support (Part 2 of 2)
Includes dual-output oscillator and quaternary frequency shift keying (QFSK) modulator example designs

**General Description**

A numerically controlled oscillator (NCO) synthesizes a discrete-time, discrete-valued representation of a sinusoidal waveform. Designers typically use NCOs in communication systems. In such systems, they are used 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 (see Figure 1–1).

Designers also use NCOs in all-digital phase-locked-loops 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.

**Figure 1–1. Simple Modulator**

![Simple Modulator Diagram](image-url)
OpenCore Plus Evaluation

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

- Simulate the behavior of a MegaCore function 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 MegaCore functions
- Program a device and verify your design in hardware

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

For more information on OpenCore Plus hardware evaluation using the NCO Compiler, see “OpenCore Plus Time-Out Behavior” on page 3–9 and AN 320: OpenCore Plus Evaluation of Megafunctions.

Performance

Table 1–3 provides performance statistics for the NCO function implemented in a Stratix II EP2S15F484C3 device.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accumulator Width</th>
<th>Angular Precision</th>
<th>Magnitude Precision</th>
<th>LEs (1)</th>
<th>M4K RAM</th>
<th>DSP Blocks</th>
<th>f_MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier-Based</td>
<td>32</td>
<td>16</td>
<td>18</td>
<td>67</td>
<td>3</td>
<td>1</td>
<td>350.39</td>
</tr>
<tr>
<td>Small ROM</td>
<td>32</td>
<td>14</td>
<td>16</td>
<td>219</td>
<td>15</td>
<td>0</td>
<td>345.30</td>
</tr>
<tr>
<td>Parallel CORDIC</td>
<td>32</td>
<td>14</td>
<td>14</td>
<td>1,623</td>
<td>0</td>
<td>0</td>
<td>385.06</td>
</tr>
<tr>
<td>Large ROM</td>
<td>32</td>
<td>12</td>
<td>12</td>
<td>56</td>
<td>24</td>
<td>0</td>
<td>350.51</td>
</tr>
</tbody>
</table>

Note to Table 1:
(1) The Quartus II software reports the number of adaptive look-up tables (ALUTs) that the design uses in Stratix II devices. The LE count is based on this number of ALUTs.
Chapter 2. Getting Started

System Requirements

The instructions in this section require the following hardware and software:

- A PC running the Windows NT/2000/XP or Red Hat Linux 7.3 or 8.0 operating system; or a Sun workstation running the Solaris 7 or 8 operating system
- Quartus® II software version 4.0
- Altera®-supported VHDL or Verilog HDL simulator (optional)

Design Flow

To evaluate the NCO Compiler MegaCore® function using the OpenCore Plus® feature, the design flow involves the following steps:

1. Download and install the NCO Compiler MegaCore function.
2. Create a custom variation of the NCO Compiler MegaCore function using IP Toolbench.
   - IP Toolbench is a toolbar from which you can quickly and easily view documentation, specify parameters, and generate all of the files necessary for integrating the parameterized MegaCore function into your design. You can launch IP Toolbench from within the Quartus® II software.
3. Implement the rest of your design using the design entry method of your choice.
4. Use the IP functional simulation model, generated by IP Toolbench, to verify the operation of your design.

   For more information on IP functional simulation models, refer to the white paper entitled, Using IP Functional Simulation Models to Verify Your System Design.
5. Use the Quartus II software to compile your design.
   - You can generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of your design in hardware for a limited time. Refer to AN 320: OpenCore Plus Evaluation of Megafonctions.
6. Purchase a license for the NCO Compiler MegaCore function.

Once you have purchased a license for the NCO Compiler, the design flow involves the following additional steps:

1. Set up licensing.
2. Generate a programming file for the Altera device(s) on your board.
3. Program the Altera device(s) with the completed design.
4. Perform complete system verification.

Download & Install the NCO Compiler

Before you can start using Altera MegaCore functions, you must obtain the MegaCore files and install them on your computer. The following instructions describe this process.

Download the NCO Compiler MegaCore Function

If you have Internet access, you can download MegaCore functions from Altera’s web site at www.altera.com. Follow the instructions below to obtain the NCO Compiler via the Internet. If you do not have Internet access, you can obtain the NCO Compiler from your local Altera representative.

1. Point your web browser to www.altera.com/ipmegastore.
2. Type NCO Compiler in the IP MegaSearch box.
3. Click Go.
4. Choose NCO Compiler from the search results page. The product description web page displays.
5. Click Download Free Evaluation on the product description web page.
6. Complete the registration form and click Submit Request.
7. Read the Altera MegaCore license agreement, turn on the I have read the license agreement check box, and click Proceed to Download Page.
8. Follow the instructions on the NCO Compiler download and installation page to download the MegaCore function.
Install the NCO Compiler MegaCore Function Files

The following instructions describe how you install the NCO Compiler on computers running the Windows, Solarix, or Linux operating systems.

**Windows**

To install the NCO Compiler on a PC running the Windows operating system, follow these steps:

1. Choose Run (Start menu).
2. Type `<path>\nco-v<version>.exe` where `<path>` is the location of the downloaded NCO Compiler, and `<version>` is the version of the MegaCore function.
3. Click OK. The NCO Compiler Installation dialog box appears. Follow the on-screen instructions to finish installation.

**Solaris**

To install the NCO Compiler on a computer running the Solaris operating systems, follow these steps:

1. Decompress the package by typing the following command:
   ```
   gunzip -d nco-v<version>_solaris.tar.gz
   ```
2. Extract the package contents by typing the following command:
   ```
   tar nco-v<version>_solaris.tar
   ```

**Linux**

To install the NCO Compiler on a computer running the Linux operating systems, follow these steps:

1. Decompress the package by typing the following command:
   ```
   gunzip -d nco-v<version>_linux.tar.gz
   ```
2. Extract the package contents by typing the following command:
   ```
   tar nco-v<version>_linux.tar
   ```
Directory Structure

Figure 2–1 shows the directory structure for the NCO Compiler where <path> is the installation directory.

Figure 2–1. Directory Structure

NCO Compiler Tutorial

This tutorial explains how to create a custom variation of the NCO MegaCore function using IP Toolbench and the Quartus II software on a PC running Windows. As you go through IP Toolbench, each step is described in detail. When you are finished generating a custom variation of the NCO MegaCore function, you can incorporate it into your overall project.

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

This tutorial consists of the following steps:

■ “Create a New Quartus II Project” on page 2–5
■ “Launch IP Toolbench” on page 2–6
■ “Step 1: Parameterize” on page 2–7
■ “Step 2: Set Up Simulation” on page 2–13
■ “Step 3: Generate” on page 2–15
Create a New Quartus II Project

Before you begin, you must create a new Quartus II project. With the New Project Wizard, you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. You will also specify the NCO Compiler user library. To create a new project, follow these steps:

1. Choose Programs > Altera > Quartus II <version> (Windows Start menu) to run the Quartus II software.

2. Choose New Project Wizard (File menu).

3. Click Next in the introduction (the introduction will not display if you turned it off previously).

4. Specify the working directory for your project. This tutorial uses the directory c:\qdesigns\nco_example.

5. Specify the name of the project. This tutorial uses NCO_DIR.

6. Click Next.

7. Click User Library Pathnames.

8. Type <path>nco-v<version>lib into the Library name box, where <path> is the directory in which you installed the NCO Compiler. The default installation directory is c:\MegaCore.

9. Click Add.

10. Click OK.

11. Click Next.

12. Click Next.

13. In the Family list, select Stratix™II. Under Do you want to select a specific device? select No, I want to allow the compiler to choose a device.

14. Click Finish.

You have finished creating your new Quartus II project.
Launch IP Toolbench

To launch IP Toolbench, follow these steps:

1. Start the MegaWizard® Plug-In Manager by choosing MegaWizard Plug-In Manager (Tools menu). The MegaWizard Plug-In Manager dialog box is displayed.

   [Refer to the Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.]

2. Specify that you want to create a new custom megafunction variation and click Next.

3. Expand the Signal Processing > Signal Generation directory under Installed Plug-Ins by clicking the + icon next to the name.

4. Select NCO v2.1.0.

5. Choose the output file type for your design; IP Toolbench supports AHDL, VHDL, and Verilog HDL. This tutorial uses VHDL.

6. Specify a name for the output file, <directory name>\<variation name>. Figure 2–2 shows MegaWizard Plug-In Manager after you have made these settings.
7. Click **Next** to launch IP Toolbench for the NCO MegaCore function.

**Step 1: Parameterize**

To create a custom variation of the NCO MegaCore function, follow these steps:

1. Click the **Step 1: Parameterize** button in IP Toolbench (see Figure 2–3).
The Parameterize - NCO Compiler window appears. You are now ready to set the parameter options for your custom NCO MegaCore function variation. See Figure 2–4.

2. With the Parameters tab selected, specify the generation algorithm, precisions, phase dithering, and generated output frequency parameters.

As you adjust the NCO parameters, you can graphically view the effects on the NCO in the Frequency Domain Response and Time Domain Response tabs. See Figure 2–4.

The NCO Compiler generates the spectral plot shown in Figure 2–4 by computing a 2,048-point FFT of bit-accurate time-domain data. Before performing the FFT, IP Toolbench windows the data using a Kaiser window of length 2,048.

You can zoom by pressing the left mouse key on the plot drawing a box around the area of interest. Right-click the plot to restore the view to its full range.
Refer to “Architectures” on page 3–5 and “Phase Dithering” on page 3–8 for more information about these parameter options.

Figure 2–4. Parameterize Tab

3. Click the Implementation tab when you are finished setting the general parameters.

4. With the Implementation tab selected, specify the frequency modulation, phase modulation, and outputs; choose the target device family. For some algorithms (e.g., multiplier-based), you can also make device-specific settings such as whether to implement the NCO Compiler in logic elements (LEs) or other hardware. The Implementation tab displays the corresponding options available for the selected algorithm in the Parameters tab.
Figure 2–5 shows the implementation parameter options when the Small ROM or Large ROM algorithm is specified.

**Figure 2–5. Implementation Tab - Small or Large ROM Algorithm**

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

Figure 2–6 shows implementation parameter options when the CORDIC algorithm is specified. With the CORDIC algorithm, you can choose a parallel or serial CORDIC implementation and specify the number of pipeline levels.
Figure 2–7 shows the implementation parameter options when the Multiplier-Based algorithm is specified. If you choose the multiplier-based algorithm and target the Stratix II, Stratix GX, or Stratix device families, you can implement the multiplier-based algorithm either by using LEs or the dedicated Stratix DSP block circuitry, i.e., Use Dedicated Multiplier(s). If you specify multiplier-based and do not target Stratix II, Stratix GX, or Stratix device families, the Quartus II software implements the NCO Compiler using LEs.
5. Click the **Resource Estimate** tab when you are finished setting the implementation parameter options.

The NCO Compiler dynamically estimates the resource usage of your custom NCO MegaCore function variation based on the parameters specified. See Figure 2–8.

The LE count for Stratix II devices, is determined by the adaptive look-up table (ALUT) count in Quartus II report files.
6. Click Finish when you are finished viewing the resource estimates.

**Step 2: Set Up Simulation**

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model file produced by the Quartus II software (version 3.0 or higher). It allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.
To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Click the Step 2: Set Up Simulation button in IP Toolbench (see Figure 2–9).

2. Turn on Generate Simulation Model (see Figure 2–10).
3. Choose the language in the **Language** drop-down box.

4. Click **OK**.

### Step 3: Generate

To generate your MegaCore function, follow these steps:

1. Click the **Step 3: Generate** button in IP Toolbench (see [Figure 2–11](#)).
2. The generation report lists the design files that IP Toolbench creates (see Figure 2–12). Click Exit.
Table 2–1 describes the standard IP Toolbench-generated files.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.vhd, .v, or .tdf</td>
<td>A MegaCore function variation file, which defines a VHDL, Verilog HDL, or AHDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.</td>
</tr>
<tr>
<td>.cmp</td>
<td>A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.</td>
</tr>
<tr>
<td>.inc</td>
<td>An AHDL include declaration file for the MegaCore function variation. Include this file with any AHDL architecture that instantiates the MegaCore function.</td>
</tr>
<tr>
<td>_bb.v</td>
<td>A Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.</td>
</tr>
<tr>
<td>.bsf</td>
<td>A Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.</td>
</tr>
</tbody>
</table>
Simulate the Design

Table 2–1. Standard IP Toolbench-Generated Files (Part 2 of 2)

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.html</td>
<td>A MegaCore function report file.</td>
</tr>
<tr>
<td>.vo or .vho</td>
<td>A VHDL or Verilog HDL IP functional simulation model.</td>
</tr>
<tr>
<td>_inst.vhd or _inst.v</td>
<td>A VHDL or Verilog HDL sample instantiation file.</td>
</tr>
</tbody>
</table>

Table 2–2 lists the NCO Compiler’s output files.

3. After you have reviewed the generation report, click Exit to close IP Toolbench and return to Quartus II software.

You can now integrate your custom megafunction variation into your design and simulate and compile.

Simulate the Design

To simulate your design, use the IP functional simulation models generated by IP Toolbench. The IP functional simulation model is the VO or VHO file generated as specified in “Step 2: Set Up Simulation” on page 2–13. Compile the VO or VHO file in your simulation environment to perform functional simulation of your custom variation of the MegaCore function.

For more information on IP functional simulation models, refer to the white paper entitled, Using IP Functional Simulation Models To Verify Your System Design.
The following instructions are ModelSim-specific; however, you can simulate the NCO Compiler in any Altera-supported VHDL or Verilog VHDL simulator.

Simulating the Design in ModelSim

To simulate the design with the MegaWizard-generated ModelSim TCL script:

1. Change your ModelSim working directory to the project directory specified in “Launch IP Toolbench” on page 2–6, and run the MegaWizard-generated TCL script. See Table 2–2.
   a. If you have selected VHDL as your functional simulation language, run the TCL script <variation_name>_vho_msim.tcl.
   b. If you selected Verilog HDL as your functional simulation language, run the TCL script <variation_name>_vo_msim.tcl

The TCL script creates a ModelSim project, maps the libraries, compiles the top level design and associated testbench, and then outputs the simulation results to the waveform viewer.

Compile the Design

Use the Quartus II software to synthesize, place, and route your design. Refer to Quartus II Help for instructions on performing compilation.

To compile your custom NCO MegaCore function variation you must have Quartus II software, version 4.0, installed on your computer.

The Quartus II user libraries for your project must include <path>nco-v<version>\lib, where <path> is the directory in which you installed the NCO MegaCore function.

To compile your design, perform the following steps:

1. If you are using the Quartus II software to synthesize your design, skip to Step 2. If you are using a third-party synthesis tool to synthesize your design, perform the following steps:
   a. Set a black box attribute for your NCO MegaCore function custom variation before you synthesize the design. Refer to Quartus II Help for instructions on setting black-box attributes per synthesis tool.
Program a Device

b. Run the synthesis tool to produce an EDIF Netlist File (.edf) or Verilog Quartus Mapping (VQM) file (.vqm) for input to the Quartus II software.

c. Add the EDIF or VQM file to your Quartus II project.

2. Choose Start Compilation (Processing menu) in Quartus II software.

Program a Device

After you have compiled your design, program your targeted Altera device, and verify your design in hardware.

With Altera’s free OpenCore Plus evaluation feature, you can evaluate the NCO Compiler MegaCore function before you purchase a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model, and produce a time-limited programming file.

For more information on IP functional simulation models, refer to the white paper entitled, Using IP Functional Simulation Models To Verify Your System Design.

You can simulate the NCO Compiler in your design, and perform a time-limited evaluation of your design in hardware.

For more information on OpenCore Plus hardware evaluation using the NCO Compiler, see “OpenCore Plus Time-Out Behavior” on page 3–1 and AN 320: OpenCore Plus Evaluation of Megafuntions.

Set Up Licensing

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

When you are satisfied with the MegaCore function you can purchase a license.

After you purchase a license for the NCO Compiler, you can request a license file from the Altera web site at www.altera.com/licensing and install it on your computer. When you request a license file, Altera e-mails you a license.dat file. If you do not have Internet access, contact your local Altera representative.

To install your license, you can either append the license to your license.dat file or you can specify the MegaCore function’s license.dat file in the Quartus II software.
Before you set up licensing for the NCO Compiler, you must already have the Quartus II software installed and licensed on your computer.

**Append the License to Your license.dat File**

To append the license, follow these steps:

1. Close the following software if it is running on your PC:
   - Quartus II software
   - MAX+PLUS® II software
   - LeonardoSpectrum synthesis tool
   - Synplify software
   - ModelSim simulator

2. Open the NCO Compiler license file in a text editor. The file should contain one `FEATURE` line, spanning 2 lines.

3. Open your Quartus II `license.dat` file in a text editor.

4. Copy the `FEATURE` line from the NCO Compiler license file and paste it into the Quartus II license file.

   Do not delete any `FEATURE` lines from the Quartus II license file.

5. Save the Quartus II license file.

   When using editors such as Microsoft Word or Notepad, ensure that the file does not have extra extensions appended to it after you save (e.g., `license.dat.txt` or `license.dat.doc`). Verify the filename at the system command prompt.

**Specify the License File in the Quartus II Software**

To specify the MegaCore function’s license file, follow these steps:

1. Start the Quartus II software.

2. Choose **License Setup** (Tools menu). The **Options** dialog box opens to the **License Setup** page.

3. In the **License file** box, add a semicolon to the end of the existing license path and filename.
4. Type the path and filename of the MegaCore function license file after the semicolon.

Do not include any spaces either around the semicolon or in the path/filename.

5. Click OK to save your changes.
An NCO synthesizes a discrete-time, discrete-valued representation of a sinusoidal waveform. There are many ways to synthesize a digital sinusoid. For example, a popular method is to accumulate phase increments to generate an angular position on the unit circle and then use the accumulated phase value to form an address to a ROM look-up table to perform the polar-to-cartesian transformation. You can reduce the ROM size by using multipliers. Multipliers provide an exponential decrease in memory usage for a given precision but require more logic.

Another method uses the coordinate rotation digital computer (CORDIC) algorithm to determine, given a phase rotation, the sine and cosine values iteratively. The CORDIC algorithm takes an accumulated phase value as input and then determines the cartesian coordinates of that angle by a series of binary shifts and compares. In all methods, the frequency at which the phase increment accumulates and the size of that input phase increment relative to the maximum size of the accumulator directly determines the normalized sinusoidal frequency.

When deciding which NCO implementation to use in programmable logic, you should consider several parameters, including the spectral purity, frequency resolution, performance, throughput, and required device resources. Often, you need to consider the trade-offs between some or all of these parameters.

This section discusses:

- Spectral purity
- Maximum output frequency

**Spectral Purity**

Typically, the spectral purity of an oscillator is measured by its signal-to-noise ratio (SNR) and its spurious free dynamic range (SFDR). The SNR of a digitally synthesized sinusoid is a ratio of the signal power relative to the unavoidable quantization noise inherent in its discrete-valued representation. SNR is a direct result of the finite precision with which NCO represents the output sine and cosine waveforms. Increasing the output precision results in an increased SNR. The following equation estimates the SNR of a given sinusoid with output precision $b$:

$$\text{SNR} = 6b - 1.8 \text{ (dB)}$$
Each additional bit of output precision leads to an additional 6 dB in SNR.

The SFDR of a digital sinusoid is the power of the primary or desired spectral component relative to the power of its highest-level harmonic component in the spectrum. Harmonic components manifest themselves as spikes or spurs in the spectral representation of a digital sinusoid and occur at regular intervals and are also a direct consequence of finite precision. However, the effect of the spurs is often severe because they can cause substantial inter-modulation products and undesirable replicas of the mixed signal in the spectrum, leading to poor reconstruction of the signal at the receiver.

The direct effect of finite precision varies between architectures, but the effect is augmented because, due to resource usage constraints, the NCO does not usually use the full accumulator precision in the polar-to-cartesian transformation. You can mitigate truncation effects with phase dithering, in which the truncated phase value is randomized by a sequence. This process removes some of the periodicity in the phase, reducing the spur magnitude in the sinusoidal spectrum by up to 12 dB.

The NCO Compiler’s graphical spectral analysis allows you to view the effects as you change parameters without regenerating the IP Toolbench output files and re-running simulation.

**Maximum Output Frequency**

The maximum frequency sinusoid that an NCO can generate is bounded by the Nyquist criterion to be half the operating clock frequency. Additionally, the throughput affects the maximum output frequency of the NCO. If the NCO outputs a new set of sinusoidal values every clock cycle, the maximum frequency is the Nyquist frequency. If, however, the implementation requires additional clock cycles to compute the values, the maximum frequency must be further divided by the number of cycles per output.
Specifications

**Functional Description**

The NCO Compiler allows you to generate a variety of NCO architectures. You can create your custom NCO using a IP Toolbench-driven interface that 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:

$$s(nT) = A \sin \left[ 2\pi (f_0 + f_{FM}) nT + \phi_{PM} + \phi_{DITH} \right]$$

where:

- $T$ is the operating clock period
- $f_0$ is the unmodulated output frequency based on the input value $\phi_{INC}$
- $f_{FM}$ is a frequency modulating parameter based on the input value $\phi_{FM}$
- $\phi_{PM}$ is the phase modulation input value
- $\phi_{DITH}$ is the internal dithering value
- $A = 2^{N-1}$ where $N$ is the magnitude precision

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

![Figure 3–1. Generic NCO Block Diagram](image)

The generated output frequency, $f_0$, for a given phase increment, $\phi_{inc}$ is determined by the equation:

$$f_0 = \frac{\phi_{inc} f_{clk}}{2^M} \text{ Hz}$$

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

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

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. If you want an output frequency of 6.25 MHz from this oscillator, then you should apply an input phase of

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

as the input phase increment to the NCO. The NCO Compiler automatically calculates this value, given the parameters you choose. IP Toolbench also sets the value of the phase increment in all testbenches and vector source files it generates.

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 $f_{\text{res}}/f_{\text{clk}}$ ratios. You can view these effects in the NCO Compiler time and frequency domain graphs as you change the NCO parameters.

This section discusses:

- Architectures
- Frequency modulation
- Phase modulation
- Timing diagrams
- OpenCore Plus time-out behavior
Architectures

The NCO Compiler 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.

Because 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**

If low LE usage and high output frequency are a high priority for your system, use the small ROM architecture to reduce your internal memory usage. 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.

Because a small ROM implementation is more likely to have periodic value repetition, the resulting waveform’s SFDR is generally lower than that of the large ROM architecture. However, you can often mitigate this reduction in SFDR with phase dithering. See “Phase Dithering” on page 3–8 for more information on this option.

**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 must compute 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. See Figure 3–2.

With the NCO Compiler, you can choose between parallel and serial CORDIC architectures:

- You can use the parallel CORDIC architecture to create a very high-performance, high-precision oscillator—implemented entirely in LEs—with a throughput of one output sample per clock cycle.

  When specifying the parallel CORDIC architecture, the NCO Compiler allows you to trade off between performance, resource usage, and initial latency using the CORDIC pipeline level parameter. Refer to “Implementation Tab - CORDIC Algorithm” on page 2–11.

- The serial CORDIC architecture uses fewer resources than the parallel CORDIC. 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 Compiler, the output sample rate and the Nyquist frequency is reduced by a factor of \( N \). This architecture is implemented entirely in LEs and is useful if your design requires low frequency, high precision waveforms.
**Multiplier-Based Architecture**

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

- Dedicated multiplier circuitry (e.g., dedicated DPS blocks) in device families that support this feature (i.e., Stratix II, Stratix GX, or Stratix devices)
- LEs

When you specify a dual output NCO, the NCO Compiler provides an additional option to reduce the throughput by a factor of two. This setting halves the resources required by the waveform generation unit, and the NCO outputs a sample every two clock cycles. Refer to “Implementation Tab- Multiplier-Based Algorithm” on page 2–12.

**Frequency Modulation**

In the NCO Compiler, 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, for example, to implement frequency shift keying (FSK) modulators like the quaternary FSK modulator in “Example Design 1” on page A–1. You can set the frequency modulation resolution input in the NCO Compiler; it must be less than or equal to the phase accumulator precision. The NCO Compiler also provides an option to increase the modulator pipeline level; however, the effect of the increase on the performance of the NCO varies across NCO architectures and variations.

**Phase Modulation**

You can use the NCO Compiler to add an optional phase modulator to your NCO 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 Compiler. The input resolution must be greater than or equal to the specified angular precision.
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 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 Compiler 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. See “Example Design 1” on page A–1 for an example using phase dithering and its effect on the spectrum of the output signal.

Timing Diagrams

Figure 3–3 shows a timing diagram with a single clock cycle per output sample. All NCO architectures—except 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.
Figure 3–4 shows the timing diagram for a two-cycle multiplier-based NCO architecture. 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 3–4. Two-Cycle Multiplier-Based Architecture Timing Diagram](image)

Figure 3–5 shows the timing diagram for a serial CORDIC NCO architecture. 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 Compiler. There is also an initial latency of \(L\) clock cycles; the exact value of \(L\) depends on the parameters that you set. Figure 3–5 shows the case where \(N = 8\).

![Figure 3–5. Serial CORDIC Timing Diagram](image)

**OpenCore Plus Time-Out Behavior**

OpenCore Plus hardware evaluation can support the following two modes of operation:

- *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 may be masked by the time-out behavior of the other megafunctions.
For MegaCore functions, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

The signals $f_{sin\_o}$ and $f_{cos\_o}$ are forced low when the evaluation time expires.

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

Signals

The NCO Compiler function has the signals shown in Table 3–1.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>Input</td>
<td>Clock.</td>
</tr>
<tr>
<td>reset</td>
<td>Input</td>
<td>Active-high reset.</td>
</tr>
<tr>
<td>clk_en</td>
<td>Input</td>
<td>Active high clock enable.</td>
</tr>
<tr>
<td>phi_inc_i [A-1:0]</td>
<td>Input</td>
<td>Input phase increment. Set the precision A in the NCO Compiler.</td>
</tr>
<tr>
<td>freq_mod_i [F-1:0]</td>
<td>Input</td>
<td>(optional) Frequency modulation input. Set the precision F in the NCO Compiler.</td>
</tr>
<tr>
<td>phase_mod_i [P-1:0]</td>
<td>Input</td>
<td>(optional) Phase modulation input. Set the precision P in the NCO Compiler.</td>
</tr>
<tr>
<td>fsin_o [M-1:0]</td>
<td>Output</td>
<td>Output sine value. Set the precision M in the NCO Compiler.</td>
</tr>
<tr>
<td>fcos_o [M-1:0]</td>
<td>Output</td>
<td>Output cosine value. Set the precision M in the NCO Compiler.</td>
</tr>
</tbody>
</table>

MegaCore Verification

Before releasing a version of the NCO Compiler, Altera runs a comprehensive regression test. 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 bit-accurate models in the MATLAB software.

The regression suite covers various parameters such as architecture options, frequency modulation, phase modulation, and precision. Figure 3–6 shows the regression flow.
Figure 3–6. Regression Flow
Appendix A. Example Designs

Example Design 1

Example design 1 is a high-precision, dual-output oscillator for use in an intermediate frequency (IF) I-Q modulator. The design targets the Altera EP2S15F484C3 Stratix II device. The top-level design file is `<installation directory>\nco-v2.1.0\example_designs\design1\design1.bdf`. The oscillator meets the following specifications:

- SFDR: 110 dB
- Output Sample Rate: 300 MSPS
- Output Frequency: 21 MHz
- Frequency Resolution: 0.05 Hz

To meet these requirements, the design uses the following NCO Compiler settings:

- **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 300 MHz and 32-bit phase accumulator precision**—These settings yield a frequency resolution of 46 mHz.

- **Angular and magnitude precision**—These settings are critical to meet the SFDR requirement, while minimizing the required device resources. Setting the angular precision to 17 bits and the magnitude precision to 18 bits results in the spectrum shown in Figure A–1 ("After Setting Angular & Magnitude Precision" plot).

- **Dither level**—The angular and magnitude precision settings described above yield an SFDR of approximately 102.8 dB, which is clearly not sufficient to meet the specification. Using the dither control in the NCO Compiler, the variance of the dithering sequence is increased until the trade-off point between spur reduction and noise level augmentation is reached for these particular clock-frequency to output frequency ratio and precision settings as shown in Figure A–1 ("After Addition of Dithering" plot). At a dithering level of 5, the SFDR is approximately 111.95 dB, which exceeds the specification.
Example Design 2

Example design 2 is a quaternary frequency shift keying (QFSK) modulator for use in a hypothetical transmitter design. The design targets the Altera EP2S15F484C3 Stratix II device. In this type of modulator, the output frequency of the oscillator varies according to an input symbol stream, the values of which map to a four-symbol alphabet. The top-level design file is `<installation directory>`\nco-v2.1.0\example_designs\design2\design2.bdf. The oscillator meets the following specifications:

- SFDR: 80 dB
- Output Frequencies:
  - \( f_c - 5.76 \) MHz
  - \( f_c - 1.92 \) MHz
  - \( f_c + 1.92 \) MHz
  - \( f_c - 5.76 \) MHz
- Output Sample Rate: 300 MSPS
- Frequency Resolution: 0.05 Hz

Where \( f_c \) is the free-running output frequency of the NCO.
Table A–1 shows the mapping of the symbols to output frequencies, assuming that the free-running frequency is set to 15.36 MHz by setting the phase increment of the oscillator to 299866808.

<table>
<thead>
<tr>
<th>Binary Symbol</th>
<th>Frequency Modulation Value</th>
<th>Output Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4182517243</td>
<td>9.60</td>
</tr>
<tr>
<td>11</td>
<td>4257483945</td>
<td>13.44</td>
</tr>
<tr>
<td>01</td>
<td>37483351</td>
<td>17.28</td>
</tr>
<tr>
<td>00</td>
<td>112450053</td>
<td>21.12</td>
</tr>
</tbody>
</table>

To meet these requirements, the design uses the following NCO Compiler settings:

- **Parallel CORDIC algorithm**—In the overall hypothetical design, other system blocks use much of the Stratix device’s internal memory and DSP blocks; however, logic elements (LEs) are in abundant supply. The need for a high precision, high performance oscillator that uses Logic Elements only makes the Parallel CORDIC algorithm a suitable choice.

- **Phase accumulator precision**—The frequency resolution specifications demand a phase accumulator precision of 32 bits.

- **Frequency modulator precision**—To maximize the frequency resolution of the modulating signal, the resolution of the frequency modulator is also set to 32 bits of precision.

- **Angular and magnitude precision and dithering**—An angular precision of 14 bits, a magnitude precision of 14 bits, and a dithering level of 8 meet the requirements of 80 dB SFDR for the four possible output frequencies as shown in Figure A–2.
Figure A–3 shows a segment of the resulting FSK modulated waveform in the time-domain.
Figure A–3. FSK Modulated Waveform