DDS Compiler v6.0

LogiCORE IP Product Guide

Vivado Design Suite

PG141 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Xilinx® LogiCORE™ IP Direct Digital Synthesizer (DDS) Compiler core implements high performance, optimized *Phase Generation* and *Phase to Sinusoid* circuits with AXI4-Stream compliant interfaces.

The core sources sinusoidal waveforms for use in many applications. A DDS consists of a *Phase Generator* and a *SIN/COS Lookup Table (phase to sinusoid conversion)*. These parts are available individually or combined using this core.

Features

- Phase Generator and SIN/COS Lookup table can be generated individually or together with optional dither to provide a complete DDS solution.
- Rasterized feature eliminates phase noise from phase truncation.
- Sine, cosine, or quadrature outputs.
- Optional per-channel resynchronization of accumulated phase.
- Lookup table can be stored in distributed or block RAM.
- Optional phase dithering spreads the spectral line energy for greater Spurious Free Dynamic Range (SFDR).
- Phase dithering or Taylor series correction options provide high dynamic range signals using minimal FPGA resources. Supports SFDR from 18 dB to 150 dB.
- Up to 16 independent time-multiplexed channels.
- Fine frequency resolution using up to 48-bit phase accumulator with DSP slice or FPAGA logic options.
- 3-bit to 26-bit signed output sample precision.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families, UltraScale™ Architecture Zynq®-7000 All Programmable SoC 7 Series
Supported User Interfaces	AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	VHDL
Constraints File	Not Provided
Simulation Model	Encrypted VHDL
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite System Generator for DSP
Simulation	For the supported simulators, see the Xilinx Design Tools: Release Notes Guide
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

Direct digital synthesizers (DDS), or numerically controlled oscillators (NCO), are important components in many digital communication systems. Quadrature synthesizers are used for constructing digital down and up converters, demodulators, and implementing various types of modulation schemes, including PSK (phase shift keying), FSK (frequency shift keying), and MSK (minimum shift keying). A common method for digitally generating a complex or real valued sinusoid employs a lookup table 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. A simple user interface accepts system-level parameters such as the desired output frequency and spur suppression of the generated waveforms.

Applications

- Digital radios and modems
- Software-defined radios (SDR)
- Digital down/up converters for cellular and PCS base stations
- Waveform synthesis in digital phase locked loops
- Generating injection frequencies for analog mixers

Licensing and Ordering Information

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Figure 2-1 provides a block diagram of the DDS Compiler core. The core consist of two main parts, a Phase Generator and SIN/COS LUT, which can be used independently or together with an optional dither generator to create a DDS capability. A time-division (TDM) multichannel capability is supported, with independently configurable phase increment and offset parameters.

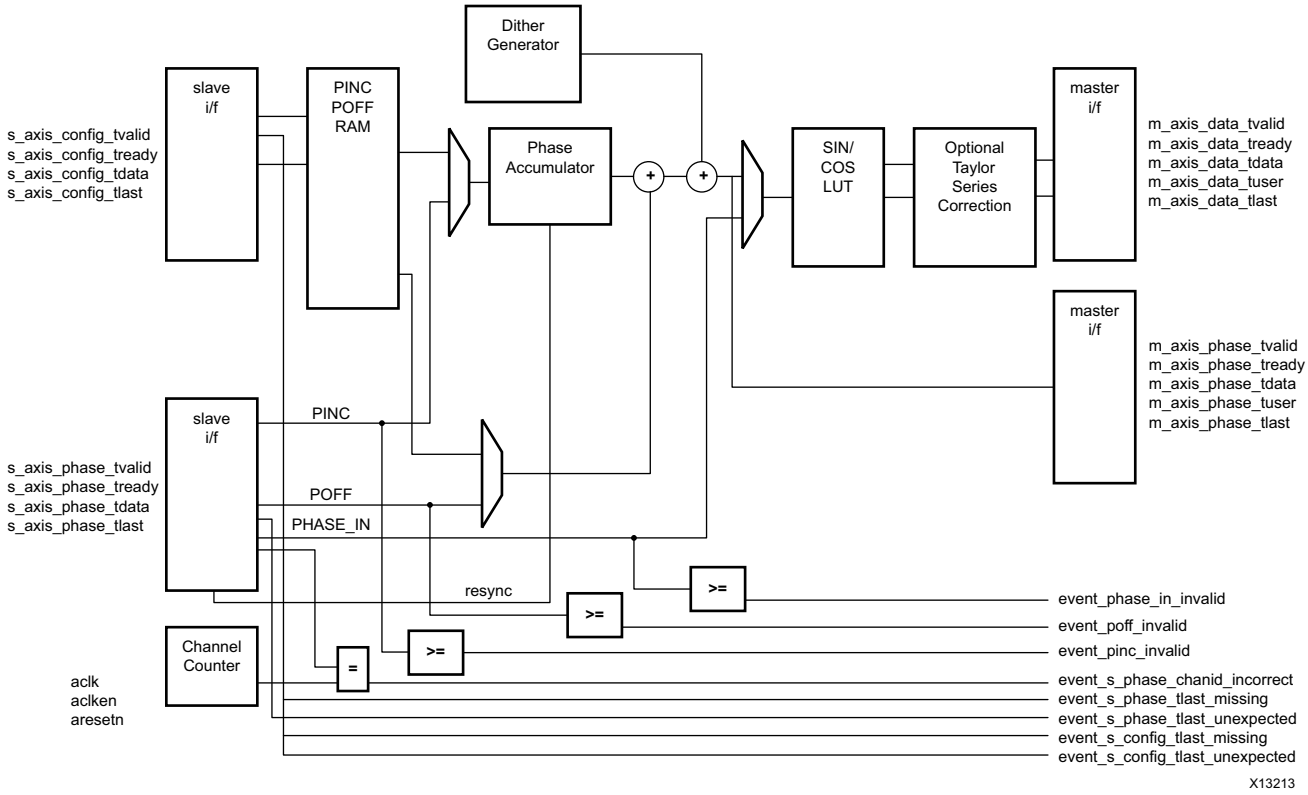

Figure 2-1: **DDS Core Architecture**

Phase Generator

The Phase Generator consists of an accumulator followed by an optional adder to provide addition of phase offset. When the core is customized, the phase increment (PINC) and phase offset (POFF) can be independently configured to be either fixed, programmable (using the CONFIG channel) or streaming (using the input PHASE channel).

When set to fixed, the DDS output frequency is set when the core is customized and cannot be adjusted after the core is embedded in a design.

When set to programmable, the CONFIG channel TDATA field has a subfield for the input in question (PINC or POFF) or both if both have been selected to be programmable. If neither PINC nor POFF is set to programmable, there is no CONFIG channel.

When set to streaming, the input PHASE channel TDATA field has a subfield for the input in question (PINC or POFF) or both if both have been selected to be streaming. If neither PINC nor POFF is set to streaming, and the core is configured to have a Phase Generator, then there is no input PHASE channel.

When PINC is set to streaming, an optional RESYNC streaming input can be configured. When asserted, this signal resets the accumulated phase of the channel in question.

When rasterized mode is selected, the hardware values of PINC and POFF that are input or configured must be 0 to Modulus-1. This corresponds to a full circle. So for negative PINC or POFF values, add the Modulus to the negative value desired to map to the required range. For example with Modulus = 100, the required range is 0 to 99. An angle of -90° would be -25 with this Modulus. Adding 100 gives 75 (270°).

When using system parameters, PINC and POFF are not input directly, but are calculated from the input Output Frequencies and Phase Angles. Beware that for small values of Modulus, the available values are relatively far apart, so the actual output frequency or phase angle may differ significantly from the desired value as displayed in the Additional Summary tab.

SIN/COS LUT

When configured as a SIN/COS LUT only, the Phase Generator is not implemented and the PHASE_IN signal is input using the input `PHASE` channel and transformed into sine and cosine outputs using a look-up table. Efficient memory usage is achieved by exploiting the symmetry of sinusoid waveforms. The core can be configured for sine only output, cosine only output or both (quadrature) output. Each output can be configured independently to be negated. Precision can be increased using optional Taylor series correction. This exploits DSP slices on FPGA families that support them to achieve high SFDR with high speed operation.

Phase Generator and SIN/COS LUT (DDS)

The Phase Generator is used in conjunction with the SIN/COS LUT to provide either a phase truncated DDS or Taylor series corrected DDS. An optional dither generator can be added between the two blocks to provide a phase dithered DDS.

Performance

This section details the performance information for various core configurations.

Latency

The latency of the core can be specified through the user interface in the Vivado Integrated Design Environment (IDE) or be automatically set to the optimum value based upon the Optimization Goal.

For streaming inputs (`s_axis_phase_t*`) the latency specifies the minimum number of cycles between input and the associated output.

For configuration inputs, the latency is the minimum latency from the first cycle `aresetn` becomes inactive until a valid output. The latency from configuration channel inputs to output channel outputs is not deterministic because configuration inputs are synchronized to the internal channel counter phase, thus introducing a delay which is unknown external to the core.

Throughput

The DDS Compiler core supports full throughput in all configurations (one output for every cycle).

Resource Utilization

For details about resource utilization, visit Performance and Resource Utilization.

Port Descriptions

The DDS Compiler core pinout is shown in Figure 2-2. All of the possible pins are shown, though the specific pins in any instance depend upon parameters specified when the core is generated.

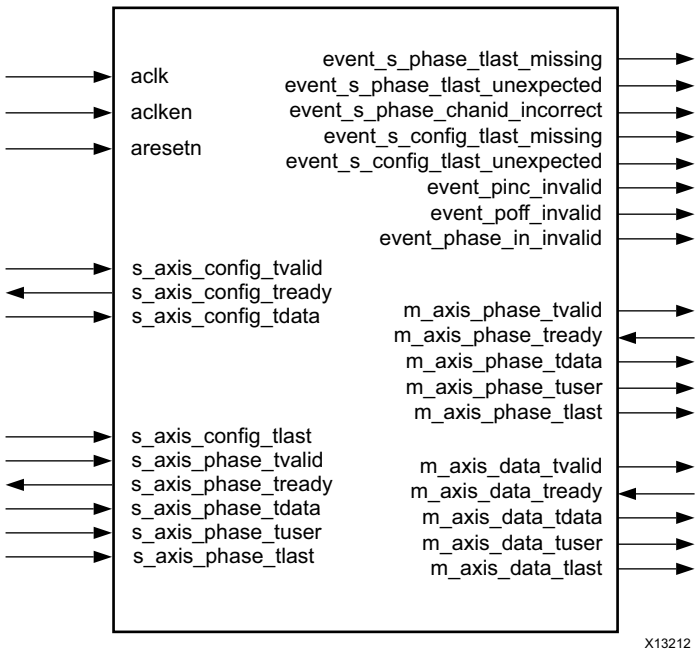

Figure 2-2: **Core Pinout**

Table 2-1 summarizes the pinout of the core. If an active-Low input is required for a specific control pin, an inverter must be placed in the path to the pin and is absorbed appropriately during mapping.

Table 2-1: **Core Signal Pinout**

Name	Direction	Optional	Description
aclk	Input	No	Rising edge clock
aclken	Input	Yes	Active-High clock enable
aresetn	Input	Yes	Active-Low synchronous clear. Always takes priority over aclken. aresetn must be driven Low for a minimum of two cycles to reset the core.
s_axis_config_tvalid	Input	Yes	TVALID for CONFIG channel
s_axis_config_tready	Output	Yes	TREADY for CONFIG channel
s_axis_config_tdata(1)	Input	Yes	TDATA for CONFIG channel. See CONFIG Channel TDATA Structure for internal structure and width,
s_axis_config_tlast	Input	Yes	TLAST for CONFIG channel. See CONFIG Channel.
s_axis_phase_tvalid	Input	Yes	TVALID for input PHASE channel
s_axis_phase_tready	Output	Yes	TREADY for input PHASE channel
s_axis_phase_tdata(1)	Input	Yes	TDATA for input PHASE channel. See Input PHASE Channel TDATA Structure for internal structure and width.
s_axis_phase_tuser	Input	Yes	TUSER for input PHASE channel. See Input PHASE Channel TUSER Structure for internal structure.

Table 2-1: **Core Signal Pinout *(Cont'd)***

Name	Direction	Optional	Description
s_axis_phase_tlast	Input	Yes	TLAST for input PHASE channel. See Input PHASE Channel TLAST Options.
m_axis_phase_tvalid	Output	Yes	TVALID for output PHASE channel.
m_axis_phase_tready	Input	Yes	TREADY for output PHASE channel.
m_axis_phase_tdata	Output	Yes	TDATA for output PHASE channel. See Output PHASE Channel TDATA Structure - Conventional DDS.
m_axis_phase_tuser	Output	Yes	TUSER for output PHASE channel. See Output PHASE Channel TUSER Structure.
m_axis_phase_tlast	Output	Yes	TLAST for output PHASE channel. See Output PHASE Channel TLAST Options.
m_axis_data_tvalid	Output	Yes	TVALID for output DATA channel.
m_axis_data_tready	Input	Yes	TREADY for output DATA channel.
m_axis_data_tdata	Output	Yes	TDATA for output DATA channel. See Output DATA Channel TDATA Structure.
m_axis_data_tuser	Output	Yes	TUSER for output DATA channel. See Output DATA Channel TUSER Structure.
m_axis_data_tlast	Output	Yes	TLAST for output DATA channel. See Output DATA Channel TLAST Options.
event_s_phase_ tlast_missing	Output	No (But leave unconnected to remove associated circuitry)	Asserted when the transfer to the s_axis_phase channel for the last channel (in multichannel configurations) does not have tlast asserted.
event_s_phase_ tlast_unexpected	Output		Asserted when TLAST is asserted for a the transfer to the s_axis_phase channel which is not for the last channel (in multichannel configurations).
event_s_phase_ chanid_incorrect	Output		Asserted when the chanid field (subfield of TUSER in s_axis_phase) does not agree with the internal expectation of channel number.
event_pinc_invalid	Output		Asserted when the value of PINC to be accumulated is out of range. Applies to rasterized configurations only.
event_poff_invalid	Output		Asserted when the value of POFF to be accumulated is out of range. Applies to rasterized configurations only.
event_phase_ in_ invalid	Output		Asserted when the value of Phase_In is out of range. Applies to rasterized and SIN_COS_LUT configurations only.
event_s_config_ tlast_missing	Output		Asserted when the last transfer (for the last channel) to the s_axis_config channel is not accompanied with TLAST asserted.
event_s_config_ tlast_unexpected	Output		Asserted when a transfer to the s_axis_config channel has TLAST asserted when it is not the transfer associated with the last channel.

Notes:

1. When in rasterized mode, the input values of PINC, POFF or PHASE_IN must be between 0 and Modulus-1.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Description

Direct digital synthesizers (DDS), or numerically controlled oscillators (NCO), are important components in many digital communication systems. Quadrature synthesizers are used for constructing digital down and up converters, demodulators, and implementing various types of modulation schemes, including PSK (phase shift keying), FSK (frequency shift keying), and MSK (minimum shift keying). A common method for digitally generating a complex or real valued sinusoid employs a lookup table 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. A simple user interface accepts system-level parameters such as the desired output frequency and spur suppression of the generated waveforms.

Theory of Operation

The standard mode of the DDS Compiler core uses phase truncation, as shown in Figure 3-1.

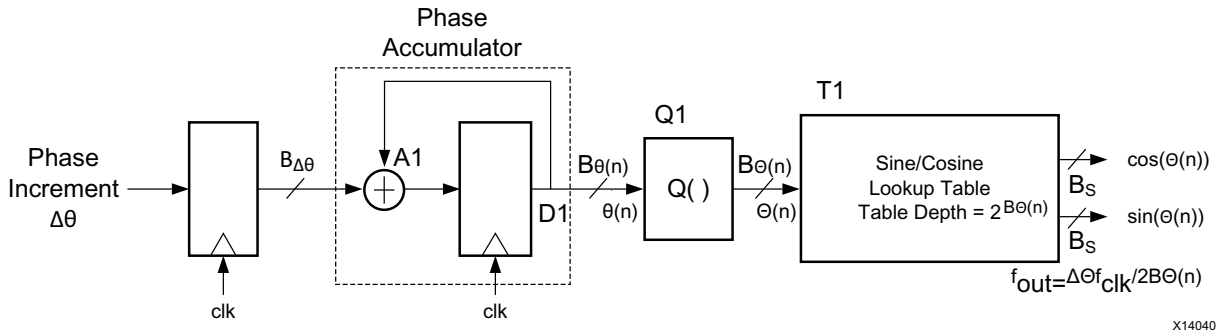

Figure 3-1: **Phase Truncation DDS (Simplified View of the DDS Core)**

The integrator (components D1 and A1) computes a phase slope that is mapped to a sinusoid (possibly complex) by the lookup table T1. The quantizer Q1, which simply

truncates, accepts the high-precision phase angle $\theta(n)$ and generates a lower precision representation of the angle denoted as $\Theta(n)$ in Figure 3-1. This value is presented to the address port of a lookup table that performs the mapping from phase-space to time.

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 depth and width of the lookup table affect the phase angle resolution and the amplitude resolution of the signal, respectively. See Spectral Purity Considerations for more details.

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 prototype complex sinusoid of length $N = 2^{B_{\Theta(n)}}$ and correspond to specific values of the sinusoid argument $\Theta(n)$ as follows:

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

where n is the time series sample index.

Quarter 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 $\Theta(n)$ are used to perform quadrant mapping. This implementation results in a more resource efficient implementation because the memory requirements are minimized, offering either fewer FPGA block RAMs or reduced distributed memory. Based on the core customization parameters, the DDS core automatically employs quarter-wave or half-wave symmetry when appropriate.

Note: For very shallow tables, FPGA logic resources are actually minimized by storing a complete cycle. No design decisions are required in this context; the Xilinx tools always produce the smallest core possible.

The rasterized mode of operation of the DDS does not truncate the accumulated phase. Rasterized operation is intended for configurations where the desired frequency is a rational fraction of the system clock (output frequency = system frequency * N/M, where 0 < N < M). Values of M from 9 to 16384 are supported. The SIN/COS LUT is configured accordingly for values from 0 to M-1, which describe a full circle. Because there is no phase truncation in the rasterized mode of operation, there is no need for dither or Taylor correction because these mitigate the effects of phase truncation. In rasterized operation, the phase noise is significantly reduced. Therefore, the output phase angle resolution and amplitude resolution are determined by the LUT table output width alone. In rasterized mode, quadrant symmetry is exploited where applicable to reduce memory use.

Output Frequency

This section covers output frequency when using the core in standard mode or rasterized mode.

Standard Mode of Operation

The output frequency, f_{out}, of the DDS waveform is a function of the system clock frequency, f_{clk}, the phase width, that is, number of bits ($B_{\theta(n)}$) in the phase accumulator and the phase increment value $\Delta\theta$. The output frequency in Hertz is defined by:

$$f_{out} = \frac{f_{clk}\Delta\theta}{2^{B_{\theta(n)}}}$$

For example, if the DDS parameters are:

- f_{clk} = 120 MHz
- $B_{\Theta(n)}$ = 10
- $\Delta\Theta$ = 12 (decimal)

Then the output frequency is calculated as follows:

$$\begin{aligned} f_{out} &= \frac{f_{clk}\Delta\theta}{2^{B_{\theta(n)}}} Hz \\ &= \frac{120 \times 10^6 \times 12}{2^{10}} \\ &= 1.406250\ MHz \end{aligned}$$

The phase increment value $\Delta\theta$ required to generate an output frequency f_{out} Hz is:

$$\Delta\theta = \frac{f_{out}2^{B_{\theta(n)}}}{f_{clk}}$$

If the DDS core is time-division multiplexed to do multiple channels, then the effective clock frequency per channel is reduced. For C channels, the phase increment required is:

$$\Delta\theta = \frac{Cf_{out}2^{B_{\theta(n)}}}{f_{clk}}$$

Rasterized Mode of Operation

The output frequency f_{out} of the DDS waveform for a single channel configuration is a function of the system clock frequency f_{clk}, the modulus M and the phase increment value ΔΘ. The output frequency in Hertz is defined by:

$$f_{out} = \frac{f_{clk}\Delta\theta}{M}$$

For example, if the DDS parameters are

- f_{clk} = 120 MHz
- M = 1000
- ΔΘ = 12 (decimal)

Then the output frequency is calculated as follows:

f_{out}= 120 x 10^6 x 12 / 1000

f_{out}= 1.44 MHz

The phase increment value ΔΘ required to generate an output frequency f_{out} Hz is:

$$\Delta\theta = \frac{f_{out}M}{f_{clk}}$$

If the DDS core is time-division multiplexed to do multiple channels, then the effective clock frequency per channel is reduced. For C channels, the phase increment required is:

$$\Delta\theta = \frac{Cf_{out}M}{f_{clk}}$$

Frequency Resolution

This section covers frequency resolution when using the core in standard mode or rasterized mode.

Standard Mode of Operation

The frequency resolution Δf of the synthesizer is a function of the clock frequency and the number of bits $B_{\theta(n)}$ employed in the phase accumulator. The frequency resolution can be determined using:

$$\Delta f = \frac{f_{clk}}{2^{B_{\theta(n)}}}$$

For example, for the following DDS parameters:

- f_{clk} = 120 MHz
- $B_{\Theta(n)}$ = 32

the frequency resolution is:

$$\Delta f = \frac{f_{clk}}{2^{B_{\theta(n)}}}$$
$$= \frac{120 \times 10^6}{2^{32}}$$
$$= 0.0279396 \text{ Hz}$$

In the time-division multichannel case, the frequency resolution is improved by the number of channels, as follows:

$$\Delta f = \frac{f_{clk}}{2^{B_{\theta(n)}} C}$$

Rasterized Mode of Operation

The frequency resolution Δf of the synthesizer is a function of the clock frequency and the modulus. The frequency resolution can be determined using:

$$\Delta f = \frac{f_{clk}}{M}$$

For example, for the following DDS parameters:

- fclk = 120 MHz
- M = 1000

The frequency resolution is

= 120 x 10^6 / 1000

= 120 kHz

In the time-division multichannel case, the frequency resolution is improved by the number of channels, as follows:

$$\Delta f = \frac{f_{clk}}{MC}$$

Phase Increment

For standard mode, phase increment values in the range 0 to 2^{N-1} describes the range [0,360)° (where N is the number of bits in the phase accumulator). For rasterized mode, phase increment values must be considered unsigned due to the internal implementation. The phase increment values [0 to Modulus-1] describe the range [0,360].

The phase increment term $\Delta\theta$ defines the synthesizer output frequency. Consider a standard DDS with the following parameterization:

$$f_{clk} = 100 \text{ MHz}$$
$$B_{\theta(n)} = 18$$

To generate a sinusoid with frequency $f_{out} = 19 \text{ MHz}$, the required phase increment is:

$$\Delta\theta = \frac{f_{out}2^{B_{\theta(n)}}}{f_{clk}}$$
$$= \frac{19 \times 10^6 \times 2^{18}}{100 \times 10^6}$$
$$= 49807.36$$

This value must be truncated to an integer giving the following actual frequency:

$$f_{out} = \frac{\Delta\theta f_{clk}}{2^{B_{\theta(n)}}}$$
$$= \frac{49807 \times 100 \times 10^6}{2^{18}}$$
$$= 18.9998627 MHz$$

Consider a DDS in rasterized mode with the following parameterization:

- F_{clk} = 100 MHz
- M (modulus) = 1536

To generate a sinusoid with frequency f_{out} = 19 MHz, the required phase increment is:

$$\Delta\theta = \frac{f_{out}M}{f_{clk}}$$

$$\Delta\theta = \frac{19 \times 10^6 \times 1536}{100 \times 10^6}$$

$$\Delta\theta = 291.84$$

The closest integer value to this is 292, giving the following actual frequency:

$$f_{out} = \frac{f_{clk}\Delta\theta}{M}$$

$$f_{out} = \frac{292 \times 100 \times 10^6}{1536}$$

$$f_{out} = 19.0104167MHz$$

Spectral Purity Considerations

This section covers spectral purity considerations when using the core in standard mode or rasterized mode.

Standard Mode of Operation

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 depth and width of the lookup table affect the phase angle resolution and the amplitude resolution of the signal, respectively. These resolution limits are equivalent to time base jitter and amplitude quantization of the signal and add spectral modulation lines and a white broad-band noise floor to the signal spectrum.

In conjunction with the system clock frequency, the phase width determines the frequency resolution of the DDS. The accumulator must have a sufficient field width to span the desired frequency resolution. For most practical applications, a large number of bits are allocated to the phase accumulator to satisfy the system frequency resolution requirements. By way of example, if the required resolution is 1 Hz and the clock frequency is 100 MHz, the required width of the accumulator is:

$$\begin{aligned} B_{\theta(n)} &= \left\lceil \log_2\left(\frac{f_{clk}}{\Delta f}\right) \right\rceil \\ &= \left\lceil \log_2\left(\frac{100 \times 10^6}{1}\right) \right\rceil \\ &= \lceil 26.5754 \rceil \\ &= 27 \text{ bits} \end{aligned}$$

where $\lceil\ \rceil$ denotes the ceiling operator. Due to excessive memory requirements, the full precision of the phase accumulator cannot be used to index the sine/cosine lookup table. A quantized (or truncated) version of the phase angle is used for this purpose. The block labeled Q1 in the phase truncation DDS, Figure 3-1, performs the phase angle quantization. The lookup table can be located in block or distributed memory.

Quantizing the phase accumulator introduces time base jitter in the output waveform. This jitter results in undesired phase modulation that is proportional to the quantization error, as shown by the following:

$$\Theta(n) = \theta(n) + \delta\theta$$
$$e^{j\Theta(n)} = e^{j[\theta(n) + \delta\theta(n)]} = e^{j\theta(n)}e^{j\delta\theta(n)}$$
$$e^{j\Theta(n)} \approx e^{j\theta(n)}[1 + j\delta\theta(n)]$$
$$\approx e^{j\theta(n)} + j\delta\theta(n)e^{j\theta(n)}$$

Figure 3-2 shows the lookup table addressing error, complex output time-series, and the spectral domain representation of the output waveform produced by the DDS structure shown in Figure 3-1. The normalized frequency for this signal is 0.022 Hz, which corresponds to phase accumulation steps of 7.92° per output sample. The angular resolution of the 256-point lookup table is 360/256 or 1.40625° per address, which is equivalent to 7.92/1.40625 or 5.632 addresses per output sample. Because the address must be an integer, the fractional part is discarded and the resultant phase jitter is the cause of the spectral artifacts. Figure 3-3 provides an exploded view of the spectral plot in Figure 3-2 (c).

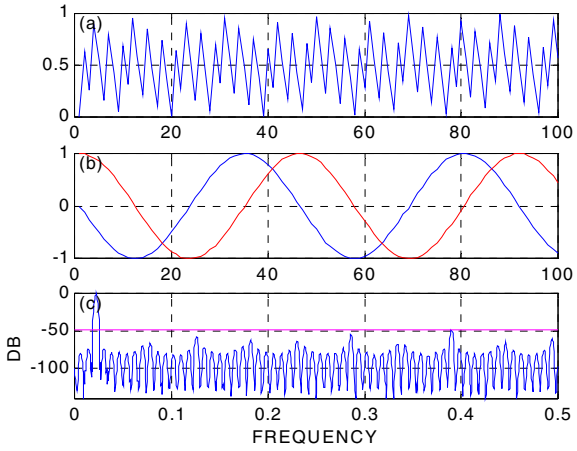

Figure 3-2: **Phase Truncation DDS. f_{out} = 0.022 Hz, Table Depth = 256 12-Bit Precision Samples. (a) Phase Angle Addressing Error (b) Complex Output Time Series (c) Output Spectrum**

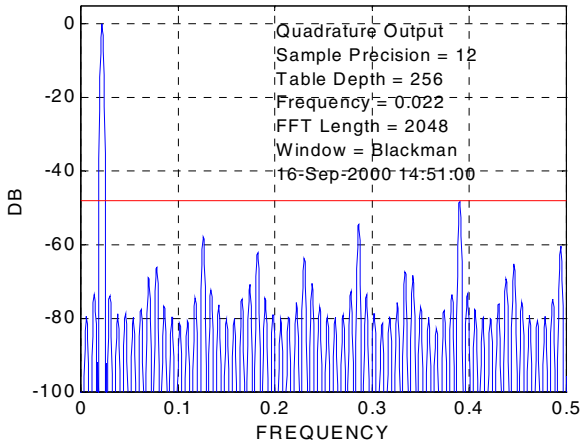

Figure 3-3: **Exploded View of Figure 3-2 (c)**

Two observations related to the phase jitter structure level can be made. First, observe that the fractional part of the address count is a periodic (sawtooth) error sequence, which is responsible for the harmonic rich (and aliased) low-level phase modulation evident in Figure 3-3. Also, the peak distortion level due to incidental phase modulation is approximately 48 dB below the desired signal level, which is consistent with 6 dB/bit of address space. Put another way, if S dB of spur suppression is required in the output waveform, as referenced to the 0 dB primary tone, the DDS lookup table must support at least $\lceil S/6 \rceil$ address bits. For example, if S = 70 dB, which means that the highest spur is 70 dB below the main signal, then the minimum number of address bits for the lookup table is $\lceil 70/6 \rceil = 12$ bits; that is, a 4096-deep table.

Figures 3-4 and 3-5 demonstrate the performance of a similar DDS to the one presented in Figure 3-2, but in this example, 16-bit precision output samples have been used. Observe that the highest spur is still at the –48 dB level, and allocating four additional bits to the output samples has not contributed to any further spur reduction. For a phase truncation DDS, the only option to further reduce the spur levels is to increase the depth of the lookup table.

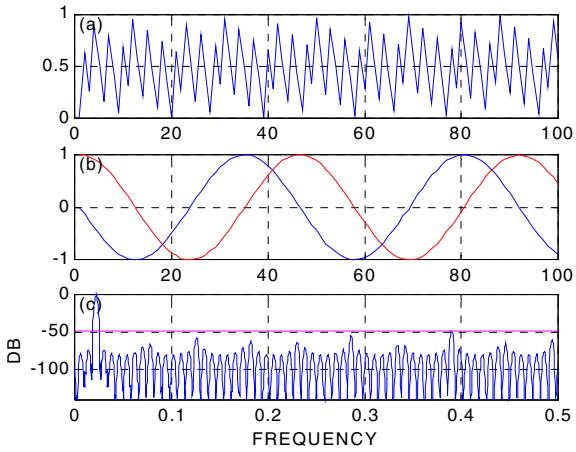

Figure 3-4: **Phase Truncation DDS. f_{out} = 0.022 Hz, Table Depth = 256 16-Bit Precision Samples. (a) Phase Angle Addressing Error (b) Complex Output Time Series (c) Output Spectrum**

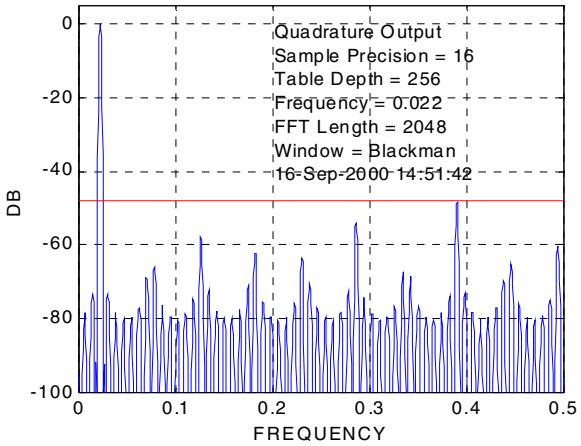

Figure 3-5: **Exploded View of Figure 3-4 (c)**

Rasterized Mode of Operation

In rasterized mode, the phase signal from the Phase Generator can be expressed exactly as an integer due to its rational fraction relationship to the system clock and the fact that no truncation of the phase vector occurs. The fidelity of the SINE/COSINE signals is therefore due only to the accuracy and precision of the SINE/COSINE table entries. Unlike the standard mode, there is no time base jitter. The effects of time base jitter described in Standard Mode of Operation do not affect a rasterized configuration.

Because there is no truncation of the phase vector from the phase accumulator and all possible values of phase have corresponding entries in the SINE/COSINE table, there is no need to compensate for or mitigate the phase error because there is no phase error. Dither and Taylor series correction are two techniques to reduce the effect of phase error. Because they are not required, they are not offered in rasterized mode.

The expected Spurious Free Dynamic Range of a rasterized design is 6 dB per bit of the output data field.

Phase Dithered DDS

IMPORTANT: *Phase Dithering is only an option in standard mode of operation.*

In the phase truncation DDS architecture shown in Figure 3-1, the quantizer Q1 introduces a phase error in the phase slope by discarding the least significant part of the high-precision phase accumulator. The phase error due to the discarded fractional part of the address count is a periodic series which results in an undesired spectral line structure. Figure 3-6 provides an example of this process for a DDS with a table depth N = 1024 and table sample precision of 16 bits. Figure 3-6 (a) is the phase error generated by taking the difference between the quantizer input and output signals, Figure 3-6 (b) is the output time series and Figure 3-6 (c) is the signal output spectrum. Observe in Figure 3-6 (a) the periodic sawtooth structure of the phase error signal. The line spectrum associated with this correlated error sequence is impressed on the final output waveform and results in spectral lines in the synthesizer output spectrum. These spurious components can be clearly seen in Figure 3-6 (c).

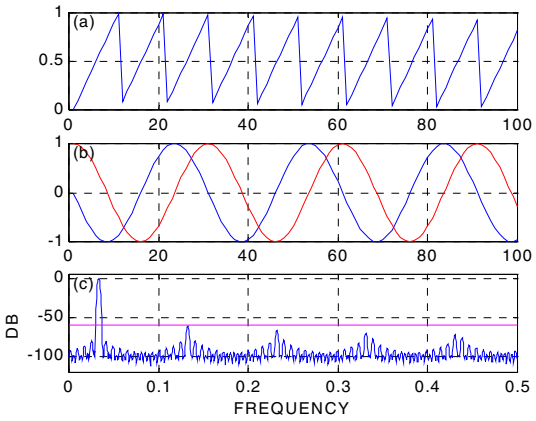

Figure 3-6: **DDS Plots Showing (a) Phase Error Time Series (b) Complex Output Time Series (c) Output Spectrum. 1024 Deep Lookup Table, 16-Bit Samples, Output Frequency 0.333 Hz**

This structure can be suppressed by breaking up the regularity of the address error with an additive randomizing signal. This randomizing sequence, called *dither*, is a noise sequence, with variance approximately equal to the least significant integer bit of the phase accumulator. The dither sequence is added to the high-precision accumulator output prior to quantization by Q1.

The dithered DDS supplies, approximately, an additional 12 dB of spurious free dynamic range (SFDR) in comparison to a phase truncation design. This is achieved by spreading the spectral energy of the phase error signal. The additional logic resources required to implement the dither sequence generator are not significant.

To provide S dB of spur suppression using a phase truncation DDS, as referenced to the 0 dB primary tone, the internal lookup table must support at least $\lceil S/6 \rceil$ address bits. To achieve this same performance using the dithered architecture requires two fewer address bits, minimizing the number of block RAMs (or logic slices for a distributed memory implementation) used in the FPGA implementation. In summary, for a dithered DDS implementation, the number of address bits needed to support S dB spur suppression is equal to $\lceil S/6 \rceil - 2$.

Figures 3-7 and 3-8 provide the results for several dithered DDS simulations. Figure 3-7 shows eight simulations for a complex dithered DDS employing a table depth N = 4096 and 16-bit precision samples. For each plot the output frequency is different and is annotated on the plot. A phase truncation design would typically generate output spurs 72 dB below the output frequency, independent of the actual value of the output frequency. Indicated on each of the plots by the parameter *A* is the peak spur level achieved for the simulation. The eight spurs are –88.12, –88.22, –86.09, –88.80, –87.21, –87.55, –87.83, –87.12 dB below the output frequency. The worst case value of –86.09 is 14.09 dB better than a similarly configured phase truncation DDS.

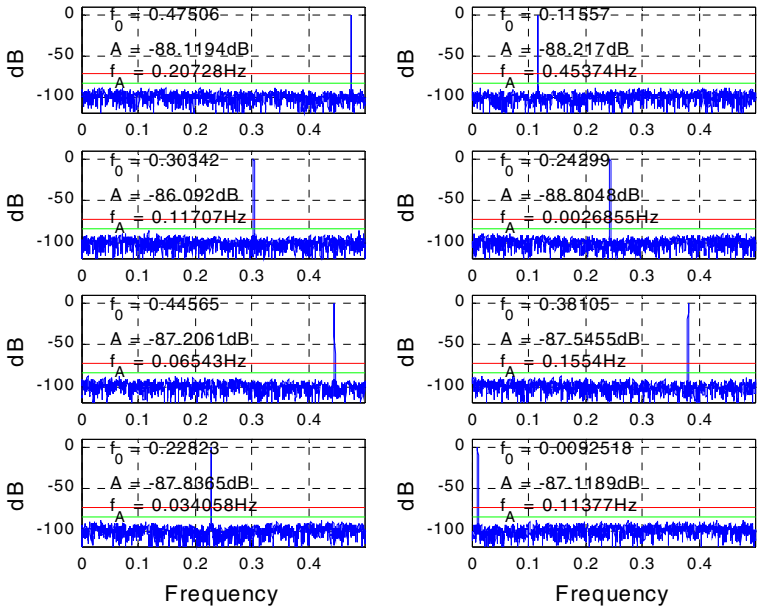

Figure 3-7: **Dithered DDS Simulations. The DDS configuration is N = 4096, B_s = 16**

The eight plots are spectral domain representations for eight different output frequencies. Each plot is annotated with the peak spur.

To achieve this same SFDR by extending the table length of a phase truncation design would require increasing the table depth by more than a factor of four.

Figure 3-8 provides one more dithered DDS simulation where the output frequency is swept over a band of frequencies. The spectrum for each discrete tone in the sweep band is overlaid to construct the final plot. The sweep start frequency, end frequency, number of tones in the sweep, and DDS configuration are annotated on the plot.

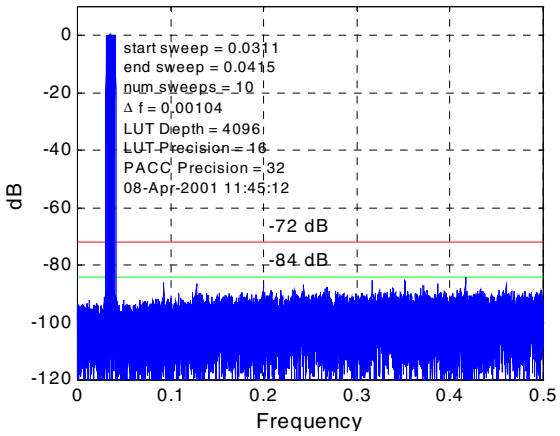

Figure 3-8: **Example Plot for Dithered DDS Simulation with Frequency Sweep**

In Figure 3-8, the synthesized signal is swept over a range of frequencies starting from 0.0311 to 0.0415 Hz. There are ten tones in the sweep separated in frequency by 0.00104 Hz. In this example, the phase truncation DDS would produce peak spurs at −72 dB with respect to the 0 dB primary signal. The dithered DDS provides approximately 12 dB better performance with the peak spur −84 dB below the output signal.

A further advantage of the dithered DDS is that the spectral line structure present in a phase truncation design is removed and the out-of-band signal is significantly whitened. This white broadband noise floor is more desirable than the line structured spectrum. In digital communication receivers that use a DDS for generating mixing signals for performing channelization functions, the spurs in a phase truncation DDS can act as low-level mixing tones and cause undesirable spectral contamination of the desired channel. For virtually all applications, the preferred implementation is the dithered DDS.

Taylor Series Corrected DDS

IMPORTANT: *Taylor series correction is only an option in standard mode of operation.*

The phase dithered DDS, as well as the phase truncation DDS, have a quantizer Q1 that produces a lower precision $\Theta(n)$ by discarding the fractional component of the high precision $\theta(n)$. The reason for this quantization step is to keep the size of the lookup memory to a reasonable size. The trade-off is spectral purity. With the availability of DSP slices in FPGAs, it is now practical to use the previously discarded fractional bits to calculate corrections that can be added to the lookup table values to produce outputs with very high SFDR.

Figure 3-9 through 3-12 show the simulation results of four different Taylor series corrected DDS simulations. The Taylor series corrected architecture in this example uses a table depth N = 4096 and 18-bit precision samples. However, the precision at the output of the feed-forward error processor is 20 bits. For each plot, the output frequency is different and annotated directly on the plot. A similarly configured phase truncation DDS would produce

spurs at –72 dB and a phase dithered DDS at –84 dB. The peak spurs for the four plots are –118.25, –118.13, –118.10, and –118.17 dB below the output frequency.

Figure 3-13 shows a swept frequency Taylor series corrected DDS. The starting frequency for this example is 0.0313 Hz, the final frequency is 0.0813, and there are 100 tones in the sweep. Using this configuration, a phase truncation DDS would produce peak spurs at approximately 72 dB below the output signal and a phase dithered DDS would produce peak spurs at approximately 84 dB below the output signal. As shown in the plot, the Taylor series corrected DDS produced spurs that are all the way down to 118 dB below the output signal. This result is 34 dB better than the phase dithering DDS, 46 dB better than the phase truncation DDS, and still only consumes a single 18Kb block RAM for the lookup storage. Figure 3-14 shows another frequency sweep simulation with 35 tones over a broader frequency range.

As shown in the plots, linear correction of the RAM values can extend the SFDR to 118 dB using only a single block RAM and three multipliers. To achieve SFDR beyond 118 dB, it is necessary to deepen the RAM or to use quadratic correction (an extra term of the Taylor series). Because the RAM size would double for each additional 6 dB, the DDS Compiler uses quadratic correction to achieve SFDR values of up to 150 dB. Introducing the extra term of the Taylor series expansion of Sine or Cosine requires an additional multiplier per sine and cosine output and an additional block RAM to both scale and square the phase error.

Optimization of Memory Usage

The Taylor Series Correction implementation in the DDS Compiler core typically results in an SFDR higher than that requested in order to guarantee SFDR. This results in extra block RAMs for values of SFDR above 102 dB. However, in many cases, depending on the phase increment values used, a specified SFDR target value of 102 dB provides higher SDFR, but with one 18K block RAM.

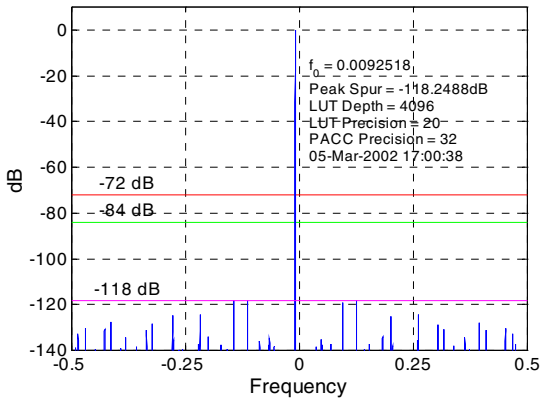

Figure 3-9: **Taylor Series Corrected DDS – Single-Tone Test, f_0 = 0.0092518**

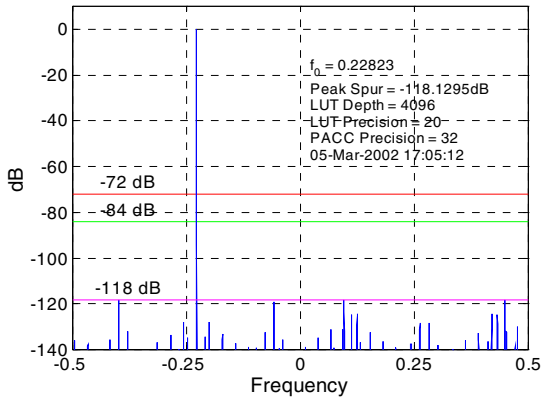

Figure 3-10: **Taylor Series Corrected DDS – Single-Tone Test, f_0 = 0.22823**

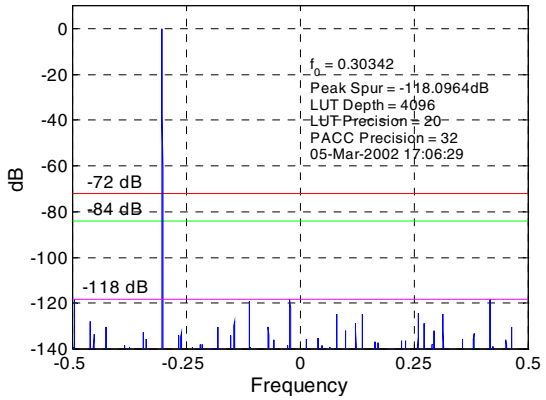

Figure 3-11: **Taylor Series Corrected DDS – Single-Tone Test, f_0 = 0.30342**

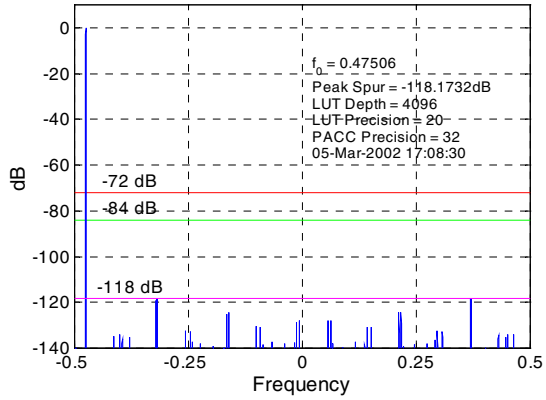

Figure 3-12: **Taylor Series Corrected DDS – Single-Tone Test, f_0 = 47506**

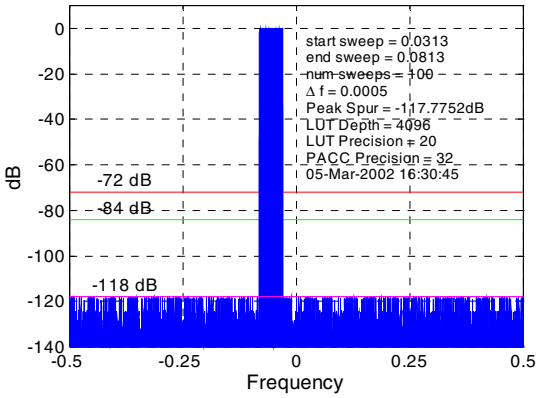

Figure 3-13: **Taylor Series Corrected DDS – Frequency Sweep Simulation, 100 Tones**

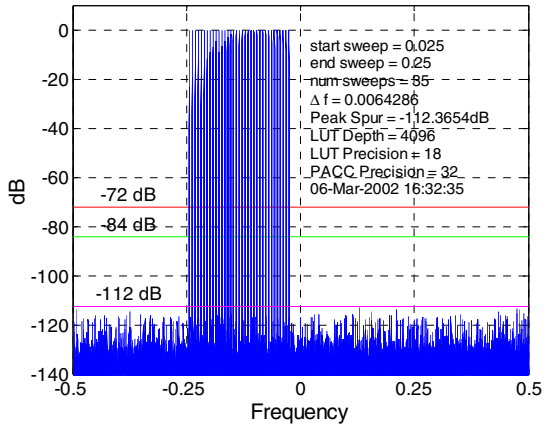

Figure 3-14: **Taylor Series Corrected DDS – Frequency Sweep Simulation, 35 Tones**

Multichannel

When configured for more than one channel, the DDS, or Phase Generator part, generates outputs for each channel in a time-multiplexed fashion. As such, the output for a particular channel occurs every N-cycles, where N is the number of channels selected when the core was customized. The outputs for channel 0 are given first.

Design Examples

The DDS Compiler user interface accepts system-level parameters instead of low-level parameters such as the width of the phase accumulator or width of the phase angle. Because of this, all preceding requirements can be entered into the Vivado IDE directly

without having to calculate low-level core details. The user interface in the Vivado IDE also provides feedback of the hardware parameters so the translation of system-level parameters to low-level parameters can been seen. Alternatively, hardware parameters can be entered directly.

Example 1

In standard mode, single-channel DDS with 1 MHz system clock, frequency resolution of 1 Hz, Phase Width is 20-bits. To synthesize an output of 23.4 kHz, an Output Frequency value of 0.0234 MHz must be entered into the user interface, which then returns a value of 5FD8 in hexadecimal, which is 24536 in decimal.

This gives a synthesized frequency of $24536/2^{20}$ *1 MHz = 23399.35 Hz.

If the application requires this to be modulated by one of 8 phase offsets, the phase offset bus need only be 3-bits precision, but these must be the top 3 bits of the phase offset input. Hence, the phase offset of 1/8 of a cycle would be entered as 0.125 in the user interface. This returns a value of 20000(hex). This could be entered on the 3-bit bus as 001(binary).

Example 2 (DDS Requiring Negative Frequencies)

In standard mode, single-channel DDS with 100 MHz System Clock, frequency resolution of 1 Hz the Phase Width is 25-bits. Frequencies of -3 MHz, -1 MHz, 1 MHz and 3 MHz are required. F_S is the frequency per channel which is System Clock/Number of Channels, that is, 25 MHz. The negative frequencies alias to every F_S Hz. The legal range to enter in the user interface is 0 to F_S, so the entered frequencies for this example must be 22 MHz (F_S-3 MHz), 24 MHz (F_S-1 MHz), 1 MHz and 3 MHz respectively.

Clocking

The DDS has only one clock signal input, `aclk`. This is the system clock referred to in equations. There are no special timing or placement constraints. Because the DDS has only one clock, there are no clock domain crossing considerations.

Resets

The active-Low reset signal, `aresetn`, is registered within the DDS Compiler core to ensure high performance. This signal has the requirement that it must be driven Low for a minimum of two cycles to guarantee that there is no violation of the AXI4-Stream protocol

on the core outputs. The registering of reset appears as a delay in entering the reset state, and the signal states after reset differ depending on whether TREADY signals are present.

The following diagram shows the core entering reset when there are no TREADY signals, using a multichannel DDS as an example The subscript on the valid data outputs indicates a channel number. The true behavioral model outputs X values for TDATA fields during reset. All channel payloads are qualified with their respective TVALID outputs.

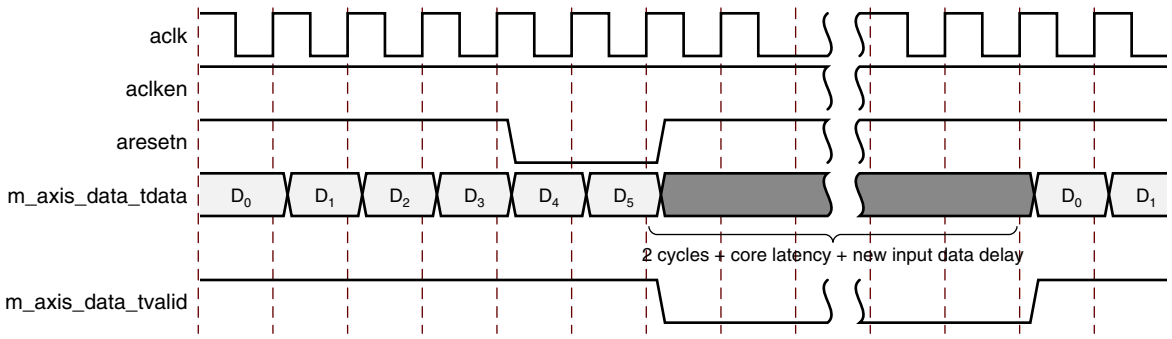

Figure 3-15: **Reset Behavior (no TREADY)**

Figure 3-16 shows the core entering reset when TREADY is present, using a multichannel DDS as an example. The subscript on the valid data outputs indicates a channel number. The reset event forces the FIFOs within the core back into their initial state; no X value is produced by the VHDL RTL-based behavioral model in this case. Again, all channel payloads are qualified with their respective TVALID outputs.

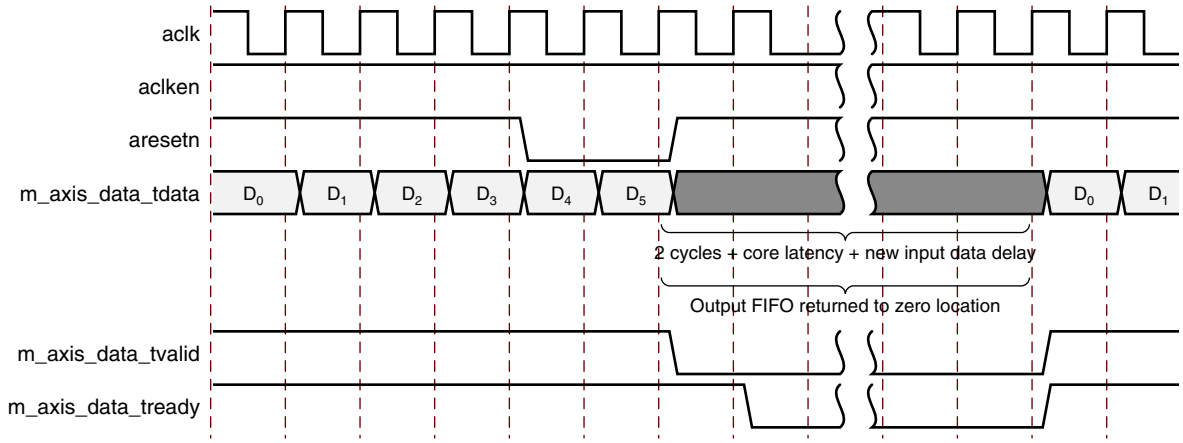

Figure 3-16: **Reset Behavior (with TREADY)**

Protocol Description

Other than the event signals, all interfaces to and from the DDS are AXI4-Stream interfaces. The event signals are registered and asserted on detection of the event they describe.

AXI4-Stream Considerations

The conversion to AXI4-Stream interfaces brings standardization and enhances interoperability of Xilinx® LogiCORE IP solutions. Other than general control signals such as `aclk`, `aclken` and `aresetn` and event indication outputs, all inputs and outputs of the DDS Compiler are conveyed on AXI4-Stream channels. A channel always consists of TVALID and TDATA, plus several optional ports and fields. In the DDS Compiler, the optional ports supported are TREADY, TLAST and TUSER. Together, TVALID and TREADY perform a handshake to transfer a message, where the payload is TDATA, TUSER and TLAST. The DDS Compiler operates on the operands contained in the TDATA field of the input channels and outputs the result in the TDATA field of the output channels. The DDS Compiler provides configuration options for the use of TUSER and TLAST.

Basic Handshake

Figure 3-17 shows the transfer of data in an AXI4-Stream channel. TVALID is driven by the source (master) side of the channel and TREADY is driven by the receiver (slave). TVALID indicates that the value in the payload fields (TDATA, TUSER and TLAST) is valid. TREADY indicates that the slave is ready to receive data. When both TVALID and TREADY are asserted in a cycle, a transfer occurs. The master and slave set TVALID and TREADY respectively for the next transfer appropriately.

The DDS Compiler "datapath" channels (all except the CONFIG channel) can be configured to have no TREADY.

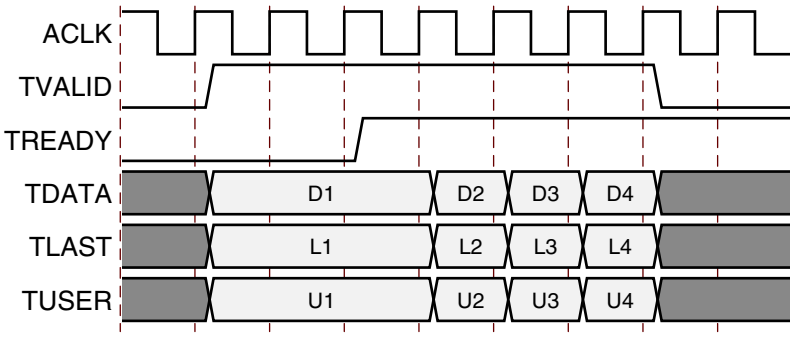

Figure 3-17: **Data Transfer in an AXI4-Stream Channel**

This is equivalent to setting TREADY for each of these channels permanently asserted. This inability to indicate backpressure simplifies the interface behavior and allows resource

savings to be made. This mode is recommended when the system can be designed to ensure that full throughput (one sample per cycle) can be assured.

The AXI4 handshake shown in Figure 3-17 forbids the output of data until the downstream circuitry is ready. This allows for easy synchronization of circuits following power-up or reset without complicated latency calculations.

For more information on AXI4-Stream interfaces see the *Xilinx AXI Design Reference Guide* (UG761) [Ref 1] and the *AMBA® AXI4-Stream Protocol Specification* (ARM IHI 0051A) [Ref 2].

CONFIG Channel

The CONFIG channel (`s_axis_config_t*`) replaces the programmatic interface of DDS Compiler v4.0. For the CONFIG channel, there is the concept of a vector. The vector in question is a complete set of values (PINC and/or POFF) for all channels. The CONFIG channel is non-blocking, which means that the other channels of the DDS Compiler do not wait upon data from the CONFIG channel. To program the CONFIG channel N transfers must occur, where N is the number of channels. Each transfer contain the PINC and/or POFF values for each channel in sequence starting with channel 0. Only the last transfer, for channel (index N-1) must have TLAST asserted. Failure to do so causes either `event_s_config_tlast_missing` or `event_s_config_tlast_unexpected` outputs to be asserted for a cycle. The packet is only deemed to be received when complete. Only when it is completely received is it eligible to be used pending a synchronization event. Synchronization events are either when the TDM channel counter rolls over (vector framing) or when the input PHASE channel is configured to receive packet TLASTs and one such TLAST is received (packet framing).

Figure 3-18 illustrates programming CONFIG data for a six-channel DDS. In the first programming cycle, TLAST is incorrectly applied, and the event outputs trigger accordingly. The second programming cycle shows correct application of TLAST.

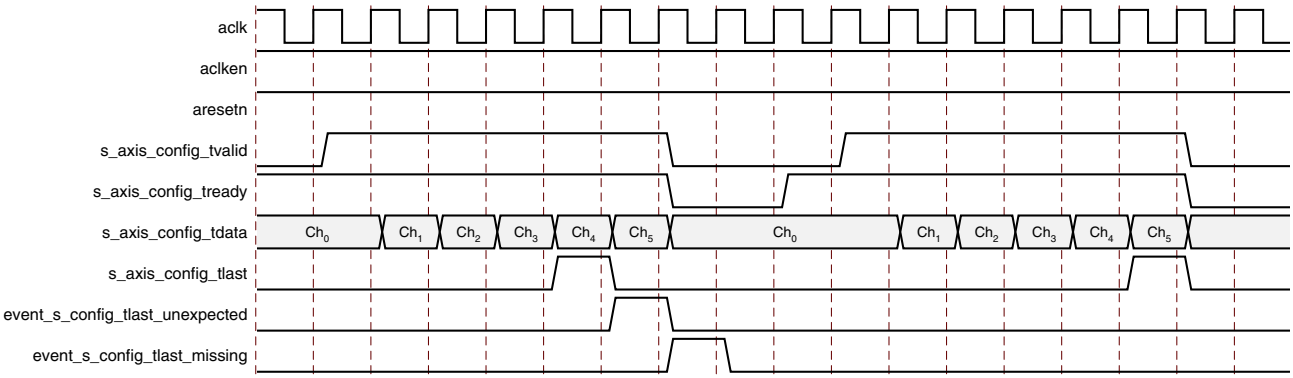

Figure 3-18: **CONFIG Channel Programming**

When the core is configured for single-channel operation, TLAST is not required and the pin is not present on the CONFIG channel.

CONFIG Channel TDATA Structure

When the CONFIG channel is configured to supply both PINC and POFF values for each TDM channel, each field is sign-extended to fit a byte boundary, then these byte-oriented fields are concatenated with PINC in the least significant positions. For example, for a phase width of 11 bits, PINC would occupy bits 10:0 and POFF would occupy 26:16. Thus `s_axis_config_tdata` would be 31:0 overall.

Figure 3-19 shows the structure for the widths in the preceding example for the following configurations:

- Both PINC and POFF are set to Programmable.
- PINC only is set to Programmable.
- POFF only is set to be Programmable

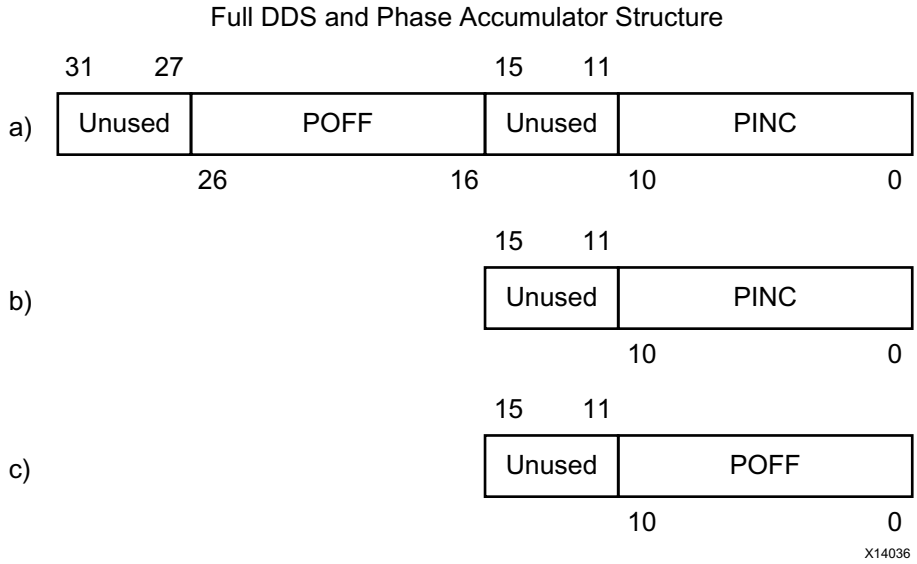

Figure 3-19: **CONFIG Channel TDATA Structure**

Input PHASE Channel

The input PHASE channel (`s_axis_phase_t*`) replaces the streaming interface (PINC_IN and POFF_IN) or PHASE_IN ports of the DDS Compiler v4.0. The input PHASE channel is intended for applications where the DDS Compiler is to perform a dynamic function such as phase or frequency modulation, where there is an output sample for each input sample. The fact that there is a one-to-one relationship between input and output means that back pressure applied to the output (TREADY deasserted) results in the deassertion of TREADY on the input PHASE channel (delayed according to internal buffer capacity). Likewise a starvation of input data on the PHASE channel (deassertion of TVALID) propagates to become a deassertion of TVALID on the output channels.

When in rasterized mode, values of PINC, POFF and PHASE_IN must be constrained between 0 and Modulus-1 (inclusive). This corresponds to a full circle. So if negative values

are required, add the Modulus to map these into the required range. For instance with modulus 360, if -90 is required, add 360 to get 270 which is in the required range. Event signals exist to detect invalid values of PINC, POFF and PHASE_IN input at run time.

When the DDS Compiler is configured to have a Phase Accumulator and either Phase Increment or Phase Offset is selected to be "Streaming" the input PHASE channel interface exists. When the DDS Compiler is configured to be a SIN/COS LUT only, the PHASE_IN field is input on the TDATA bus of the input PHASE channel. These two configurations are mutually exclusive.

Input PHASE Channel TDATA Structure

As noted earlier, the two configurations in which the DDS Compiler can have an input PHASE channel are mutually exclusive, so although there are four fields which can occur in the TDATA, all four cannot occur together.

When the DDS Compiler is configured to be a SIN/COS LUT only, the PHASE_IN field is mapped to `s_axis_phase_tdata`. The PHASE_IN field occupies a byte-oriented field in the least significant portion of the bus. So the width of `s_axis_phase_tdata` is the minimum multiple of 8 bits required to accommodate the PHASE_IN width. Because this is an input, any additional bits required to achieve this byte orientation are ignored by the core and are optimized away during synthesis or mapping.

Figure 3-20 shows the structure of `s_axis_phase_tdata` where Phase_Width = 11 for the following configurations:

- Both PINC and POFF are set to Streaming.
- PINC only is set to Streaming.
- POFF only is set to Streaming.
- The DDS is configured to be a SIN/COS LUT only.

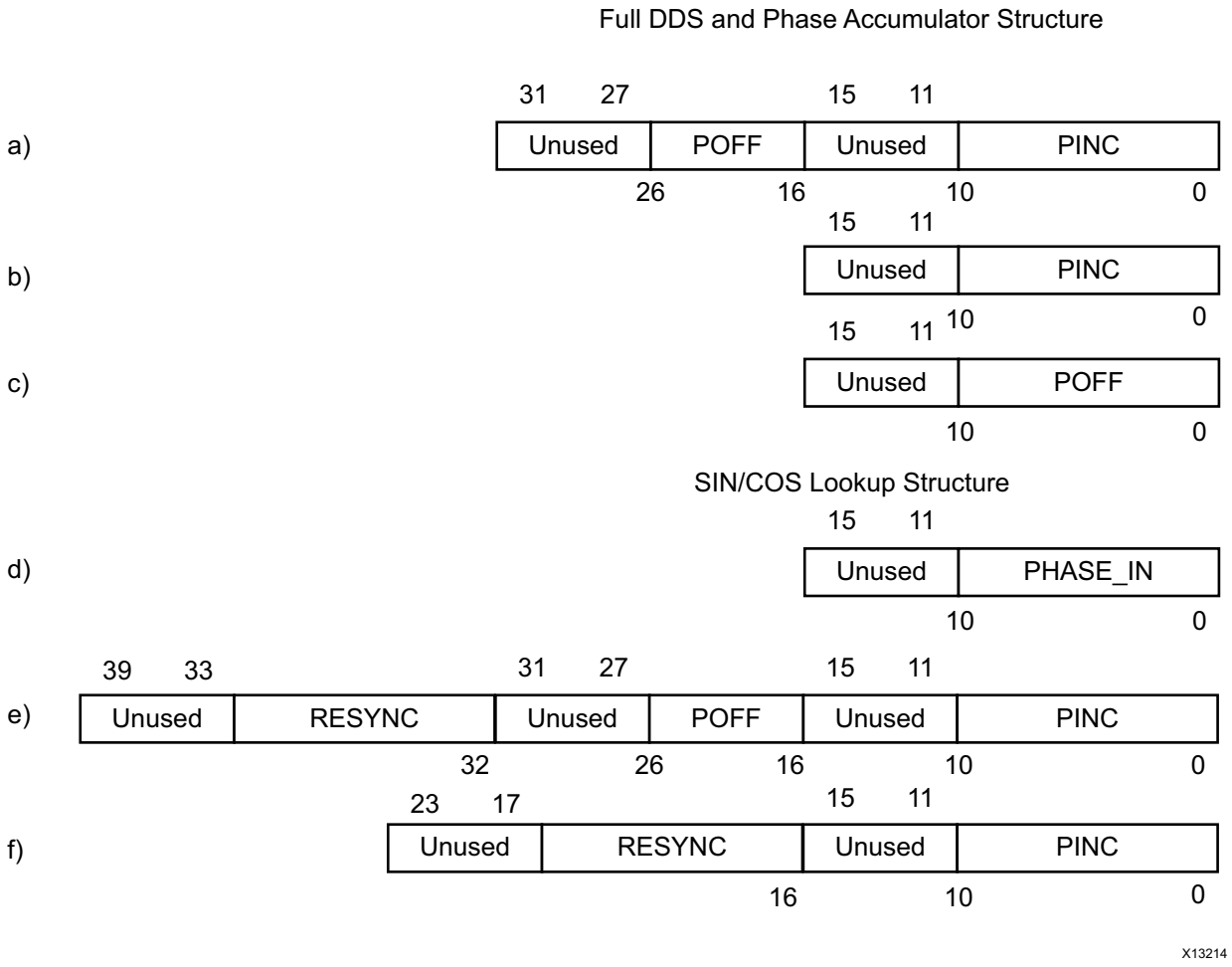

Figure 3-20: **Input PHASE Channel TDATA Structure**

Input PHASE Channel TUSER Structure

The input PHASE channel can be configured to have no TUSER port, to have a user field or to carry the TDM Channel index, or both a user field and TDM Channel index. There is no byte orientation to these fields. The TDM channel index, if configured, has the minimum width required to describe the number of TDM channels. The width of the user field is determined by user selection from 1 to 256 bits. The two fields are concatenated with the TDM channel ID field in the least significant place. If only one field exists, it occupies the least significant bits of `s_axis_phase_tuser`.

Figure 3-21 shows the 3 possible combinations; both user field and chan_id field, chan_id field only and user field only.

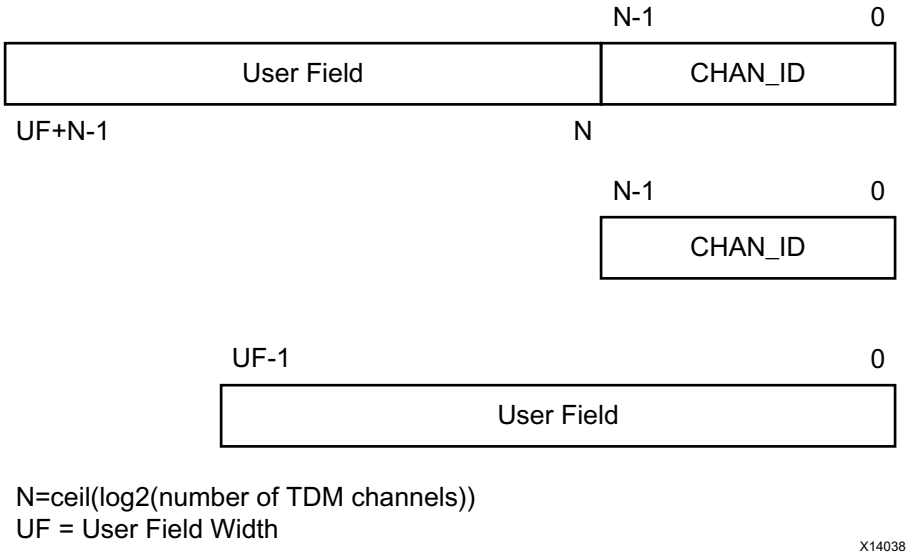

Figure 3-21: **Input PHASE Channel TUSER Structure**

Input PHASE Channel TLAST Options

The input PHASE channel can be configured to have no TLAST, to have a vector framing TLAST or to have a packet framing TLAST.

When Vector Framing is selected, TLAST is expected to indicate the last channel in a TDM cycle of channels. If TLAST does not match the internal expectation of when TLAST should arrive, one of two event signals is asserted for a clock cycle.

When Packet Framing is selected, the core does not have any expectation of the timing of TLAST so the event signals are not present, but TLAST is conveyed to the output channels with the same latency as the TDATA input.

Output DATA channel

The Output Data channel exists whenever the DDS Compiler is configured to have a SIN/COS LUT. This channel replaces the SINE and COSINE outputs of DDS Compiler v4.0. These former outputs now exist as fields of `m_axis_data_tdata`.

Output DATA Channel TDATA Structure

The sine and cosine output fields are sign extended to the next byte boundary then concatenated, with cosine in the least significant portion, to create `m_axis_data_tdata`. If only one of sine or cosine is selected, then it is sign extended and put into the least significant portion of `m_axis_data_tdata`.

Figure 3-22 shows the internal structure of TDATA for the three configurations; quadrature outputs, cosine only and sine only. An 11-bit output has been shown in the diagram for example, sign extended to 16 bits. The <<< denotes sign extension.

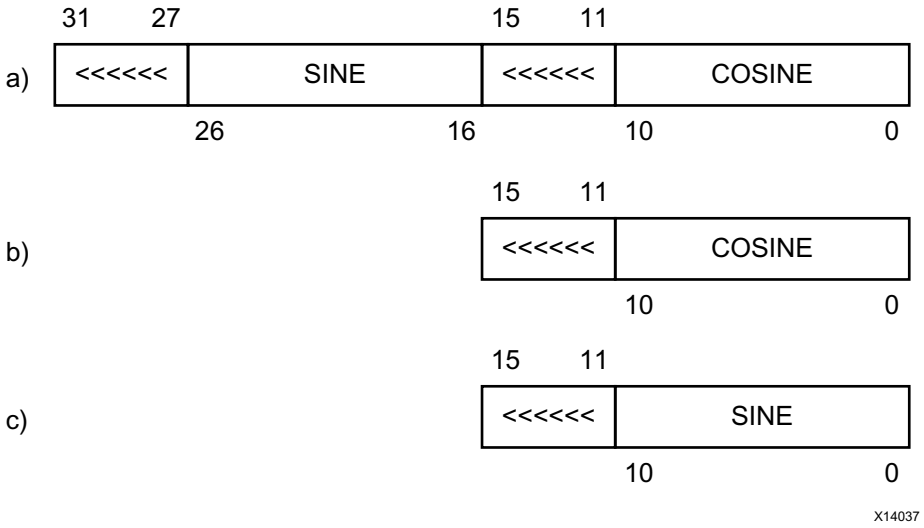

Figure 3-22: **Output DATA Channel TDATA Structure**

Output DATA Channel TUSER Structure

The output DATA channel can be configured to have no TUSER field, or for TUSER to hold a user field, or a TDM channel index or both a user field and a TDM channel ID. When both user field and TDM channel ID are selected the fields are concatenated with the TDM channel ID in the least significant position. The TDM channel ID qualifies the fields in the TDATA bus for that transfer as belonging to the TDM channel described. See Figure 3-21 for the structure, as this is identical to the structure for the Output DATA channel TUSER port.

The user field is not used nor interpreted by the core. It is provided as a service to allow the system designer to pass information through the core with latency identical to the main datapath (input PHASE channel to output channels). For instance, the user field could contain flags and other ancillary information irrelevant to the DDS, but relevant to some core downstream from the DDS.

Output DATA Channel TLAST Options

The output DATA channel can be configured to have no TLAST, to have vector framing, to have packet framing or to have a TLAST which is "Configuration triggered".

When set to vector framing TLAST is asserted for the transfer which contains the TDATA of the last TDM channel of a cycle of TDM channels (for example, channel 12 of 12).

When set to packet framing, the TLAST of the input PHASE channel is passed unaltered with latency equal to the latency of the main datapath. This is intended to be a service to the system designer, where TLAST might have a meaning which is irrelevant to the DDS, but relevant to some core downstream.

When set to "Configuration Triggered" the TLAST is generated internally by the DDS rather than conveyed from the input PHASE channel. It is asserted on the last channel of a TDM cycle immediately before a configuration change is effected. In other words, if a

configuration change is provoked using the CONFIG channel, TLAST is asserted on the last sample of the old configuration.

Output PHASE Channel

The output PHASE channel replaces the PHASE_OUT port of DDS Compiler v4.0. The PHASE_OUT port now exists as a field of `m_axis_phase_tdata`.

Output PHASE Channel TDATA Structure - Conventional DDS

The PHASE_OUT field is sign extended to the next multiple of 8 bits and becomes `m_axis_phase_tdata`. For example, if PHASE_OUT is 20 bits, `m_axis_phase_tdata` is 24 bits wide [23:0], occupied by a sign extended PHASE_OUT.

Figure 3-23 shows this for an example width of 11 bits sign extended to 16 bits.

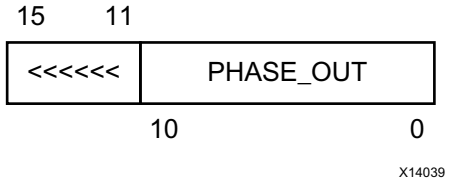

Figure 3-23: **Output PHASE Channel TDATA Structure**

Output PHASE Channel TDATA Structure - Rasterized DDS

The phase values in rasterized mode are positive integers in the range 0 to Modulus-1. Therefore the PHASE_OUT field is padded with zeros up to the next multiple of 8 bits and becomes `m_axis_phase_tdata`.

Output PHASE Channel TUSER Structure

The TUSER field has the same configuration options as the output DATA channel, but the options are independent for the two output channels, so, for instance, one might be configured to have a user field while the other has a TDM channel ID field. Figure 3-21 is identical to the structure options for the output PHASE Channel TUSER port.

Output PHASE Channel TLAST Options

The output PHASE channel uses the same TLAST setting as the output DATA channel.

Event Interface

To allow users to synchronize correctly to the internal channel counter, and flag errors when writing multichannel data to the CONFIG and/or PHASE channels, the DDS Compiler core provides several event outputs to indicate when unexpected conditions have occurred. Event outputs obey `aclken` and `aresetn` conditions, occur immediately and are asserted for each cycle that a discrepancy is present.

If the event outputs are not required, they can be left unconnected and the associated logic is optimized away by the Xilinx tools.

CONFIG Channel Event Outputs

For multichannel configuration, the CONFIG channel expects a single pulse on `s_axis_config_tlast` to indicate the last sample in a sequence of channels (a "vector"). If `s_axis_config_tlast` is asserted when the core does not expect it to be, `event_s_config_tlast_unexpected` is asserted. If the `s_axis_config_tlast` pulse is not asserted with the configuration data of the last channel, `event_s_config_tlast_missing` is asserted.

PHASE Channel Event Outputs

When the PHASE channel is present with multiple channels, the PHASE channel expects a single pulse on `s_axis_phase_tlast` to indicate the last sample in a sequence of channels (a "vector2). If `s_axis_phase_tlast` is asserted when the core does not expect it to be, `event_s_phase_tlast_unexpected` is asserted. If the `s_axis_phase_tlast` pulse is not asserted with the data of the final channel, `event_s_phase_tlast_missing` is asserted.

When the DDS is configured for rasterized mode, both PINC and POFF values (fixed, programmable or streaming) are expected to be in the range 0 to Modulus-1. Values outside this range are not supported and can lead to erroneous output. The signals `event_pinc_invalid` and `event_poff_invalid` are asserted when a value outside the supported range is detected.

The `s_axis_phase_tuser` configuration options allow you to input a Channel ID value to facilitate synchronization with the internal channel counter. If this option is selected, the `event_s_phase_chanid_incorrect` output is present on the core and is asserted for every cycle where the input Channel ID on `s_axis_phase_tuser` does not match the current channel of the core.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 3]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 5]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 3] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4] and the *Vivado Design Suite User Guide: Getting Started* ((UG910) [Ref 5].

Component Name: The name of the core component to be instantiated. The name must begin with a letter and be composed of the following characters: a to z, A to Z, 0 to 9 and "_".

Configuration Tab

Configuration Options: The full DDS or optionally the Phase Generator part or SIN/COS LUT part can be generated.

- **Phase Generator and SIN/COS LUT:** DDS is provided by combining Phase Generator and SIN/COS LUT with an optional dither circuit.
- **Phase Generator only:** Only the phase generator is provided.
- **SIN/COS LUT only:** Only the SIN/COS LUT with optional Taylor series correction circuit is provided.

System Requirements: The general context of the DDS is set by this group of parameters:

- **System Clock:** The frequency at which the DDS core is clocked. The value provided influences architectural choices, and is used to calculate the value of phase increment from output frequency (it is the relative value of output frequency to system clock that specifies phase increment, and so doubling system clock while maintaining output frequency results in a doubling of phase increment).

IMPORTANT: *The specified clock rate might not be achievable by the final implementation, because this depends on the FPGA family and how much is being packed into the device.*

- **Number of Channels:** The DDS and phase generator can support up to 16 channels. The channels are time-multiplexed, which reduces the effective clock frequency per channel.
- **Mode of Operation**: The DDS supports standard mode where the accumulated phase can be truncated before being used to access the SIN/COS LUT, or rasterized mode which can be used when the desired frequencies and system clock are related by a rational fraction. See Theory of Operation in Chapter 3 for more details.
- **Modulus** (rasterized mode only): Describes the relationship between the system clock frequency and the desired frequencies. See Theory of Operation in Chapter 3 for more details.
- **Frequency per Channel (F_s):** Because of time division multiplexing, the effective system clock to each channel is the real system clock divided by the number of channels.

Parameter Selection: DDS key parameters can be specified using **System Parameters**, which are aimed at system architects (frequency domain parameters) or **Hardware Parameters**, which are aimed at hardware engineers (time-domain parameters). The Phase Generator and SIN/COS LUT are only specified in terms of Hardware parameters.

System Parameters

- **Spurious Free Dynamic Range (SFDR):** The targeted purity of the tone produced by the DDS. This sets the Output Width as well as internal bus widths and various implementation decisions.
- **Frequency Resolution:** Specified in Hz, this specifies the minimum frequency resolution and is used to determine the Phase Width, as employed by the phase accumulator and its associated phase increment (`PINC`) and phase offset (`POFF`) values. Small values give high frequency resolution and require larger accumulators. Larger values reduce hardware resources. Depending upon the choice of Noise Shaping, the Phase Width can be increased, and the frequency resolution higher than that specified. For rasterized mode, the frequency resolution is fixed by the system clock, the number of channels, and the modulus selected. See Theory of Operation in Chapter 3 for more details.

Noise Shaping: This controls whether phase truncation, dithering, or Taylor series correction is used. The options are:

- **None:** Phase truncation DDS is produced.
- **Dithering:** Phase dither is used to improve SFDR at the expense of increased noise floor. See Phase Dithered DDS in Chapter 3.
- **Taylor Series Corrected:** Sine/cosine values are interpolated using the otherwise discarded bits from phase truncation. See Taylor Series Corrected DDS in Chapter 3.
- **Auto:** Noise-shaping is automatically determined, based on System Parameters such as SFDR. The selected noise shaping option is presented in the core summary pages. Auto is only available when Parameter Selection is System Parameters.

The availability of particular noise shaping options depends upon the configuration option selected and Parameter Selection method. System Parameter entry automatically constrains whether a particular Noise Shaping option is possible. When Hardware Parameter entry is selected, the options summarized in Table 4-1 are made available, and the choice of the Noise Shaping option then constrains the hardware parameter to ranges to those supported by the selected option.

Table 4-1: **Availability of Noise Shaping Options for Hardware Parameters**

Setting	DDS Part	Phase Generator Part	SIN/COS LUT Part
None	Available	Available	Available
Dithering	Available		
Taylor	Available		Available
Auto	Available		

Based upon the System Parameters entered and Noise Shaping selected, the minimum Phase Width and Output Width are derived by the Vivado IDE in the following way. The

Phase Width can be increased to enable a particular Noise Shaping option. For example, Taylor Series Correction requires a minimum Phase Width of 12 bits.

$$\text{Phase Width} = \left\lceil \log_2\left(\frac{\text{DDS Clock Rate}}{\text{Channels} \times \text{Frequency Resolution}}\right) \right\rceil$$

Table 4-2: **Calculation of Output Width from SFDR and Noise Shaping**

Noise Shaping	Output Width
None and Dithering	$\text{Output Width} = \left\lceil \frac{\text{SFDR}}{6} \right\rceil$
Taylor	$\text{Output Width} = \left\lceil \frac{\text{SFDR}}{6} \right\rceil + 1$

Figure 4-1 shows the regions of SFDR and Phase Width over which each Noise Shaping option operates. There are three overlapping regions for **None**, **Phase Dithering** and **Taylor Series Correction**, and deeper levels of shading have been used to show where regions overlap. The darkest region is where all three regions overlap and all three noise shaping options are possible. The lower dashed line signifies that Taylor series correction is only valid for SDFR > 66.0 dB (and not 66.0 dB). Phase Width can be increased to maximize the number of noise shaping options for a particular SFDR target.

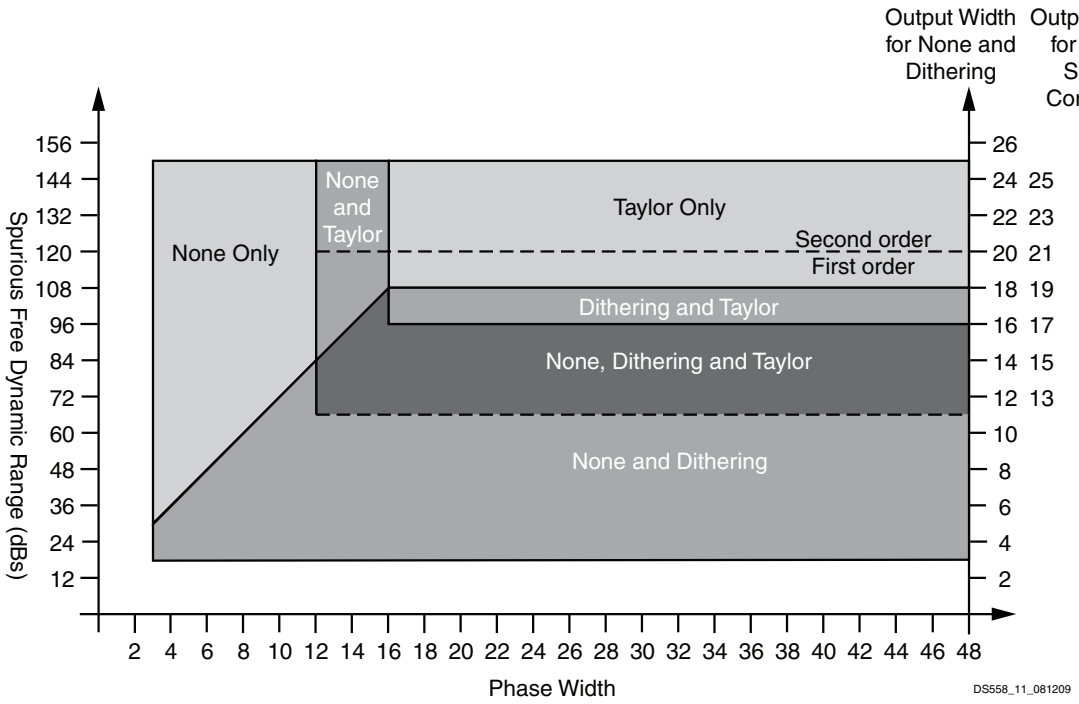

Figure 4-1: **Noise Shaping Regions**

Hardware Parameters

- **Phase Width:** Sets the width of the PHASE_OUT field within m_axis_phase_tdata, the phase field within s_axis_phase_tdata when the DDS is configured to be a SIN/COS LUT only, the phase accumulator, associated phase increment and offset registers and the phase field in s_axis_config_tdata. For rasterized mode, the phase width is fixed as the number of bits required to describe the valid input range [0, Modulus-1], that is $\log_2$ (Modulus-1) rounded up.
- **Output Width:** Only enabled when DDS or SIN/COS LUT part selected, as it is not required by the Phase Generator part. Sets the width of SINE and COSINE fields within m_axis_data_tdata. The SFDR that this provides is dependent on the Noise Shaping option previously selected. The equations in Table 4-3 can be used to estimate the SFDR that can be achieved:

Table 4-3: **Calculation of SFDR for given Noise Shaping**

Noise Shaping	SFDR
None, Dither	$SFDR = \text{Output Width} \times 6$
Taylor	$SFDR = (\text{Output Width} - 1) \times 6$

Implementation Tab

Phase Increment Programmability: Selects the means by which the PINC value is set.

- **Fixed:** PINC is fixed at generation time and cannot be changed at run-time. Fixed requires minimal resource.
- **Programmable:** PINC value can be changed at run-time using the CONFIG channel. This is recommended when the DDS frequency is to change between modes of operation.
- **Streaming:** PINC value is taken directly from the input PHASE channel. This is recommended when the PINC value has to change often, or for example when frequency modulation is required.

Resync: When selected, the s_axis_phase channel has a RESYNC field. This bit, when asserted, mutes the accumulated phase for the channel in question. The value of the accumulated phase for that cycle is the value of PINC accompanying the RESYNC assertion, plus the POFF value.

Phase Offset Programmability: Selects the means by which the POFF value is set.

- **None:** No phase offset facility and the required hardware is not generated. This saves FPGA resources.
- **Fixed:** POFF is fixed at generation time and cannot be changed at run-time.

- **Programmable:** POFF value can be changed using the CONFIG channel. This is recommended when the DDS phase is to change between modes of operation.
- **Streaming:** POFF value can be changed using the input PHASE channel. This is recommended when the POFF value has to change often, or for example when phase modulation is required.

Output

- **Output_Selection:** The DDS may have a quadrature SINE and COSINE field in the m_axis_data_tdata bus, or only one of these two fields. See Output DATA Channel TDATA Structure in Chapter 3 for m_axis_data_tdata internal structure.
- **Polarity:** The SINE and COSINE fields of m_axis_data_tdata can be inverted. This allows conversion of a DDS used as a transmitter mixer to a receiver mixer, using conjugated outputs; hence both instantiations are identical except for the values of the two selections here.
  - **Negative Sine:** Checking this selection results in the SINE field being negated at run-time.
  - **Negative Cosine:** Checking this selection results in the COSINE field being negated at run-time.
- **Amplitude Mode:** This selection allows for one of two amplitudes from the DDS.
  - **Full Range:** Aimed at communications applications where the maximum amplitude within the two's complement representation is desired, but the value of amplitude is less important due to the expectation of automatic gain control. The target amplitude for full range mode is $1–2^{-(\text{Output Width}–2)}$ for non-Taylor modes, whereas for Taylor mode the amplitude varies between this value and 1. Note that amplitude here is normalized to the output width with the binary point in the first place. An 8-bit output would have a binary amplitude of 100000000 – 10 giving values between 01111110 and 11111110 corresponding to just less than 1 and just more than –1 respectively.
  - **Unit Circle:** For applications where the value of the DDS output amplitude is important, say for FFT twiddle factor generation. When **Unit Circle**, the DDS output amplitude is half full range (that is, values range from 01000..(+0.5). to 110000..(-0.5)). As the amplitude is reduced over **Full Range** by a factor of 2, the SDFR is reduced by 6 dB. Increase SFDR or Output Width to accommodate this requirement.

Implementation Options

- **Memory Type:** This controls the implementation of the SIN/COS LUT. The **Auto** setting selects **Distributed ROM** for small cases where the table can be contained in a single layer of memory and selects **Block ROM** for larger cases. (That is, **Distributed ROM** is selected when **Phase Width** ≤ 5-bits). This selection can be overridden by selecting **Distributed ROM** or **Block ROM** explicitly.

- **Optimization Goal:** In some cases, circuit clock speed can be increased at the expense of extra pipelining registers. This selection controls whether the implementation decisions target highest speed or lowest resource.
- **DSP48 Use:** This controls the implementation of the phase accumulator and following addition stages (for phase offset and/or dither noise addition). When set to **Minimal**, the phase accumulator and following stages are implemented in FPGA logic. When **Maximal**, all is implemented using DSP slices. In the case of single channel, the DSP slice can also provide the register to store programmable phase increment and/or phase offset and thereby save further fabric resources. This is not done if either **Phase Increment** or **Phase Offset** is **Streaming** and only when the Optimization Goal is Area. When this optimization is performed, the initial value of the `PINC` and/or `POFF` register must be zero. This is enforced by the Vivado IDE by setting the initial value of `PINC` and/or `POFF` to zero and disabling entry.

Detailed Implementation Tab

AXI Channel Options: The action of certain AXI interface signals can be configured.

- **DATA Has TLAST**: Enabled when there is more than one DDS channel (as opposed to AXI channel). Limited options are also available when only the PHASE channel is present. Options are:
  - **Not Required:** In this mode, no TLAST signals are present on the input PHASE channel or the output channels. In multichannel configurations, TLAST on the CONFIG channel is used to denote the last channel to be reconfigured, and is always present, regardless of this setting.
  - **Vector Framing:** A TLAST pulse on the input PHASE channel and output channels denotes the last channel in a cycle of channels (for example, 12^{th} of 12 channels). If the TLAST pulse is not applied at the correct time to match the channel state of the core, an event is flagged on the `event_s_phase_tlast_missing` or `event_s_phase_tlast_unexpected` event outputs.
  - **Packet Framing:** A TLAST pulse is conveyed from the input PHASE channel to the output channels with the same latency as TDATA. TLAST in this configuration may be used to trigger a reconfiguration. See Synchronization Mode on page 46. This mode is intended as a service to ease system design for cases where signals must accompany the datastream, but which have no application in the DDS.
  - **Config Triggered:** This option causes the core to generate an output TLAST on the last TDM channel before a new configuration is applied to the core. Subsequent output samples are generated using the new core configuration. This mode is only available when the CONFIG channel is present.

- **Output TREADY**: When selected, the output channels have a TREADY and hence support the full AXI handshake protocol with inherent back pressure. If there is an input PHASE channel, the presence of its TREADY is also determined by this control, so that the datapath from input PHASE channel to output channels as a whole supports back pressure or not.
- **TUSER options**: The core supports two distinct uses of the TUSER field; to denote the time-division-multiplex channel index or as conduit to pass a user field (auxiliary data associated with TDATA) from input PHASE channel to output channels. These choices are independent for the input PHASE channel. However, since the selection of a user field implies the desire to convey the TUSER field from input to output, the selection of a user field on the input PHASE channel forces a user field to be present in each of the output channel TUSER ports. Options for the input PHASE channel are shown below. Options for the each output channel are constrained by the input PHASE channel choice, but are otherwise independent.
  - **Not required**: Neither of the above uses is required; the channel in question does not have a TUSER field.
  - **Chan ID field**: In this mode, the TUSER field identifies the time-division-multiplexed channel for the transfer. For the input PHASE channel, this gives the user a mechanism to synchronize to the internal DDS channel state. If the applied Channel ID does not match the internal state of the core, an event is flagged on the `event_s_phase_chanid_incorrect` output.
  - **User Field**: In this mode, the core ignores the content of the TUSER field, but passes it unaltered from input PHASE channel to the output channels.
  - **User and Chan ID field**: In this mode the TUSER field has both a user field and a channel ID field, with the channel ID field in the least significant bits. The minimal number of bits required to describe the channel determines the width of the channel ID field; for example, seven channels requires three bits.
  - **User field width**: This field determines the width of the bit field which is conveyed from input to output unaltered by the DDS. It does not include the width of the Channel ID field, if it is present.
- **Output Form**: In general, the output of SINE and COSINE is in twos complement form. However, when quadrant symmetry is used, the output form can be changed to sign and magnitude. Selecting sign and magnitude removes the inverters following the SIN/COS LUT. This cuts resource use relative to twos complement. It is intended for use with a mixer stage where the sign information can be handled separate to the magnitude by changing the sign of the components in a complex multiplication. A mixer (complex multiplier with inputs $a+jb$ and $c+jd$) can be expressed as:

  real = $ac-bd$

  imaginary = $ad + bc$

In this case, *a* and *b* from the DDS are signed, twos complement. However, if *a* and *b* are expressed as sign and magnitude, the sign from each of *a* and *b* can be used to modify the sign of each of the terms in the mixer equation. For instance, if *a* alone is negative, the mixer equation becomes:

real = $-ac\ -bd$

imaginary = $-ad + bc$

If the mixer stage is implemented using DSP slices, this sign manipulation can be implemented by changing the DSP48 Slice opmode signal.

- **Synchronization Mode**: This selection deals with the timing of reconfiguration when both CONFIG and PHASE channels are present. The configuration channel takes configuration data asynchronously to the phase of the channel counter and stores the reconfiguration data in a buffer. This selection determines when that new configuration data takes effect on the datapath:
  - **On Vector**: In this mode, the reconfiguration data is applied when the channel counter rolls over to start a new cycle of time-division-multiplexed channels.
  - **On Packet:** In this mode, available when TLAST is set to packet framing, a TLAST on the input PHASE channel triggers the reconfiguration. This mode is targeted at cases where each set of configuration data is to be associated with the packets implied by the input TLAST indicator.

Latency Options: Select whether Latency should be configured automatically by the Vivado IDE or manually:

- **Auto:** Causes the DDS to be pipelined for optimal performance (taking into account the Optimization Goal).
- **Configurable:** Where optimal performance is beyond requirements, Latency can be set to configurable and a smaller value of latency selected. This reduces the number of pipeline stages and generally results in resource savings. A minimum value of latency is imposed, where a cycle of latency arises from each of the following sources:
  - Streaming phase increment
  - Block ROM within SIN/COS LUT (can be avoided by selecting Distributed ROM).
  - Block ROM within second order Taylor series correction (used for SFDR above 120 dB).

  Note that when TREADY is selected the AXI interfaces buffer data as a FIFO. This buffering results in non-deterministic latency. However, this action can only increase in latency. In this case, the minimum latency possible is 6 cycles plus the minimum latency described above.

Optional Pins: Certain inputs and outputs can be disabled to save resources.

- **Has Phase Out:** When checked the core has the output `PHASE` channel.
- **ACLKEN:** When checked the core has an `aclken` (active-High clock enable) port.
- **ARESETn:** When checked the core has an `aresetn` (active-Low synchronous reset) port.

Parameter Entry Pages: The following pages appear for entry of parameters when either Phase Increment or Phase Offset are either Fixed or Programmable. If Programmable, the initial value of the register is specified through the Parameter Entry Pages. If an DSP slice register is used, as described under Implementation Options, the initial value of phase increment and/or offset is assumed to be zero.

System Parameters:

- **Output Frequencies:** This page appears when **Parameter Selection** is set to **System Parameters** and **Phase Increment Programmability** is **Fixed** or **Programmable**. For each channel, an independent frequency (MHz) can be entered into the table. The allowable range is displayed as 0 to F_S (where F_S is the frequency per channel). Values from $F_S/2$ to F_S alias to $-F_S/2$ to 0 respectively, so can be used to input negative frequencies.
- **Phase Offset Angles:** This page appears when **Parameter Selection** is set to **System Parameters** and **Phase Offset** is set to **Fixed** or **Programmable**. This table allows the phase offset to be specified for each channel as a fraction of a cycle. The valid range is -1.0 to 1.0 for standard mode. For rasterized mode, the valid range is 0 to 1.0. For example enter 0.5 for 180° (that is, π radians). The range for standard mode is greater than a single cycle, but is allowed, because negative values map to equivalent positive values.

Hardware Parameters:

- **Phase Angle Increment Values:** This page appears when **Parameter Selection** is set to **Hardware Parameters** and **Phase Increment Programmability** is **Fixed** or **Programmable**. Values must be entered in binary. The range is 0 to the modulus value minus 1. For standard mode, the modulus value is $2^{\text{PhaseWidth}}$. For rasterized mode, the modulus is a value selected earlier in the Vivado IDE. The angle in radians is the input number divided by the modulus and multiplied by 2π. Entries are extended to Phase Width bits by zero padding to the left.
- **Phase Offset Values:** This page appears when **Parameter Selection** is set to **Hardware Parameters** and **Phase Offset** is set to **Fixed** or **Programmable**. Values must be entered in binary. The range for standard mode is 0 to the weight of the accumulator, that is, $2^{\text{Phase Width}}-1$, which corresponds to a single cycle. For rasterized mode, the valid range is 0 to 1.0 – 1/modulus. The angle in radians can be obtained by converting the unsigned fractional number to decimal and multiplying by 2π. Entries are extended to Phase Width bits by zero padding to the left.

Summary (2 pages): The final two tabs provide feedback fields.

- **Summary (Page 1):** This page presents the resolved values of the selected part. For instance, these fields indicate the result of automatic memory type and latency allocation. They also indicate the expected SFDR and frequency resolution for the DDS when hardware parameters are used for input, or vice versa. There are also resource estimates (DSP slices and 18 kbit block RAM primitives).
- **Summary (Page 2):** This is only presented when **Phase Increment** and/or **Phase Offset** are fixed or programmable, and provides a summary of the hexadecimal values used to obtain a particular frequency or phase offset. The actual value of frequency and phase (the latter as a fraction of a cycle) is also given as a floating-point number.

User Parameters

Table 4-4 shows the relationship between the Vivado IDE fields in the Vivado IDE (described in Customizing and Generating the Core) and the User Parameters (which can be viewed in the Tcl console).

Table 4-4: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter	User Parameter	Default Value
Configuration Options	partspresent	Phase_Generator_and_SIN_COS_LUT
System Clock (MHz)	dds_clock_rate	100
Number of Channels	channels	1
Mode of Operation	mode_of_operation	Standard
Modulus	modulus	9
Parameter Selection	parameter_entry	System_Parameters
Spurious Free Dynamic Range	spurious_free_dynamic_range	45
Frequency Resolution	frequency_resolution	0.4
Noise Shaping	noise_shaping	Auto
Phase Width	phase_width	16
Output Width	output_width	12
Phase Increment Programmability	phase_increment	Fixed
Resync	resync	False
Phase Offset Programmability	phase_offset	None
Output Selection	output_selection	Sine_and_Cosine
Negative Sine	negative_sine	False
Negative Cosine	negative_cosine	False
Amplitude Mode	amplitude_mode	Full_range
Memory Type	memory_type	Auto

Table 4-4: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
Optimization Goal	optimization_goal	Auto
DSP48 Use	dsp48_use	Minimal
Has Phase Out	has_phase_out	True
DATA has TLAST	data_has_tlast	Not_required
Output TREADY	has_tready	False
TUSER Options: Input	s_phase_has_tuser	Not_required
TUSER Options: User Field Width	s_phase_tuser_width	1
TUSER Options: DATA Output	m_data_has_tuser	Not_required
TUSER Options: PHASE Output	m_phase_has_tuser	Not_required
Synchronization Mode	s_config_sync_mode	On_Vector
Output Form	output_form	Twos_complement
Latency Configuration	latency_configuration	Auto
Latency	latency	1
ARESETn	has_aresetn	False
ACLKEN	has_aclken	false

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

System Generator for DSP

The DDS Compiler core is in the Xilinx Blockset in the DSP section of the System Generator. The block is called the DDS Compiler. See the System Generator for DSP Help page for the DDS Compiler block for more information on parameters not mentioned here.

The System Generator for DSP GUI offers the same parameters as the Vivado Design System GUI. However, there is a minor difference for the hardware parameters. In the Vivado design tools GUI, the hardware parameters are hidden when **System Parameter** is selected. In the System Generator for DSP GUI, the hardware parameters are disabled. Likewise, the system parameters are disabled when **Hardware Parameter** is selected.

For more information on System Generator for DSP see the *System Generator for DSP User Guide* (UG640) [Ref 6]

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

IMPORTANT: *For cores targeting 7 series or Zynq®-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado® Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

C Model

The DDS Compiler core bit accurate C model is a self-contained, linkable, shared library that models the functionality of this core with finite precision arithmetic. This model provides a bit accurate representation of the various modes of the DDS Compiler core, and it is suitable for inclusion in a larger framework for system-level simulation or core-specific verification.

The C model is an optional output of the Vivado® Design Suite (see the Vivado documentation to set up the C model output).

Features

- Bit accurate with DDS Compiler core
- Available for 64-bit Linux platforms
- Available for 64-bit Windows platforms
- Supports all features of the DDS Compiler core with the exception of those affecting timing or AXI4-Stream configuration
- Designed for integration into a larger system model
- Example C code showing how to use the C model functions

Overview

The model consists of a set of C functions that reside in a shared library. Example C code is provided to demonstrate how these functions form the interface to the C model. Full details of this interface are given in C Model Interface.

The model is bit accurate but not cycle-accurate; it performs exactly the same operations as the core. However, it does not model the core latency, its interface signals or TUSER feature.

Unpacking and model contents

There are separate ZIP files containing all the files necessary for use. Each ZIP file contains:

- C model shared library
- C model header file
- Example code showing how to call the C model

Table 5-1 and Table 5-2 list the contents of each ZIP file.

Table 5-1: **C Model ZIP File Contents: Linux**

File	Description
dds_compiler_v6_0_bitacc_cmodel.h	Header file which defined the C model API
libIp_dds_compiler_v6_0_bitacc_cmodel.so	Model shared object library
Run_bitacc_cmodel.c	Example program for calling the C model.
dds_compiler_v6_0_bitacc_mex.cpp	MATLAB MEX function source;
make_dds_compiler_v6_0_mex.m	MATLAB MEX function compilation script;
run_dds_compiler_v6_0_mex.m	MATLAB MEX function example script;
@dds_compiler_v6_0_bitacc	MATLAB MEX function class directory

Table 5-2: **C Model ZIP File Contents: Windows**

File	Description
dds_compiler_v6_0_bitacc_cmodel.h	Header file which defined the C model API
libIp_dds_compiler_v6_0_bitacc_cmodel.dll	Model dynamically linked library
libIp_dds_compiler_v6_0_bitacc_cmodel.lib	Model LIB file for compiling
run_bitacc_cmodel.c	Example program for calling the C model
dds_compiler_v6_0_bitacc_mex.cpp	MATLAB MEX function source;
make_dds_compiler_v6_0_mex.m	MATLAB MEX function compilation script;
run_dds_compiler_v6_0_mex.m	MATLAB MEX function example script;
@dds_compiler_v6_0_bitacc	MATLAB MEX function class directory

Installation

Linux

- Unpack the contents of the ZIP file.
- Ensure that the directory where the `libIP_dds_compiler_v6_0_bitacc_cmodel.so` resides is included in the path of the environment variable LD_LIBRARY_PATH.

Windows

- Unpack the contents of the ZIP file.
- Ensure that the directory where the `libIP_dds_compiler_v6_0_bitacc_cmodel.dll` resides is:
  - Included in the path of the environment variable PATH, or
  - In the directory in which the executable that calls the C model is run.

C Model Interface

An example file, `run_bitacc_cmodel.c`, is included. This demonstrates how to call the C model. See this file for examples of using the interface described in this section.

The Application Programming Interface (API) of the C model is defined in the header file `dds_compiler_v6_0_bitacc_cmodel.h`. The interface consists of data structures and functions as described in the following sections.

Data Types

Table 5-3 shows the types defined for the DDS Compiler C model.

Table 5-3: **C Model Data Types**

Name	Type	Description
xip_dds_v6_0_data	Double	Used for Phase input/output and for SIN/COS output.
xip_uint	Unsigned Int	Used for configuration parameter of integer or Boolean type. For Boolean: 0=false 1=true
xip_dds_v6_0_config	Struct	Structure of configuration parameters.

Table 5-3: **C Model Data Types *(Cont'd)***

Name	Type	Description
xip_dds_v6_0_config_pkt	Struct	Data structure of Phase Increment (PINC) and/or Phase Offset (POFF) values input to core at run time. Equivalent to data on the S_AXIS_CONFIG channel of the core.
xip_dds_v6_0_status	Int	Error code return from many C model functions. 0 indicated success. Any other value indicates failure.
xip_status	Int	Same as xip_dds_v6_0_status, but used for functions which are not core-specific.
xip_real	Double	
xip_array_real	Struct	Structure used to hold data for input to DDS (on S_AXIS_PHASE channel) or output from DDS.
xip_dds_v6_0_config	Struct	The configuration of the core itself. The members of this structure are listed in the dds_compiler_v6_0_bitacc_cmodel file. The names closely match the same names in XCO/XCI files. The dds_compiler_v6_0_bitacc_cmodel file also contains #defined values for all. All member variables beginning with "res" are for internal use and need not be set.
xip_dds_v6_0	Struct	Type defined which C (not C++) can use as a handle (pointer) to a C++ object – the C model itself.

`xip_dds_v6_0_data` is of type double because this has sufficient precision to describe even the largest input and output vectors (48 bits) without the complications of long which can vary from platform to platform or compiler to compiler.

xip_array_real is a structure with the following members:

- **data**: A pointer to the array of data values.
- **data_size**: Of type size_t, which describes the total size of the data array.
- **data_capacity**: Also of type size_t, which describes how much of the array is currently populated.
- **dim**: A pointer to a size_t array of values which indicate the size of each dimension.
- **dim_size** (size_t): Indicates the number of dimensions of the data array.
- **dim_capacity**: Indicates how much of the dimension array is currently populated.
- **owner**, unsigned int: This is provided as a handle for when the data structure is intended to be passed from one core to another, but is not used by any of the DDS C model functions.

Functions

There are several accessible C model functions.

Information Functions

Table 5-4 lists the information functions.

Table 5-4: **Information Functions**

Name	Return	Arguments	Description
xip_dds_get_version	Const char*	Void	Return the DDS Compiler C model version as a null terminated string. For v6.0, this is "6.0".
xip_dds_v6_0_ get_default_config	xip_dds_v6_0_status	xip_dds_v6_0_config*	Populates the contents of structure pointed to by the input argument with the values of a default configuration.

Initialization Functions

The functions to create, configure and destroy the C model and associated data structures are listed in Table 5-5.

Table 5-5: **Initialization Functions**

Name	Return	Arguments	Description
xip_dds_v6_0_create	Pointer to structure holding configuration of C model object	Pointer to structure holding configuration	Creates new C model object and returns pointer to config structure (which is pointer to C model itself).
xip_dds_v6_0_destroy	xip_dds_v6_0_status	Pointer to xip_dds_v6_0 (C model itself)	Deallocates memory owned by C model and destroys C model itself.
xip_dds_v6_0_get_config	xip_dds_v6_0_status	Pointer to C model, pointer to configuration structure	Copies the contents of the configuration of the C model indicated to the designated configuration structure.
xip_array_create	Pointer to created data structure	None	Allocates memory for the structure itself, not the array members within it.
xip_array_reserve_data	xip_status	Pointer to data structure, maximum number of elements in data array	(Re)allocates enough memory for the maximum size. Error is returned if structure data_capacity is greater than space allocated.

Table 5-5: **Initialization Functions *(Cont'd)***

Name	Return	Arguments	Description
xip_array_reserve_dim	xip_status	Pointer to data structure, maximum number of dimensions	Allocates a small array which is to contain the size of each dimension of the data array. For example, 100 samples x 4 channels x 3 fields.
xip_array_destroy	xip_status	Pointer to data structure	Frees up the memory allocated for the data array, the dimension array, and the data structure itself.
xip_dds_v6_0_alloc_ config_pkt	xip_dds_v6_0_status	Pointer to xip_dds_v6_0_ config_pkt, xip_uints for number of PINC values and number of POFF values	Allocates memory for arrays within config_pkg, but not config_pkt itself.
xip_dds_v6_0_free_ config_pkt	xip_dds_v6_0_status	Pointer to xip_dds_ v6_0_config_pkt	Deallocates memory for arrays within config pkt.

Execution Functions

The run time functions of the C model are described in Table 5-6.

Table 5-6: **Execution Functions**

Name	Return	Arguments	Description
xip_dds_v6_0_reset	xip_dds_v6_0_status	Pointer to C model (xip_dds_v6_0)	Resets the internal state of the core to power-on state.
xip_dds_v6_0_config_do	xip_dds_v6_0_status	Pointer to C model, pointer to config_pkt to input to C model	Writes config pkt to C model internal storage. This does not prompt execution of the C model, but the config data is picked up on the next xip_dds_v6_0_ data_do call
xip_dds_v6_0_data_do	xip_dds_v6_0_status	Pointer to C model, Pointer to input data structure, Pointer to output data structure, Number of samples, number of channels, number of fields input and number of fields output	The function which prompts execution of the C model. The number of samples, channels and fields must match the size of the array passed or an error is returned.

Table 5-6: **Execution Functions *(Cont'd)***

Name	Return	Arguments	Description
xip_array_real_set_data	xip_status	Pointer to array structure, the value to be written, the sample, channel and field to be written to	Used to populate the input data structure.
xip_array_real_get_data	xip_status	Pointer to the array structure, pointer of real type (returned value), sample, channel and field to be read	Used to read the output (or input) data structure.

Data Format

The DDS Compiler sine and cosine outputs are generated as unsigned integer values which are cast to doubles on the interface. To restore the sign for printing to stdout or use outside the C model, you must cast back to integer and scale accordingly. As both sine and cosine can be a maximum of 26 bits wide, casting to an int is sufficient. To scale the data, Xilinx recommends that the data is shifted up by the difference between the width of an int (usually 32 bits) and the config.Output_Width C model parameter to set the sign bit correctly, then scaled down by the same value, maintaining the sign.

For example, for a 12-bit output, the scaling value is (32 - 12) = 20, and the correctly-signed sine output can be printed using the following code:

```
std::cout << "sine " << ((int)sine_output << 20) >> 20) << std::endl;
```

See the C model smoke test, `run_bitacc_cmodel.c` for a generalized example.

The phase output from the C model can be up to 48 bits wide, and can be either signed or unsigned, depending on the configuration. See Output PHASE Channel for more details. In the general case, for a signed phase output, cast the double to a 64-bit integer type (for example, `int64_t`) and for an unsigned phase output, cast the double to an unsigned 64-bit integer type (for example, `uint64_t`).

Compiling

Compilation of user code requires access to the `dds_compiler_v6_0_bitacc_cmodel.h` header file. The header file should be copied to a location where it is available to the compiler. Depending on the location chosen, the include search path of the compiler might need to be modified.

When compiling on Windows, the symbol `NT` must be defined, either by a compiler option, or in user source code before the `dds_compiler_v6_0_bitacc_cmodel_v6_0_bitacc_cmodel.h` header file is included.

Linking

To use the C model, the user executable must be linked against the correct libraries for the target platform.

Linux

The executable must be linked against the `libIp_dds_compiler_v6_0_bitacc_cmodel.so` library.

Using GCC, linking is typically achieved by adding the following command line options:

```
-L. -Wl, -rpath,. -lIp_dds_compiler_v6_0_bitacc_cmodel
```

This assumes the object library is in the current directory. If this is not the case, the `-L.` option should be changed to specify the library search path to use.

Using GCC, the provided example program `run_bitacc_cmodel.c` can be compiled and linked using the following command:

```
gcc -x c++ -I. -L. -lIp_dds_compiler_v6_0_bitacc_cmodel -Wl, -rpath,. -o
run_bitacc_cmodel run_bitacc_cmodel.c
```

Windows

The executable must be linked against the `libIp_dds_compiler_v6_0_bitacc_cmodel.dll` dynamic link library.

Depending on the compiler, the `libIp_dds_compiler_v6_0_bitacc_cmodel.lib` import library might also be required.

To link to an import library using Microsoft Visual Studio, add the library to the Additional Dependencies entry under the Linker section of Project Properties.

Example

The `run_bitacc_cmodel.c` file contains example code to show basic operation of the C model. For example, the file shows declaration for the model, configuration of the model, allocation of required input and output data structures, execution of the model, resetting of the core, deallocation of the data structures, and destruction of the model.

The example file does not verify the output data for correctness, but allows a debug mode in which output data can be printed to a standard output channel (stdout).

MATLAB Interface

A MEX function and MATLAB® software class is provided to simplify the integration with MATLAB. The MEX function provides a low-level wrapper around the underlying C model, while the class file provides a convenient interface to the MEX function.

Compiling

Source code for a MATLAB MEX function is provided. This can be compiled within MATLAB by changing to the directory which contains the code and running the `make_dds_compiler_v6_0_bitacc_mex.m` script.

Installation

To use the MEX function the compiled MEX function must be present on the MATLAB search path. This can be achieved by either of the following:

1. Add the directory where the compiled MEX function is located to the MATLAB search path (see the MATLAB `addpath` function) OR
2. Copy the files to a location already on the MATLAB search path.

As with all uses of the C model, the correct C model libraries also need to be present on the platform library search path (that is, PATH or LD_LIBRARY_PATH).

MATLAB Class Interface

The `@dds_compiler_v6_0_bitacc` class handles the create/destroy semantics on the C model. The class provides objects for each of the data and control structures defined for the C model and described in Data Types. MATLAB arrays are used for the mapping of types as in Table 5-7.

Table 5-7: **C Model Type Mapping**

C Model Type	MATLAB Type
xip_real	double

The class provides the following methods:

Constructor

```
[model] = dds_compiler_v6_0_bitacc
[model]=dds_compiler_v6_0_bitacc(config)
[model]=dds_compiler_v6_0_bitacc(field, value [, field,value]*)
```

* indicates an optional parameter

The first version of the function call constructs a model object using the default configuration. The second version constructs a model object from a structure that specified the configuration parameter values to use. The third version is the same as the second, but allows the configuration to be specified as a series of (parameter name, value) pairs rather than a single structure. The names and valid values of configuration parameters are identical to those described for the C model in Data Types.

The MATLAB configuration structure can contain an additional element, PersistentMemory. When the element is set to TRUE the internal data memory state of the model is retained following a call to the run function. Otherwise, the model is reset after the data is returned. PersistentMemory is set to FALSE by default.

Get Version

```
[version] = get_version(model)
```

This method returns the version strinf of the C model library used.

Get Configuration

```
[config] = get_configuration(model)
```

This method returns the current parameters structure of a model object. If the model object is empty the method returns the default configuration. If the model object has been created, the method returns the configuration parameters that were used to create it.

Reset

```
[model] =reset(model)
```

This function resets the model.

Config_do

```
config_do(model, config_pkt)
```

This method applies new values of PINC and/or POFF to the model, depending on whether Phase Increment and Phase Offset have been configured as programmable. This function only applies if at least one of Phase Increment or Phase Offset have been configured as programmable.

Run

```
[data_out]=run(model, data_samples)
[data_out]=run(model, data_samples, data_in)
```

Each of these methods causes the model to be run for the number of cycles specified by the data_samples value.

The first form applies if the model has been configured with neither Phase Increment nor Phase Offset configured as Streaming.

The second form applies an array of values for PINC and/or POFF. Each element of data_in should have a value for PINC (if Phase Increment has been configured as Streaming) and/or POFF (if Phase Offset has been configured as Streaming). The number of elements in data_in should match data_samples.

Data out is an array of output samples. Each element contains a phase, sin and cosine value output depending on the configuration of the model. The number of output elements is the data_samples value.

Destroy

```
destroy(model)
```

This method destroys all memory associated with the model.

Example

The `run_dds_compiler_v6_0_bitacc_mex.m` file contains a MATLAB script with several examples of differently configured DDS models showing how to configure and run each. Each example configuration and run produces a plot of the output values.

To run the sample script:

1. Compile the MEX function with the `make_dds_compiler_v6_0_bitacc_mex.m` script (see Compiling).
2. Install the MEX function (see Installation).
3. Execute the `run_dds_compiler_v6_0_bitacc_mex.m` script.

Test Bench

This chapter contains information about the demonstration test bench provided in the Vivado® Design Suite.

Demonstration Test Bench

When the core is generated using the Vivado Design Suite, a demonstration test bench is created. This is a simple VHDL test bench that exercises the core.

The demonstration test bench source code is one VHDL file, `demo_tb/tb_<component_name>.vhd`, located in the Vivado output directory. The source code is comprehensively commented.

Using the Demonstration Test Bench

The demonstration test bench instantiates the generated DDS Compiler core. Compile the netlist and the demonstration test bench into the work library (see your simulator documentation for more information on how to do this). Then simulate the demonstration test bench. View the test bench signals in your simulator waveform viewer to see the operations of the test bench.

Demonstration Test Bench in Detail

The demonstration test bench performs the following tasks:

- Instantiate the core
- Generate a clock signal
- Drive the core input signals to demonstrate core features
- Checks that the core output signals obey AXI protocol rules (data values are not checked in order to keep the test bench simple)
- Provide signals showing the separate fields of AXI TDATA and TUSER signals

The operations performed by the demonstration test bench are appropriate for the configuration of the generated core:

- If phase increment and offset are fixed:
  - Run to produce sine / cosine / phase outputs
- If phase increment and/or offset are programmable, and neither is streaming:
  - Program an initial configuration
  - Run to produce sine / cosine / phase outputs
  - Program a different configuration
  - Run again to produce sine / cosine / phase outputs
- If one of phase increment or offset are streaming and the other is fixed:
  - Stream in constant phase increment or offset to produce sine / cosine / phase outputs
  - If phase offset is streaming, stream in incrementing phase offset to produce higher frequency sine / cosine / phase outputs
- If one of phase increment or offset are streaming, and the other is programmable:
  - Program an initial configuration
  - Stream in constant phase increment or offset to produce sine / cosine / phase outputs
  - If phase offset is streaming, stream in incrementing phase offset to produce higher frequency sine / cosine / phase outputs
  - Continue streaming in phase increment or phase offset, and simultaneously program a different configuration
- If phase increment and offset are both streaming:
  - Stream in constant phase increment and zero phase offset to produce sine / cosine / phase outputs
  - Stream in zero phase increment and incrementing phase offset to produce sine / cosine / phase outputs
- For SIN/COS LUT only:
  - Stream in incrementing phase to produce sine / cosine outputs
- For all configurations:
  - Demonstrate back pressure by deasserting TREADY of master channels (if TREADY is present)
  - Demonstrate use of clock enable (if present)
  - Demonstrate use of reset (if present)

Customizing the Demonstration Test Bench

It is possible to modify the demonstration test bench to drive the core inputs with different data or to perform different operations.

All operations performed by the demonstration test bench to drive the core inputs are done in the `stimuli` process. This process is comprehensively commented, to explain clearly what is being done. New operations, potentially with different input data, can be added by copying and modifying sections of this process.

The clock frequency of the core can be modified by changing the `CLOCK_PERIOD` constant.

Migrating and Upgrading

This appendix contains information about migrating a design from the ISE® Design Suite to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included. This appendix describes migrating from older versions of the IP to the current IP release.

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 8].

Upgrading in the Vivado Design Suite

Parameter Changes

There are no changes to parameters from v5.0 to v6.0. However, there are some new parameters in v6.0 that default to values so that v6.0 is backwards compatible with v5.0. Table A-1 lists the parameter changes from v4.0 to v6.0.

Table A-1: **Parameter Changes**

Version 4.0	Version 6.0	Notes
PartPresent	PartPresent	Unchanged
DDS_Clock_Rate	DDS_Clock_Rate	Unchanged
Channels	Channels	Unchanged
	Mode_of_Operation	New to v6.0
	Modulus	New to v6.0
Parameter _Entry	Parameter _Entry	Unchanged
Spurious_Free_Dynamic_Range	Spurious_Free_Dynamic_Range	Unchanged
Frequency_Resolution	Frequency_Resolution	Unchanged
Noise_Shaping	Noise_Shaping	Unchanged

Table A-1: **Parameter Changes *(Cont'd)***

Version 4.0	Version 6.0	Notes
Phase_Width	Phase_Width	Unchanged
Output_Width	Output_Width	Unchanged
Phase_Increment	Phase_Increment	Unchanged
	Resync	New to v6.0
Phase_Offset	Phase_Offset	Unchanged
Output_Selection	Output_Selection	Unchanged
Negative_Sine	Negative_Sine	Unchanged
Negative_Cosine	Negative_Cosine	Unchanged
Amplitude_Mode	Amplitude_Mode	Unchanged
Memory_Type	Memory_Type	Unchanged
Optimization_Goal	Optimization_Goal	Unchanged
DSP48_Use	DSP48_Use	Unchanged
Latency_Configuration	Latency_Configuration	Unchanged
Latency	Latency	Unchanged
Has_Phase_Out	Has_Phase_Out	Unchanged
SCLR_pin	Has_ARESETn	Name change only. Note that the signal itself, aresetn is active-Low and must be driven Low for at least two cycles.
Clock_Enable	Has_ACLKEN	Name change only.
RFD		Deprecated. Closest equivalent is Has_TREADY.
RDY		Deprecated. Equivalent to AXI4-Stream TVALID, which is not optional.
Channel_Pin		Deprecated. The presence of a channel indication field is now specified by M_DATA_Has_TUSER and M_PHASE_Has_TUSER.
Output_Frequency(1 to 16)	Output_Frequency(1 to 16)	Unchanged
PINC(1 to 16)	PINC(1 to 16)	Unchanged
Phase_Offset_Angles(1 to 16)	Phase_Offset_Angles(1 to 16)	Unchanged
POFF(1 to 16)	POFF(1 to 16)	Unchanged
POR_mode	POR_mode	Unchanged
	DATA_Has_TLAST	New to v5.0
	S_PHASE_Has_TUSER	New to v5.0
	S_PHASE_TUSER_Width	New to v5.0
	Has_TREADY	New to v5.0

Table A-1: **Parameter Changes *(Cont'd)***

Version 4.0	Version 6.0	Notes
	M_DATA_Has_TUSER	New to v5.0
	M_PHASE_Has_TUSER	New to v5.0
	S_CONFIG_Sync_Mode	New to v5.0
	OUTPUT_FORM	New to v6.0

Port Changes

Version 6.0 includes optional new ports or outputs that might not be relevant to those present in v5.0. These new optional ports default to a state that are ignored by v5.0 designs. Table A-2 details the changes to port naming, additional or deprecated ports and polarity changes from v4.0 to v6.0.

Table A-2: **Port Changes from Version 4.0 to Version 6.0**

Version 4.0	Version 6.0	Notes
CLK	aclk	Rename only.
CE	aclken	Rename only.
SCLR	aresetn	Rename, change of sense (now active-Low), requirement to drive reset Low for a minimum of 2 cycles.
ADDR	Deprecated	Replaced by s_axis_config_t* (CONFIG channel).
REG_SELECT		
WE		
DATA		
PINC_IN	Deprecated	Replaced by s_axis_phase_t* (input PHASE channel).
POFF_IN		
PHASE_IN	Deprecated	Replaced by s_axis_phase_t* (Input PHASE channel).
RDY	Deprecated	Nearest equivalent is m_axis_data_tvalid.
RFD	Deprecated	Nearest equivalent is s_axis_phase_tready.
CHANNEL	Deprecated	Channel ID can be carried as a subfield of m_axis_phase_tuser or m_axis_data_tuser.
COSINE	Deprecated	Both these fields are now subfields of m_axis_data_tdata. See Output DATA Channel TDATA Structure in Chapter 3.
SINE		
PHASE_OUT(N-1:0)(1)	m_axis_phase_tdata(byte(N)-1: 0)(1)	PHASE_OUT is now the payload of m_axis_phase_tdata.

Table A-2: **Port Changes from Version 4.0 to Version 6.0 *(Cont'd)***

Version 4.0	Version 6.0	Notes
-	s_axis_config_tvalid	TVALID (AXI4-Stream channel handshake signal) for each channel.
-	s_axis_phase_tvalid	
-	m_axis_phase_tvalid	
-	m_axis_data_tvalid	
-	s_axis_config_tready	TREADY (AXI4-Stream channel handshake signal) for each channel.
-	s_axis_phase_tready	
-	m_axis_phase_tready	
-	m_axis_data_tready	
-	s_axis_config_tlast	TLAST (AXI4-Stream packet signal indicating the last transfer of a data structure) for each channel. See the TLAST User section for each channel, in AXI4-Stream Considerations
-	s_axis_phase_tlast	
-	m_axis_phase_tlast	
-	m_axis_data_tlast	
-	s_axis_phase_tuser (E-1:0)(1)	TUSER (AXI4-Stream ancillary field for application-specific information) for each channel. See the TUSER Packing section for each channel, in AXI4-Stream Considerations
-	m_axis_phase_tuser (F-1:0)(1)	
-	m_axis_data_tuser (G-1:0)(1)	
-	event_s_phase_tlast_missing	Asserted on the last transaction of an incoming vector if s_axis_phase_tlast is not seen asserted.
-	event_s_phase_tlast_unexpected	Asserted on every transaction where s_axis_phase_tlast is unexpectedly seen asserted.
-	event_s_phase_chaind_incorrect	Asserted on every transaction where the s_axis_phase_tuser Channel ID field does not match the value expected by the core.
-	event_s_config_tlast_missing	Asserted on the last transaction of an incoming vector if s_axis_config_tlast is not seen asserted.
-	events_s_config_tlast_unexpected	Asserted on every transaction where s_axis_config_tlast is unexpectedly seen asserted.
-	event_pinc_invalid	New to v6.0. Indicates an invalid value of PINC when in rasterized mode.
-	event_poff_invalid	New to v6.0. Indicates an invalid value of POFF when in rasterized mode.
-	event_phase_in_invalid	New to v6.0. Indicates an invalid value of PHASE_IN when in rasterized mode.

Notes:

1. N, E, F and G are all arbitrary independent integers.

Latency Changes

In version 6.0, latency and initial conditions have changed for some configurations to ensure compliance with the equation for PHASE_OUT, Equation A-1.

The latency of DDS Compiler v6.0 is different compared to v4.0 and greater in general. The update process cannot account for this and guarantee equivalent performance.

IMPORTANT: *When in Blocking Mode (that is, TREADY handshaking is present), the latency of the core is variable, so only the minimum possible latency can be determined. When in Non-Blocking Mode (no TREADY), the latency of the core is as shown in the latency field of the Vivado IDE and is constant.*

Behavioral Changes

SINE/COSINE lookup values remain bit accurate with respect to DDS Compiler v5.0 and v4.0. However, versions of the DDS Compiler core prior to v6.0 had different latencies for the programmable and streaming interfaces (the number of cycles between applying an input and that input affecting the calculated phase), which resulted in a different output sequence depending on the configuration.

This behavior has been standardized in DDS Compiler v6.0; however, this might not match the behavior of previous versions. The generated phase is described by the following equation:

$$\text{PHASE\_OUT}(n) = \sum_{i=0}^{n} \text{PINC}(n) + \text{POFF(n)} \qquad \textit{Equation A-1}$$

Instructions for Minimum Change Migration

The upgrade function alone produces a configuration of v6.0 equivalent to any configuration of v5.0. To configure the DDS Compiler v6.0 to most closely mimic the behavior of v4.0 the translation is as follows:

Parameters

Most parameters remain unchanged. Uncheck Output TREADY. All other new parameters default as required for legacy operation.

- If the CHANNEL output was used, set DATA Output TUSER setting to “Chan_ID_Field”

Ports

Rename ports as described in Table A-2.

- WE, REG_SELECT, ADDR and DATA are replaced by the CONFIG channel, where WE is equivalent to TVALID, ADDR has no equivalent, but is replaced internally by an incrementing count with `s_axis_config_tlast` denoting the last transfer of the TDM sequence, DATA(T-1:0) is replaced by `s_axis_config_tdata` (See CONFIG Channel TDATA Structure in Chapter 3). REG_SELECT is no longer required, because both PINC and POFF can be written in a single transfer.
- PINC_IN and POFF_IN are mapped to `s_axis_phase_tdata` as described in Input PHASE Channel TDATA Structure in Chapter 3. Connect `s_axis_phase_tvalid` to logical 1.
- SINE(P-1:0) and COSINE(P-1:0) are mapped to `m_axis_data_tdata` as described in Output DATA Channel TDATA Structure in Chapter 3. RDY becomes `s_axis_data_tvalid`.
- CHANNEL(N-1:0) becomes `m_axis_data_tuser`(N-1:0).
- PHASE_OUT(W-1:0) becomes `m_axis_phase_tdata`(W-1:0) as described in Output PHASE Channel TDATA Structure - Conventional DDS in Chapter 3.

Miscellaneous Changes

The synchronous, active-Low reset pin `aresetn` must be driven Low for a minimum of two clock cycles to correctly reset the core. The reset is registered within the core, causing some delay between the assertion/deassertion of `aresetn` and the effect being seen on the interface.

Performance

To achieve equivalent performance to v4.0 Latency should be set to the original latency value plus 1. Alternatively, set Latency to be the same as Latency for v4.0, but there might be drop in performance. Resource allocation for this configuration is greater than v4.0 in flip-flop count by approximately the number of bits in any fields in `s_axis_phase_tdata`.

Functionality Changes

Version 6.0 is bit accurate with v5.0 for any configuration of v5.0. Some configurations might differ in initial output or latency compared to v5.0 to comply with Equation A-1. When AXI4-Stream interface considerations have been handled according to AXI4-Stream Considerations in Chapter 3, there are no other functionality changes between v4.0 and v6.0.

Simulation

Starting with DDS Compiler v6.0 (2013.3 version), behavioral simulation models have been replaced with IEEE P1735 Encrypted VHDL. The resulting model is bit and cycle accurate with the final netlist. For more information on simulation, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the DDS Compiler core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the DDS Compiler core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Records for the DDS Compiler Core

AR: 54498

Sub-Harmonic Frequencies

The equations for SFDR rely on the assumption that rounding errors from the finite precision of phase and amplitude are incoherent. This assumption is violated for values of Phase Increment that are not mutually prime with the weight of the Phase Accumulator. The anomalies, such as spurii, are more obvious for larger common factors between the Phase Increment Value and the weight of the accumulator (2^{Phase_Width}). This is because such values might not access every location in the SIN/COS LUT, so the rounding errors are not randomly spread. To avoid this, do not use values of Output Frequency that are simple rational fractions of the frequency per channel, Fs, such as 3/8, 1/64.

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

There are many tools available to address DDS Compiler core design issues.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx® devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 9].

Reference Boards

Various Xilinx development boards support the DDS Compiler. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series FPGA evaluation boards
  - KC705
  - KC724

AXI4-Stream Interface Debug

If data is not being transmitted or received, check the following conditions:

- If transmit <*interface_name*>_tready is stuck Low following the **<*interface_name*>_tvalid** input being asserted, the core cannot send data.
- If the receive **<*interface_name*>**_tvalid is stuck Low, the core is not receiving data.
- Check that the ACLK inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed, as shown in Figure 3-15, Figure 3-16, Figure 3-17, and Figure 3-18.
- Check core configuration.

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. *Vivado® Design Suite: AXI Reference Guide* (UG1037)
2. *AMBA® AXI4-Stream Protocol Specification* (ARM IHI 0051A)
3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
4. *Vivado Design Suite User Guide: Designing with IP* (UG896)
5. *Vivado Design Suite User Guide: Getting Started* (UG910)
6. *System Generator for DSP User Guide* (UG640)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *ISE® to Vivado Design Suite Migration Guide* (UG911)
9. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	6.0	Added support for UltraScale+ families.
06/24/2015	6.0	• Added MATLAB interface description to C Model chapter.
04/01/2015	6.0	• Data Format section in C Model chapter updated to clarify sine, cosine and phase output format.
10/01/2014	6.0	• Minor corrections to Phase Increment section of Designing with the Core to account for automatic gain control. • Minor corrections to Output section of Implementation Tab description in Design Flow Steps to account for Taylor mode. • Added Data Format section to C Model.
04/02/2014	6.0	• Added link to resource utilization figures. • Added User Parameter table (Table 4-4).
12/18/2013	6.0	Added support for UltraScale™ architecture.
10/02/2013	6.0	• Minor updates to IP Facts table and Migrating appendix. • Corrections to Designing with the Core chapter. • Document version number advanced to match the core version number.
03/20/2013	1.0	Initial Xilinx release. Replaces DS794, *LogiCORE IP DDS Compiler Data Sheet*. • Updated for core v6.0. • Added support for Zynq-7000 and Artix-7 devices. • Removed support for ISE Design Suite. • Removed support for Virtex-6 and Spartan-6 devices.

Please Read: Important Legal Notices