CoreDDS Handbook

Actel®

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200078-0

Release: September 2006

Trademarks

Table of Contents

Introduction

Core Overview

CoreDDS is an RTL generator that produces an Actel FPGA-optimized direct digital synthesizer (DDS) core. DDS digitally generates a complex or real-valued sine wave. Due to the digital nature of the DDS functionality, it offers fast switching between output sine wave frequencies, fine frequency resolution, and operations over broad frequency range.[1]

A DDS or a numerically controlled oscillator (NCO) is an important component in many digital communication systems. Quadrature synthesizers are used for constructing digital down and up converters, constructing demodulators, and implementing various types of modulation schemes, including phase shift keying (PSK), frequency shift keying (FSK), and minimum shift keying (MSK). A common method for digitally generating a complex or real valued sinusoid employs a lookup table (LUT) scheme. The lookup table stores samples of a sinusoid. A digital integrator is used to generate a suitable phase argument that is mapped by the lookup table to the desired output waveform.

Figure 1 depicts a simplified view of the CoreDDS inputs and outputs.

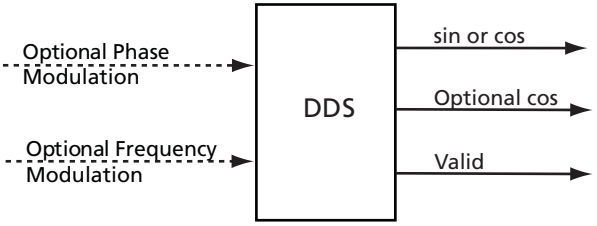

Figure 1. CoreDDS Simplified View

Depending on the core configuration, CoreDDS can generate a sine or cosine waveform, as well as the complex sinusoid. In the latter case, the core generator creates two data outputs with a sine wave (imaginary part of the complex sinusoid) present on the first output, and the cosine wave (real part of the complex sinusoid) present on the second output. Output "valid" goes active when the DDS generates its first valid output sample.

The DDS can be set to generate a waveform of a constant frequency and phase shift. Particular values of these parameters and the waveform bit resolution are set at configuration time. One can also configure the core to have optional phase and/or frequency modulation ports so that the output frequency and phase can be modulated at run time.

1. Goldberg, Bar-Giora, 1999, *Digital Frequency Synthesis Demystified*, LLH Technology Publishing.

A user configures the DDS via a configuration text file by setting desired parameter values. For a detailed description of the configuration parameters, refer to "Configuration Parameters" on page 26.

A typical application using the DDS is shown in Figure 2 on page 6. The DDS is used in a communication system as a quadrature intermediate frequency (IF) carrier oscillator to implement a simple IF modulator. Two mixers modulate baseband quadrature data, I(t) – Q(t), to create a composite output IF signal. Similarly, the DDS can be used to demodulate an I-Q composite signal.

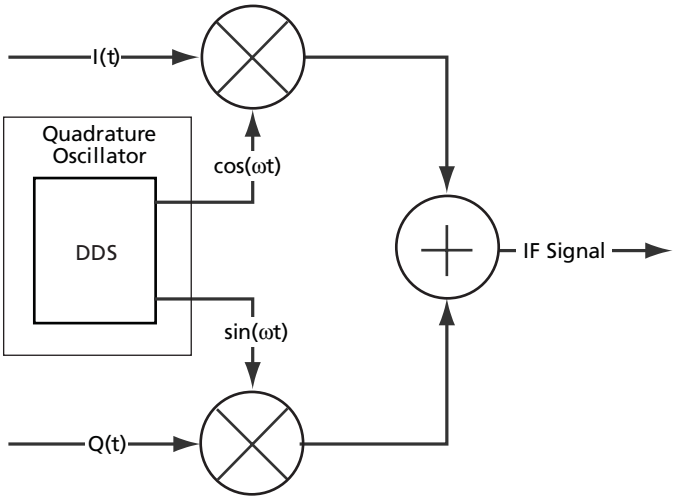

Figure 2. DDS Typical Application

Hardware Implementation

The DDS engine implements three different hardware architectures:

- Small LUT
- Big LUT
- Parallel CORDIC

A user specifies a desired architecture at configuration time.

"Architectures" on page 12 provides a detailed explanation of the DDS architectures.

Utilization and Performance

The CoreDDS engine has been implemented in several Actel FPGA families. A summary of the data for CoreDDS is listed in Table 1.

Table 1. CoreDDS Device Utilization and Performance

Device	Engine Architect	Cells or Tiles			Utilization %	RAM Blocks		Clock Rate (MHz)
		Comb	Seq	Total		Utilized	Available on Chip	
Fusion			**Speed Grade –2**					
AFS600	Small LUT	1,185	506	1,691	12%	4	24	104
AFS600	Big LUT	548	214	762	6%	1	24	109
AFS600	CORDIC	7,500	1,654	9,154	66%	0	24	82
ProASIC®3/E			**Speed Grade –2**					
A3P600	Small LUT	1,194	508	1,702	12%	4	24	112
A3P600	Big LUT	556	217	773	6%	1	24	101
A3P600	CORDIC	7,433	1,643	9,076	66%	0	24	85
ProASIC PLUS®			**Speed Grade STD**					
APA600	Small LUT	1,726	447	2,173	11%	8	56	68
APA600	Big LUT	852	213	1065	5%	2	56	64
APA600	CORDIC	10,342	1,623	11,965	56%	0	56	45
Axcelerator®			**Speed Grade –2**					
AX500	Small LUT	858	417	1,275	16%	4	16	141
AX500	Big LUT	399	226	625	8%	1	16	191
AX500	CORDIC	3,751	1,603	5,354	66%	0	16	138
RTAX			**Speed Grade –1**					
RTAX250S	Small LUT	858	417	1,275	32%	4	16	100
RTAX250S	Big LUT	399	226	625	16%	1	16	127
RTAX1000S	CORDIC	3,751	1,603	5,354	33%	0	16	102

Note: Data in this table were achieved using typical synthesis and layout settings. CoreDDS configuration parameters were set as shown in Table 2 on page 8.

Table 2. DDS Test Configurations

Parameter		Small LUT	Big LUT	CORDIC
Name	Comment			
module_name		testDDS	testDDS	testDDS
function	Output waveform: ain(0), xos(1), quad(2)	0	1	2
architecture	Small LUT(0), Big LUT(1), CORDIC(2)	0	1	2
phAcc_mode	Phase increment: constant(0), port (1)	0	0	0
phase_const	Phase increment constant value	977	317	977
phase_modulation	None(0), constant(1), port(2)	1	1	1
phase_mod_const	Phase modulation constant value	12	12	12
dithering	Off(0), On(1)	1	1	0
dith_attenuation	Dithering attenuation value	1	1	x
phAcc_bitSize	Bit width of the phase accumulator	24	15	24
slicer_bitSize	Bit width of a slicer	12	7	12
wave_bitSize	Waveform bit resolution	14	12	14
fpga_family				
lang		verilog	verilog	verilog

1 Functional Description

Theory of Operation

A high-level view of the DDS Core is presented in Figure 1-3. The phase accumulator computes a phase slope with high bit resolution that is mapped to a sinusoid (possibly complex) by the lookup table (LUT). The phase quantizer (slicer) accepts the high-precision phase angle and generates a lower precision representation of the angle denoted as Φ(n) in Figure 1-3. The slicer output bit resolution is denoted as b_Φ(n) bits.

This value is presented to the address port of a lookup table that contains the pre-computed value of sin(Φ(n)) and/or cos(Φ(n)) in a proper memory cell.

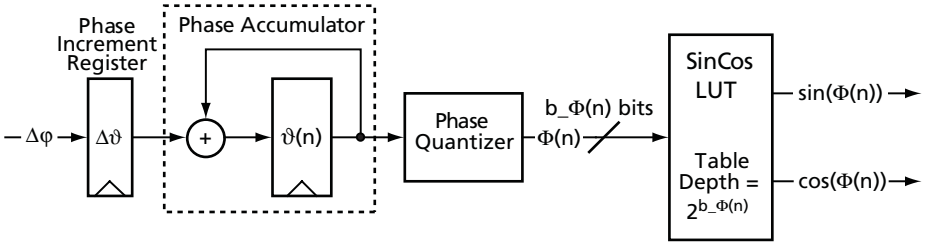

Figure 1-1. LUT-Based DDS Simplified Block Diagram

The fidelity of a signal formed by recalling samples of a sinusoid from a lookup table is affected by both the phase and amplitude quantization of the process. The length and width of the lookup table affect the signal's phase angle resolution and the signal's amplitude resolution respectively. These resolution limits are equivalent to time base jitter and to amplitude quantization of the signal, and add spectral modulation lines and a white broad-band noise floor to the signal's spectrum.

Direct digital synthesizers use an addressing scheme with an appropriate lookup table to form samples of an arbitrary frequency sinusoid. If an analog output is required, the DDS presents these samples to a digital to analog converter (DAC) and a low-pass filter to obtain an analog waveform with the specific frequency structure. Of course, the samples are also commonly used directly in the digital domain. The lookup table traditionally stores uniformly spaced samples of a cosine and a sine wave. These samples represent a single cycle of a complex sinusoid. The cycle contains $N = 2^{b\_\Phi(n)}$ complex samples and corresponds to specific values of the sinusoid's argument, Φ(n), as shown in EQ 1.

$$\Phi(n) = n\frac{2\pi}{N}$$

EQ 1

where n is the time series sample index.

Quarter period wave symmetry in the basis waveform can be exploited to construct a DDS that uses shortened tables. In this case, the two most significant bits of the quantized phase angle are used to

perform quadrant mapping. This implementation results in a more area-efficient implementation because the memory requirements are minimized; the required LUT can be implemented using one quarter of the necessary RAM capacity. Since this memory-saving technique may slightly slow down the DDS implementation, it is left up to the user to decide which implementation better matches design needs[2].

Output Frequency

The output frequency, f_{out}, of the DDS waveform is a function of the system clock frequency, f_{clk}, the number of bits in the phase accumulator, b_ϑ(n), and the phase increment value, Δϑ. Output frequency in Hertz is defined as shown in EQ 2.

$$f_{out} = \frac{f_{clk}\Delta\vartheta}{2^{b\_\vartheta(n)}} \text{Hz}$$

EQ 2

For example, if the DDS parameters are

$$f_{clk} = 96 \text{ MHz}$$

$$b\_\vartheta(n) = 12$$

$$\Delta\vartheta = 62$$

the output frequency will be $f_{out} = \frac{96 \cdot 10^6 \cdot 62}{2^{12}} = 1,453,125 \text{ MHz}$.

The phase increment value, Δϑ, required to generate an output frequency, f_{out} Hz, is shown in EQ 3.

$$\Delta\vartheta = \frac{f_{out} 2^{b\_\vartheta(n)}}{f_{clk}}$$

EQ 3

2. Also for short tables, FPGA logic resources are actually minimized by storing a complete cycle.

Frequency Resolution

The frequency resolution of the synthesizer is a function of the clock frequency and the number of bits employed in the phase accumulator. The frequency resolution can be determined using EQ 4.

$$\Delta f = \frac{f_{clk}}{2^{b\_\vartheta(n)}}$$

EQ 4

For example, for the DDS parameters

$$f_{clk} = 96 \text{ MHz}$$

$$b\_\vartheta(n) = 32$$

the frequency resolution is $\Delta f = \frac{f_{clk}}{2^{b\_\vartheta(n)}} = \frac{96 \cdot 10^6}{2^{32}} = 0.0223517 \text{ Hz}$.

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 the DDS represents the output sine and cosine waveforms. Increasing the output precision results in an increased SNR. EQ 5 estimates the SNR of a given sinusoid with output precision b_s bits:

$$\text{SNR} = 6\ b\_s\ - 1.8 \text{ (dB)}$$

EQ 5

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 DDS does not usually use the full accumulator precision in the polar-to-Cartesian transformation. One can mitigate the quantization effects with phase dithering, which attempts to convert the spurious signal energy to evenly distributed noise. Dithering randomizes the truncated phase value by adding a small amount of noise prior to phase truncation. This process removes some of the periodicity in the phase, reducing the spur magnitude in the sinusoidal spectrum.

Architectures

CoreDDS supports Big LUT, Small LUT, and CORDIC architectures.

Big LUT Architecture

The Big LUT architecture, shown in Figure 1-2, stores a complete cycle (360 degrees) of the sine/cosine waveforms in the LUT. Use the Big LUT architecture if the design requires very high speed sinusoidal waveforms, and the design can use large quantities of internal memory.

The phase accumulator uses either constant phase increment, $\Delta\vartheta$, stored in a phase increment register, or an input signal, $\Delta\varphi$, coming from an optional frequency modulation port. A phase offset can be added to a phase signal at a phase accumulator output. Depending on user configuration, the phase offset is a constant user-defined value of ϑ_0, or a variable signal, φ_0, coming from an optional phase modulation port. It is also possible to add a dithering signal to the phase prior to truncating extra phase bits. Output of the phase quantizer directly addresses the LUT. 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 Big LUT architecture also uses the fewest fabric tiles for a given set of precision parameters.

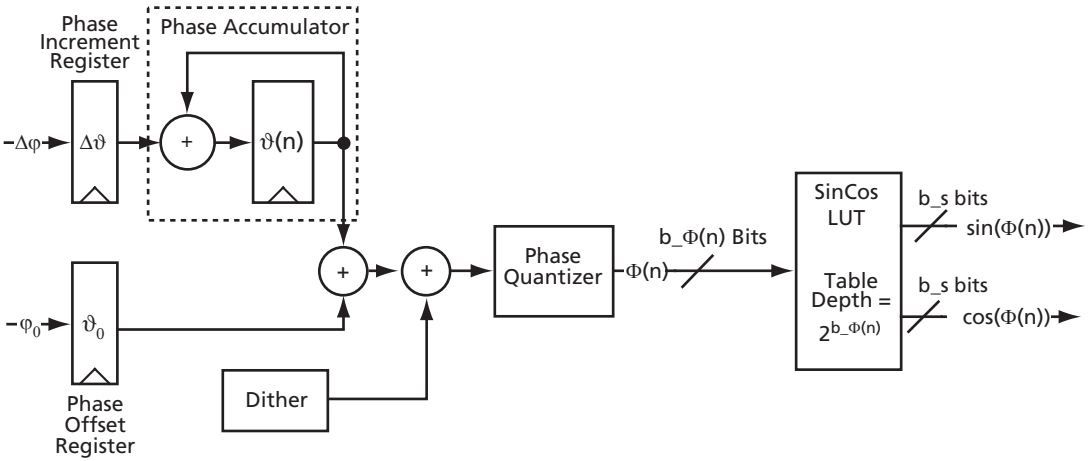

Figure 1-2. Big LUT Architecture

Small LUT Architecture

In a Small LUT architecture, depicted in Figure 1-3, the LUT only stores 90 degrees of the sine or cosine waveform (or 45 degrees of the sine and cosine waveforms). All other output values are derived from these values based on the angle value Φ(n). Use the Small LUT architecture to reduce on-chip memory usage.

The phase generation part of the Small LUT architecture is identical to that of the Big LUT architecture described in "Big LUT Architecture" on page 12. The quantized phase, Φ(n), is mapped to the first quadrant (0–90 degrees) of a polar angle. The sine/cosine samples obtained from the LUT are then processed to restore proper polar quadrant mapping.

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, it is often possible to mitigate this reduction in SFDR with phase dithering.

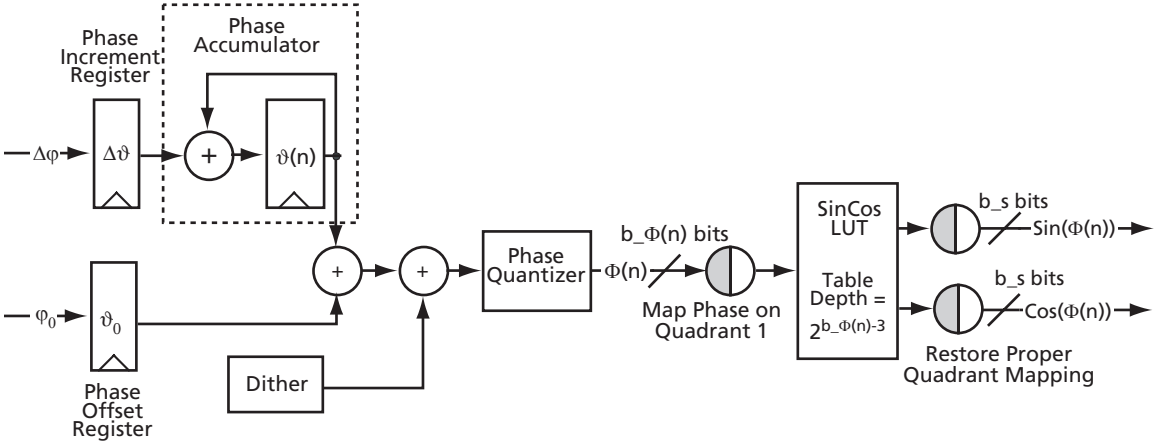

Figure 1-3. Small LUT Architecture

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 shifts and additions/subtractions.

In a DDS, 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 the *CoreCORDIC CORDIC RTL Generator* datasheet for more information.

Figure 1-4 shows the CORDIC-based DDS architecture. Since the CORDIC architecture eliminates the sine/cosine LUT by real-time waveform computation, it can provide higher phase resolution, thus reducing spurious signals caused by phase quantization. While in the LUT-based architectures the feasible LUT size limits bit resolution at the phase quantizer output, the CORDIC DDS architecture is virtually free of this limitation.

In many cases, the CORDIC-based architecture does not utilize the phase quantizer, since the CORDIC can accept all bits of the phase generator. CoreDDS still enables a user to infer the phase quantizer as well as dither the phase signal if there is a desire to do so.

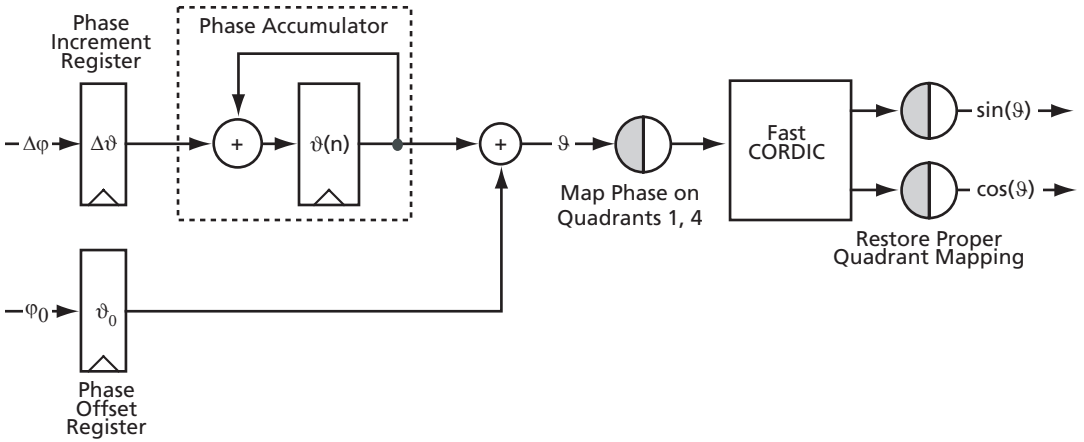

Figure 1-4. CORDIC-Based DDS Architecture

LUT Initialization

To support the Actel one-chip solution, CoreDDS implements an internal sine/cosine table generator to initialize the SinCos LUT (Figure 1-5 on page 15). The generator is based on a small bit-serial CORDIC engine that calculates the necessary sin/cos table entries. CORDIC-generated sine wave samples are approximations of a precise sine wave. In order to store the LUT precise sine wave values, the approximations sin' and cos' need to be truncated to discard bits that are not accurate. The bit-serial CORDIC engine is relatively slow, but since the initialization takes place only on power-on, this does not impact the DDS performance.

A power-on signal (asynchronous reset) kicks off an initialization state machine. It generates the necessary signals to control the CORDIC engine as well as the RAM block, and provides a write

address for the RAM block. Once the initialization is done, the RAM block switches to read-only mode.

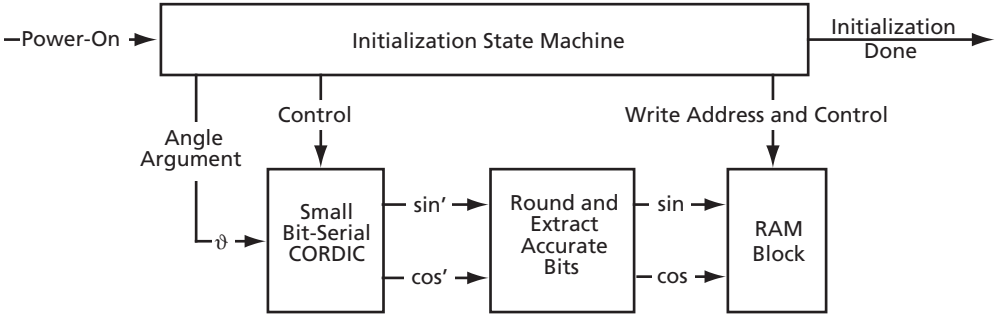

Figure 1-5. Sine/Cosine LUT Initialization

Phase Dithering and Phase Quantizer

A source of dithering noise generates a uniformly distributed zero-mean pseudorandom signal in a range from –B/2 to B/2, where B is the most significant bit to be discarded as a result of phase quantization. From the spur level standpoint, it is often beneficial to use a dithering signal with a smaller variance. CoreDDS supports configurable attenuation of the dithering signal by a factor of $1/2^{dith\_attenuation}$.

The phase quantizer implements the round-to-nearest-even algorithm, which is commonly recognized as a standard way of rounding numbers. Round-to-nearest-even is the default rounding scheme of the IEEE floating–point standard.[3]

3. Parhami, Behrooz, 2000, *Computer Arithmetic, Algorithms and Hardware Designs*, Oxford University Press, pp 287-291.

2 Tool Flows

CoreDDS is licensed in two ways. Depending on your license, tool flow functionality may be limited.

Evaluation: RTL source code with limited parameters and corresponding testbench are provided. Evaluation version is fully supported in the Actel Libero® Integrated Design Environment (IDE)

Full version: Complete fully user configurable RTL source code is provided along with behavioral testbenches.

CoreDDS is a combination of Verilog or VHDL source files, Microsoft Windows® command line programs that generate other source files, and auxiliary files necessary to integrate the core into Libero IDE design flow. The RTL sources are optimized for Actel FPGA devices.

The majority of tool flows are common to both the Full and Evaluation versions of the core. Any differences are outlined below.

Installation Flow

This section describes CoreDDS installation flow and provides implementation hints. Check the CoreDDS web page for updates to the installation instructions.

In order to install CoreDDS you must create a Libero IDE project first.

Installing CoreDDS Evaluation Version

To install the CoreDDS evaluation version:

1. Start Libero IDE and create a new project.
   - Choose a project name (testDDS, for example), a location (F:\Actelprj, for example), and an HDL type (Verilog or VHDL).
   - Select the targeted FPGA family, device, and package (Fusion family, AFS600 device, and 256 FBGA package, for example). Click **Next**.
   - Select integrated tools **Synplify**® and **Model*Sim***®. Click **Next**.
   - Click **Finish**.
2. Using Windows Explorer, create the subfolders *\source* and *\package* in the project folder *\testDDS*.

3. Unzip **evalDDS.zip** to extract files in the newly created subfolder *\testDDS\source*. The subfolder now contains the following files:
   - *coreDDS_eval.exe*
   - *coreCORDIC.exe*
   - *dds_config.txt*
   - *dds_top.sdc*
   - Subfolder *coreDDS_verilog*, which contains the following files:

     dds.v

     kit.v

     dds_tb.v
   - Subfolder *coreDDS_vhdl*, which contains the following files:

     dds.vhd

     kit.vhd

     rtl_pack.vhd

     bhv_pack.vhd

     bhv.vhd

     dds_tb.vhd
4. Modify the **lang** parameter of the *dds_config.txt* file as desired, using a text editor.

   Note: The evaluation version can only accept the user-defined parameters outlined in Table A-1 on page 32.
5. Run **CoreDDS_eval** by typing the following command at the command prompt:

   `F:\Actelprj\testDDS\source > coreDDS_eval dds_config.txt`

   This creates a few more files in the *\source* subfolder, *dds_batch.bat* among them.
6. Run **dds_batch.bat** by double-clicking its icon in Windows Explorer. This creates three new folders within the *\source* subfolder: *\hdl*, *\package*, and *\stimulus,* filled with necessary source files. All source files stored in these folders are imported into the Libero IDE project.
7. Go back to the Libero IDE and import the source files from the folders *\testDDS\source\hdl* and *\testDDS\source\stimulus*. If the language selected is VHDL, import the source files from the folder *\testDDS\source\package* as well.

   Note: Normally you need to import files from the *\testDDS\source\hdl* subfolder to the HDL Source Files folder of the Libero IDE project; import files from the *\testDDS\source\stimulus* subfolder to the Stimulus Files folder of the Libero IDE project; and import files from the *\testDDS\source\package* subfolder to the Package folder of the Libero IDE project. Refer to Libero IDE documentation for specific instructions.

8. Import the constraint file *\testDDS\source\dds_top.sdc* to the Constraint Files folder of the Libero IDE project.
9. In the Libero IDE, set **dds_top** as a root.
10. Successful installation of CoreDDS source files results in a Libero IDE screen similar to the one shown in Figure 2-1 (Verilog project) or Figure 2-2 on page 20 (VHDL project).

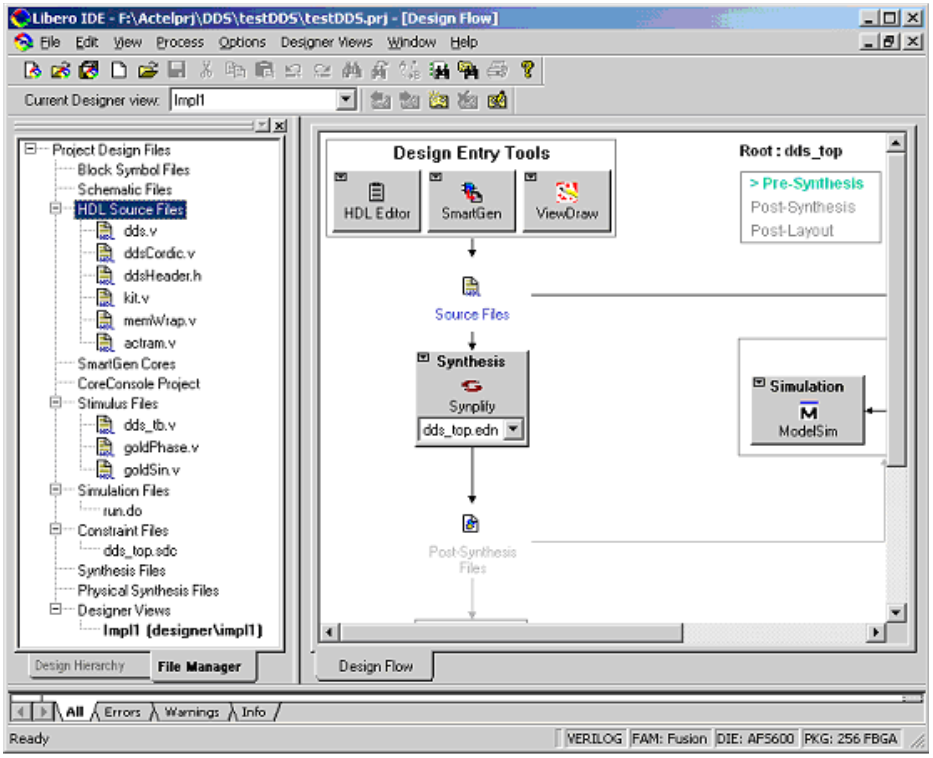

Figure 2-1. Libero IDE DDS Verilog Project Settings

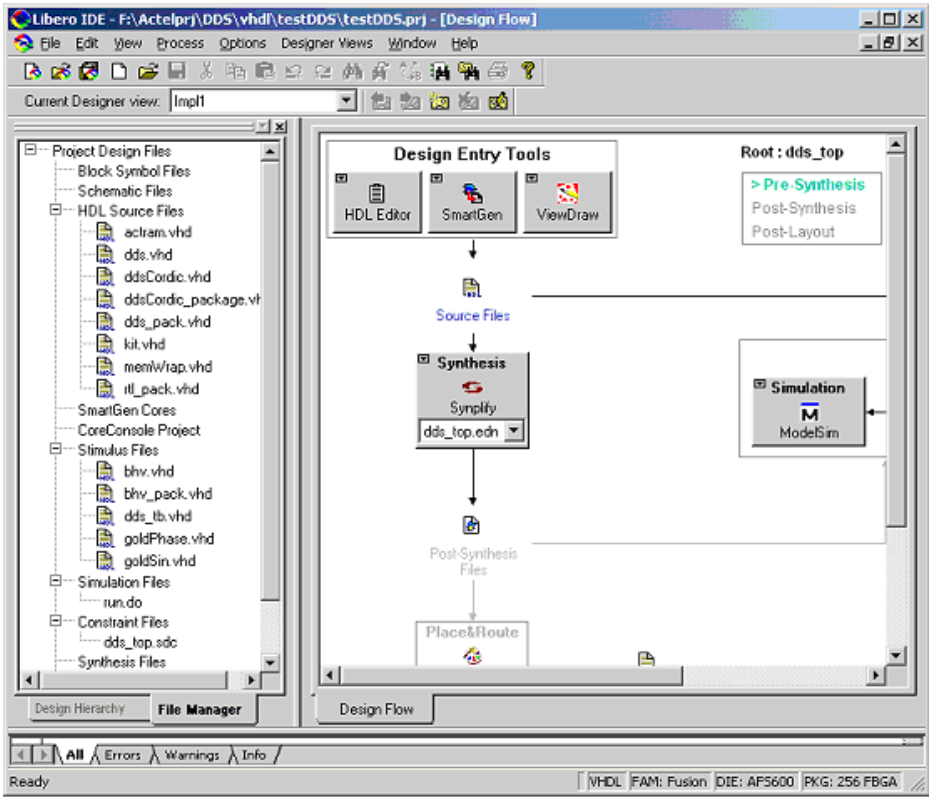

Figure 2-2. Libero IDE DDS VHDL Project Settings

Installing CoreDDS Full Version

To install the CoreDDS full version:

1. Start Libero IDE and create a new project.
   - Choose a project name (testDDS, for example), a location (F:\Actelprj, for example), and an HDL type (Verilog or VHDL).
   - Select the targeted FPGA family, device, and package (Fusion family, AFS600 device, and 256 FBGA package, for example). Click **Next**.
   - Select integrated tools **Synplify** and **Model*Sim***. Click **Next**.

- Click **Finish**.

2. Using Windows Explorer, create the subfolders *\source* and *\package* in the project folder *\testDDS*.
3. Unzip **corelDDS.zip** to extract files in the newly created subfolder *\testDDS\source*. The subfolder now contains the following files:
   - *coreDDS.exe*
   - *coreCORDIC.exe*
   - *dds_config.txt*
   - *dds_top.sdc*
   - Subfolder *coreDDS_verilog*, which contains the following files:
     dds.v
     kit.v
     dds_tb.v
   - Subfolder *coreDDS_vhdl*, which contains the following files:
     dds.vhd
     kit.vhd
     rtl_pack.vhd
     bhv_pack.vhd
     bhv.vhd
     dds_tb.vhd
4. Modify the configuration parameter of the *dds_config.txt* file as desired, using a text editor. Valid values of the core parameters are shown in Table 3-2 on page 26.

   Note: Keep the original format of the configuration file. Modify only the parameter values.
5. Run CoreDDS by typing the following command at the command prompt:

   ```
   F:\Actelprj\testDDS\source > coreDDS dds_config.txt
   ```

   This creates a few more files in the *\source* subfolder, *dds_batch.bat* among them.
6. Run **dds_batch.bat** by double-clicking its icon in Windows Explorer. This creates three new folders within the *\source* subfolder: *\hdl*, *\package*, and *\stimulus* filled with necessary source files. All source files stored in these folders are to be imported into the Libero IDE project.
7. Go back to the Libero IDE and import the source files from the folders *\testDDS\source\hdl* and *\testDDS\source\stimulus*. If the language selected is VHDL, import the source files from the folder *\testDDS\source\package* as well.

   Note: Normally you need to import files from the *\testDDS\source\hdl* subfolder to the HDL Source Files folder of the Libero IDE project; import files from the *\testDDS\source\stimulus* subfolder to the Stimulus Files folder of the Libero IDE project;

and import files from the *\testDDS\source\package* subfolder to the Package folder of the Libero IDE project. Refer to Libero IDE documentation for specific instructions.

8. Import the constraint file *\testDDS\source\dds_top.sdc* to the Constraint Files folder of the Libero IDE project.
9. In the Libero IDE, set **dds_top** as a root.
10. Successful installation of CoreDDS source files results in a Libero IDE screen similar to the one shown in Figure 2-1 on page 19 (Verilog project) or Figure 2-2 on page 20 (VHDL project).

Pre-Synthesis Simulation Flow

Pre-synthesis simulation verifies the correctness of the code generated.

Use the following procedure to run pre-synthesis simulation in Libero IDE. The same procedure applies to both Evaluation and Full versions.

To run pre-synthesis simulation:

1. In the Libero IDE GUI, select **Options** > **Project Settings**. In the Project Settings window, choose the **Simulation** tab. Specify the simulation run time **–all**. Click **OK**.
2. Run simulation.
   - Click the **Simulation – Model*Sim*** button in the Libero IDE GUI Design Flow window to start pre-synthesis simulation.
   - When prompted, set up stimulus:

     Select **Associate with stimulus**. Click **OK.**

     If in a Verilog project, select the following stimulus files:

     dds_tb.v

     goldSin.v

     goldPhase.v

     If in a VHDL project, select the following stimulus files:

     dds_tb.vhd

     goldSin.vhd

     goldPhase.vhd

     bhv_pack.vhd

     bhv.vhd

     Click **OK.**

   Model*Sim* starts automatically.

3. Since LUT-based architectures may take significant time, depending on the actual core configuration parameters, the Model*Sim* screen notifies you of the estimated simulation length, as follows:

```
# ----------------------------------------------------------------
#  Initializing a LUT...  It takes simulation time of about 216 us
# ----------------------------------------------------------------
```

4. Successful simulation results in the following message on the Model*Sim* screen:

```
##############################
#  DDS test passed
##############################
```

Synthesis in the Libero IDE

Click the **Synthesis** button in the Libero IDE GUI. The synthesis window appears, displaying the Synplicity project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To start synthesis, click the **RUN** button.

Place-and-Route in Libero IDE

Once the synthesis has successfully completed, click the **Place&Route** button in the Libero IDE GUI to invoke Designer. Make sure Designer imports the constraints file *dds_top.sdc*.

3 Interface Descriptions

Ports

The port signals for CoreDDS are defined in Table 3-1 and illustrated in Figure 3-1 on page 26.

Table 3-1. I/O Signal Descriptions

Name	Type	Description
nGrst	Input	Global asynchronous reset. Active low.
rstInit	Input	Synchronous reset for the LUT initialization circuitry. Active high.
rstPh	Input	Synchronous reset for the phase generation module. Active high. Can be used to reset phase generation circuitry without disturbing the initialized LUT.
clk	Input	System clock. Active rising edge.
clkEnInit	Input	Clock enable signal for the LUT initialization circuitry. Active high.
clkEnPh	Input	Clock enable signal for the phase generation circuitry. Active high.
PortPhMod [PHACC_WIDTH-1:0]	Input	Phase modulation port. Unsigned data present at the port add up to phase accumulator output. The port bit width equals the configurable phase accumulator bit width of PHACC_WIDTH.
PortFreqMod [PHACC_WIDTH-1:0]	Input	Frequency modulation port. Unsigned data present at the port are used to increment phase accumulator. The port bit width equals the configurable phase accumulator bit width of PHACC_WIDTH.
outSin [WORDSIZE-1:0]	Output	Output data bus carrying digitized waveform. User defined bus width = WORDSIZE. The bus carries a digitized sine wave if the core is configured to generate a real or complex sinusoid. It carries a cosine waveform if the core is configured to generate a real cosine function.
outCos [WORDSIZE-1:0]	Output	Output data bus carrying digitized cosine waveform. User defined bus width = WORDSIZE. The bus carries a digitized cosine wave if the core is configured to generate a complex sinusoid. If configured differently, the signal of the bus is undefined.
initDone	Output	Goes active (high) upon completion of the LUT initialization.
validOut	Output	Goes active (high) when the core starts generating valid output waveform samples.

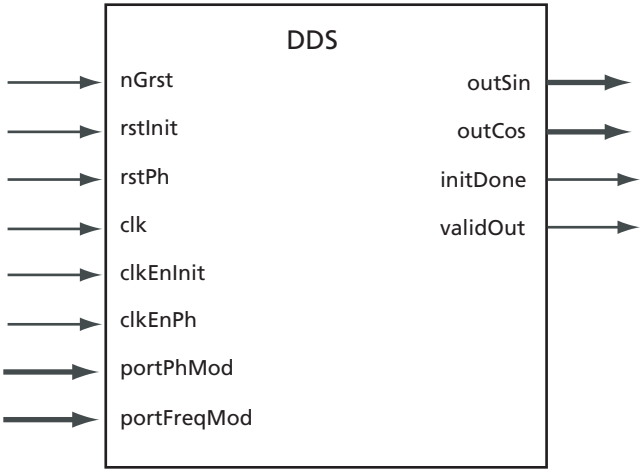

Figure 3-1. CoreDDS I/O Signals

Configuration Parameters

CoreDDS generates the DDS engine RTL code based on parameters set by the user. CoreDDS supports the variations specified in Table 3-2.

Table 3-2. CoreDDS Configuration Parameters

Name	Valid Value	Description
module_name	String of ≤ 50 characters	Name of the generated RTL code module or entity
function	0, 1, or 2	Output waveform type: sine (0), cosine (1), or quadrature (2)
architecture	0, 1, or 2	Defines DDS engine architecture: Small LUT (0), Big LUT (1), or CORDIC(2)
phAcc_mode	0 or 1	Sets a source of the phase increment value for the phase accumulator: constant (0), or the frequency modulation port portFreqMod (1).
phase_const	Unsigned PHACC_WIDTH-bit number	Value of the constant phase increment for the phase accumulator. It is used if phAcc_mode is set to 0.

Table 3-2. CoreDDS Configuration Parameters (Continued)

Name	Valid Value	Description
phase_modulation	0, 1, or 2	Sets a phase modulation mode: no phase modulation (0), constant phase shift (1), phase shift is obtained from the phase modulation port, portPhMod (2).
phase_mod_const	Unsigned PHACC_WIDTH-bit number	Value of the constant phase shift. It is used if phase_modulation is set to 1.
dithering	0 or 1	Enables dither mode. Phase gets dithered if this parameter is set to 1. Dithering takes effect only if the phase accumulator bit width PHACC_WIDTH is larger than bit width SLICE_WIDTH at the output of the phase quantizer.
dith_attenuation	0 to 7	Pseudorandom dithering signal attenuation factor. Dithering takes effect if the phase accumulator bit width, PHACC_WIDTH, is not less than the sum of the phase quantizer output bit width, SLICE_WIDTH, and the attenuation factor. dith_attenuation = 0 means there is no attenuation; dith_attenuation = 7 means the max attenuation applies.
phaseAcc_bitSize	4 to 32	Sets the bit width, PHACC_WIDTH, of the phase accumulator.
slicer_bitSize	4 to 32	Sets the bit width, SLICE_WIDTH, of the phase quantizer output. With LUT-based DDS architectures, the quantizer bit width is limited by the on-chip RAM capacity available on a selected part.
wave_bitSize	4 to 32	Sets the generated waveform bit resolution.
fpga_family	af, pa3, apa, or ax	Identifies Actel family of FPGA devices: Fusion (af), ProASIC3/E (pa3), ProASIC PLUS (apa), or Axcelerator/ RTAX (ax).
lang	verilog or vhdl	Identifies hardware description language for the RTL code and testbench to be generated.

4

Testbench Operation and Modification

Verification Testbench

Included with the releases of CoreDDS is a verification testbench that verifies operation of the CoreDDS engine. A simplified block diagram of the verification testbench is shown in Figure 4-1. The verification testbench instantiates the DDS Engine configured by the user, as well as a signal generator that provides necessary clock, reset, and other signals. The testbench compares the actual DDS output(s) and golden test vectors, Golden Sine and/or Golden Cosine, automatically generated by CoreDDS. CoreDDS generates Verilog or VHDL testbench behavioral code based on the user selection of the core language.

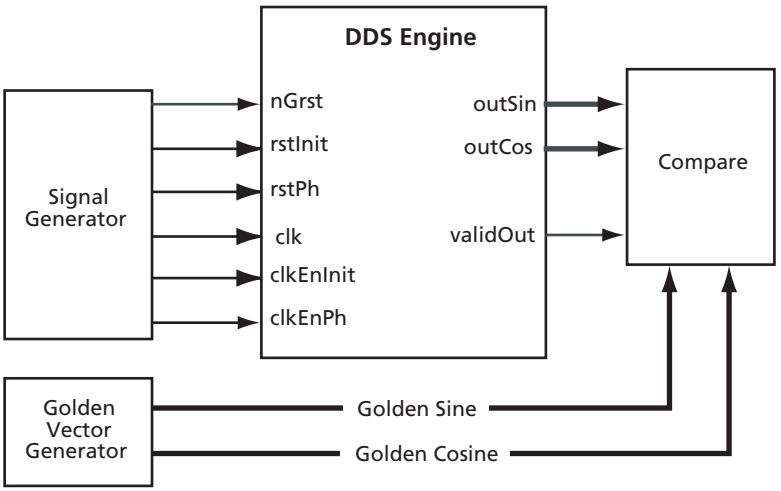

Figure 4-1. CoreDDS Verification Testbench

The testbench instantiates the DDS engine and verifies it is configured according to the user-defined parameters[4]. The same testbench can be used for pre-synthesis and post-synthesis simulation. A simulation tool displays the verification result.

4. The verification testbench assumes no modulation ports are present.

A

CoreDDS Configurations

The user defines the DDS engine configuration via a configuration file.

Sample Configuration File

```
module_name
testDDS

function
0

architecture
0

phAcc_mode
0

phase_const
977

phase_modulation
1

phase_mod_const
12

dithering
1

dith_attenuation
1

phaseAcc_bitSize
```

```
24

slicer_bitSize
12

wave_bitSize
14

fpga_family
ax

lang
verilog
```

Evaluation Version Configuration of CoreDDS

The Evaluation version implements the DDS engine configuration shown in Table A-1.

Table A-1. CoreDDS Evaluation Version Configuration

Name	Value
module_name	testDDS
function	0
architecture	0
phAcc_mode	0
phase_const	569
phase_modulation	1
phase_mod_const	20
dithering	1
dith_attenuation	1
phaseAcc_bitSize	20
slicer_bitSize	8
wave_bitSize	8
fpga_family	af
lang	verilog or vhdl

The user can only select the HDL type by changing the 'lang' parameter value. Any other modifications of the evaluation configuration file are ignored. The evaluation configuration file is shown below:

```
module_name
testDDS

function
0

architecture
0

phAcc_mode
0

phase_const
569

phase_modulation
1

phase_mod_const
20

dithering
1

dith_attenuation
1

phaseAcc_bitSize
20
```

```
slicer_bitSize
8

wave_bitSize
8

fpga_family
af

lang
verilog
```

B

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/contact/offices/index.html.

Index

For more information about Actel's products, visit our website at http://www.actel.com

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 USA*
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd*. • Dunlop House, Riverside Way • Camberley, Surrey GU15 3YL • United Kingdom*
Phone +44 (0) 1276 401 450 • Fax +44 (0) 1276 401 490

Actel Japan *• EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • www.jp.actel.com

Actel Hong Kong *• Suite 2114, Two Pacific Place • 88 Queensway, Admiralty Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200078-0/9.06