

# Zynq UltraScale+ RFSoc RF Data Converter v2.2

## *LogiCORE IP Product Guide*

Vivado Design Suite

PG269 (v2.2) October 30, 2019



# Table of Contents

<b>Chapter 1: IP Facts</b> .....	<b>5</b>
Features.....	5
IP Facts.....	6
<b>Chapter 2: Overview</b> .....	<b>7</b>
RF-ADC.....	9
RF-DAC.....	13
Applications.....	15
Licensing and Ordering.....	15
<b>Chapter 3: Product Specification</b> .....	<b>16</b>
Performance.....	18
Resource Use.....	18
Port Descriptions.....	18
Register Space.....	25
<b>Chapter 4: Designing with the Core</b> .....	<b>36</b>
IP Core Configuration in the Vivado Design Suite.....	36
Software Driver.....	36
RF-ADC.....	37
RF-DAC.....	75
Quadrature Modulator Correction .....	98
Coarse Delay.....	101
Dynamic Update Events.....	102
PLL.....	104
Interrupt Handling.....	105
Clocking.....	110
Resets.....	115
Power-up Sequence.....	116
Bitstream Reconfiguration.....	118
Interfacing to the AXI4-Stream Interface.....	118
Applications Overview.....	119

<b>Chapter 5: Design Flow Steps.....</b>	<b>138</b>
Customizing and Generating the Core.....	138
Simulation.....	152
Synthesis and Implementation.....	152
<b>Chapter 6: Example Design.....</b>	<b>153</b>
RF-ADC Data Capture Block.....	154
RF-DAC Data Stimulus Block.....	156
Digital Data Format.....	158
<b>Chapter 7: Test Bench.....</b>	<b>159</b>
Analog Signaling.....	161
<b>Appendix A: Upgrading.....</b>	<b>164</b>
Changes from V2.1 to V2.2 .....	164
Changes from V2.0 to V2.1.....	164
Changes from V1.2 to V2.0.....	166
Changes from V1.1 to V1.2.....	166
<b>Appendix B: Debugging.....</b>	<b>168</b>
Finding Help on Xilinx.com.....	168
Debug Tools.....	169
Hardware Debug.....	170
Interface Debug.....	170
<b>Appendix C: Zynq UltraScale+ RFSoc RF Data Converter Bare-metal/Linux Driver.....</b>	<b>172</b>
Overview.....	172
Data Structures.....	173
User API Functions.....	190
Interrupt Handling.....	228
In-line Functions.....	234
<b>Appendix D: RF Analyzer.....</b>	<b>250</b>
Clocking.....	252
Address Space.....	254
User Interface.....	254
Hardware Trigger.....	255

<b>Appendix E: Additional Resources and Legal Notices.....</b>	<b>257</b>
Xilinx Resources.....	257
Documentation Navigator and Design Hubs.....	257
References.....	257
Revision History.....	258
Please Read: Important Legal Notices.....	260

## IP Facts

The Xilinx<sup>®</sup> LogiCORE™ IP Zynq<sup>®</sup> UltraScale+™ RFSoc RF Data Converter IP core provides a configurable wrapper to allow the RF-DAC and RF-ADC blocks to be used in IP integrator designs.



---

**IMPORTANT!** *In this guide reference is made to the Dual and Quad tiles; for the actual sampling rate specifications, see the Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926). Note that Dual and Quad refers to the tile configuration and not the number of converters.*

---

---

## Features

- Up to 16 14-bit RF-DACs
- Eight 12-bit Dual RF-ADCs, or 16 12-bit Quad RF-ADCs, depending on device
- Supports alignment between multiple converters
- Pre-programs RF-DAC and RF-ADC with key user-defined parameters
- Multiple AXI4-Stream data interfaces for RF-ADCs and RF-DACs
- Single AXI4-Lite configuration interface
- 1x (bypass), 2x, 4x, 8x interpolation
- 1x (bypass), 2x, 4x, 8x decimation
- Digital complex mixers
- Numerical Controlled Oscillator (NCO)
- Quadrature Modulation Correction (QMC)
- On-chip PLL and VCO per tile

# IP Facts

LogiCORE IP Facts Table	
<b>Core Specifics</b>	
Supported Device Family <sup>1</sup>	Zynq® UltraScale+™ RFSoc
Supported User Interfaces	AXI4-Stream, AXI4-Lite Control/Status
Resources	<a href="#">Performance and Resource Use web page</a>
<b>Provided with Core</b>	
Design Files	RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Verilog
Supported S/W Driver <sup>2</sup>	Standalone and Linux
<b>Tested Design Flows<sup>3</sup></b>	
Design Entry	Vivado® IP Integrator
Simulation	For supported simulators, see the <a href="#">Xilinx Design Tools: Release Notes Guide</a> .
Synthesis	Vivado Synthesis
<b>Support</b>	
Release Notes and Known Issues	Master Answer Record: <a href="#">69907</a>
All Vivado IP Change Logs	Master Vivado IP Change Logs: <a href="#">72775</a>
<a href="#">Xilinx Support web page</a>	

**Notes:**

- For a complete list of supported devices, see the Vivado® IP catalog.
- Stand-alone driver details can be found in the software development kit <Install Directory>/Vitis/<Release>/data/embeddedsw/doc/Xilinx\_drivers.htm.  
Bare Meta/Linux documentation is available in [Appendix C: Zynq UltraScale+ RFSoc RF Data Converter Bare-metal/Linux Driver](#).
- For the supported versions of third-party tools, see the [Xilinx Design Tools: Release Notes Guide](#).

# Overview

The Xilinx<sup>®</sup> Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc family integrates the key subsystems required to implement a complete software-defined radio including direct RF sampling data converters, enabling eCPRI and Gigabit Ethernet-to-RF on a single, highly programmable SoC.

Each RFSoc offers multiple RF-sampling analog-to-digital (RF-ADC) and RF-sampling digital-to-analog (RF-DAC) data converters. The data converters are high-precision, high-speed and power-efficient. Both are highly configurable and tightly integrated with the programmable logic (PL) resources of the Zynq UltraScale+ RFSoc.

The RF-ADC supports device-dependent sample rates and input signal frequencies listed in the *Zynq UltraScale+ RFSoc Data Sheet: Overview* ([DS889](#)), with excellent dynamic range performance.

The RF-DAC generates output carrier frequencies at rates defined in the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics* ([DS926](#)), depending on the device (see the *Zynq UltraScale+ RFSoc Data Sheet: Overview* ([DS889](#)) for device information).

The RF data converters also include power efficient digital down converters (DDCs) and digital up converters (DUCs) that include programmable interpolation and decimation rates, a numerically controlled oscillator (NCO), and a complex mixer. The DDCs and DUCs can also support multi-band operation. The following figure shows the block diagram of the Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc RF Data Converter.

The RF-ADCs and RF-DACs are organized into tiles, each containing two or four RF-ADCs or four RF-DACs. Multiple tiles are available in each Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc (see the specific device data sheet for the number of tiles and converters per device). Each tile also includes a block with a PLL and all the necessary clock handling logic and distribution routing for the analog and digital logic.

This guide describes the Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc RF Data Converter IP core and software drivers that are used to configure the data converters and instantiate them for use in a design.

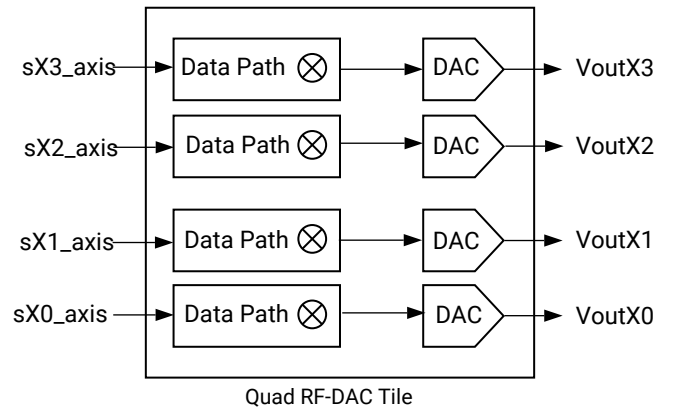
For device specifications and additional information, see:

- *Zynq UltraScale+ RFSoc Data Sheet: Overview* ([DS889](#))
- *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics* ([DS926](#))
- *Zynq UltraScale+ Device Technical Reference Manual* ([UG1085](#)).





Figure 3: RF-DAC Tile Structure



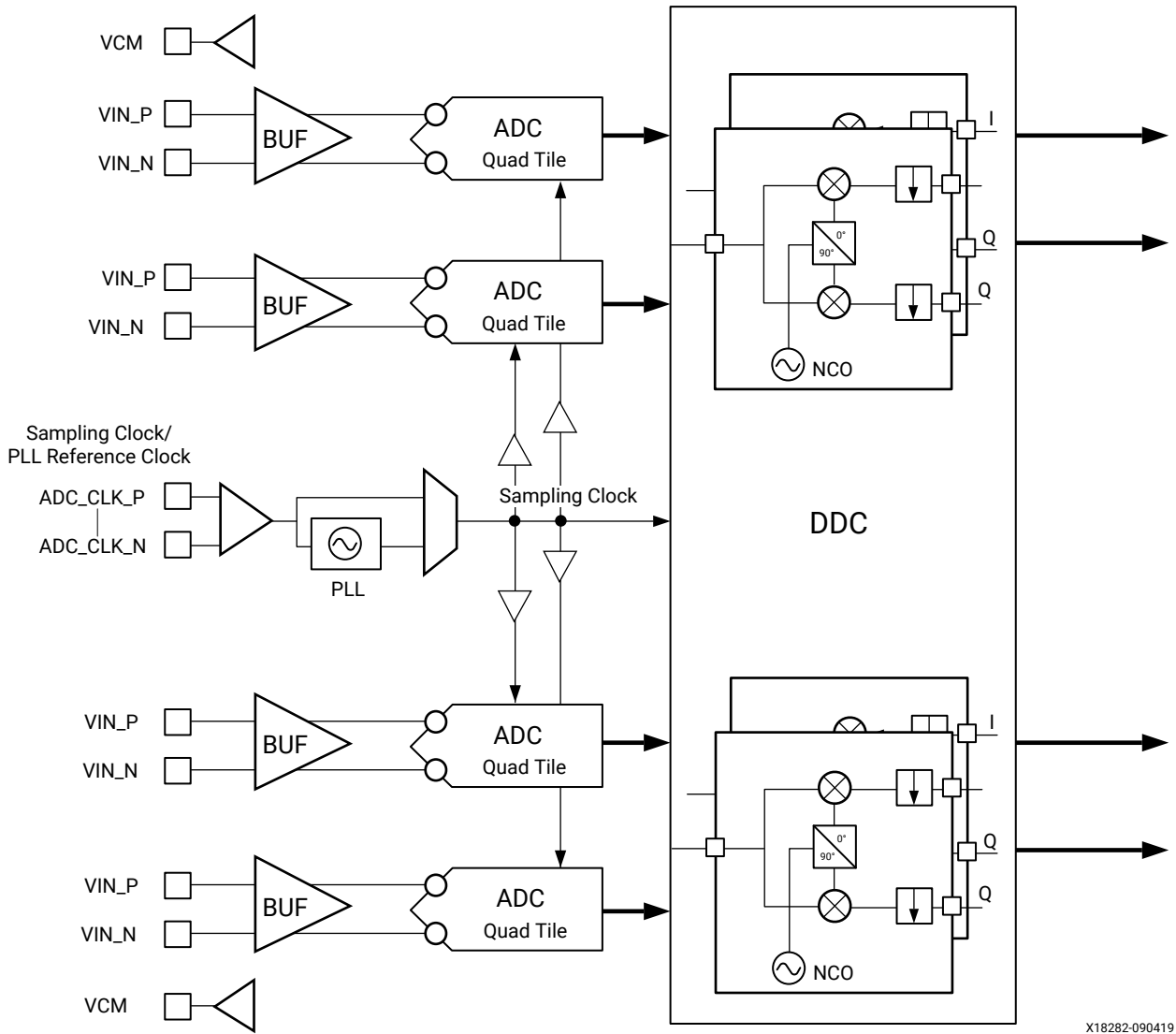
X23274-092319

## RF-ADC

There are two types of RF-ADC tile, the Dual RF-ADC and the Quad RF-ADC tile. The type of tile available is device dependent (see the *Zynq UltraScale+ RFSoc Data Sheet: Overview (DS889)*). Each tile includes a PLL and clocking instance. All RF-ADCs within a tile share this common clocking infrastructure.

The Quad RF-ADC tile consists of four RF-ADCs, arranged in two pairs. Each of these converters can be configured individually for real input signals or, as a pair, for I/Q input signals. The following figure shows an overview of the Quad RF-ADC tile.

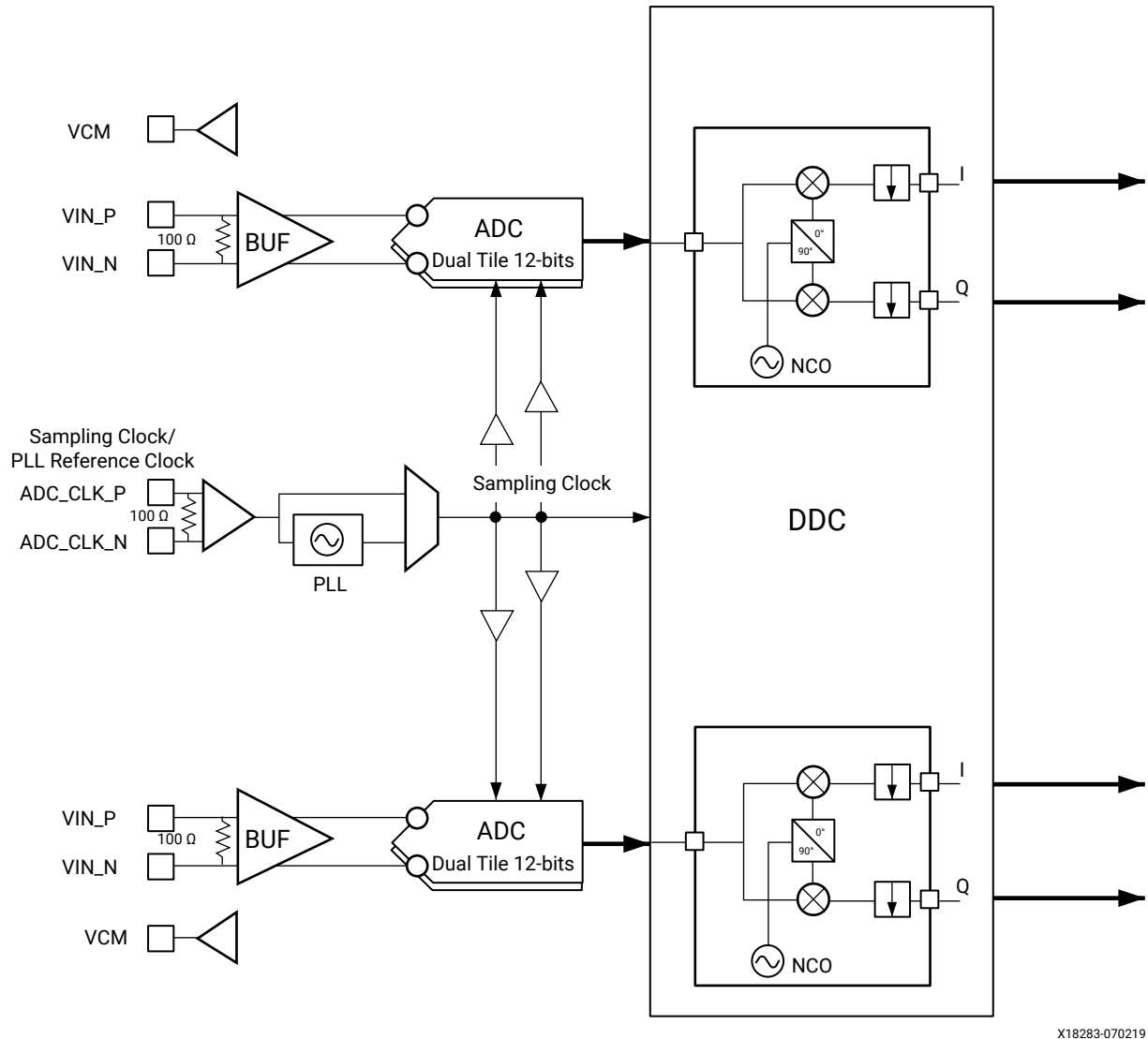
Figure 4: Quad RF-ADC Tile Overview



X18282-090419

The Dual RF-ADC tile consists of two RF-ADCs. These converters can be configured individually for real input signals or, as a pair, for I/Q input signals. The following figure shows an overview of the Dual RF-ADC tile.

Figure 5: Dual RF-ADC Tile Overview



## RF-ADC Features

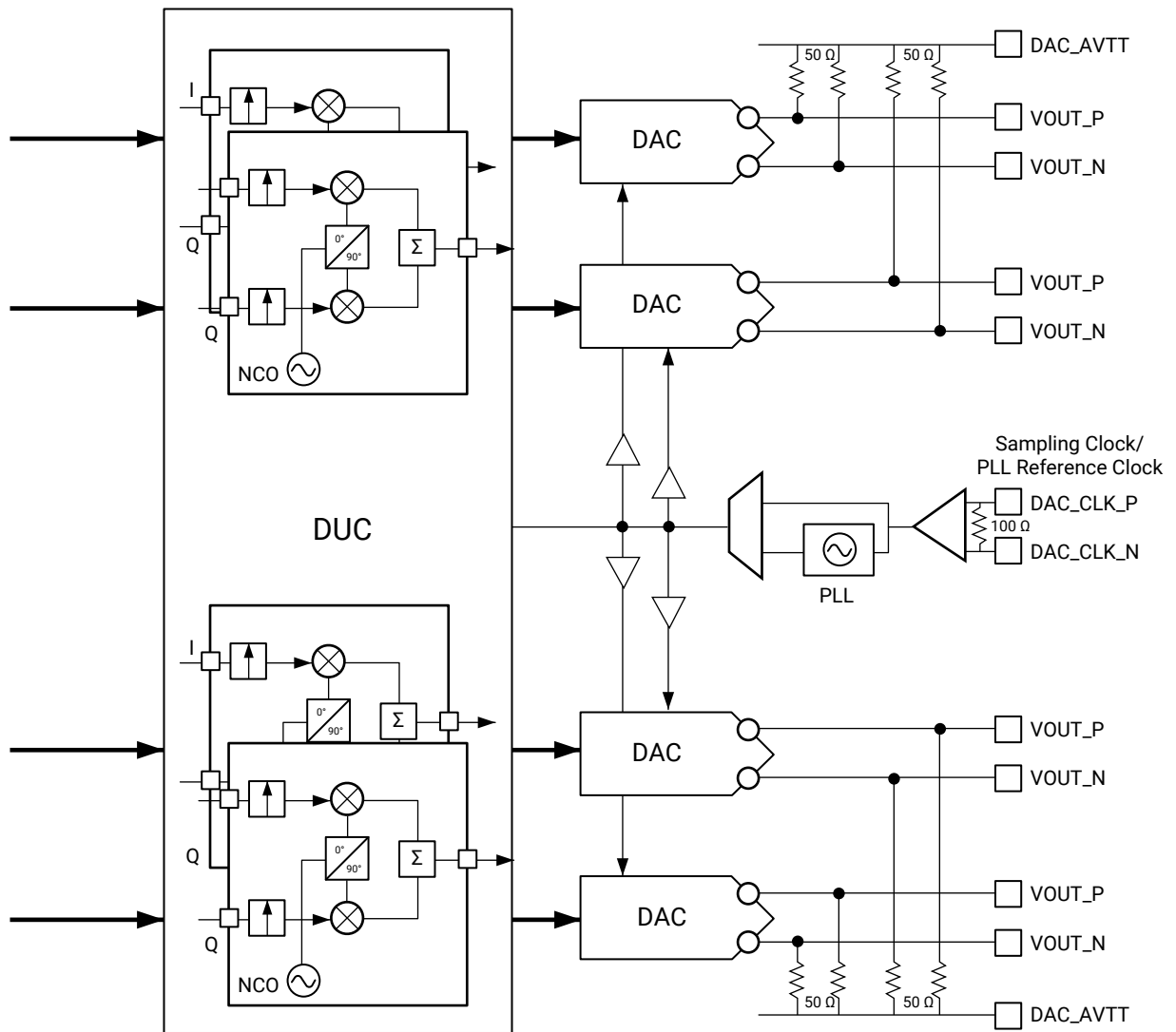
- Tile configuration
  - Four or two RF-ADCs and one PLL per tile
  - 12-bit RF-ADC resolution, with 16-bit digital signal processing datapath; each 12-bit data stream is MSB-aligned to 16 bit samples at the output of the RF-ADC core before passing to the DDC block

- Implemented as either four channels (Quad) or two channels (Dual) (the sampling rate is device dependent; for the actual sampling rate specifications, see the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)*)
- Decimation filters
  - 1x (bypass filter), 2x, 4x, 8x
  - 80% of Nyquist bandwidth, 89 dB stop-band attenuation
- Digital Complex Mixers
  - Full complex mixers support real or I/Q inputs from the RF-ADC
  - 48-bit Numeric Controlled Oscillator (NCO) per RF-ADC
  - Fixed  $F_s/4$ ,  $F_s/2$  low power frequency mixing mode, where  $F_s$  is the sample frequency
  - I/Q and real input signals supported
- Single/multi-band flexibility
  - 2x bands per RF-ADC pair
  - 4x bands per Quad RF-ADC tile
  - Can be configured for real or I/Q inputs
- Full bandwidth of the RF-ADC can be accessed in bypass mode
- Input signal amplitude threshold: Two programmable threshold flags per RF-ADC
- Built-in digital correction for external analog quadrature modulators:
  - Supports gain, phase, and offset correction for an I/Q input pair (two RF-ADCs)
- SYSREF input signal for multi-channel synchronization
- Flexible AXI4-Stream interface supports a wide range of programmable logic clock rates and converter sample rates
- Per tile current-mode logic (CML) clock input buffer with on-chip calibrated 100 $\Omega$  termination; supplies the RF-ADC sampling clocks or provides a reference clock for the on-chip PLL
- Dedicated high-speed, high-performance, differential input buffer per RF-ADC with on-chip calibrated 100 $\Omega$  termination (on-die termination)
- Output common mode reference voltage for DC-coupling RF-ADC inputs

# RF-DAC

Each RF-DAC tile consists of four RF-DACs that can be configured individually for real output signals or, as a pair, for I/Q output signal generation. Each RF-DAC runs at a data rate specified in the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)*. The RF-DAC tile has one PLL and a clocking instance. The following figure shows an overview of the RF-DAC.

Figure 6: RF-DAC Overview



X18281-082019

## RF-DAC Features

- Tile configuration
  - Four RF-DACs and one PLL per tile
  - 14-bit RF-DAC resolution with 16-bit digital signal processing path; the data is MSB-aligned to 16 bits
  - Device-dependent sampling speed; see the *Zynq UltraScale+ RFSoc Data Sheet: Overview (DS889)*
  - Device-dependent full power output bandwidth; see the *Zynq UltraScale+ RFSoc Data Sheet: Overview (DS889)*
- Interpolation
  - 1x (bypass filter), 2x, 4x, 8x
  - 80% pass band, 89 dB stop band attenuation
- Digital Complex Mixers
  - Full complex mixers support real or I/Q output signals to the RF-DACs
  - 48-bit NCO per RF-DAC
  - Fixed  $F_s/4$ ,  $F_s/2$  low-power frequency mixing mode
  - Supports mixed mode RF-DAC functionality which maximizes RF-DAC power in the second Nyquist zone
- Single/multi-band flexibility
  - 2x bands per RF-DAC pair
  - 4x bands per RF-DAC tile
  - Can be configured for real or I/Q outputs
- Full Nyquist bandwidth in bypass mode
- Digital Correction for external analog quadrature modulators:
  - Supports gain, phase, and offset correction for an I/Q output pair (two RF-DACs)
- Sinc correction for first Nyquist zone
- External input signal (SYSREF) for multi-channel synchronization of data converter channels
- Per tile current mode logic (CML) clock input buffer with on-chip calibrated 100Ω termination; supplies the RF-DAC sampling clocks or provides a reference clock for the on-chip PLL
- Supports 20 mA or 32 mA output power mode

For the RF-ADC and RF-DAC operating and absolute maximum/minimum parameters see:

- [Zynq UltraScale+ RFSoc Data Sheet: Overview \(DS889\)](#)
- [Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics \(DS926\)](#)

---

## Applications

- Multi-band, multi-mode 3G, 4G, and 5G cellular radios
- Multiple antenna systems (Massive MIMO, AAS, AAA)
- Cable infrastructure (DOCSIS 3.x)
- Software defined radios
- Microwave and millimeter wave radios
- Test and measurement applications

### Related Information

[Applications Overview](#)

---

## Licensing and Ordering

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](#).

**Note:** To verify that you need a license, check the License column of the IP Catalog. Included means that a license is included with the Vivado® Design Suite; Purchase means that you have to purchase a license to use the core.

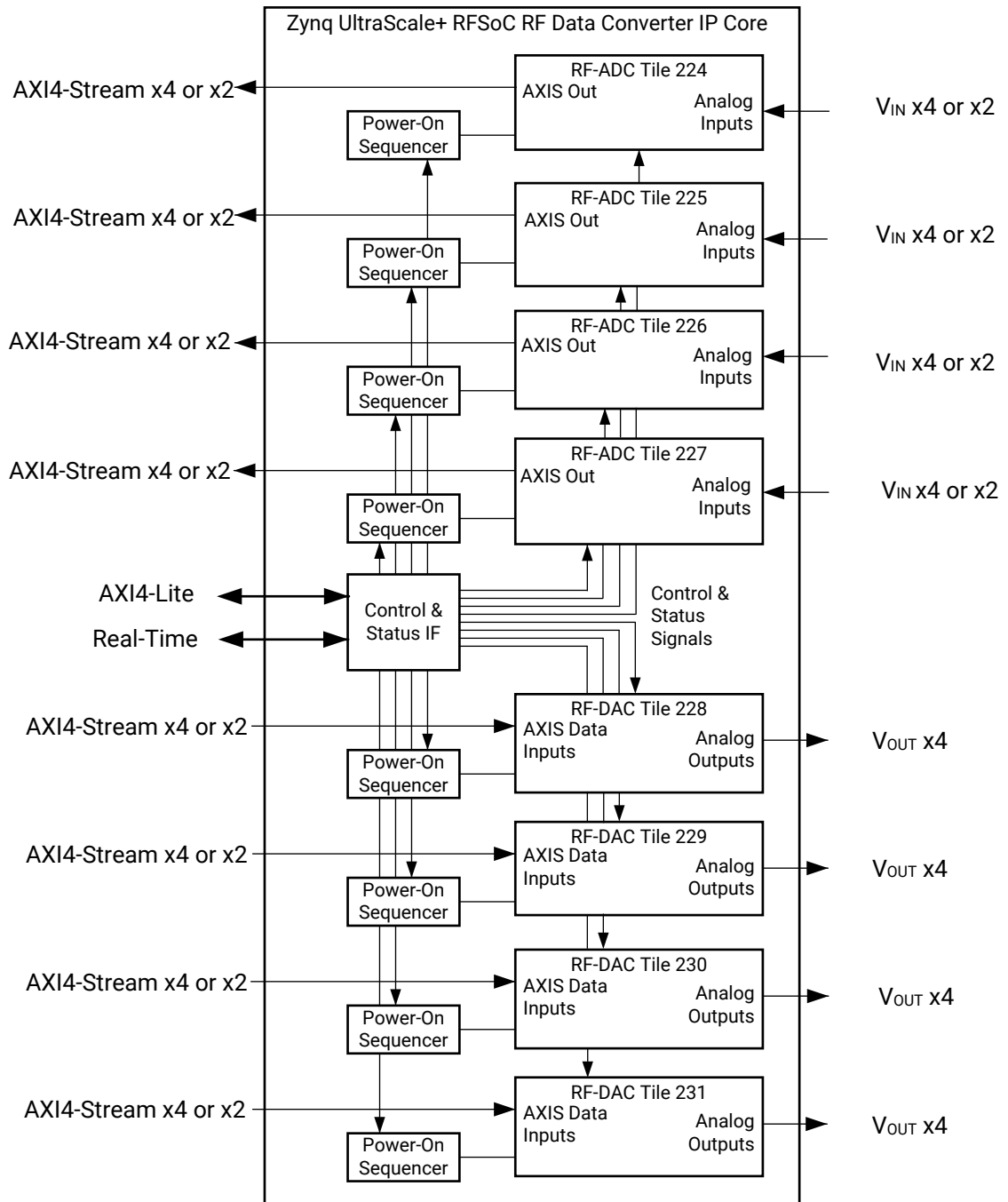
Information about other Xilinx® LogiCORE™ IP modules is available at the [Xilinx Intellectual Property](#) page. For information about pricing and availability of other Xilinx® LogiCORE IP modules and tools, contact your [local Xilinx sales representative](#).

# Product Specification

The Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc RF Data Converter IP core provides a way of instantiating all the RF-DAC and RF-ADC blocks in Zynq UltraScale+ RFSoc in IP integrator. A single IP core instance allows access to all converters in the device. The IP ensures that all enabled blocks are powered up and that unused converters are disabled.



Figure 7: IP Core Overview



X18850-082019

An RF-ADC tile has two or four RF-ADCs; an RF-DAC tile has four RF-DACs. The number of converters and the maximum sample rate depend on the device and package. The converters in each tile are the same type.

Related Information  
[Power-up Sequence](#)

## Performance

To see the performance of the RF-ADC and RF-DAC blocks, see the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)*.

Related Information  
[Clocking](#)

## Resource Use

For full details about performance and resource utilization, visit the [Performance and Resource Utilization web page](#) (registration required).

## Port Descriptions

### Configuration Interface Ports

Table 1: Configuration Interface Ports

Port Name	I/O	Clock	Description
s_axi_aclk	In	N/A	AXI clock input (continuous clock)
s_axi_aresetn	In	N/A	Reset for the aclk domain. The deassertion of the reset should be synchronous to s_axi_aclk.
s_axi_awaddr[17:0]	In	s_axi_aclk	Write Address
s_axi_awvalid	In	s_axi_aclk	Write Address Valid
s_axi_awready	Out	s_axi_aclk	Write Address Ready
s_axi_wdata[31:0]	In	s_axi_aclk	Write Data
s_axi_wstrb[3:0]	In	s_axi_aclk	Write Data Byte Strobe
s_axi_wvalid	In	s_axi_aclk	Write Data Valid
s_axi_wready	Out	s_axi_aclk	Write Data Ready
s_axi_bresp[1:0]	Out	s_axi_aclk	Write Response
s_axi_bvalid	Out	s_axi_aclk	Write Response Valid
s_axi_bready	In	s_axi_aclk	Write Response Ready

Table 1: Configuration Interface Ports (cont'd)

Port Name	I/O	Clock	Description
s_axi_araddr[17:0]	In	s_axi_aclk	Read Address
s_axi_arvalid	In	s_axi_aclk	Read Address Valid
s_axi_arready	Out	s_axi_aclk	Read Address Ready
s_axi_rdata[31:0]	Out	s_axi_aclk	Read Data
s_axi_rresp[1:0]	Out	s_axi_aclk	Read Response
s_axi_rvalid	Out	s_axi_aclk	Read Data Valid
s_axi_rready	In	s_axi_aclk	Read Data Ready
irq	Out	s_axi_aclk	Interrupt output

## Multi-Tile Synchronization Ports

Table 2: Multi-Tile Synchronization Ports

Port Name	I/O	Clock	Description <sup>1</sup>
sysref_in_p	In	N/A	External analog SYSREF input
sysref_in_n	In	N/A	External analog SYSREF input
user_sysref_adc	In	m0_axis_aclk	RF-ADC SYSREF input from programmable logic (PL)/ user design; synchronous to RF-ADC tile 0 PL clock
user_sysref_dac	In	s0_axis_aclk	RF-DAC SYSREF input from programmable logic (PL)/user design; synchronous to RF-DAC tile 0 PL clock

**Notes:**

1. See the Multi-Tile Synchronization section in the Applications sub-section for more information.

### Related Information

[Multi-Tile Synchronization](#)

## Clock Ports Common to RF-DAC Tile

Table 3: Clock Ports Common to RF-DAC Tile

Port Name <sup>1</sup>	I/O	Clock	Description
dacX_clk_p	In	N/A	RF-DAC on-chip PLL reference clock or sampling clock input
dacX_clk_n	In	N/A	RF-DAC on-chip PLL reference clock or sampling clock input
clk_dacX	Out	N/A	Output clock to user logic

**Notes:**

1. X refers to the location of the tile in the converter column.

## AXI4-Stream Related Ports for RF-DACs

Table 4: AXI4-Stream Related Ports for RF-DACs

Port Name <sup>1</sup>	I/O	Clock	Description
sX_axis_aclk	In	N/A	Clock input for RF-DAC data input
sX_axis_aresetn	In	N/A	Synchronous reset for the sX_axis_aclk domain. This should be held low until sX_axis_aclk is stable.
sXY_axis_tdata[M:0]	In	sX_axis_aclk	AXI4-Stream data input
sXY_axis_tvalid	In	sX_axis_aclk	AXI4-Stream valid
sXY_axis_tready	Out	sX_axis_aclk	AXI4-Stream ready
voutXZ_p	Out	N/A	Analog output
voutXZ_n	Out	N/A	Analog output

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DUC block in the tile (0 to 3). Z refers to the location of the RF-DAC in the tile (0 to 3). M is the number of samples per AXI4-Stream word \* 16 for converter XY.

## Real-Time Signal Interface Ports for RF-DACs

Table 5: Real-Time Signal Interface Ports for RF-DACs

Port Name <sup>1</sup>	I/O	Clock	Description
dacXZ_fast_shutdown[2:0]	In	N/A	RF-DAC fast shutdown 001 - Scale output data by 0.5 011 - Scale output data by 0.25 111 - Scale output data by 0 Others - Normal operation
dacXY_pl_event	In	clk_dacX	RF-DAC PL event Assert to update RF-DAC settings from the PL

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DUC block in the tile (0 to 3). Z refers to the location of the RF-DAC in the tile (0 to 3).

## Real-Time NCO Signal Interface Ports for RF-DACs

Table 6: Real Time NCO Signal Interface Ports for RF-DACs

Port Name <sup>1</sup>	I/O	Clock	Description
dacXY_nco_freq[47:0]	In	s_axi_aclk	Requested NCO frequency setting. This is a 48 bit signed input representing the NCO frequency. The value ranges from $-F_s/2$ to $F_s/2$ , where $F_s$ is the sampling rate.
dacXY_nco_phase[17:0]	In	s_axi_aclk	Requested NCO Phase setting. This is a 18 bit signed number representing the NCO phase. The value ranges from -180 to 180 degrees.

Table 6: Real Time NCO Signal Interface Ports for RF-DACs (cont'd)

Port Name <sup>1</sup>	I/O	Clock	Description
dacXY_nco_phase_rst	In	s_axi_aclk	NCO phase reset. Used to align the NCO phases across the converter.
dacXY_update_enable[5:0]	In	s_axi_aclk	Enable register writes <ul style="list-style-type: none"> <li>• Bit 5: Enable write to phase reset</li> <li>• Bit 4: Enable write to NCO phase bits 17:16</li> <li>• Bit 3: Enable write to NCO phase bits 15:0</li> <li>• Bit 2: Enable write to NCO frequency bits 47:32</li> <li>• Bit 1: Enable write to NCO frequency bits 31:16</li> <li>• Bit 0: Enable write to NCO frequency bits 15:0</li> </ul>
dacX_update_request	In	s_axi_aclk	Asserted High to request an update of the NCO settings.
dacX_nco_update_busy[1:0]	Out	s_axi_aclk	Update busy register <ul style="list-style-type: none"> <li>• Bit 1: High while SYSREF is disabled. Applicable for RF-DAC tile 228 when multi-tile synchronization has been selected</li> <li>• Bit 0: High when the NCO update is in progress.</li> </ul>
dac0_sysref_int_gating	In	s_axi_aclk	Applicable for RF-DAC tile 228 when multi-tile synchronization is enabled. When asserted the disabling of the SYSREF is carried out by the IP core.
dac0_sysref_int_reenable	In	s_axi_aclk	Applicable for RF-DAC tile 228 when multi-tile synchronization is enabled. When asserted the IP core re-enables the SYSREF after the NCO update process has completed. In multi-device systems all devices should be re-enabled at the same time.

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DUC block in the tile (0 to 3).

## Clock Ports Common to RF-ADC Tile

Table 7: Clock Ports Common to RF-ADC Tile

Port Name <sup>1</sup>	I/O	Clock	Description
adcX_clk_p	In	N/A	RF-ADC on-chip PLL reference clock or sampling clock input
adcX_clk_n	In	N/A	RF-ADC on-chip PLL reference clock or sampling clock input
clk_adcX	Out	N/A	Output clock to user logic

**Notes:**

1. X refers to the location of the tile in the converter column.

## AXI4-Stream Related Ports for RF-ADCs

Table 8: AXI4-Stream Related Ports for RF-ADCs

Port Name <sup>1</sup>		I/O	Clock	Description
mX_axis_aclk		In	N/A	Clock input for RF-ADC data output
mX_axis_aresetn		In	N/A	Synchronous reset for the mX_axis_aclk domain. This should be held low until mX_axis_aclk is stable.
mXY_axis_tdata[M:0]		Out	mX_axis_aclk	AXI4-Stream data output
mXY_axis_tvalid		Out	mX_axis_aclk	AXI4-Stream valid
mXY_axis_tready		In	mX_axis_aclk	AXI4-Stream ready
Quad RF-ADC Tile	vinXZ_p	In	N/A	Analog input
	vinXZ_n	In	N/A	Analog input
Dual RF-ADC Tile	vinX_ZZ_p	In	N/A	Analog input
	vinX_ZZ_n	In	N/A	Analog input

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DDC block in the tile (0 to 3). In Quad RF-ADC tile devices, Z refers to the location of the RF-ADC in the tile (0 to 3). In Dual RF-ADC tile devices, ZZ is either 01 (the lower RF-ADC in the tile) or 23 (the upper RF-ADC in the tile). M is the number of samples per AXI4-Stream word \* 16 for converter XY.

## Real-Time Signal Interface Ports for Quad RF-ADCs

Table 9: Real-Time Signal Interface Ports for Quad RF-ADCs

Port Name <sup>1,2</sup>		I/O	Clock	Description
adcXY_pl_event		In	clk_adcX	RF-ADC PL event Assert to update RF-ADC settings from the PL
adcXZ_over_range		Out	Async	Over range output. A High on this output indicates that the signal exceeds the full-scale input of the RF-ADC.
adcXZ_over_threshold1		Out	clk_adcX	Over threshold1 output Signal amplitude level is above programmable threshold 1
adcXZ_over_threshold2		Out	clk_adcX	Over threshold2 output Signal amplitude level is above programmable threshold 2
adcXZ_over_voltage		Out	Async	Over voltage output An Over Voltage condition occurs when a signal far exceeds the normal operating input-range.

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DDC block in the tile (0 to 3). Z refers to the location of the RF-ADC in the tile (0 to 3).
2. See RF-ADC Threshold and Over Range Settings for details on the real-time signals.

### Related Information

[RF-ADC Threshold and Over Range Settings](#)

## Real-Time Signal Interface Ports for Dual RF-ADCs

Table 10: Real-Time Signal Interface Ports for Dual RF-ADCs

Port Name <sup>1,2</sup>	I/O	Clock	Description
adcXY_pl_event	In	clk_adcX	RF-ADC PL event Assert to update RF-ADC settings from the PL
adcX_ZZ_over_range	Out	Async	Over range output. A High on this output indicates that the signal exceeds the full-scale input of the RF-ADC.
adcX_ZZ_over_threshold1	Out	clk_adcX	Over threshold1 output Signal amplitude level is above programmable threshold 1
adcX_ZZ_over_threshold2	Out	clk_adcX	Over threshold2 output Signal amplitude level is above programmable threshold 2
adcX_ZZ_over_voltage	Out	Async	Over voltage output An Over Voltage condition occurs when a signal far exceeds the normal operating input-range.

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DDC block in the tile (0 to 3). ZZ is either 01 (the lower RF-ADC in the tile) or 23 (the upper RF-ADC in the tile)
2. See RF-ADC Threshold and Over Range Settings for details on the real-time signals.

### Related Information

[RF-ADC Threshold and Over Range Settings](#)

## Real-Time NCO Signal Interface Ports for Quad RF-ADCs

Table 11: Real-Time NCO Signal Interface Ports for Quad RF-ADCs

Port Name <sup>1</sup>	I/O	Clock	Description
adcXY_nco_freq[47:0]	In	s_axi_aclk	Requested NCO frequency setting. This is a 48-bit signed input representing the NCO frequency. The value ranges from $-F_s/2$ to $F_s/2$ , where $F_s$ is the sampling rate
adcXY_nco_phase[17:0]	In	s_axi_aclk	Requested NCO phase setting. This is a 18-bit signed number representing the NCO phase. The value ranges from -180 to 180 degrees
adcXY_nco_phase_rst	In	s_axi_aclk	NCO phase reset. Used to align the phases across the converter

Table 11: Real-Time NCO Signal Interface Ports for Quad RF-ADCs (cont'd)

Port Name <sup>1</sup>	I/O	Clock	Description
adcXY_update_enable[5:0]	In	s_axi_aclk	Enable register writes <ul style="list-style-type: none"> <li>Bit 5: Enable write to phase reset</li> <li>Bit 4: Enable write to NCO phase bits 17:16</li> <li>Bit 3: Enable write to NCO phase bits 15:0</li> <li>Bit 2: Enable write to NCO frequency bits 47:32</li> <li>Bit 1: Enable write to NCO frequency bits 31:16</li> <li>Bit 0: Enable write to NCO frequency bits 15:0</li> </ul>
adcX_update_request	In	s_axi_aclk	Asserted High to request an update of the NCO settings
adcX_update_busy	Out	s_axi_aclk	Update busy register. High when the NCO update is in progress

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DDC block in the tile (0 to 3).

## Real-Time NCO Signal Interface Ports for Dual RF-ADCs

Table 12: Real-Time NCO Signal Interface Ports for Dual RF-ADCs

Port Name <sup>1</sup>	I/O	Clock	Description
adcX_ZZ_nco_freq[47:0]	In	s_axi_aclk	Requested NCO frequency setting. This is a 48-bit signed input representing the NCO frequency. The value ranges from $-F_s/2$ to $F_s/2$ , where $F_s$ is the sampling rate.
adcX_ZZ_nco_phase[17:0]	In	s_axi_aclk	Requested NCO phase setting. This is a 18-bit signed number representing the NCO phase. The value ranges from -180 to 180 degrees.
adcX_ZZ_nco_phase_rst	In	s_axi_aclk	NCO phase reset. Used to align the phases across the converter.
adcX_ZZ_update_enable[5:0]	In	s_axi_aclk	Enable register writes <ul style="list-style-type: none"> <li>Bit 5: Enable write to phase reset</li> <li>Bit 4: Enable write to NCO phase bits 17:16</li> <li>Bit 3: Enable write to NCO phase bits 15:0</li> <li>Bit 2: Enable write to NCO frequency bits 47:32</li> <li>Bit 1: Enable write to NCO frequency bits 31:16</li> <li>Bit 0: Enable write to NCO frequency bits 15:0</li> </ul>
adcX_update_req	In	s_axi_aclk	Asserted High to request an update of the NCO settings.
adcX_nco_update_busy	Out	s_axi_aclk	Update busy register. High when the NCO update is in progress.

**Notes:**

1. X refers to the location of the tile in the converter column. ZZ is either 01 (the lower RF-ADC in the tile) or 23 (the upper RF-ADC in the tile)



## Calibration Freeze Ports for Quad RF-ADC Tiles

Table 13: Calibration Freeze Ports for Quad RF-ADC Tiles

Port Name <sup>1</sup>	I/O	Clock	Description
adcXY_int_cal_freeze	In	s_axi_aclk	Signal from the PL to indicate that IP should freeze the calibration. This is typically asserted when the RF-ADC output is below a certain threshold.
adcXY_cal_frozen	Out	s_axi_aclk	Asserted when the calibration is frozen.

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the converter location in the tile (0 to 3).

## Calibration Freeze Ports for Dual RF-ADC Tiles

Table 14: Calibration Freeze Ports for Dual RF-ADC Tiles

Port Name <sup>1</sup>	I/O	Clock	Description
adcX_ZZ_int_cal_freeze	In	s_axi_aclk	Signal from the PL to indicate that IP should freeze the calibration. This is typically asserted when the RF-ADC output is below a certain threshold.
adcX_ZZ_cal_frozen	Out	s_axi_aclk	Asserted when the calibration is frozen.

**Notes:**

1. X refers to the location of the tile in the converter column. ZZ is either 01 (the lower RF-ADC in the tile) or 23 (the upper RF-ADC in the tile).

## Register Space

The address map, shown in the following table, is split on a per-tile basis. All banks are 16 KB. The first bank contains the functions common to all tiles. Each tile has a bank for control and status.

Table 15: Address Space

AXI4-Lite Address Range ADDR[17:0]	Function
0x00000 - 0x03FFF	IP Common Control and Status
0x04000 - 0x07FFF	RF-DAC Tile 0 registers (Tile <n> Registers)
0x08000 - 0x0BFFF	RF-DAC Tile 1 registers (Tile <n> Registers)
0x0C000 - 0x0FFFF	RF-DAC Tile 2 registers (Tile <n> Registers)
0x10000 - 0x13FFF	RF-DAC Tile 3 registers (Tile <n> Registers)
0x14000 - 0x17FFF	RF-ADC Tile 0 registers (Tile <n> Registers)
0x18000 - 0x1BFFF	RF-ADC Tile 1 registers (Tile <n> Registers)
0x1C000 - 0x1FFFF	RF-ADC Tile 2 registers (Tile <n> Registers)

Table 15: Address Space (cont'd)

AXI4-Lite Address Range ADDR[17:0]	Function
0x20000 - 0x23FFF	RF-ADC Tile 3 registers (Tile <n> Registers)
0x24000 - 0x3FFFF	Reserved

## IP Common Control and Status

Table 16: IP Common Control and Status

Address Range ADDR[13:0]	Function
0x0000	IP Versioning Information
0x0004	Master Reset Register
0x0100	Common Interrupt Status Register
0x0104	Common Interrupt Enable Register

## IP Version Information (0x0000)

Table 17: IP Version Information (0x0000)

Bit	Default Value	Access Type	Description
31:24	02	RO	Major
23:16	00		Minor
15:8	00		Revision
7:0	00		Reserved

## Master Reset Register (0x0004)

Table 18: Master Reset Register (0x0004)

Bit	Default Value	Access Type	Description
31:1	-	-	Reserved
0	0	R/W Auto Clear	Reset All Tiles. Write 1 to this bit to reset all logic in the core and restart the power-on sequence of all converters in the core. Each converter is configured as per the settings chosen during core generation. The AXI4-Lite registers are unaffected by this reset with the exception of bits 15:8 in the Restart State Register for each tile which is set to 0 automatically. The end state (bits 7:0) is not affected by this reset so the power-up sequence for each tile starts from state 0 and runs to the value programmed in bits 7:0 of the Restart State register for each tile (the default end state is 0x0F).

**Related Information**
[Restart State Register \(0x0008\)](#)
[XRFdc\\_Reset](#)

## Common Interrupt Status Register (0x0100)

Table 19: Common Interrupt Status Register (0x0100)

Bit	Default Value	Access Type	Description
31	0	RO, Clear on Read	AXI timeout interrupt
30:8	-	RO	Reserved (read 0)
7	0		RF-ADC Tile 3 interrupt
6			RF-ADC Tile 2 interrupt
5			RF-ADC Tile 1 interrupt
4			RF-ADC Tile 0 interrupt
3			RF-DAC Tile 3 interrupt
2	0		RF-DAC Tile 2 interrupt
1	0		RF-DAC Tile 1 interrupt
0	0		RF-DAC Tile 0 interrupt

**Related Information**
[Interrupt Handling](#)

## Common Interrupt Enable Register (0x0104)

Table 20: Common Interrupt Enable Register (0x0104)

Bit	Default Value	Access Type	Description
31	0	R/W	AXI timeout interrupt enable
30:8	-		Reserved
7	0		RF-ADC Tile 3 interrupt enable
6	0		RF-ADC Tile 2 interrupt enable
5	0		RF-ADC Tile 1 interrupt enable
4	0		RF-ADC Tile 0 interrupt enable
3	0		RF-DAC Tile 3 interrupt enable
2	0		RF-DAC Tile 2 interrupt enable
1	0		RF-DAC Tile 1 interrupt enable
0	0		RF-DAC Tile 0 interrupt enable

## Related Information

[Interrupt Handling](#)

[XRFdc\\_IntrEnable](#)

## Tile <n> Registers

Do not attempt to write to any tile specific registers while the power-on state machine is operating. To ensure the power-on state machine is not running prior to any register access, poll the Restart Power-On State Machine register (for tile <n>) and wait for it to read all zeros.

Table 21: Tile <n> Registers

ADDR[12:0]	Function
0x0000	Reserved
0x0004	Restart Power-On State Machine Register
0x0008	Restart State Register
0x000C	Current State Register
0x0010 - 0x0034	Reserved
0x0038	Reset Count Register
0x003C - 0x00FC	Reserved
0x0100	Post-Implementation Simulation Speedup Register
0x0104 - 0x01FC	Reserved
0x0200	Interrupt Status Register
0x0204	Interrupt Enable Register
0x0208	Converter 0 Interrupt Register
0x020C	Converter 0 Interrupt Enable Register
0x0210	Converter 1 Interrupt Register
0x0214	Converter 1 Interrupt Enable Register
0x0218	Converter 2 Interrupt Register <sup>1</sup>
0x021C	Converter 2 Interrupt Enable Register <sup>1</sup>
0x0220	Converter 3 Interrupt Register <sup>1</sup>
0x0224	Converter 3 Interrupt Enable Register <sup>1</sup>
0x0228	RF-DAC/RF-ADC Tile <n> Common Status Register
0x022C	Reserved
0x0230	RF-DAC/RF-ADC Tile <n> Disable Register
0x0234-0x3FFF	Reserved

### Notes:

1. Converter 2 and 3 registers are not applicable for Dual RF-ADC tiles.

## Restart Power-On State Machine Register (0x0004)

Table 22: Restart Power-On State Machine Register (0x0004)

Bit	Default Value	Access Type	Description
31:1	-	-	Reserved
0	0	R/W Auto Clear	Write 1 to this bit to start the power-on state machine. The state machine starts and stops at the stages programmed in the Restart State Register. This bit stays High until the state machine has reached the chosen end state.

### Related Information

[Restart State Register \(0x0008\)](#)

## Restart State Register (0x0008)

Table 23: Restart State Register (0x0008)

Bit	Default Value	Access Type	Description	
31:16	-	-	Reserved	
15:8	00	R/W	Start	Enabled tiles only. Start and End states for the power-on sequence. The default start state of 0x00 and end state of 0x0F should be used to enable the converters and a start state of 0x00 and an end state of 0x03 should be used to stop the converters. When a 1 is written to the bit in the Restart Power-on State Machine Register for tile<n>, the power-on state machine is started from the start state and runs to the end of the end state specified in this register. See Power-up Sequence for details about restarting and power-down.
7:0	0F	R/W	End	

### Related Information

[Power-up Sequence](#)

## Current State Register (0x000C)

Table 24: Current State Register (0x000C)

Bit	Default Value	Access Type	Description
31:8	-	-	Reserved
7:0	00	RO	Current state of Power-on state machine. See Power-on Sequence Steps.

### Related Information

[Power-on Sequence Steps](#)

[XRFdc\\_GetIPStatus](#)

## Reset Count Register (0x0038)

Table 25: Reset Count Register (0x0038)

Bit	Default Value	Access Type	Description
31:8	-	-	Reserved
7:0	00	RO Clear on Reset	If the IP automatically restarts due to instability in the supply power, a loss of clock integrity or a loss of PLL lock, the reset counter is incremented. The counter saturates at 255 resets.

## Post-Implementation Simulation Speedup Register (0x0100)

Table 26: Post-Implementation Simulation Speedup Register (0x0100)

Bit	Default Value	Access Type	Description
31:1	-	-	Reserved
0	0	RW	Simulation speed-up for post-implementation simulations. Set High to speed up post-implementation simulations. This register must not be set in hardware.

## Interrupt Status Register (0x0200)

Table 27: Interrupt Status Register (0x0200)

Bit	Default Value	Access Type	Description
31:5	-	-	Reserved (read back 0)
4	0	RO Clear On Read	Common interrupt bit. Asserted when the power-on state machine has encountered an error during the start up process
3	0	RO	Converter 3 interrupt bit
2	0		Converter 2 interrupt bit
1	0		Converter 1 interrupt bit
0	0		Converter 0 interrupt bit

### Related Information

[Interrupt Handling](#)

[XRFdc\\_IntrClr](#)

[XRFdc\\_GetIntrStatus](#)

## Interrupt Enable Register (0x0204)

Table 28: Interrupt Enable Register (0x0204)

Bit	Default Value	Access Type	Description
31:5	-	-	Reserved (read back 0)
4	1	R/W	Common interrupt enable bit
3	1		Converter 3 interrupt enable bit
2	1		Converter 2 interrupt enable bit
1	1		Converter 1 interrupt enable bit
0	1		Converter 0 interrupt enable bit

## Converter 0 Interrupt Register (0x0208)

Table 29: Converter 0 Interrupt Register (0x0208)

Bit	Default Value	Access Type	Description
31:16	-	-	Reserved Enable Read back 0
15	0	RO Clear on Reset	Flags a FIFO overflow in converter when High
14	0		Flags a datapath overflow in converter when High <sup>1</sup>
13:4	-	-	Reserved (read back 0)
3	0	RO Clear on Reset	Flags an Over Range interrupt in converter <sup>2</sup>
2	0	RO Clear on Read	Flags an Over Voltage interrupt in converter <sup>2</sup>
1:0	N/A		Reserved (read back 0)

**Notes:**

- A datapath overflow indicates one of the following conditions has occurred:
  - Interpolation filter overflow in the RF-DAC
  - Decimation filter overflow in the RF-ADC
  - Overflow in the Quadrature Modulation Correction block
  - Overflow in the RF-DAC Inverse Sinc filter
- RF-ADC only.

## Converter 0 Interrupt Enable Register (0x020C)

Table 30: Converter 0 Interrupt Enable Register (0x020C)

Bit	Default Value	Access Type	Description
31:16	-	-	Reserved (read back 0)

Table 30: Converter 0 Interrupt Enable Register (0x020C) (cont'd)

Bit	Default Value	Access Type	Description
15	1	R/W	Enable FIFO overflow interrupt in converter
14	1		Enable datapath overflow interrupt in converter
13:4	-	-	Reserved (read back 0)
3	1	R/W	Enable Over Range interrupt in converter <sup>1</sup>
2	1		Enable Over Voltage interrupt in converter <sup>1</sup>
1:0	-	-	Reserved (read back 0)

**Notes:**

1. RF-ADC only.

## Converter 1 Interrupt Register (0x0210)

Table 31: Converter 1 Interrupt Register (0x0210)

Bit	Default Value	Access Type	Description
31:16	-	-	Reserved (read back 0)
15	0	RO Clear on Reset	Flags a FIFO overflow in converter when High
14	0		Flags a datapath overflow in converter when High
13:4	-	-	Reserved (read back 0)
3	0	RO Clear on Reset	Flags an Over Range interrupt in converter <sup>1</sup>
2	0	RO Clear on Read	Flags an Over Voltage interrupt in converter <sup>1</sup>
1:0	-	-	Reserved (read back 0)

**Notes:**

1. RF-ADC only.

## Converter 1 Interrupt Enable Register (0x0214)

Table 32: Converter 1 Interrupt Enable Register (0x0214)

Bit	Default Value	Access Type	Description
31:16	-	-	Reserved (read back 0)
15	1	R/W	Enable FIFO overflow interrupt in converter
14	1		Enable datapath overflow interrupt in converter
13:4	-	-	Reserved (read back 0)
3	1	R/W	Enable Over Range interrupt in converter <sup>1</sup>
2	1		Enable Over Voltage interrupt in converter <sup>1</sup>



Table 32: Converter 1 Interrupt Enable Register (0x0214) (cont'd)

Bit	Default Value	Access Type	Description
1:0	-	-	Reserved (read back 0)

**Notes:**

- RF-ADC only.

## Converter 2 Interrupt Register (0x0218)

Table 33: Converter 2 Interrupt Register (0x0218)

Bit	Default Value	Access Type	Description
31:16	-	-	Reserved (read back 0)
15	0	RO Clear on Reset	Flags a FIFO overflow in converter when High
14	0		Flags a datapath overflow in converter when High
13:4	-	-	Reserved (read back 0)
3	0	RO Clear on Reset	Flags an Over Range interrupt in converter <sup>1</sup>
2	0	RO Clear on Read	Flags an Over Voltage interrupt in converter <sup>1</sup>
1:0	-	-	Reserved (read back 0)

**Notes:**

- RF-ADC only.

## Converter 2 Interrupt Enable Register (0x021C)

Table 34: Converter 2 Interrupt Enable Register (0x021C)

Bit	Default Value	Access Type	Description
31:16	-	-	Reserved (read back 0)
15	1	R/W	Enable FIFO overflow interrupt in converter
14	1		Enable datapath overflow interrupt in converter
13:4	-	-	Reserved (read back 0)
3	1	R/W	Enable Over Range interrupt in converter <sup>1</sup>
2	1		Enable Over Voltage interrupt in converter <sup>1</sup>
1:0	-	-	Reserved (read back 0)

**Notes:**

- RF-ADC only.

## Converter 3 Interrupt Register (0x0220)

Table 35: Converter 3 Interrupt Register (0x0220)

Bit	Default Value	Access Type	Description
31:16	-	-	Reserved (read back 0)
15	0	RO Clear on Reset	Flags a FIFO overflow in converter when High
14	0		Flags a datapath overflow in converter when High
13:4	-	-	Reserved (read back 0)
3	0	RO Clear on Reset	Flags an Over Range interrupt in converter <sup>1</sup>
2	0	RO Clear on Read	Flags an Over Voltage interrupt in converter <sup>1</sup>
1:0	-	-	Reserved (read back 0)

**Notes:**

1. RF-ADC only.

## Converter 3 Interrupt Enable Register (0x0224)

Table 36: Converter 3 Interrupt Enable Register (0x0224)

Bit	Default Value	Access Type	Description
31:16	-	-	Reserved (read back 0)
15	1	R/W	Enable FIFO overflow interrupt in Converter
14	1		Enable datapath overflow interrupt in Converter
13:4	-	-	Reserved (read back 0)
3	1	R/W	Enable Over Range interrupt in converter <sup>1</sup>
2	1		Enable Over Voltage interrupt in converter <sup>1</sup>
1:0	-	-	Reserved (read back 0)

**Notes:**

1. RF-ADC only.

## RF-DAC/RF-ADC Tile <n> Common Status Register (0x0228)

Table 37: RF-DAC/RF-ADC Tile <n> Common Status Register (0x0228)

Bit	Default Value	Access Type	Description <sup>1,2</sup>
31:4	-	RO	Reserved
3			PLL locked. Asserted when the tile PLL has achieved lock.
2			Power-up state. Asserted when the tile is in operation.
1			Supplies up. Asserted when the external supplies to the tile are stable.
0			Clock present. Asserted when the reference clock for the tile is present.

**Notes:**

1. <n> is 0 to 3.
2. See Register Space for register <n> address.

### Related Information

[Register Space](#)

[XRFdc\\_GetIPStatus](#)

[XRFdc\\_GetBlockStatus](#)

## RF-DAC/RF-ADC Tile <n> FIFO Disable Register (0x0230)

Table 38: RF-DAC/RF-ADC Tile <n> FIFO Disable Register (0x0230)

Bit	Default Value	Access Type	Description <sup>1,2</sup>
31:1	-	-	Reserved
0	0	R/W	Disable the interface FIFO for converter <n>

**Notes:**

1. <n> is 0 to 3.
2. See Register Space for register <n> address.

### Related Information

[Register Space](#)

# Designing with the Core

The RF Data Converter solution consists of the Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc RF Data Converter IP core configuration in the Vivado<sup>®</sup> Integrated Design Environment (IDE) and the RF data converter (RFdc) driver Application Programming Interface (API).

---

## IP Core Configuration in the Vivado Design Suite

The Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc RF Data Converter IP core configuration screen in the Vivado<sup>®</sup> IDE sets up the physical configuration of the RF-ADCs and RF-DACs in the Zynq UltraScale+ RFSoc. The configuration screen is used to enable tiles, configure decimation, interpolation, and mixing, set up converter sample rates, the Programmable Logic (PL) interface word widths and data types, and enables the optional interface ports. The IP core also handles the configuration and power-up of the data converters. This ensures that the settings specified in the Vivado IDE are applied to the RF-ADCs and RF-DACs immediately after the PL configuration completes.

---

## Software Driver

The RFdc driver API provides runtime interaction and monitoring of the data converters. This includes responding to interrupts, changing some settings, such as mixer frequency, and also interacting with the IP core to power up or down RF-ADC or RF-DAC tiles. The RFdc driver API is available as a bare-metal or Linux driver validated on both MicroBlaze<sup>™</sup> and Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc Processing System (PS) APU or RPU processors. Your application should always use these driver APIs to read status from or change the configuration of the IP core on the fly. Accessing internal registers directly is not allowed with the exception of some status/interrupt registers which have been identified in this document.

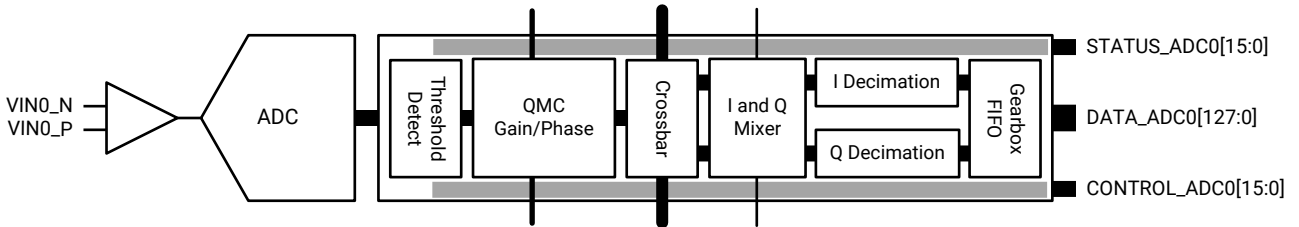
### Related Information

[Zynq UltraScale+ RFSoc RF Data Converter Bare-metal/Linux Driver](#)

# RF-ADC

Every RF-ADC in a tile has its own dedicated high-performance input buffer and includes features optimized for direct RF applications including quadrature modulator correction (QMC), full complex mixers, and decimation filters.

Figure 8: RF-ADC Functionality Block Diagram (Simplified)



X18254-091117

Certain functions can only be executed when the RF-ADCs in a tile are paired. The even-numbered RF-ADCs are used for I datapaths and the odd-numbered RF-ADCs are used for Q datapaths. All of the available built-in functionality of a tile and each of the RF-ADCs within a tile are configured with the supporting RFdc driver API and/or core configuration screen in the Vivado® IDE.

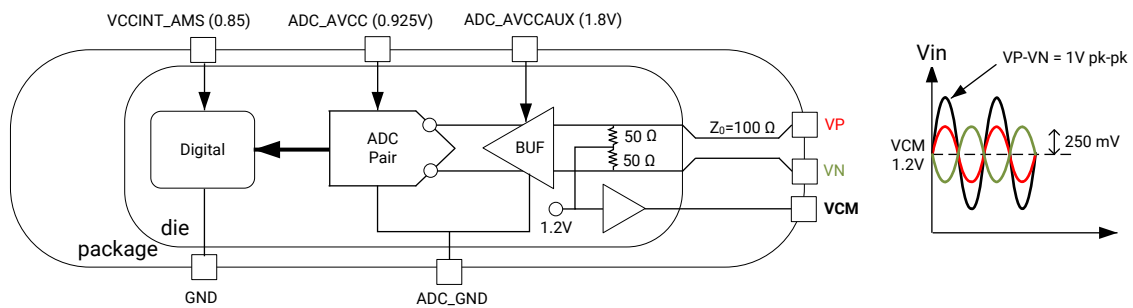
## Related Information

[Quadrature Modulator Correction](#)

## RF-ADC Analog Input

Every RF-ADC in a tile has its own differential analog input buffer. This input is optimized for performance and requires source impedance matching for best dynamic performance.

Figure 9: RF-ADC Analog Input



X19533-082819

**Note:** In the figure above the waveform diagram is an oscilloscope representation with a DC offset.

There are several ways to drive an RF-ADC in a tile. Driving the RF-ADC can be either active or passive. However, optimum performance is achieved by driving the analog input differentially. Using a differential amplifier, AC- or DC-coupled, to drive the RF-ADC provides a flexible interface with excellent performance. For AC-coupled mode, the input signals should be AC-coupled in using capacitors. For DC-coupled mode, the output VCM buffer is enabled as shown in the figure above. This buffer is only enabled when DC-coupled mode is selected. This VCM buffer allows the user to align the common mode of the external active driving circuit with the ADC internal common requirements. Two VCM buffers (VCM01, VCM23) are available for each tile.

For all applications where the signal-to-noise ratio (SNR) is a key parameter, Xilinx recommends using a differential transformer or balun configuration. See the *UltraScale Architecture PCB Design User Guide* ([UG583](#)) for details on how to design the input networks and PCB.

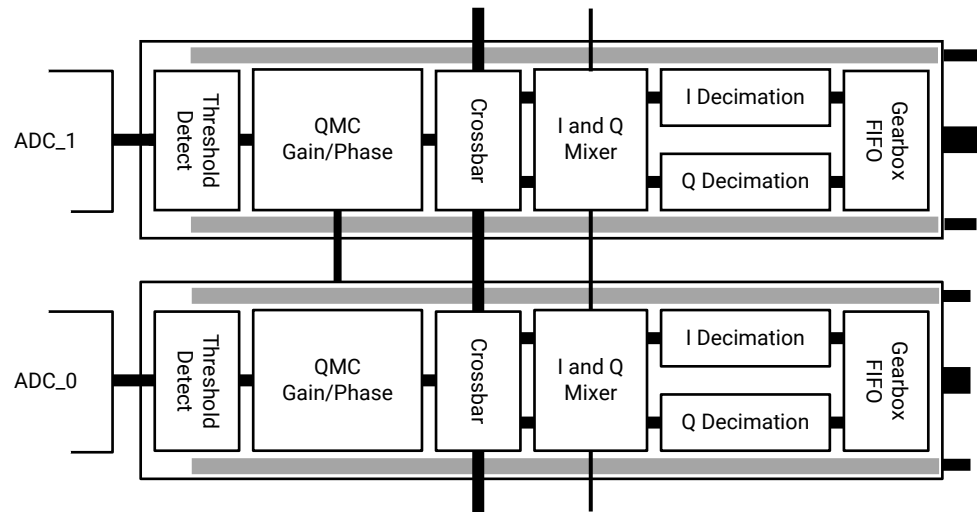
## RF-ADC Digital Datapath

An RF-ADC component in a tile has integrated DSP features which can be enabled by the user to pre-process the sampled data from the RF-ADC device before it is passed to the PL. The different DSP function blocks are as follows:

- **Detection functionality:** Contains a dual level programmable threshold that provides two flags to the internal interconnect logic, and is asserted when the absolute output value of the RF-ADC is greater or smaller than the programmed threshold values.
- **Compensation functionality:** Contains a quadrature modulator correction (QMC) block with a coarse delay adjustment block
- **Digital Down Converter (DDC):** consists of
  - Mixer—coarse (quarter and half rate) and fine (NCO with 48-bit frequency resolution)
  - Signal decimation functionality—decimation by 1 (bypass), 2, 4, or 8 is supported

Single, multiple, or all DSP functions can be used or bypassed. Some functions, such as the QMC, require activation of the same function in both I and Q RF-ADCs. Even numbered RF-ADCs are always used for I datapaths, and odd numbered RF-ADCs are used for Q datapaths. You can implement, configure, or modify the functionality of one or multiple functions using the IP core. The following figure shows the available functions in an RF-ADC and the functions are described in this section.

Figure 10: Signal Treatment of the RF-ADC Peripherals



X18259-091117

## RF-ADC Threshold and Over Range Settings

As with any ADC, the input analog signals must be kept within the full-scale range of the ADC, and at the correct input levels. Any signals that do not comply with these conditions result in data loss. To help prevent this, the threshold-detector feature can be used to adjust signal chain gains to keep the signal within the ideal full-scale ranges. However, in the event that a signal does exceed the full-scale range, each RF-ADC channel has built-in detection and protection for this with the Over Range and Over Voltage features.

The Over Voltage and Over Range signals are provided to both the interrupt feature on the IP core, as well as to the RF-ADC real-time signals bus on the IP core for direct access to the PL design.

### Over Range

An Over Range condition occurs when a signal exceeds the full-scale input of the RF-ADC. When this condition is detected, the converted data is saturated (clipped) to limit the data corruption, and the signal is flagged to the user by both the interrupt mechanism, and by asserting the Over Range real-time output signal. Because Over Range events can be as short as one RF-ADC sample, the output signal is sticky. To clear the Over Range output and the associated interrupt, the API interrupt handling mechanism is used.

### Related Information

[Interrupt Hierarchy](#)

[Real-Time Signal Interface Ports for Quad RF-ADCs](#)

[Real-Time Signal Interface Ports for Dual RF-ADCs](#)

## Over Voltage

An Over Voltage condition occurs when a signal far exceeds the normal operating input range. Because an excessive voltage on the inputs can damage the device, automatic protection mechanisms are provided.

An Over Voltage event results in the automatic shutdown of the input buffer to protect it. The Over Voltage circuit monitors each of the signals of the differential inputs independently, and flags the condition when any individual input signal exceeds the maximum input voltage or is less than the minimum input voltage of the RF-ADC input buffer.

The Over Voltage feature offers protection for signals in the range defined in the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)*. Signals exceeding this maximum are not permitted, and care must be taken externally to ensure that such voltages are not presented to the RF-ADC inputs.

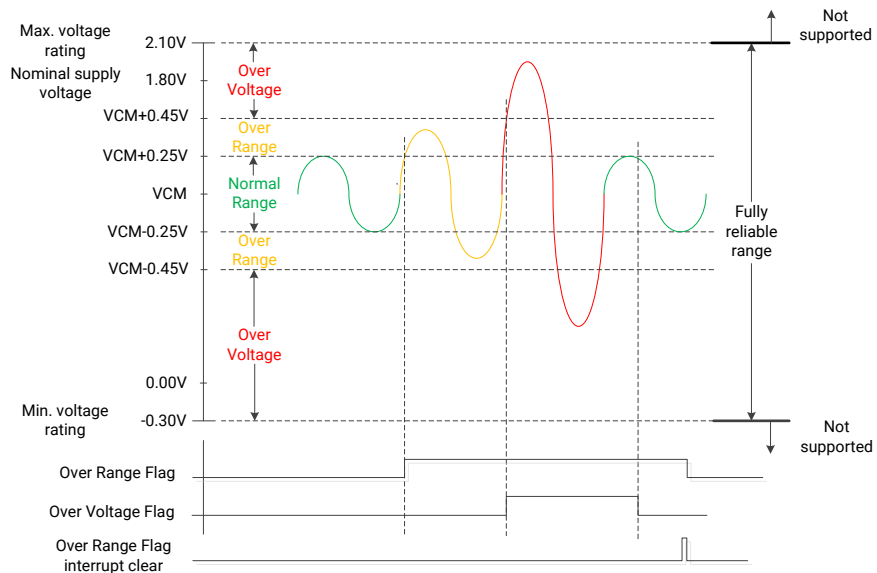
When an Over Voltage condition is detected the signal is flagged to the user by both the interrupt mechanism, and by asserting the Over Voltage real-time output signal. The Over Voltage real-time output is asserted and deasserted asynchronously, and provides immediate notification of the event. As a result, the Over Voltage output self-clears when the Over Voltage condition is no longer present. The associated interrupt is sticky, so requires clearing by the API interrupt handling routines.

After an Over Voltage event the input buffer automatically re-enables and the RF-ADC resumes operation as before. However, an over voltage condition should be considered as grounds to stop the traffic and restart the receiver chain. In the event of an Over Voltage condition, the buffer shuts down and the resulting RF-ADC output digital codes are mostly just noise. As a result, the threshold information become irrelevant and this can affect an AGC implementation. Therefore it is important that an AGC implementation takes account of the Over Voltage signal and sets up the two threshold detectors accordingly.

The following figure shows the Threshold, Over Range, and Over Voltage levels and the response of these with an increasing input analog signal.



Figure 11: Threshold, Over Range, and Over Voltage Levels



X20472-050919

**Related Information**

- [Interrupt Hierarchy](#)
- [Real-Time Signal Interface Ports for Quad RF-ADCs](#)
- [Real-Time Signal Interface Ports for Dual RF-ADCs](#)

**Threshold Settings**

Instead of waiting for a signal to propagate through the signal processing blocks, the threshold feature provides an early indication of the incoming signal level. This early indication of the signal level can be used by the automatic gain control (AGC) implemented in the PL. Threshold levels used to indicate the input signal level are set using the RFdc driver API.

Threshold monitoring occurs when the RF-ADC sampled data enters the datapath. This data is compared to a user-defined threshold. A threshold status signal is sent to the outputs on the IP core, to indicate that a user-defined threshold has been exceeded. There are two real-time over threshold output signals for each RF-ADC. The modes of the threshold monitoring circuit are listed in the following table.

Table 39: Threshold Signaling Modes

Mode	Description
Off	The threshold circuit is disabled and the status outputs are Low.
Sticky over	The threshold status signal is High when the data from the RF-ADC exceeds the programmed upper threshold value. The status is kept until a clear action is sent. <sup>1</sup>

Table 39: Threshold Signaling Modes (cont'd)

Mode	Description
Sticky under	The threshold status signal is High when the data from the RF-ADC remains below the programmed lower threshold value for the duration of a user-specified time or delay. The status is kept until a clear action is sent <sup>1</sup> . This delay value for the lower threshold is defined by a 32-bit counter. The counter is set using the RFdc driver API. Using this mechanism prevents short duration excursions triggering a threshold event.
Hysteresis	The status output is set when the programmed upper threshold value is exceeded, and is cleared when the signal remains below the lower threshold value for the duration of a user-specified delay value. This delay value for the lower threshold is defined by a 32-bit counter. The counter is set using the RFdc driver API. The delay adds hysteresis to the threshold detection to prevent short duration excursions triggering a threshold event.

**Notes:**

1. To clear see Clearing Threshold Flags.

Threshold levels are set as 14-bit unsigned values, with any value from 0 to 16383 allowed. The maximum value, 16383 represents the absolute value of the full-scale input of the RF-ADC. The 32-bit programmable delay counts RF-ADC samples. To relate this count to a specific time, the following equation can be used:

$$\text{Counter Value} = (\text{delay}(\text{ms}) \times \text{ADC}_{\text{SampleRate}} (\text{GSPS}) \times 10^6) / \text{Interleaving Factor}$$

where the Interleaving Factor is 4 for the Quad RF-ADC tile and is 8 for the Dual RF-ADC tile.

The delay can also be expressed in samples instead of ms as below:

$$\text{Counter Value} = \text{delay (RF-ADC samples)} / \text{Interleaving Factor} = \text{delay (sub-ADC samples)}$$

**Related Information**

[Real-Time Signal Interface Ports for Quad RF-ADCs](#)

[Real-Time Signal Interface Ports for Dual RF-ADCs](#)

**Clearing Threshold Flags**

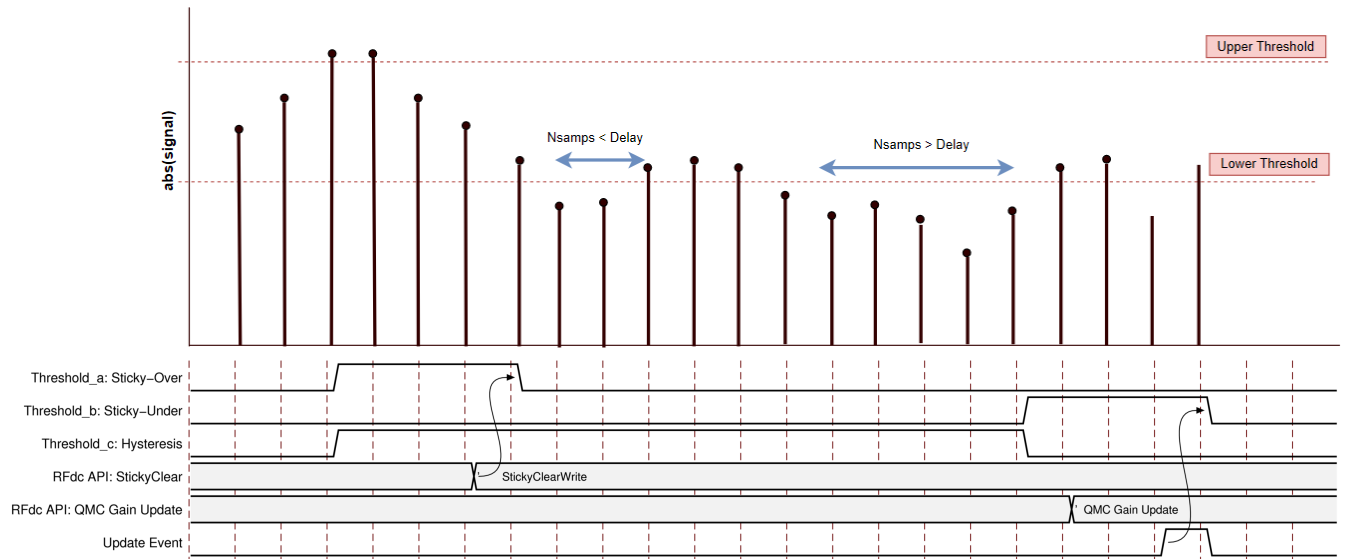
After the threshold values assert in the Sticky Over and Sticky Under modes, the flags can be cleared in two ways; by issuing the clear command directly, or by using the AutoClear feature. The AutoClear mode clears the threshold whenever the QMC Gain value associated with that RF-ADC is updated.

**Threshold Operation Example**

The following figure illustrates the threshold unit operation. The diagram shows three thresholds, a, b, and c, each configured in a different threshold mode. In hardware all threshold units have independent threshold levels and delay values. In this example the threshold levels are indicated by the dashed horizontal lines, and all units use the same levels for the purpose of illustration. The delay value is set to 5 (the delay value has no effects for Sticky Over mode), and the sticky thresholds have been set up to be cleared by different clear modes. The behavior is described in the following sections.

**Note:** In the following figure, samples are counted by one sub-ADC, but not RF-ADC.

**Figure 12: Threshold Operation**



X20238-012219

### Threshold-A, Sticky Over Mode

The flag asserts when the first sample exceeds the upper threshold value. In this example, the flag is cleared using a call to the `XRFdc_ThresholdStickyClear` API function.

### Threshold-B, Sticky Under Mode

The flag asserts when the samples have been continuously below the lower threshold for the delay number of samples (6, in the example). In this example, the flag is cleared through an update of the QMC Gain value, which is issued using a call to the `XRFdc_SetQMCSettings` API function, followed by an Update event, which applies the QMC gain update.

### Threshold-C, Hysteresis Mode

The flag asserts when the first sample exceeds the upper threshold value. The flag is cleared when the samples have been continuously below the lower threshold for the delay number of samples.

### Related Information

[Dynamic Update Events](#)

## Threshold RFdc Driver API Commands

The threshold levels and delay values are configured using the RFdc driver API.

```
// Initial Setup

XRFdc_Threshold_Settings Threshold_Settings;
Threshold_Settings.UpdateThreshold = XRFDC_UPDATE_THRESHOLD_BOTH; // Setup
values
for threshold 0 and 1
Threshold_Settings.ThresholdMode[0] = XRFDC_TRSHD_STICKY_UNDER; // Set
threshold0
mode to Sticky Under
Threshold_Settings.ThresholdUnderVal[0] = 1000; // Measured in 14-bit
unsigned LSBs
Threshold_Settings.ThresholdAvgVal[0] = 10; // Data must be below lower
threshold for
10*8 4 GSPS RF-ADC samples

// Write threshold values to the selected Tile / RF-ADC

XRFdc_SetThresholdSettings(ptr, Tile, Block, &Threshold_Settings);
```

The threshold clear operation is shown in the following code examples.

- Clear the thresholds by writing directly using the RFdc driver API:

```
// Initial Setup

XRFdc_SetThresholdClrMode(ptr, Tile, Block, Threshold#,
XRFDC_THRESHOLD_CLRMD_MANUAL_CLR);
...
// During application run-time (after a threshold asserts)

XRFdc_ThresholdStickyClear(ptr, Tile, Block, Threshold#);
```

- Clear the thresholds using the AutoClear function; the threshold clears with the QMC Gain Update:

```
// Initial Setup

XRFdc_QMC_Settings QMC_Settings;
XRFdc_SetThresholdClrMode(ptr, Tile, Block, Threshold#,
XRFDC_THRESHOLD_CLRMD_AUTO_CLR);
...
// During application run-time (after a threshold asserts - update t)

QMC_Settings.GainCorrectionFactor = new_gain_value;
XRFdc_SetQMCSettings(ptr, XRFDC_ADC_TILE, Tile, Block, &QMC_Settings);
...
// QMC Gain applied by an Update Event
```

**Related Information**[XRFdc\\_SetThresholdSettings](#)[XRFdc\\_SetThresholdClrMode](#)[XRFdc\\_ThresholdStickyClear](#)[XRFdc\\_SetQMCSettings](#)**Threshold Applications**

A common use for threshold detectors is in Automatic Gain Control (AGC) applications.

**Related Information**[Automatic Gain Control Systems](#)**Over Range Settings**

There are two types of Over Range signals available as status outputs to the programmable logic, Over Voltage and Over Range. The IP core exposes these signals using the interrupt mechanism, so any violation is immediately flagged to the user application. These are triggered when normal or expected operating levels are exceeded.

**Over Voltage Signal**

An Over Voltage signal is detected whenever the input signal exceeds a safe input range from the RF-ADC input buffers.

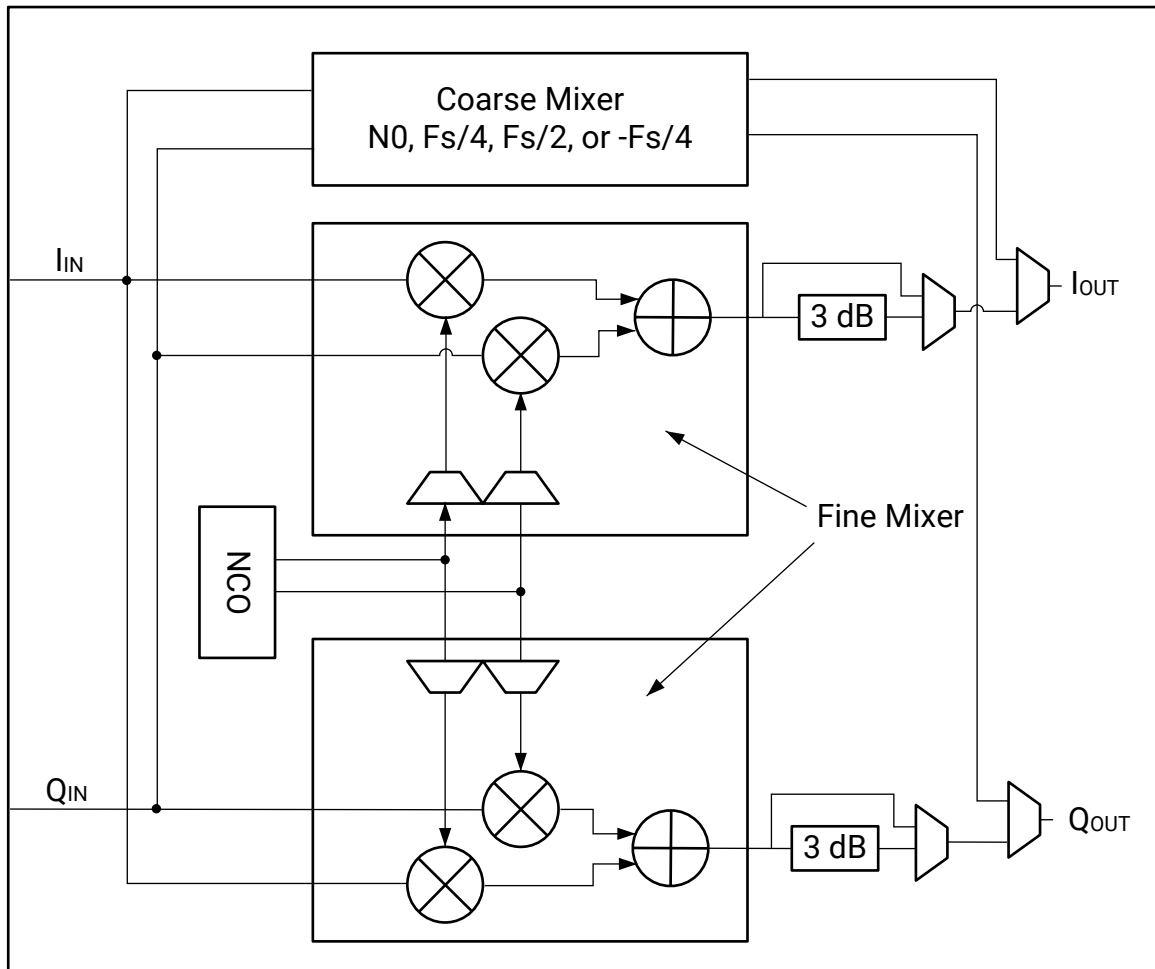
**Over Range Signal**

- When the input signal exceeds the  $\pm$  digital full-scale range of an RF-ADC, an Over Range signal is detected.
- An Over Range signal is measured at the raw digital output of the RF-ADC.

***RF-ADC Mixer with Numerical Controlled Oscillator***

The mixer function has three modes: bypass (no mixing), coarse mixing or fine mixing. Fine mixing automatically enables the NCO which is used to generate the carrier frequency. The mixer supports full quadrature mixing, with both real to I/Q and I/Q to I/Q modes supported.

Figure 13: RF-ADC Mixer with NCO DSP Block



X18261-032318

#### Coarse Mixer:

- The coarse mixer allows the data to be mixed with a carrier of 0,  $F_s/2$ ,  $F_s/4$ , or  $-F_s/4$ .

**Note:** The selection of 0 is only available using the RFdc driver API.

- Mixing with a 0 carrier bypasses the mixer component.

#### Fine Mixer:

- The fine mixer allows the data to be shifted up or down in frequency by an arbitrary amount.
- The frequency shift amount is obtained by programming the mixer frequency generated in the NCO. The fine mixer also supports 18-bit phase adjustment.
- The NCO phase can be synchronized within a tile using `XRFdc_UpdateEvent`.
- The NCO phase can be synchronized across tiles using an external event signal (SYSREF or MARKER).

- To manage potential overflow, the fine mixer output includes 3 dBV attenuation, as shown in the figure above. This attenuation is not relevant in R2C mode, so the automatic mode selection from the API selects the correct attenuation level following the RF-ADC mixer scaling output factor (see the table below). A manual selection is also possible, allowing 0 dBV or -3 dBV.

Table 40: RF-ADC Mixer Scaling Output Factor

Tile Usage	Coarse Mixer	Auto Fine Mixer
IQ (C2C)	1 (0 dBV)	0.707 (-3 dBV)
Real (R2C)	1 (0 dBV)	0.997 (-0 dBV)

The mixer settings can be configured in the core, or by using the RFdc driver API. The core is used to set the initial mixer settings (for example, mixer type and mixer mode), and the RFdc driver API is used to adjust the settings at runtime. Both the RFdc driver API and the core compute the required register settings based on the supplied sample rates and desired frequencies. A sample configuration screen is shown in the following figure. See the RF-ADC Converter Configuration section for information on the settings.

Figure 14: RF-ADC Mixer Settings Configuration

**ADC 0**

Enable ADC     Invert Q Output  
 Dither

**Data Settings**

Digital Output Data: I/Q

Decimation Mode: 1x

Samples per AXI4-Stream Word: 8

Required AXI4-Stream clock: 250.000 MHz

**Mixer Settings**

Mixer Type: Fine

Mixer Mode: Real->I/Q

NCO Frequency (GHz): 0.0

NCO Phase: 0

**Analog Settings**

Nyquist Zone: Zone 1

Calibration Mode: Mode2

### Related Information

[XRFdc\\_UpdateEvent](#)

[RF-ADC Converter Configuration](#)

## RF-ADC Mixer RFdc API Example

Related RFdc driver API functions are shown in the following code. This code illustrates the use of the NCO Phase reset function. This function must be used at startup to initialize the phase of the fine mixer to a valid state. Note that the following code resets the NCOs in all tiles.

```
XRFdc_Mixer_Settings Mixer_Settings;

for(tile=0;tile<4; tile++) {
    // Make sure the mixer settings update use the Tile event
    for(block=0; block<2; block++) {
        XRFdc_GetMixerSettings (ptr, XRFDC_ADC_TILE, tile, block,
        &Mixer_Settings);
        Mixer_Settings.EventSource = XRFDC_EVNT_SRC_TILE; //Mixer Settings
        are updated
        XRFdc_SetMixerSettings (ptr, XRFDC_ADC_TILE, tile, block,
        &Mixer_Settings);
    }

    // Reset NCO phase of both DDCs in Tile0 (assuming both are active)
    XRFdc_ResetNCOPhase(ptr, XRFDC_ADC_TILE, tile, 0); // DDC Block0
    XRFdc_ResetNCOPhase(ptr, XRFDC_ADC_TILE, tile, 1); // DDC Block1

    XRFds_UpdateEvent(ptr, XRFDC_ADC_TILE, tile, 1, XRFDC_EVENT_MIXER); //
    Generate a Tile Event
}
```

### Related Information

[XRFdc\\_GetMixerSettings](#)

[XRFdc\\_SetMixerSettings](#)

[XRFdc\\_ResetNCOPhase](#)

## NCO Setting Example

The NCO is in the digital domain and its effective frequency range is always from  $-F_s/2$  to  $F_s/2$ .

When the RF-ADC or the RF-DAC operates at sub-sampling frequencies ( $F_c > F_s/2$ ), the application must first calculate the signal location at the 1st Nyquist band and then set the effective NCO value to shift signals.

Xilinx defines the Zynq UltraScale+ RFSoc NCO frequency range from -10 GHz to 10 GHz in the API and in the IP configuration for convenience, so the application does not need to calculate the frequency in the 1st Nyquist band, to set any required NCO value in -10 GHz to 10 GHz, the API or the Vivado® Design Suite converts it to an effective NCO configuration automatically. This works for both RF-ADCs and RF-DACs.



In the digital domain, the spectrum shows inverse in adjacent Nyquist bands, set NCO frequency as positive or negative to pick up particular spectrum base on applications.

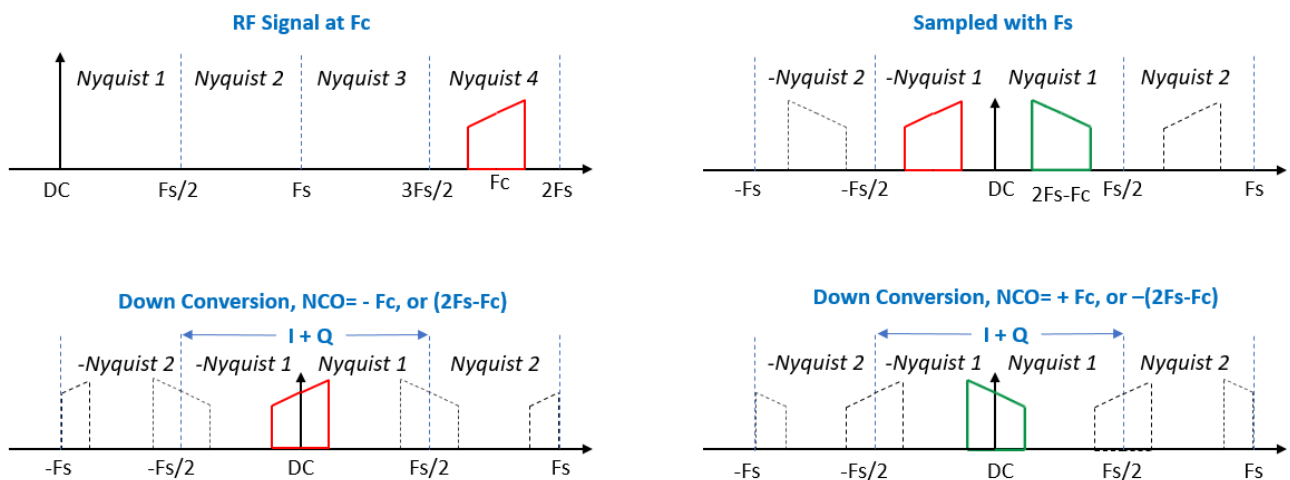
For example, the DDC, in general, shifts the carrier from  $F_c$  to DC, so with this absolute NCO range (-10 GHz to 10 GHz), setting the NCO to  $-F_c$  always shifts the desired signal at DC; setting the NCO to  $F_c$  always shifts the inverse signal to DC, no matter which Nyquist band the signal is located in.

When setting the NCO within the effective NCO range ( $-F_s/2$  to  $F_s/2$ ), to convert the carrier to DC, the NCO frequency should be positive when the signal is at even Nyquist bands, and negative when the signal is at odd Nyquist bands.

The DUC works similarly to the DDC but with a reverse signal flow direction. Setting the NCO to  $F_c$  always up-converts the signal from DC to  $F_c$ , and setting the NCO to  $-F_c$  always gets an inverse signal at  $F_c$ , no matter in which Nyquist band the carrier is located.

The following figure shows an example of down conversion.

Figure 15: Down Conversion Example



## RF-ADC Decimation Filters

Decimation filters are required to implement the down-sampling and filtering part of the digital-down conversion (DDC) process. The overall filter response is determined by the number of decimation stages used. The decimation chain consists of three FIR filter stages which can be combined to implement variable decimation rates. When a FIR stage is not used it is automatically powered down. The decimation filters allow for the creation of the following:

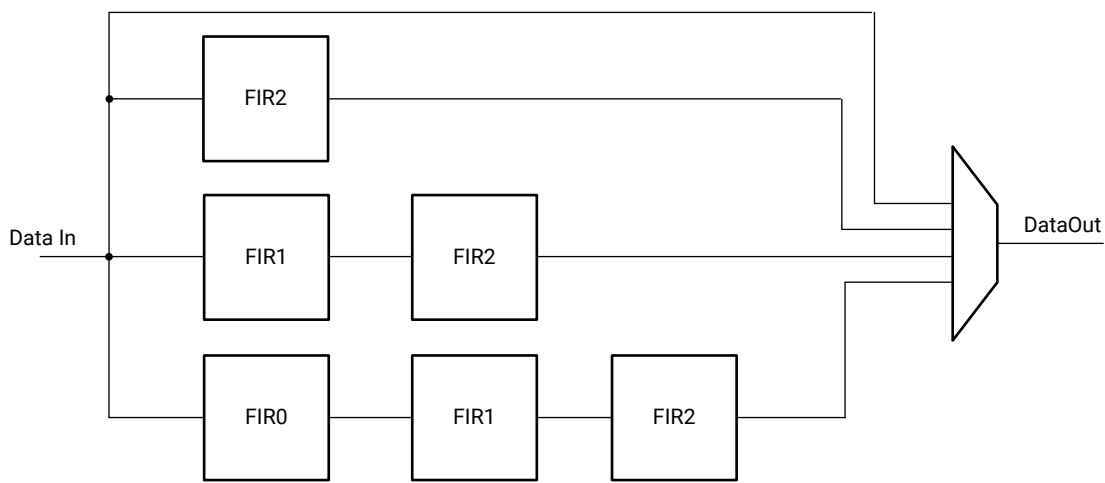
- **1x:** All filter stages are bypassed
- **2x:** Decimation filtering using a single stage

- **4x:** Decimation filtering using two stages
- **8x:** Decimation filtering using all three available stages

Each decimation filter element has a different number of taps and the stop-band attenuation and ripple are shown in Decimation Filter Details. The decimation filter chains can operate on either I/Q data or real data. Unused filter chains are powered down.

Each of the filter stages can overflow given the step-response of a FIR filter, especially when full-scale data is on the input. To detect and protect the datapath from overflow, each filter stage and sub-phase has a signed overflow status signal and saturation at the output. When a filter stage is not used, the flag is forced zero. These flags are connected to the datapath interrupt mechanism which is described in Interrupt Handling. The multiplexer in the following figure shows the decimation level selected in the IP configuration with the corresponding selection of decimation filter blocks.

Figure 16: RF-ADC Digital Down Converter



X21610-050919

Table 41: Decimation Filter Operating Modes

Mode	Description
<b>Quad and Dual RF-ADC Tile</b>	
OFF	The entire filter is disabled/powered down (applies when RF-ADC is disabled)
1x	The entire filter is bypassed
2x	2x decimation, 80% Nyquist passband <sup>1</sup>
4x	4x decimation, 80% Nyquist passband
8x	8x decimation, 80% Nyquist passband

**Notes:**

1. 80% Nyquist passband is  $0.4 \cdot F_s$

## Related Information

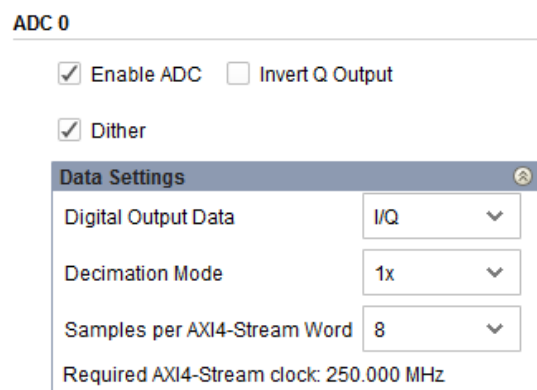
[Interrupt Hierarchy](#)

[Decimation Filter Details](#)

## Decimation Filter Use

The IP core is used to set the decimation rate. This is set in the Vivado® IDE because changing the decimation rate directly affects the physical interface as the bandwidth to the PL changes. The filters that are enabled are as shown in the following figure. Enable the RF-ADC by checking the **Enable ADC** check box. See RF-ADC Converter Configuration for information on the settings.

Figure 17: RF-ADC Decimation Filter Configuration



## Related Information

[RF-ADC Converter Configuration](#)

## Related API Commands

The RFdc driver API can be used to get the decimation rate set in the IP core using the following code.

```
// Get Decimation factor for Tile0, DDC Block1

int Tile = 0;
u32 Block = 1;
u32 Decimation_Factor;
if( XRFdc_GetDecimationFactor (ptr, Tile, Block, &DecimationFactor) ==
XST_SUCCESS) {
    xil_printf("ADC Tile%1d,%1d Decimation Factor is: %d", Tile, Block,
Decimation_Factor);
}
```

## Decimation Filter Details

The decimation filter chain consists of three FIR filters: FIR2, FIR1, and FIR0, which can be enabled to give successive decimation by a factor of two per stage. The filter transfer functions are as shown in the following figures.

Figure 18: FIR2 and FIR1 Frequency Response

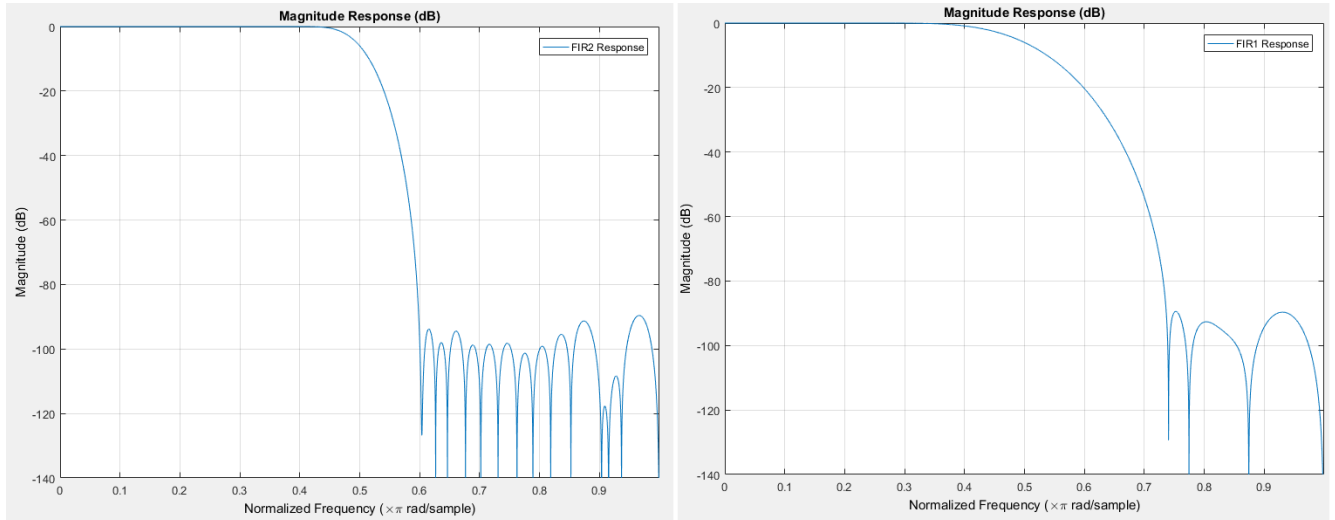
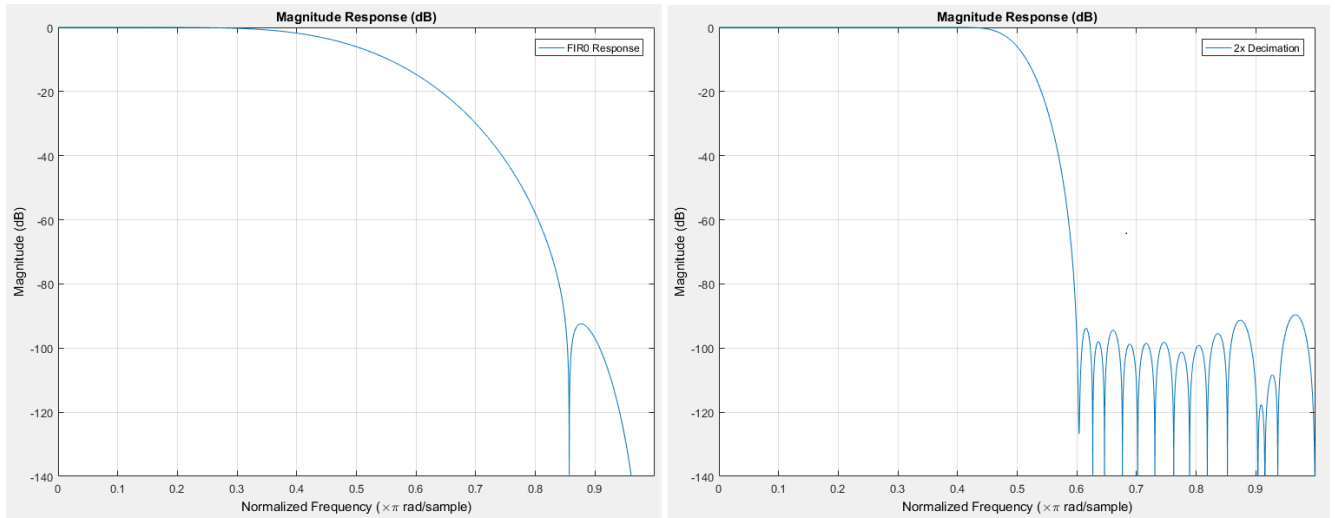
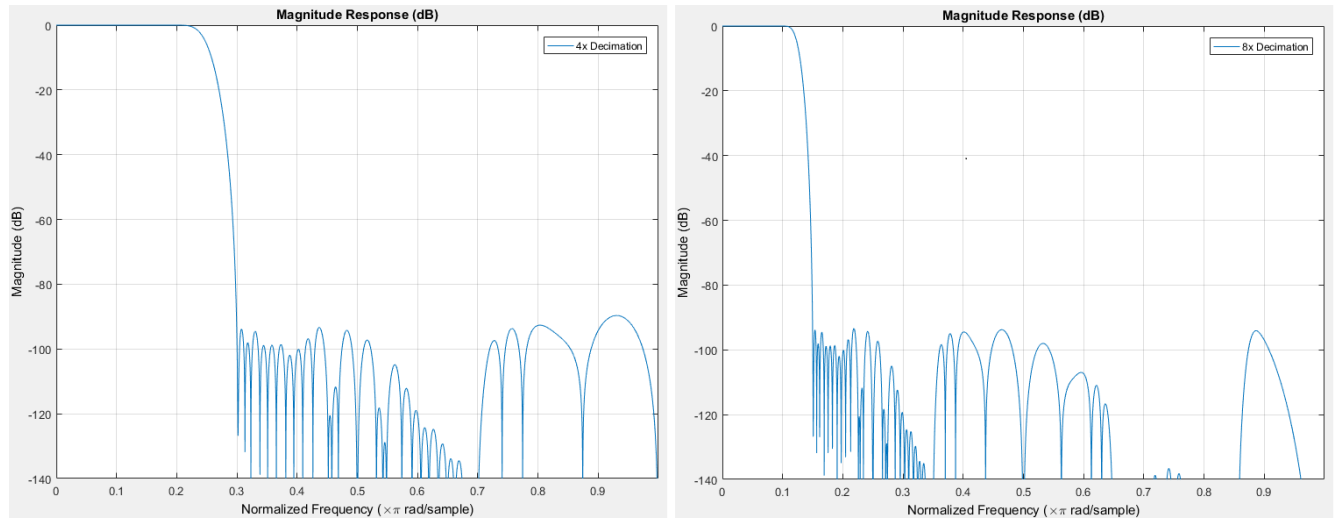


Figure 19: FIR0 and 2x Decimation Frequency Response



**Figure 20: 4x Decimation and 8x Decimation Frequency Response**


The filter coefficients for the decimation filters are shown in the following tables.

FIR2	1st Half	5	0	-17	0	44	0	-96	0	187	0	-335	0	565	0	-906
		0	1401	0	-2112	0	3145	0	-4723	0	7415	0	-13331	0	41526	
	Center Tap	65536														
	2nd Half	5	0	-17	0	44	0	-96	0	187	0	-335	0	565	0	-906
0		1401	0	-2112	0	3145	0	-4723	0	7415	0	-13331	0	41526		

FIR1	1st Half	-12	0	84	0	-337	0	1008	0	-2693	0	10142	
	Center Tap	16384											
	2nd Half	-12	0	84	0	-337	0	1008	0	-2693	0	10142	

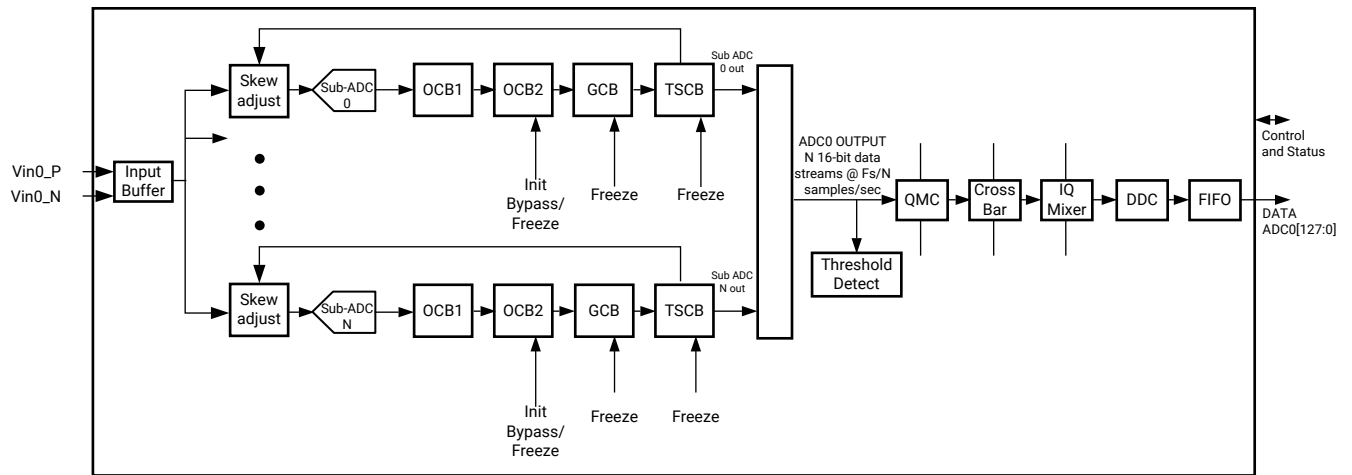
FIR0	1st Half	-6	0	54	0	-254	0	1230
	Center Tap	2048						
	2nd Half	-6	0	54	0	-254	0	1230

## RF-ADC Calibration Mechanism

Each of the RF-ADCs in the Zynq UltraScale+ RFSoc is built on multiple sub-RF-ADCs in an interleaving architecture. The nature of the interleaving process requires that an intricate calibration algorithm to be carried out to obtain the best dynamic range performance from the RF-ADC. Each of the RF-ADCs (Dual or Quad variant) in the RFSoc has a built-in calibration process that includes a foreground calibration (FG CAL) step and a background calibration (BG

CAL) step. The FG CAL step gets executed during the RF-ADC power-on state machine (startup initialization) only. The BG CAL step is a process that is designed to operate optionally during the RF-ADC run-time. Both of the calibration steps for each RF-ADC is carried out in parallel and independently. The following figure shows the block diagram of one of the RF-ADCs with the internal calibration blocks for each of the N sub-RF-ADCs.

Figure 21: RF-ADC Calibration Block Diagram



The calibration sub-system comprises of three major blocks that are designed to estimate and correct the various imperfections and mismatches of the interleaving sub-RF-ADCs.

### Time Interleaved Offset Calibration Block (OCB)

This block corrects for the DC offset of each of the sub-RF-ADCs. Any residual DC offset not corrected appears as a spur at  $k \cdot F_s / N$ , where  $F_s$  is the composite sampling rate of the RF-ADC,  $N$  is the number of sub-RF-ADCs interleaved together, and  $k = 0, 1, 2, \dots, N$ .  $N$  is 8 for the Dual RF-ADC tile variant and  $N$  is 4 for the Quad RF-ADC tile variant. This block is further divided down to two sub-blocks called OCB1 and OCB2. OCB2 only runs during FG CAL while OCB1 runs during both FG and BG CAL. The OCB2 block cancels offsets of the sub RF-ADCs at start-up. The OCB1 block cancels sub-RF-ADC offsets continuously in the BG calibration process without affecting the input signal content at the offset locations.

### Gain Calibration Block (GCB)

This block corrects for the gain difference among the interleaving sub-RF-ADCs. Any residual difference results in spurious signal at  $\pm f_{in} + (k/N) \cdot F_s$ , where  $F_s$  is the sample rate of the RF-ADC,  $N$  is the number of sub-RF-ADCs, and  $f_{in}$  is the frequency of the input signal. The GCB requires the presence of an input signal with power greater than -40 dBFs for an accurate correction of the gain skews. Signal bins at  $f = k \cdot F_s / (2N)$  are ignored and do not count towards the signal power. This block should be in the freeze mode whenever the input signal power drops below -40 dBFs for longer than 100  $\mu$ s.

## ***Time Skew Calibration Block (TSCB)***

This block corrects for the time skews among the interleaving sub-RF-ADCs. Any residual difference can result in spurious signal at  $\pm f_{in} + (k/N)F_s$ , where  $F_s$  is the sample rate of the RF-ADC,  $N$  is the number of sub-RF-ADCs,  $f_{in}$  is the frequency of the input signal. The TSCB also relies on the presence of an input signal with input power greater than -40 dBFs to provide accurate estimate and correction of the time skews among sub RF-ADCs. This block should be in the freeze mode whenever the input signal power drops below -40 dBFs for longer than 100  $\mu$ s.

## ***Foreground Calibration Process***

The foreground calibration step is part of the RF-ADC startup initialization state machine sequence. This is managed and carried out by the startup IP. The purpose of the foreground calibration is to provide correction through the OCB1 and OCB2 blocks for sub-RF-ADC offsets and skews of the sampling switches. The OCB2 block is then frozen and not updated again during operation mode. During this process there should ideally be no signal energy at any of the interleaving offset locations, namely frequency =  $kF_s/N$ . Thus the input should be muted during this process. This is typically accomplished by sequencing the receiver front end enable after the RF-ADC startup process is complete. At the end of the foreground calibration process the OCB2 block is frozen, and OCB1, Time Skew and Gain Calibration blocks are set to enable to run in the background.

## ***Background Calibration Process***

At the end of a successful startup initialization of the RF-ADC the background calibration is enabled. The purpose of the background calibration is to provide real time adjustment for the various skews and mismatches introduced by environmental changes to the sub-RF-ADCs, primarily that of temperature. The blocks that are running in the background (in real time) include OCB1, GCB, and TSCB. As noted, OCB1 runs in the background to adjust for residual sub-RF-ADC offset levels introduced by temperature change without affecting the input signal content. Signal content present at  $kF_s/N$  is unaffected by this block.

The gain calibration block (GCB) and the time skew calibration block (TSCB) provide correction to the sub-RF-ADC gain and time skews respectively. Both blocks use and depend on the input signal to estimate and correct for the skews. When the input signal drops below -40 dBFs it is no longer effective to use the signal for the computation. Thus the optimal usage of these two blocks is to dynamically control the coefficient update (freeze and resume) of these two blocks. The control of these two blocks for freezing and resuming is implemented using the Vivado® IP port RF-ADC calibration freeze ports in the ADVANCE control panel. The CAL freeze port is independently available for each RF-ADC and includes a freeze control signal (`int_cal_freeze`) and a status signal (`cal_frozen`). An active-High on the freeze control signal freezes both blocks for computation and coefficient update. Coefficients computed prior to the freeze event continue to be in effect until the next enable (unfreeze) event, which

corresponds to a Low signal on the freeze control port. The time it takes the blocks to change state is approximately 7  $\mu$ s upon a change in the control signal level. Note that this is significantly shorter than the time constants for both the GCB and TSCB. Care does have to be taken so that the freeze control port does not change faster than 7  $\mu$ s intervals. The freeze control mechanism can be provided in one of many different methods, depending on application needs.

A simple signal level detector can be implemented in the PL to monitor the digital output level of the RF-ADC and control the state of the background calibration. Typically a leaky integrator of the absolute value of the RF-ADC output and a hysteresis counter will suffice.

For applications that have a pre-determined duty cycle on the receiver, such as TDD-LTE wireless radios, the TX/RX switching signal can be used in conjunction with the level detector to provide more precise control of the GCB and TSCB blocks.

Note that at the initial startup of the RF-ADC the GCB and TSCB are not at the optimal performance because they both need the presence of some signal to train up the coefficients. The time constant for the convergence for these two blocks is approximately  $2^{22}$  RF-ADC sampling clocks. For applications that requires fast convergence, consider providing a training signal that can be eventually shut off to train the GCB and TSCB blocks right after the startup state machine is completed. This is particularly useful for applications where the input signal is very bursty and with low duty cycle and low power during system startup.

### ***Sub-RF-ADC Outputs***

As shown in the block diagram, each of the sub-RF-ADCs are independently in separate data streams until they get mixed by the QMC block, complex mixer, and then filtered by the DDC. The OCB2, GCB, and TSCB coefficients are applied to each sub-RF-ADC independently and can be effectively disabled if you permanently sets the freeze port to active-High. If the QMC block and the DDC block (hence the mixer) are bypassed, then the sub-RF-ADC data streams are output to the logic independently.

### ***Key CAL Features and Guidance Summary***

Input signal contents at  $F_s/N$ , where  $N = 8$  and  $4$  for the Dual and Quad RF-ADC tile respectively, must be muted during foreground calibration of OCB2. The signal component at the  $k \cdot F_s/N$  bins should be less than  $-95$  dBFs. Gain and Time Skew calibration blocks (GCB, TSCB) should be put in freeze mode when the input signal drops below  $-40$  dBFs level for longer than  $100 \mu$ s. A simple detector block can be used to manage the dynamic control of the freeze port. Dithering mode should be disabled when operating the RF-ADC at  $0.75$  times the maximum sampling rate or slower for the Dual and Quad parts. For applicable systems, a training signal can also be used to calibrate the GCB and TSCB before switching the system to real time operation. Sub-RF-ADC data output streams are available at the programmable logic interface if the DDC features are not used.



## Getting/Setting Calibration Coefficients

As well as automatic calibration, all four calibration blocks (OCB1, OCB2, GCB, TSCB) are available for getting and setting user coefficients. The application reads back the coefficients generated when calibration is un-frozen, and restores them when needed; this helps to maintain the RF-ADC performance when the input signal does not meet calibration requirements. This feature is available for each RF-ADC in the IP wizard. Enabling this feature increases the size of the IP.

Restoring calibration coefficients disables the real-time calibration control port automatically; an API is provided to disable this user coefficients override mode and re-enable the real-time calibration control port.

**Note:** The following two limitations apply::

1. The user coefficients getting/setting for GCB is only valid when the DDC channel is in 1x (bypass), NCO off mode, and the samples per AXI4-Stream must be four samples for Quad RF-ADC and eight samples for Dual RF-ADC
2. The calibration block OCB1 does not support calibration coefficients get/set.

### Related Information

[XRFdc\\_GetCalCoefficients](#)

[XRFdc\\_SetCalCoefficients](#)

[XRFdc\\_GetCalFreeze](#)

[XRFdc\\_SetCalFreeze](#)

[XRFdc\\_DisableCoefficientsOverride](#)

## Setting Coefficients RFdc API Example

The following example code shows the setting of user coefficients for TSCB.

```

u32 Status = XRFDC_FAILURE;
XRFdc_Calibration_Coefficients Coeffs;

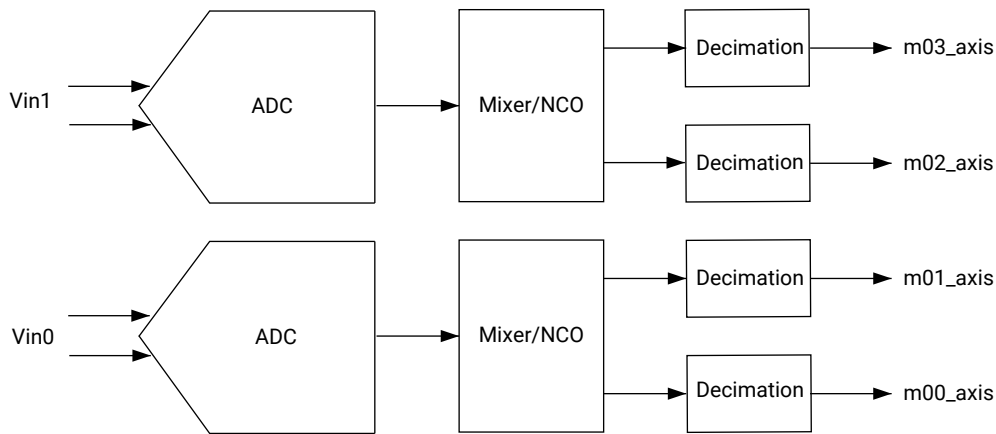
/*use sample coeffs below*/
Coeffs.Coeff0 = 146;
Coeffs.Coeff1 = 255;
Coeffs.Coeff2 = 255;
Coeffs.Coeff3 = 255;
Coeffs.Coeff4 = 113;
Coeffs.Coeff5 = 255;
Coeffs.Coeff6 = 255;
Coeffs.Coeff7 = 255;

Status = XRFdc_SetCalCoefficients( RFdcInstPtr, Tile, Block,
XRFDC_CAL_BLOCK_TSCB, &Coeffs);
If (Status != XRFDC_SUCCESS) {
    /*handle error*/
}
    
```

## RF-ADC Programmable Logic Data Interface

The data interface between the RF-ADC tiles and the PL is implemented using parallel data streams, using the AXI4-Stream protocol. These data streams are output through the gearbox FIFOs which provide a flexible interface between the user application and the RF-ADC tile. The maximum interface width is 128 bits per stream, representing up to eight 16-bit little endian words. The data streams and associated FIFOs have a configurable number of words which provide the flexibility to choose between the number of words and clock frequency to interface with the PL design. There are four streams per tile, and the naming convention is mXY\_axis, where X represents the RF-ADC tile number and Y represents a stream (FIFO) output from that tile. The following figure shows the interfaces.

Figure 22: Dual RF-ADC Tile Programmable Logic Data Interface



X19538-082819

### Interface Data Formats

The data streams represent real or I/Q data, depending on the RF-ADC tile configuration. For Dual RF-ADC tiles, a given stream is either real, I or Q. If an RF-ADC is configured with I/Q output data, then the streams with an even number represent I data and the streams with an odd number represent Q data. These Dual real and I/Q configurations are shown in RF-ADC IP Configuration .

For Quad RF-ADC tiles, a given stream is either real or I/Q interleaved. If an RF-ADC is configured with I/Q output data, then the even-numbered samples of the stream represent I data and the odd-numbered samples represent Q data. These Quad real and I/Q configurations are illustrated in the following sections.

#### Related Information

[RF-ADC IP Configuration](#)

## RF-ADC Interface Data and Clock Rates

The total data rate per channel to the PL is determined by a number of factors, RF-ADC sample rate, decimation factor, and I/Q/Real data formats. The gearbox FIFOs provide a way of interfacing this data rate to the clock frequency of the PL design, by allowing the number of words per clock to be altered. The only requirements are that the number of words and clock rate combine to match the output data rate of the RF-ADC and the decimation rate, if enabled. All RF-ADCs in a tile share a common interface clock frequency. This is shown by the following equations, where  $2G_{IQMode}$  is set to 2 for a Quad RF-ADC tile and I/Q mode is enabled and set to 1 otherwise.

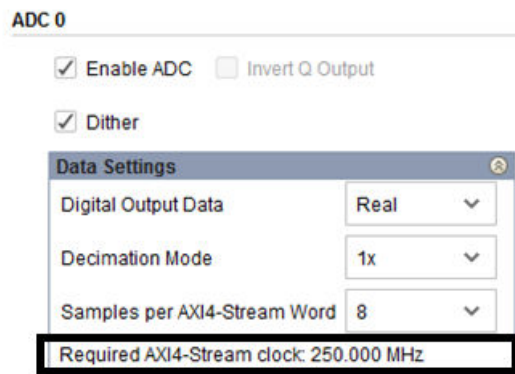
$$PL_{DataRate} = (ADC_{DataRate} \times 2G_{IQMode}) / DecimationRate$$

$$AXI4\text{-Stream Clock} \times PL_{NumWords} = PL_{DataRate}$$

$$AXI4\text{-Stream Clock} = PL_{DataRate} / PL_{NumWords}$$

The core automatically calculates the data rates based on the RF-ADC sample rate and datapath settings. This is shown in the following figure. See the RF-ADC Converter Configuration for information on the settings.

Figure 23: RF-ADC Interface Data and Clock Rates Configuration



Because each tile has independent clocking, sample rates, clock rates, PL rates, and configurations can be specified on a per-tile basis.

### Related Information

[RF-ADC Converter Configuration](#)

## ***PL Clock Interface***

The AXI4-Stream data for all four tile streams is synchronous to a clock from the PL, which has a naming convention of `mX_axis_aclk`, where X represents the RF-ADC tile number. This clock must be at the frequency specified by the Required AXI4-Stream clock displayed on the IP core configuration screen.

The RF-ADC tile also outputs a clock that can be used by the PL. This output clock is a divided version of the RF-ADC sample clock, and is therefore frequency locked to it. This clock has a naming convention of `clk_adcX`, where X represents the RF-ADC tile number.

## ***Interface FIFO Overflow***

The data rate through the interface gearbox FIFOs must be constant during runtime of the RF-ADC tile, with no frequency drift between the PL clock and RF-ADC sample clock domains. If there is a frequency mismatch between these domains, a FIFO overflow might occur. The interface FIFOs have a built-in feature to determine if FIFO overflow has occurred, which is flagged to the PL using the IP interrupt mechanism.

There are two types of overflow, actual and marginal. Actual overflow indicates that the FIFO read/write pointers are overlapping, which means data is not being transferred safely between domains, and action must be taken. Marginal overflow is a warning and indicates that the FIFO read/write pointers are close to overlapping. Overflow should not occur during normal operation, and if overflow is observed it is an indication that the clocking infrastructure of the PL /PCB/ IP is incorrectly configured.

### **Related Information**

[Interrupt Hierarchy](#)

## ***Synchronization***

The gearbox FIFOs provide a flexible data and clock interface for the RF-ADC tiles. However, as with all dual clock FIFOs, latency can vary between one tile and another. While all channels within a tile have the same latency, some applications might require more than one RF-ADC tile to be used, and require the latencies to be matched across all RF-ADC channels. These applications can use the multi-tile synchronization (MTS) feature to achieve this inter-tile synchronization.

### **Related Information**

[Multi-Tile Synchronization](#)

## RF-ADC Multi-Band Operation

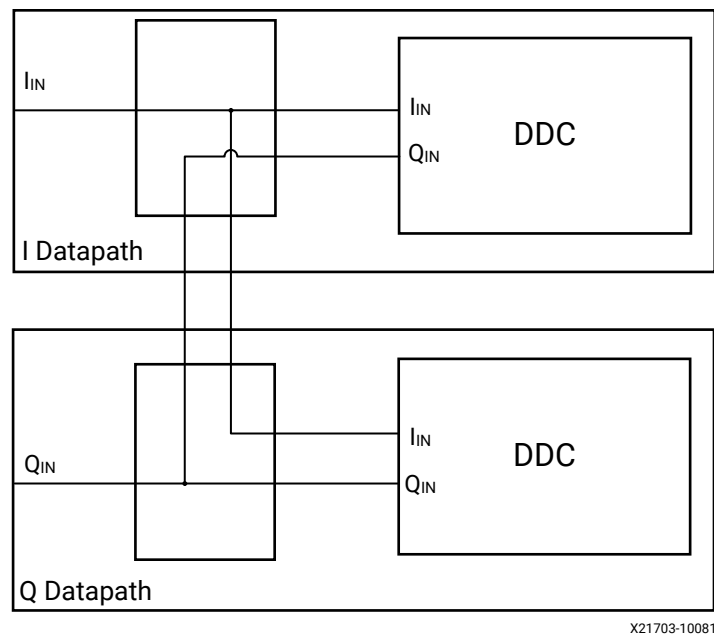
The RF-ADC can be configured to operate in multi-band mode. This is where the incoming analog input consists of baseband signals up-converted (mixed) to different carrier frequencies. Multiple DDC blocks are used to down-convert the analog input to recover the separate baseband signals.

The RF-ADC multi-band feature supports the following configurations:

- 2x multi-band real data per pair. One RF-ADC analog input is active. Both RF-ADC outputs are enabled.
- 2x multi-band I/Q data per pair. Both RF-ADC inputs are active, one for I and one for Q. Both RF-ADC outputs are enabled.
- 4x multi-band real data per tile (Quad ADC tile only). One RF-ADC analog input is active. All 4 RF-ADC outputs are enabled.
- 4x multi-band I/Q data per tile (Quad ADC tile only). Two RF-ADC inputs are active, one for I and one for Q. All 4 RF-ADC outputs are enabled.

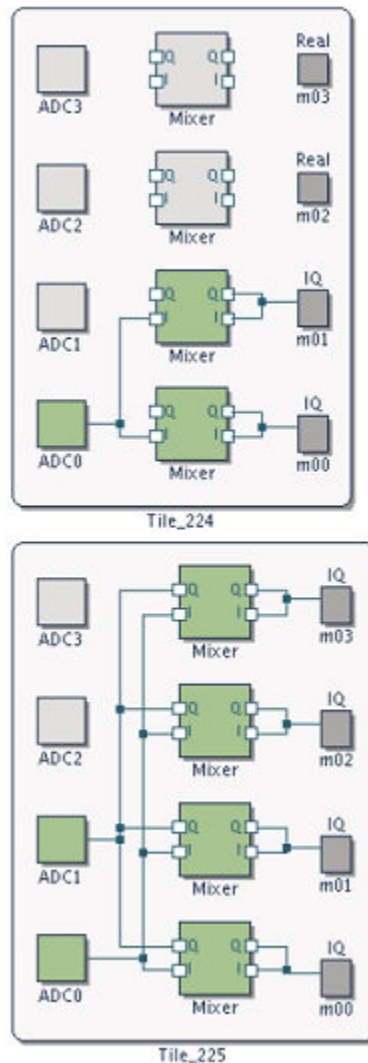
When multi-band is off the I and Q inputs pass straight through the multi-band routing logic. When multi-band is on the I and Q inputs are routed to multiple DDC blocks in the tile.

Figure 24: Multi-Band Routing Logic for Quad RF-ADC tile



RF-ADC multi-band is implemented by routing the output of one RF-ADC analog block to multiple RF-ADC DDC blocks. Each block handles one band of data, and can mix from multiple carriers to baseband. This is shown for a Quad ADC tile in the following figure.

Figure 25: RF-ADC Multi-Band Configurations



Here, tile 224 is configured in real input to I/Q output mode. ADC0 converts the dual-band signal; ADC1 is off. The top pair can be configured as independent RF-ADCs. The dual-band output is routed to the DDC blocks of ADC0 and ADC1. The mixers in the DDC blocks can be configured to extract the correct band from the incoming data. Tile 225 is configured in 4x multi-band I/Q input to I/Q output mode. Here ADC0 carries the quad-band I signal and ADC1 carries the Q data. ADC2 and ADC3 are off. The outputs of the RF-ADCs are routed to all four DDC blocks. Each DDC can be configured to extract the data from the desired frequency band.

## RF-ADC Nyquist Zone Operation

Each RF-ADC channel can sample signals in the first or second Nyquist zones. To ensure the RF-ADC performance is optimal the RF-ADC configuration settings should indicate the intended zone of operation.

- First Nyquist zone is defined as a signal between 0 and  $F_s/2$
- Second Nyquist zone is defined as a signal between  $F_s/2$  and  $F_s$

Other Nyquist zones can also be used as long as the signal meets the RF-ADC input bandwidth requirements. Zones 1, 3, 5, ... are referred to as odd zones, and zones 2, 4, ... are referred to as even zones.

### Related Information

[RF-ADC Converter Configuration](#)

[XRFdc\\_GetNyquistZone](#)

[XRFdc\\_SetNyquistZone](#)

## RF-ADC IP Configuration

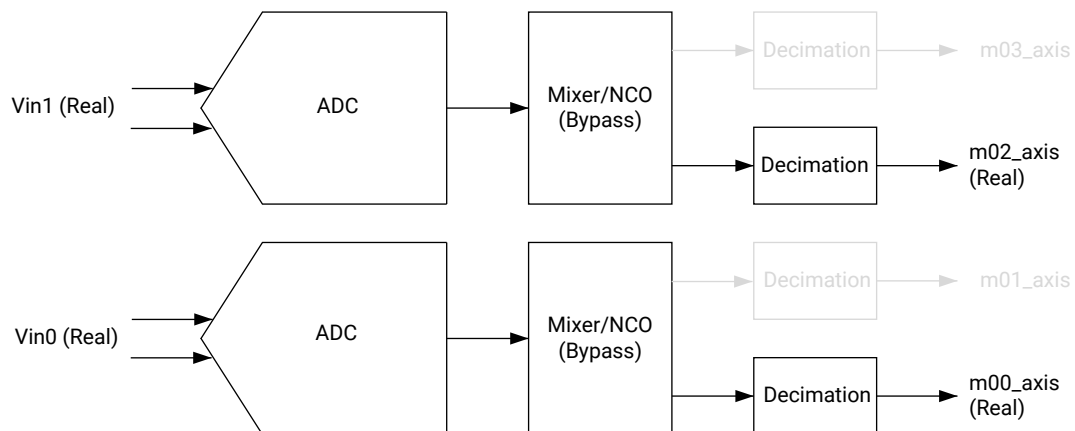
The RF-ADC tile can be configured in several modes. The basic configuration options are available on the IP core configuration screen in the Vivado® IDE and advanced operating modes can be configured using the RFdc driver API.

The RF-ADC is available in Quad or Dual configurations depending on the tile (see the *Zynq UltraScale+ RFSoc Data Sheet: Overview (DS889)* and the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)*).

### Dual RF-ADC Configuration Options

#### Dual RF-ADC Real Input to Real Output

Figure 26: Dual RF-ADC Real Input to Real Output



X18274-082019

Figure 27: Dual RF-ADC Real Input to Real Output IP Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [1.0 - 4.116]      Clock Out (MHz)

Reference Clock (MHz)       AXI4-Stream Clock (MHz)

---

**Multi Tile Sync**      **Link Coupling**

Enable Multi Tile Sync      Link Coupling

---

**Converter Configuration**

**ADC 0**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 250.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

**Analog Settings**

Nyquist Zone

Calibration Mode

**ADC 1**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

**Mixer Settings**

Mixer Type

Mixer Mode

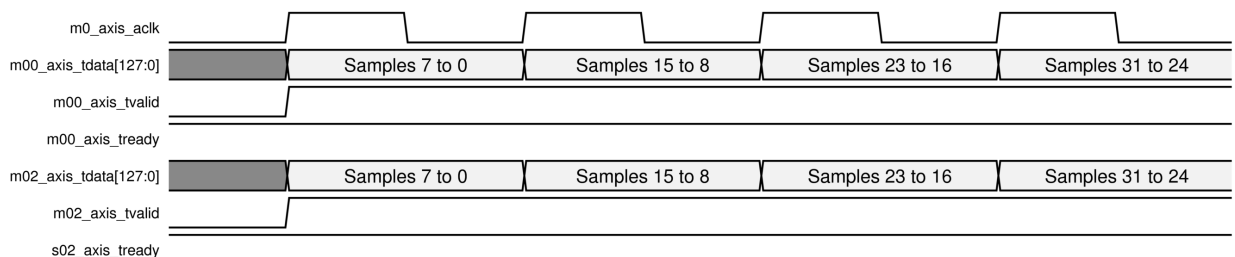
**Analog Settings**

Nyquist Zone

Calibration Mode

The following figure shows a Dual RF-ADC with real data input to real data output, 1x decimation, the mixer bypassed and running at a 500 MHz AXI4-Stream clock.

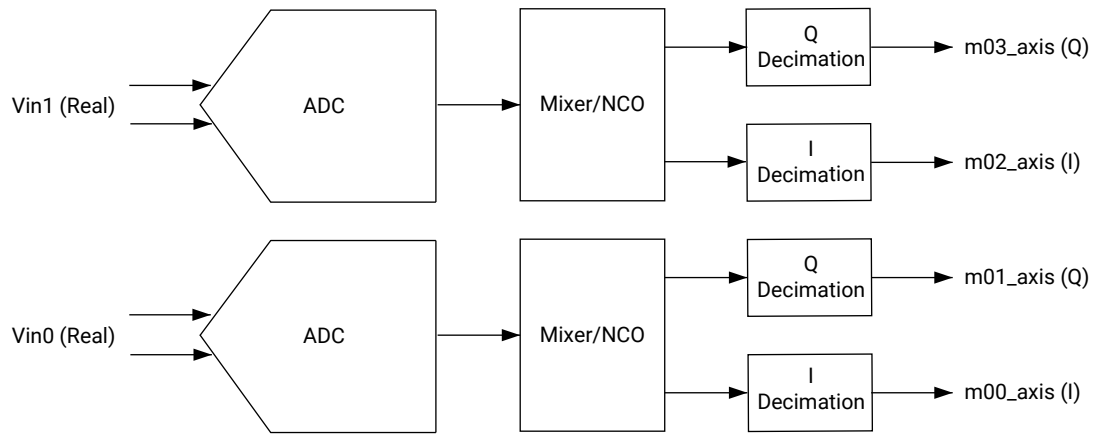
Figure 28: Dual RF-ADC Real Input to Real Output Data Timing





### Dual RF-ADC Real Input to I/Q Output

Figure 29: Dual RF-ADC Real Input to I/Q Output



X18273-082719

Figure 30: Dual RF-ADC Real Input to I/Q Output IP Core Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [1.0 - 4.116]      Clock Out (MHz)

Reference Clock (MHz)       AXI4-Stream Clock (MHz)

---

**Multi Tile Sync**      **Link Coupling**

Enable Multi Tile Sync      Link Coupling

---

**Converter Configuration**

**ADC 0**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 500.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

NCO Phase

**Analog Settings**

Nyquist Zone

Calibration Mode

**ADC 1**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 500.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

NCO Phase

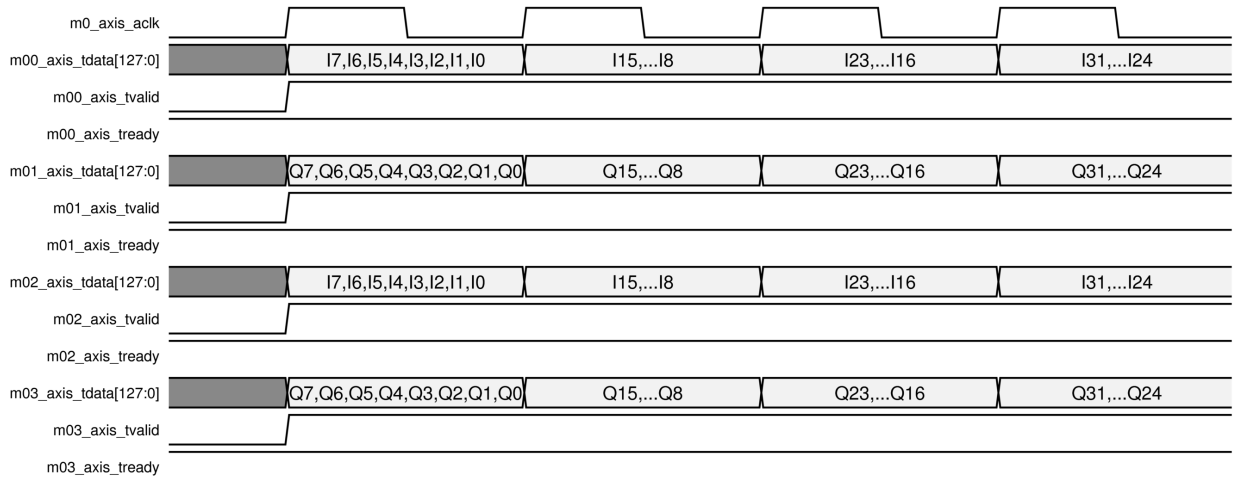
**Analog Settings**

Nyquist Zone

Calibration Mode

The following figure shows a Dual RF-ADC with real data input to I/Q data output, 1x decimation, the mixer enabled, and running at a 500 MHz AXI4-Stream clock.

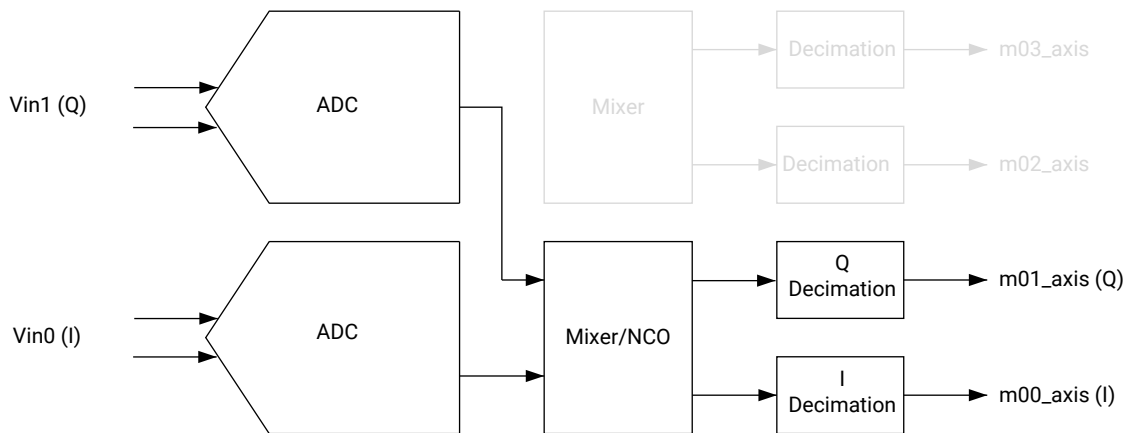
Figure 31: Dual RF-ADC Real Input to I/Q Output Data Timing



### Dual RF-ADC I/Q Input to I/Q Output

For I/Q input to I/Q output, the RF-ADCs are paired.

Figure 32: Dual RF-ADC I/Q Input to I/Q Output



X18275-082719

Figure 33: Dual RF-ADC I/Q Input to I/Q Output IP Core Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [1.0 - 4.116]      Clock Out (MHz)

Reference Clock (MHz)       AXI4-Stream Clock (MHz)

---

**Multi Tile Sync**      **Link Coupling**

Enable Multi Tile Sync      Link Coupling

---

**Converter Configuration**

**ADC 0**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 500.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

NCO Phase

**Analog Settings**

Nyquist Zone

Calibration Mode

**ADC 1**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 500.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

NCO Phase

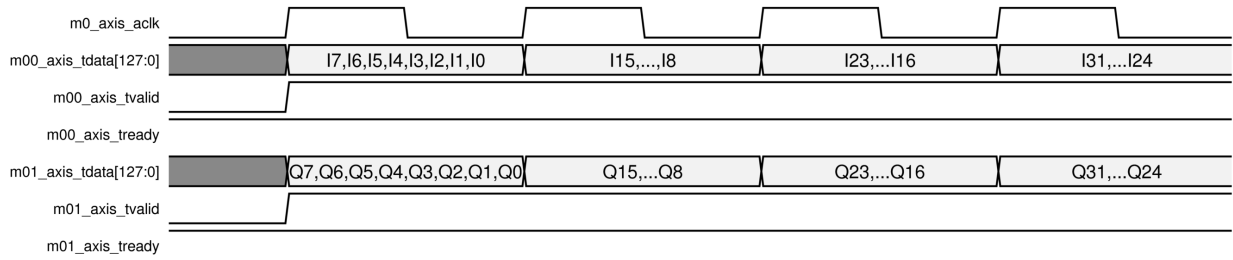
**Analog Settings**

Nyquist Zone

Calibration Mode

The following figure shows a Dual RF-ADC with I/Q input to I/Q output, 1x decimation, the mixer enabled, and running at a 500 MHz AXI4-Stream clock.

Figure 34: Dual RF-ADC I/Q Input to I/Q Output Timing

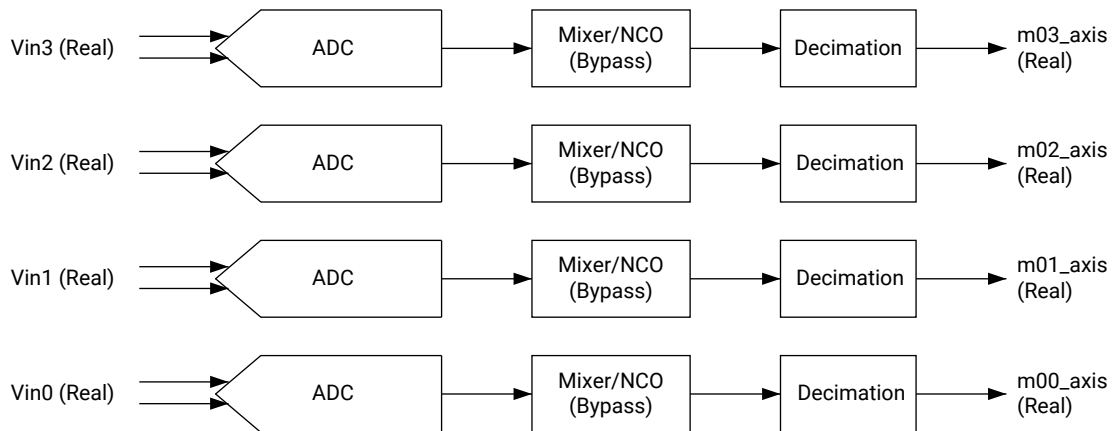


### Quad RF-ADC Configuration Options

This option is only available on selected devices (see the *Zynq UltraScale+ RFSoc Data Sheet: Overview* (DS889) for device information).

### Quad RF-ADC Real Input to Real Output

Figure 35: Quad RF-ADC Real Input to Real Output



X18276-082719

Figure 36: Quad RF-ADC Real Input to Real Output IP Core Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [0.5 – 2.058]    Clock Out (MHz)

Reference Clock (MHz)     AXI4-Stream Clock (MHz)

---

**Multi Tile Sync**      **Link Coupling**

Enable Multi Tile Sync      Link Coupling

---

**Converter Configuration**

ADC Pair 0,1    ADC Pair 2,3

**ADC 0**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 500.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

**Analog Settings**

Nyquist Zone

Calibration Mode

**ADC 1**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

**Mixer Settings**

Mixer Type

Mixer Mode

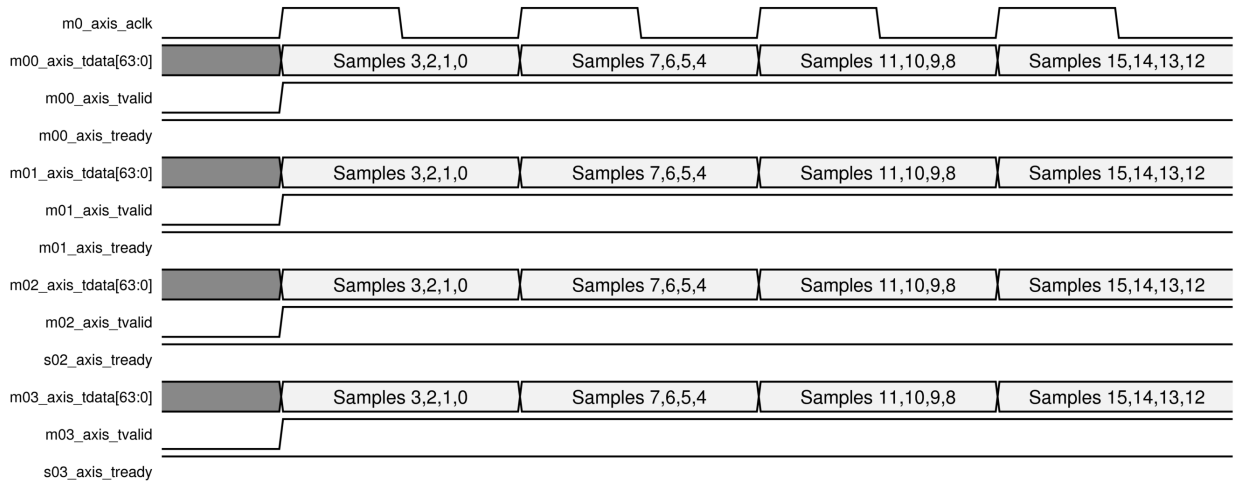
**Analog Settings**

Nyquist Zone

Calibration Mode

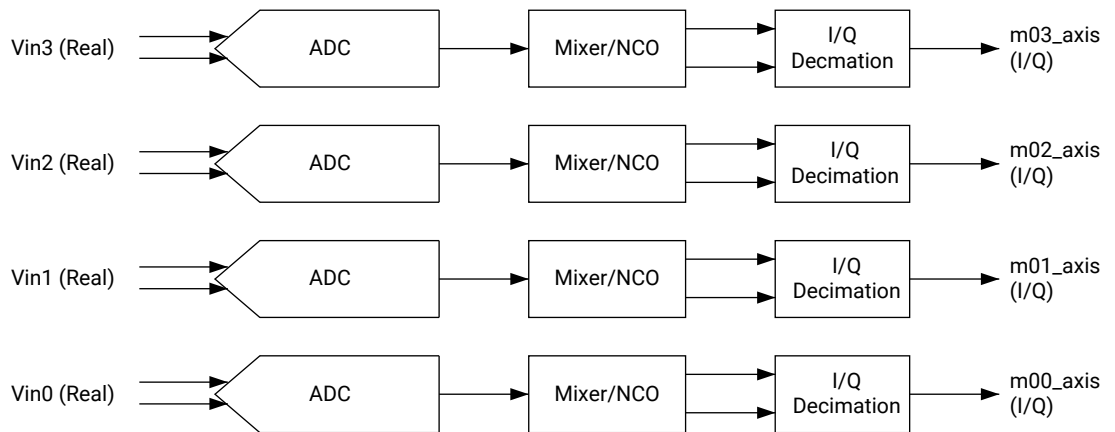
The following figure shows a Quad RF-ADC real data input to real data output, 1x decimation, the mixer bypassed, and running at a 500 MHz AXI4-Stream clock.

Figure 37: Quad RF-ADC Real Input to Real Output Timing



### Quad RF-ADC Real Input to I/Q Output

Figure 38: Quad RF-ADC Real Input to I/Q Output



X18277-082719

Figure 39: Quad RF-ADC Real Input to I/Q Output IP Core Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [0.5 - 2.058]      Clock Out (MHz)

Reference Clock (MHz)       AXI4-Stream Clock (MHz)

---

**Multi Tile Sync**      **Link Coupling**

Enable Multi Tile Sync      Link Coupling

---

**Converter Configuration**

ADC Pair 0,1    ADC Pair 2,3

**ADC 0**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 500.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

NCO Phase

**Analog Settings**

Nyquist Zone

Calibration Mode

**ADC 1**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

**Mixer Settings**

Mixer Type

Mixer Mode

**Analog Settings**

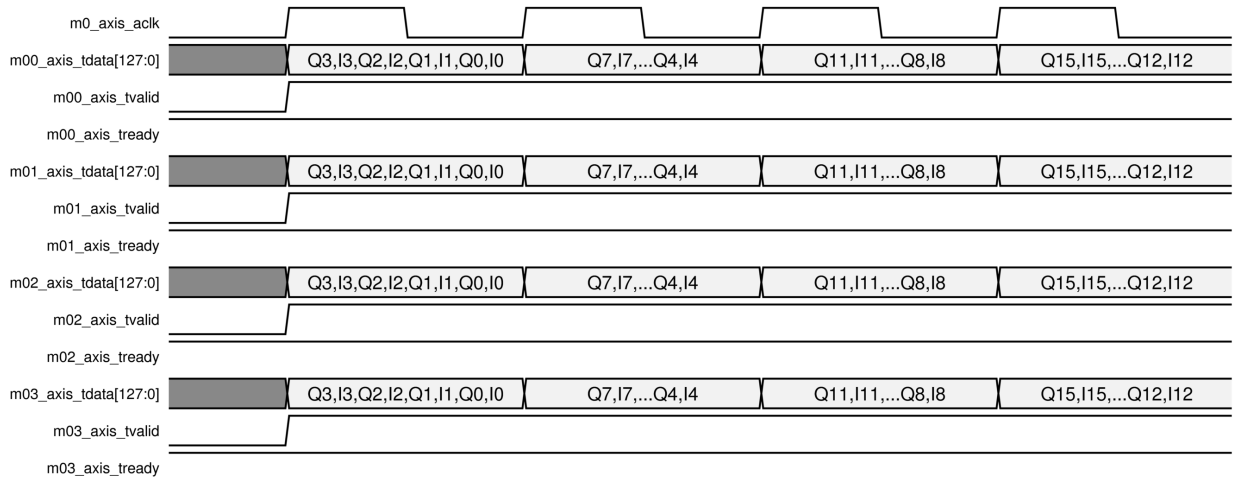
Nyquist Zone

Calibration Mode

The following figure shows a Quad RF-ADC real input data to I/Q output data, 1x decimation, the mixer enabled, and running at a 500 MHz AXI4-Stream clock. Note that each I/Q channel is interleaved on the output data stream.



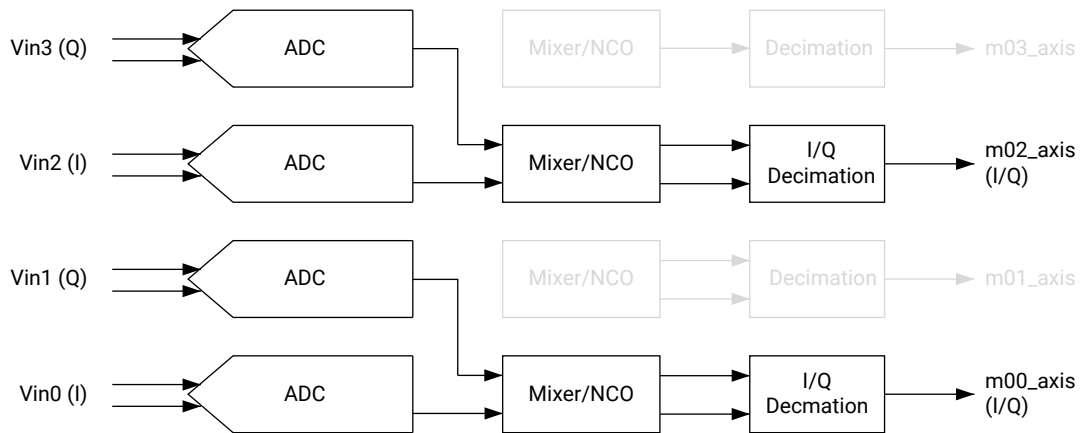
Figure 40: Quad RF-ADC Real Input to I/Q Output Timing



### Quad RF-ADC I/Q Input to I/Q Output

For I/Q input to I/Q output, the RF-ADCs are paired.

Figure 41: Quad RF-ADC I/Q Input to I/Q Output



X18278-082719

Figure 42: Quad RF-ADC I/Q Input to I/Q Output IP Core Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [0.5 – 2.058]    Clock Out (MHz)

Reference Clock (MHz)     AXI4-Stream Clock (MHz)

---

**Multi Tile Sync**      **Link Coupling**

Enable Multi Tile Sync      Link Coupling

---

**Converter Configuration**

**ADC Pair 0,1**    **ADC Pair 2,3**

**ADC 0**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 500.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

**Analog Settings**

Nyquist Zone

Calibration Mode

**ADC 1**

Enable ADC     Invert Q Output

Dither

**Data Settings**

Digital Output Data

Decimation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 500.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

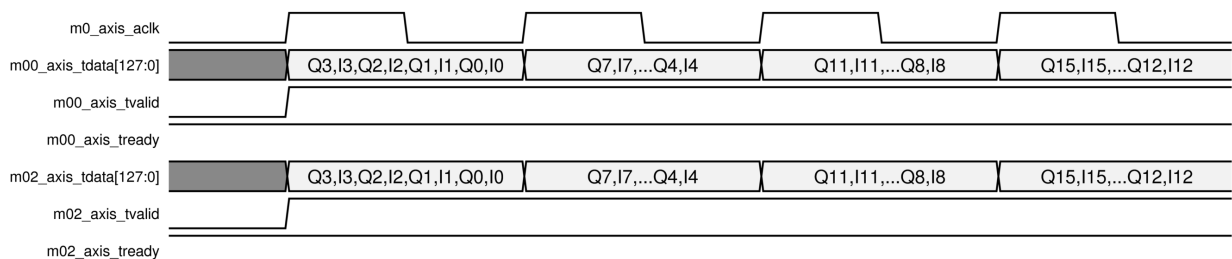
**Analog Settings**

Nyquist Zone

Calibration Mode

The following figure shows a Quad RF-ADC I/Q to I/Q, 1x decimation, the mixer enabled, and running at a 500 MHz AXI4-Stream clock.

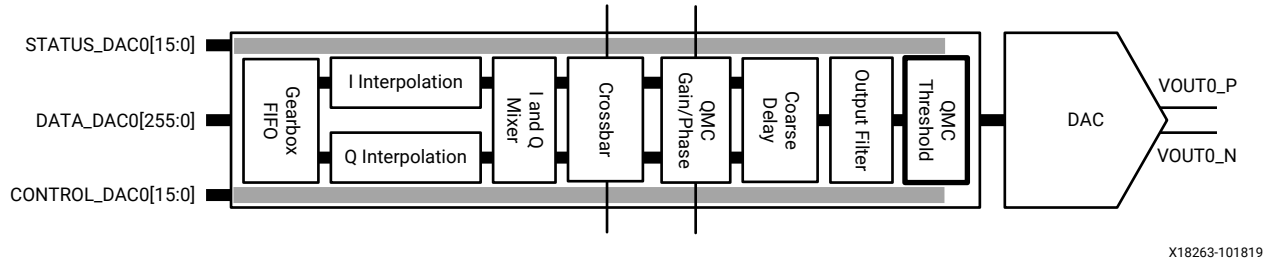
Figure 43: Quad RF-ADC I/Q Input to I/Q Output Timing



## RF-DAC

Each RF-DAC tile includes a complete clocking support structure with a PLL and the necessary synchronization logic. Every RF-DAC in a tile has a highly configurable FIFO allowing the internal interconnect logic to have direct high-speed access to the RF-DAC (see figure below).

Figure 44: Simplified RF-DAC Functionality Block Diagram



Certain functions can only be executed when the RF-DACs in a tile are paired. Even numbered RF-DACs are used for I datapaths and the odd numbered RF-DACs are used for Q datapaths.

All of the available built-in functionality of a tile and each of the RF-DACs in a tile are user programmable. The Zynq® UltraScale+™ RFSoc RF Data Converter core configuration screen in the Vivado® IDE and the RFdc driver API can be used to configure the digital and analog functionality of the RF-DACs.

## RF-DAC Analog Outputs

Each RF-DAC in a tile has its own differential analog current output buffer/driver. The output currents are complementary; the sum of the two currents always equals the full-scale current of the RF-DAC. The digital input code determines the effective differential current delivered to the load. The differential RF-DAC outputs are typically AC-coupled out using capacitors. RF-DAC output DC-coupling is also supported, assuming that the correct common mode biasing and load impedance requirements are met. See *UltraScale Architecture PCB Design User Guide (UG583)* for more details.

### Transmit Transfer Function

The differential output provides the maximum output current when all digital input bits are High. The output current (using binary format) is shown in the following equations.

$$I_{VOUTP} = (\text{BinDataIn} / 2^{N-1}) \times I_{OUTFS}$$

$$I_{VOUTN} = I_{OUTFS} - I_{VOUTP}$$

where:

1. BlnDataIn = 14-bit digital input
2. I<sub>OUTFS</sub> = Full-scale output current

The output current (using a twos complement) is shown in the following equations.

$$I_{VOUTP} = (\text{TwosDataIN} + 2^{N-1}) / 2^{N-1} \times I_{OUTFS}$$

$$I_{VOUTN} = I_{OUTFS} - I_{VOUTP}$$

where:

1. TwosDataIN = 14-bit digital twos complement input ranging from  $(-2^{N-1})$  to  $(2^{N-1}-1)$
2. I<sub>OUTFS</sub> = Full-scale output current.

### RF-DAC Output Current Mode

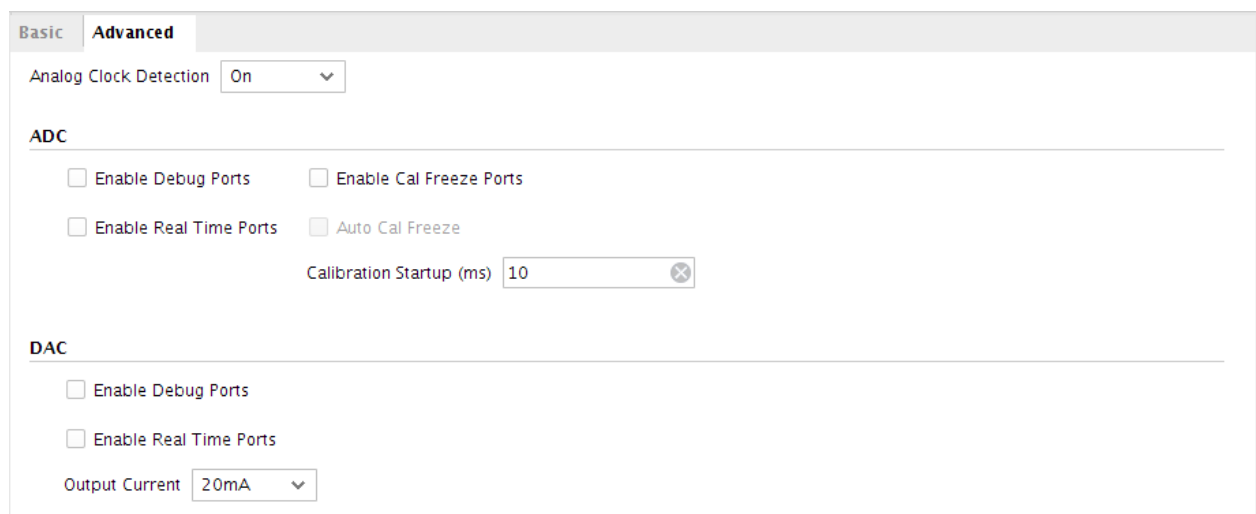
The RF-DAC output current is configurable, with an option of operating in either 20 mA or 32 mA modes. In the 20 mA mode, set the DAC\_AVTT to 2.5V and in the 32 mA mode, set it to 3.0V to maintain the linearity performance.



**CAUTION!** A 3.0V DAC\_AVTT should not be used in the 20 mA mode. This risks exceeding the maximum ratings of the device and also risks affecting device reliability.

The output current can be set in the Advanced tab in the IP core configuration screen (see the following figure).

Figure 45: RF-DAC Output Current Configuration



### RFdc Driver API Commands

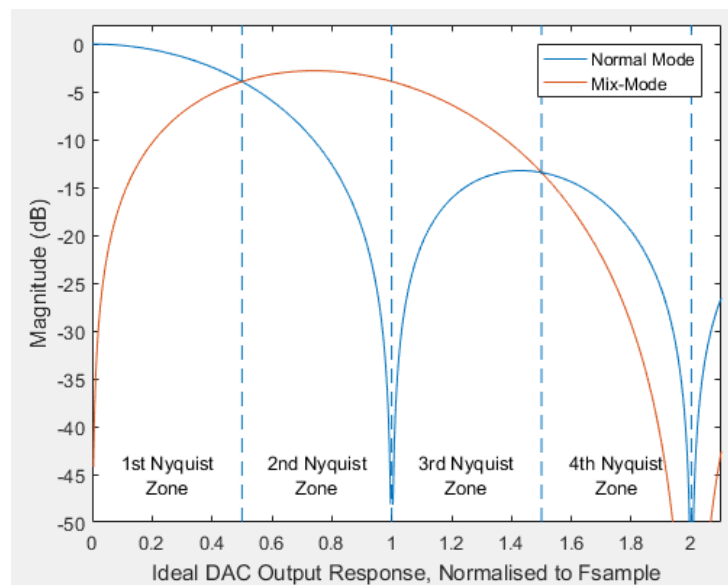
```
// Get output current of RF-DAC Tile1, Block2 to 32mA
XRfDc_GetOutputCurr(ptr, 1, 2, XRFDC_OUTPUT_CURRENT_32MA);
```

**Related Information**
[XRFdc\\_GetOutputCurr](#)

## RF-DAC Nyquist Zone Operation

Each RF-DAC can optimize its output response in the second Nyquist zone by using the mix-mode feature. This feature mixes the RF-DAC data with the sample clock, and, as a result, increases the output power in the second Nyquist zone, while attenuating it in the first Nyquist zone. This is shown in the following figure.

Figure 46: RF-DAC Nyquist Zone Operation



For normal (non-mix mode) operation, the blue line represents the ideal RF-DAC output roll-off sinc response. As can be seen, an output image in the second Nyquist zone in this mode would be severely attenuated. An inverse sinc filter is available to compensate for the roll-off in the first Nyquist zone for this mode. See RF-DAC Inverse Sinc Filter for more information.

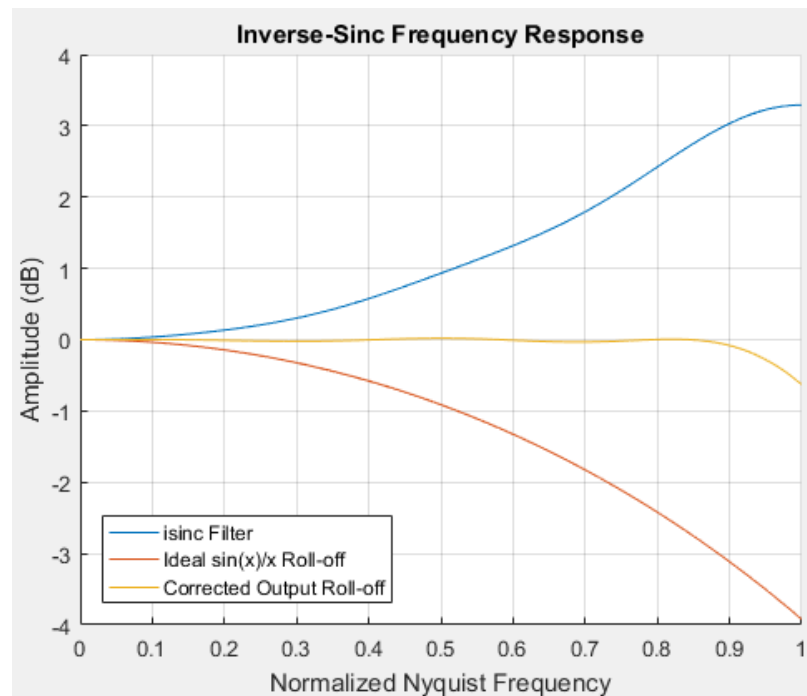
In RF-DAC mix-mode, the red line represents the ideal RF-DAC output response. In this mode, the output power of the image in the second Nyquist zone is significantly increased, and it also has an approximately flat response across the majority of the zone. The Nyquist zone can be set in the Vivado® IDE.

**Related Information**
[RF-DAC Inverse Sinc Filter](#)
[XRFdc\\_GetNyquistZone](#)
[XRFdc\\_SetNyquistZone](#)

## RF-DAC Inverse Sinc Filter

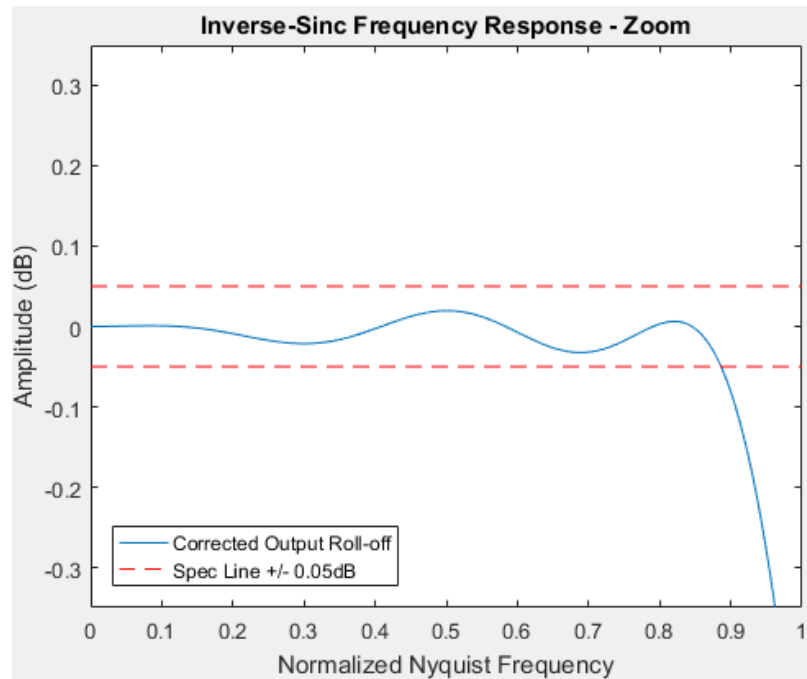
The analog output response of the RF-DAC follows a characteristic  $\text{sinc}(x)/x$ , or sinc shape. For applications that require flat-output response over a wide bandwidth, an inverse sinc filter is available to achieve this. The inverse sinc filter is an 11-tap FIR, which provides a flat-response with less than  $\pm 0.05$  dB of ripple up to 89% of Nyquist, or  $\pm 0.033$  dB of ripple up to 80% of Nyquist. The following figure shows the inverse sinc performance.

Figure 47: Inverse Sinc Frequency Response



The line in blue represents the frequency response of the filter itself. As can be seen, it increases with frequency to compensate for the sinc response of the output, as shown by the red line. The composite output response is given by the yellow trace, and shows a flat pass-band up to 89% of Nyquist. The following figure shows the detail of the corrected response, with ripple specification levels highlighted in red.

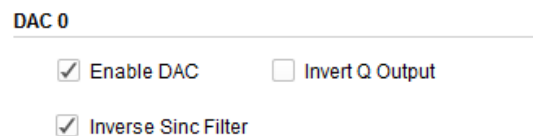
Figure 48: Inverse Sinc Detail



## Inverse Sinc Filter Use

The filter can be enabled or disabled per RF-DAC in the Vivado® IDE by using the Inverse Sinc Filter check box, as shown in the following figure.

Figure 49: Inverse Sinc Configuration



## RFdc Driver API Commands

The RFdc driver API can be used to get the inverse sinc enable state using the following code.

```
// Get Inverse Sinc State for Tile0, DAC Block1

int Tile = 0;
u32 Block = 1;
u32 isinc_enabled;
isinc_enabled = XRFdc_GetInverseSincFilter (ptr, Tile, Block);
xil_printf("DAC%d,%d Inverse Sinc Enabled: %d\n\r", Tile, Block,
isinc_enabled);
```

**Related Information**
[XRFdc\\_GetInverseSincFilter](#)
[XRFdc\\_GetInvSincFIR](#)
[XRFdc\\_SetInvSincFIR](#)

## Overflow

As seen in the Inverse Sinc Frequency response, the filter gain is  $>1$ , and increases with frequency up to approximately 2.4 dB at 80% of Nyquist. To ensure that the filter output does not overflow, the input signal amplitude must be backed-off to account for this gain factor. The inverse sinc block has automatic overflow detection and saturation, and if overflow is detected, it is flagged using the interrupt mechanism to the RFDc driver API, using the XRFDC\_DAC\_IXR\_INVSNC\_OF\_MASK interrupt. For more details see Interrupt Handling.

**Related Information**
[Interrupt Hierarchy](#)

## Inverse Sinc Filter Details

Inverse Sinc FIR	-1	3	-6	15	-52	594	-52	15	-6	3	-1
------------------	----	---	----	----	-----	-----	-----	----	----	---	----

## RF-DAC Digital Datapath

Each RF-DAC in a tile has a number of optimized DSP features that can be used to implement Digital Up Conversion (DUC) and transmit signal filtering. These features are:

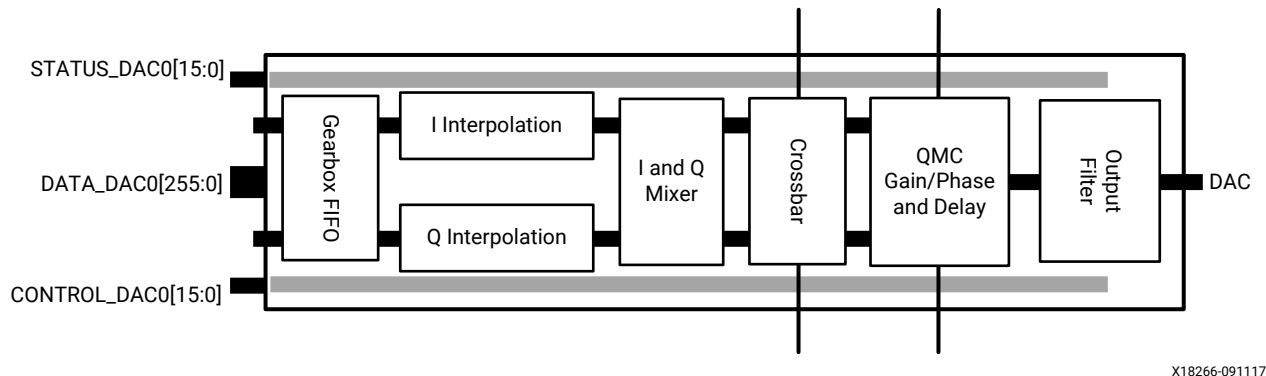
- Signal interpolation functionality—interpolation by 1 (bypass), 2, 4, or 8 is supported
- Coarse mixing (quarter and half rate) or fine mixing with a 48-bit frequency resolution numerically controlled oscillator (NCO)
- Compensation functionality containing a quadrature modulator correction (QMC) block with coarse delay adjustment block
- Signal conditioning containing an inverse sinc FIR filter

Single, multiple, or all DSP functions can be used or bypassed, using the Vivado® IDE.

The following figure shows the available functions in an RF-DAC. Each function is described in the following sections.



Figure 50: Signal Treatment of the RF-DAC Peripherals

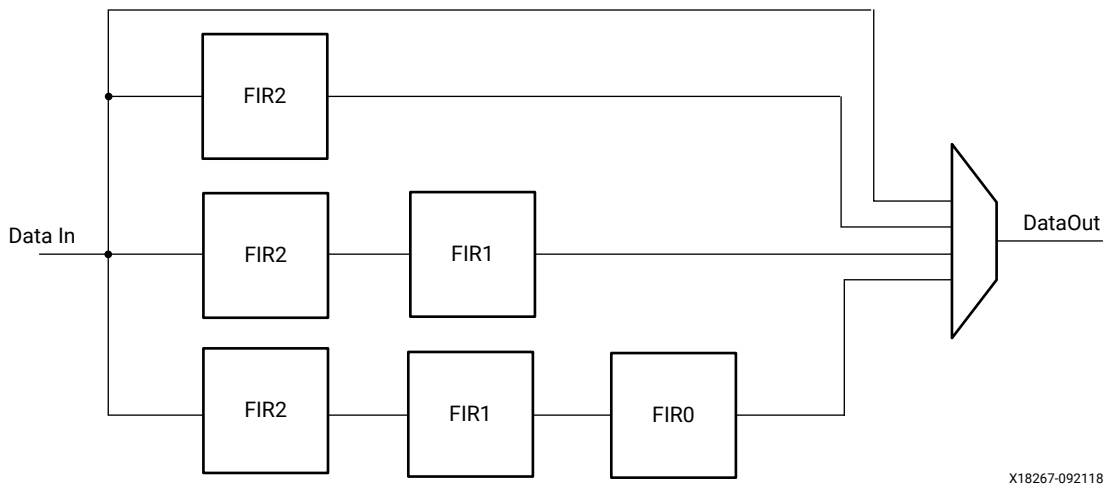


### RF-DAC Interpolation Filters

Interpolation filters are required to implement the up-sampling and filtering portions of the DUC process. The implemented filter operation consists of three low pass FIR filters, each with a predetermined, fixed set of coefficients, shown in the following figure. Each filter block can be bypassed and the output of each filter block can be routed to the output of the filter.

- 1x—all filter stages are bypassed
- 2x—interpolation using a single stage
- 4x—interpolation using two stages
- 8x—interpolation using all three available stages

Figure 51: RF-DAC Interpolation Filter



The output of an interpolated signal is the exact representation of the original signal presented at a higher sampling rate. Up-sampling of a signal sampled at  $F_1$  to a higher sample rate ( $N \cdot F_1$ ) results in  $N$  replications of the original spectrum. Low pass filtering of the result helps to remove unwanted replications resulting in the preservation of the original signal at the required higher sample rate.

**Table 46: Interpolation Filter Operating Modes**

Mode	Description
OFF	Filter is disabled, RF-DAC is not available.
1x	Filter is bypassed.
2x	2x interpolation, 80% Nyquist passband.
4x	4x interpolation, 80% Nyquist passband.
8x	8x interpolation, 80% Nyquist passband.

Each of the filter stages can overflow—given the step-response of a FIR filter, especially when full-scale data is on the input. Each filter stage has an overflow status signal and output saturation to detect an overflow and protect the datapath. When a filter stage is not used, the overflow flag is pulled Low. These flags are connected to the datapath interrupts mechanism.

### Related Information

[Interrupt Hierarchy](#)

### Interpolation Filter Use

The IP core is used to set the interpolation rate. This is an IP core setting because changing the interpolation rate directly affects the physical interface as the bandwidth to the PL changes. The filters that are enabled are shown in the following figure.

Figure 52: RF-DAC Interpolation Filter Configuration

**DAC 0**

Enable DAC     Invert Q Output  
 Inverse Sinc Filter

**Data Settings**

Analog Output Data: Real

Interpolation Mode: 2x

Samples per AXI4-Stream Word: 16

Required AXI4-Stream clock: 400.000 MHz

**Mixer Settings**

Mixer Type: Fine

Mixer Mode: I/Q->Real

NCO Frequency (GHz): 0.0

NCO Phase: 0

**Analog Settings**

Nyquist Zone: Zone 1

Decoder Mode: SNR Optimized

**Note:** I/Q input data and mixing requires a minimum of 2x interpolation. If full bandwidth I/Q data is required, two RF-DAC data converters can be operated as independent real streams, containing the I and Q data respectively. Phase adjustment cannot be done when using QMC.

### Related Information

[RF-DAC Converter Configuration](#)

### RFdc Driver API Commands

The RFdc driver API can be used to get the interpolation rate set in the Vivado® IDE using the following code.

```
// Get Interpolation factor for Tile0, DUC Block1

int Tile = 0;
u32 Block = 1;
u32 Interpolation_Factor;
if( XRFdc_GetInterpolationFactor (ptr, Tile, Block, &Interpolation_Factor)
==
XST_SUCCESS) {
    xil_printf("DAC Tile%1d,%1d Interpolation Factor is: %d", Tile, Block,
Interpolation_Factor);
}
}
```

**Related Information**

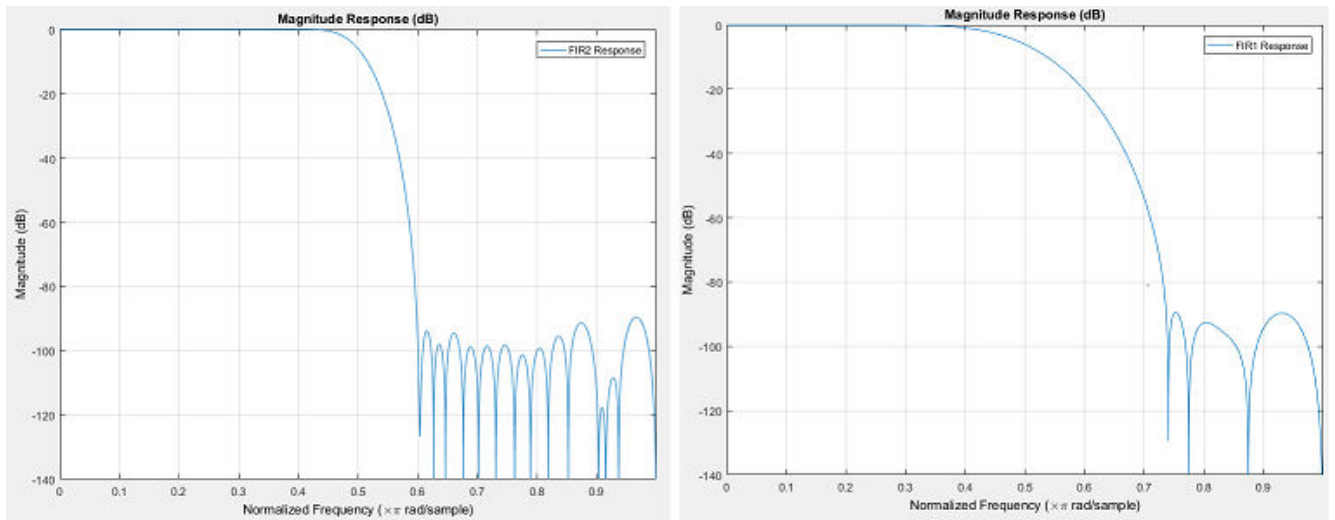
[XRFdc\\_GetInterpolationFactor](#)

[XRFdc\\_SetInterpolationFactor](#)

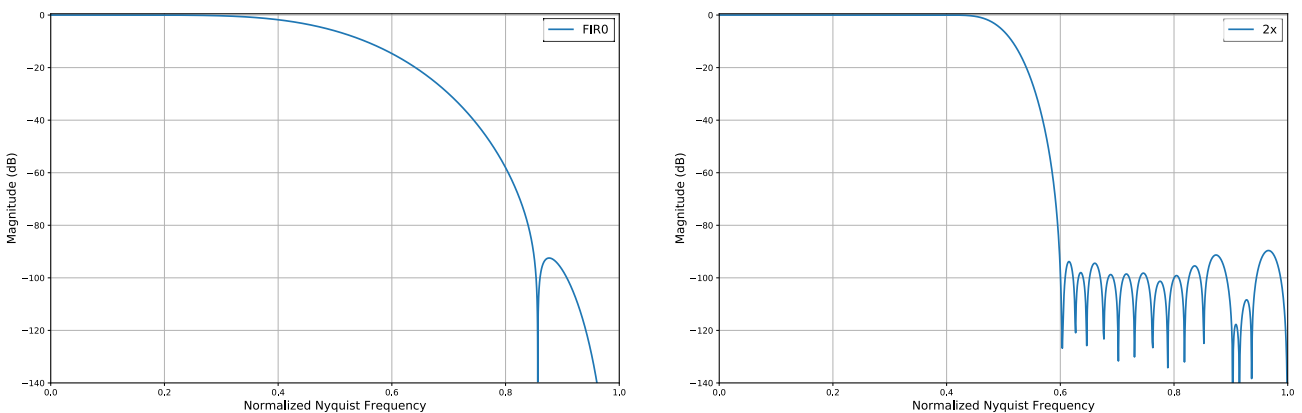
**Interpolation Filter Details**

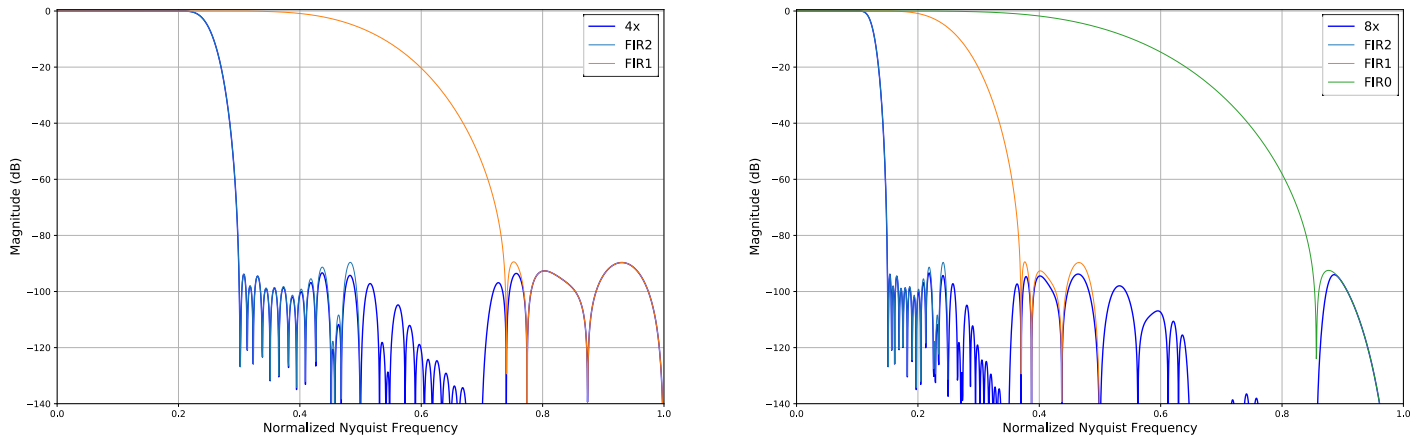
The interpolation filter chain consists of three FIR filters, FIR2, FIR1, and FIR0, which can be enabled to give successive interpolation by a factor of two per stage. The filter transfer functions are shown in the following figures.

*Figure 53: FIR2 and FIR1 Frequency Response*



*Figure 54: FIR0 and 2x Interpolation Frequency Response*



**Figure 55: 4x Interpolation and 8x Interpolation Frequency Response**


The filter coefficients for the interpolation filters are shown in the following tables.

FIR 2	1st Half	5	0	-17	0	44	0	-96	0	187	0	-335	0	565	0	-906
		0	1401	0	-2112	0	3145	0	-4723	0	7415	0	-13331	0	41526	
	Center Tap	65536														
	2nd Half	5	0	-17	0	44	0	-96	0	187	0	-335	0	565	0	-906
0		1401	0	-2112	0	3145	0	-4723	0	7415	0	-13331	0	41526		

FIR1	1st Half	-12	0	84	0	-337	0	1008	0	-2693	0	10142
	Center Tap	16384										
	2nd Half	-12	0	84	0	-337	0	1008	0	-2693	0	10142

FIR0	1st Half	-6	0	54	0	-254	0	1230
	Center Tap	2048						
	2nd Half	-6	0	54	0	-254	0	1230

The interpolation filter chains can operate in real or I/Q modes, depending on the data type selected for the RF-DAC interface.

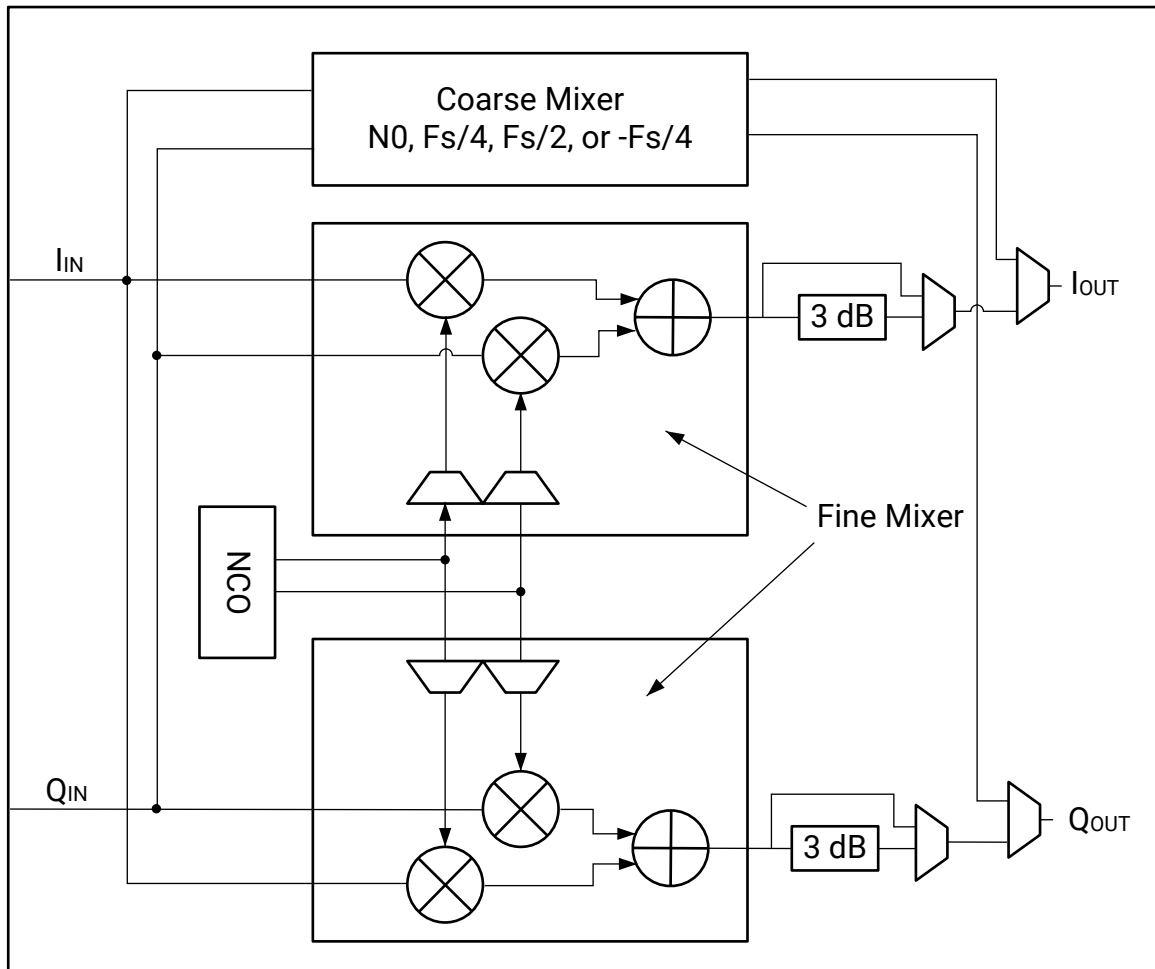
### Related Information

[RF-DAC Programmable Logic Data Interface](#)

## ***RF-DAC Numerical Controlled Oscillator and Mixer***

The mixer function has three modes, bypass (no mixing), coarse mixing or fine mixing. Fine mixing automatically enables the NCO which is used to generate the carrier frequency. The mixer supports full quadrature mixing, with both real to I/Q and I/Q to I/Q modes supported.

Figure 56: RF-DAC Mixer with NCO DSP Block



X18261-032318

#### Coarse Mixer:

- The coarse mixer allows the data to be mixed with a carrier of 0,  $F_s/2$ ,  $F_s/4$ , or  $-F_s/4$ .

**Note:** The selection of 0 is only available using the RFdc driver API.

- Mixing with a 0 carrier bypasses the mixer component.

#### Fine Mixer:

- The fine mixer allows the data to be shifted up or down in frequency by an arbitrary amount.
- The frequency shift amount is obtained by programming the mixer frequency generated in the NCO. The fine mixer also supports 18-bit phase adjustment.
- The NCO phase can be synchronized within a tile using `XRFdc_UpdateEvent`.
- The NCO phase can be synchronized across tiles using an external event signal (SYSREF or MARKER).

- To manage potential overflow, the fine mixer output includes 3 dBV attenuation, as shown in the figure above. This attenuation is set to -3 dBV (multiplication factor = 0.707) regardless of the RF-DAC setting in the RFdc driver API. A manual selection is also possible, allowing 0 dBV or -3 dBV.

The mixer settings can be configured in the core, or by using the RFdc driver API. The core is used to set the initial mixer settings (for example, mixer type and mixer mode), and the RFdc driver API is used to adjust the settings at runtime. Both the RFdc driver API and the core compute the required register settings based on the supplied sample rates and desired frequencies. See the RF-DAC Converter Configuration section for information on the settings.

### Related Information

[RF-DAC Converter Configuration](#)

## RF-DAC Multi-Band Operation

Multi-band operation is where two or more baseband signals are up-converted (mixed) to individual carriers and then combined to generate a single composite analog output. In the RF-DAC tiles, this involves combining multiple DUC blocks together to drive the analog outputs.

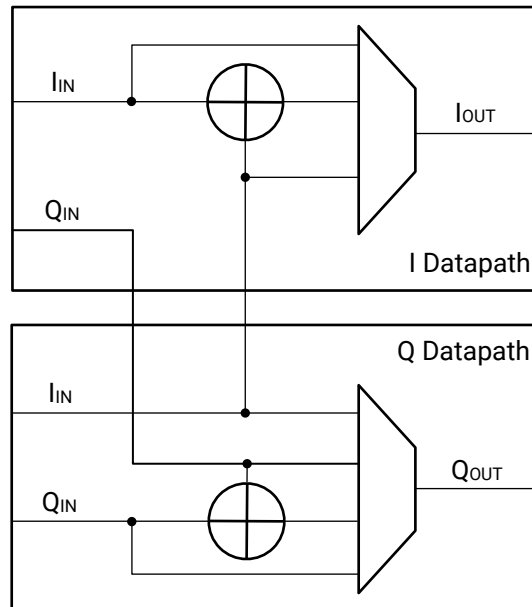
The RF-DAC multi-band feature supports the following configurations:

- 2x multi-band real data per pair. One RF-DAC output of the pair is enabled; the other is off.
- 2x multi-band I/Q data per pair. Both RF-DACs in the pair are enabled, one for I and one for Q.
- 4x multi-band real data per tile. Four input streams are combined to drive the DAC0 output in real mode. All other RF-DACs are off.
- 4x multi-band I/Q data per tile. Four input streams are combined to drive the DAC0 output as I and DAC1 output as Q. All other RF-DACs are off.

Latency between outputs in a multi-band pair is matched and latency between pairs is matched, irrespective of the mode.

When multi-band is off, the I and Q datapaths pass straight through the multi-band logic block (as shown below). When multi-band is on, I and Q of each datapath are combined and passed to the next DSP block in the chain in front of the RF-DAC. Multi-band can be turned on per RF-DAC, for only the I datapath or the Q datapath, or for both I and Q.

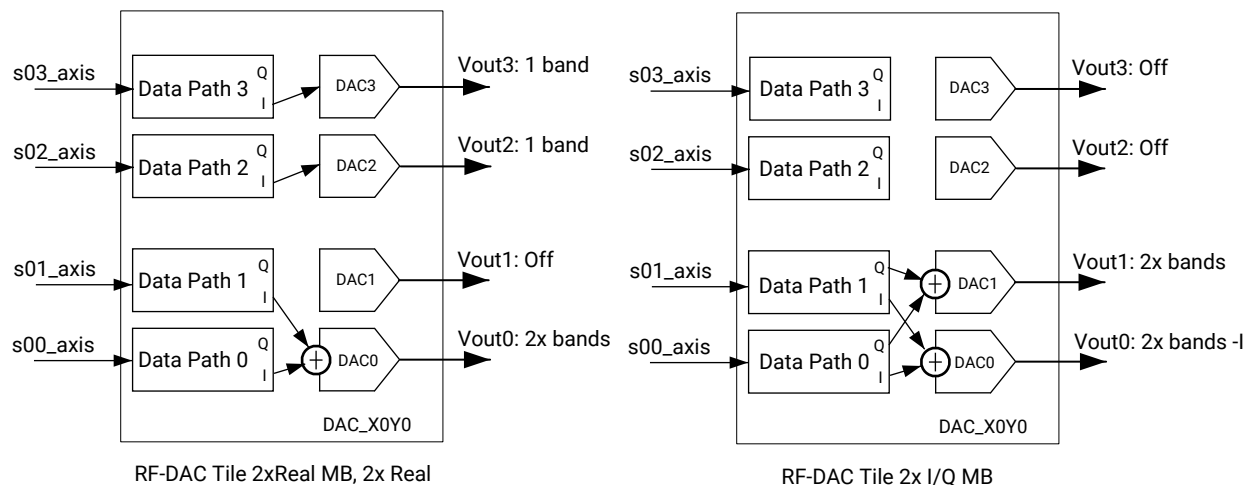
Figure 57: Multi-Band Logic



X18269-081017

RF-DAC multi-band is implemented by connecting multiple RF-DAC DUC blocks to an RF-DAC analog block. Each DUC block handles one band of data, and can mix this up to any carrier frequency. The output is then summed before being sent to the analog datapath and RF-DAC output. This is shown in the following figure.

Figure 58: RF-DAC Digital Datapaths



X19542-011918

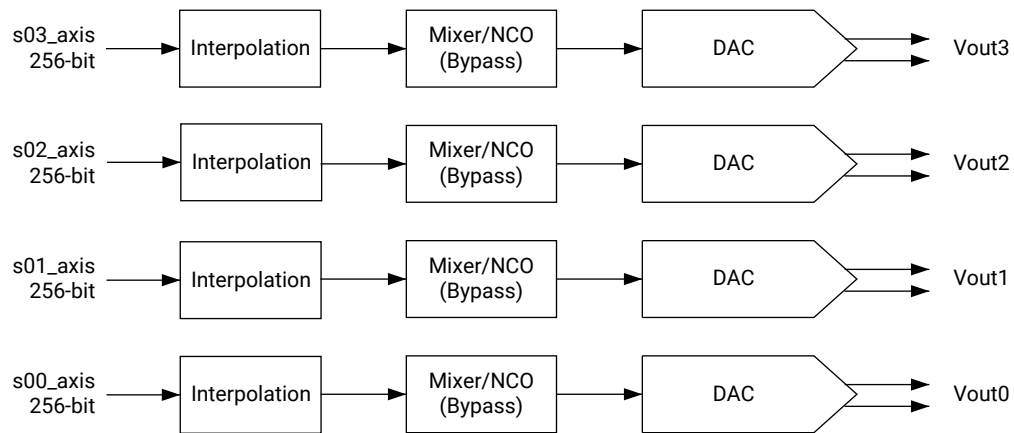


Two example multi-band configurations are shown. The RF-DAC tile consists of four DUC blocks (digital datapaths) and four RF-DAC-analog blocks (analog datapaths). In the left hand example, the lower RF-DAC pair in the tile is configured as 2x real multi-band, while the top pair are independent RF-DACs. Because the bottom pair uses two DUC blocks, DAC1 is off, and DAC0 outputs the dual-band signal. In the right hand example, the lower pair is configured as 2x I/Q multi-band, while the top pair is off.

## RF-DAC Programmable Logic Data Interface

The data interface between the RF-DAC tiles and the PL is implemented using parallel data streams, using the AXI4-Stream protocol. These data streams are input to gearbox FIFOs that provide a flexible interface between the user application and the RF-DAC tile. The maximum interface width is 256 bits per stream, representing up to 16 16-bit little endian words. The data streams and associated FIFOs have a configurable number of words which provide the flexibility to choose between the number of words and the clock frequency to interface with the PL design. There are four streams per tile, and the naming convention is `sXY_axis`, where X represents the RF-DAC tile number, and Y represents a stream input to the FIFO in that tile. The following figure shows the interfaces.

Figure 59: RF-DAC Data Interfaces



X19539-081919

### Interface Data Formats

The data streams represent real or I/Q data, depending on the RF-DAC tile configuration. For RF-DAC tiles, a given stream is either real or I/Q interleaved. If an RF-DAC is configured with I/Q input data then the even numbered samples of the stream represent I data and the odd-numbered samples represent Q data. These real and I/Q configurations are shown in RF-DAC IP Configuration.

In each configuration all the enabled RF-DAC FIFOs in a tile begin to accept data when the power-up sequence has completed. This is indicated by the assertion of the `sXY_axis_tready` outputs. If the data for any stream is not valid at this point, logic to suppress it should be included in the PL.

### Related Information

[RF-DAC IP Configuration](#)

## RF-DAC Interface Data and Clock Rates

The total data rate per channel to the PL is determined by a number of factors, RF-DAC sample rate, interpolation factor, and I/Q or Real data formats. The gearbox FIFOs provide a way of interfacing this data rate to the clock frequency of the PL design, by allowing the number of words per clock to be chosen. The only requirements are that the interface number of words and clock rate combine to match the data rate required by the RF-DAC channel, and all RF-DACs in a tile share a common interface clock frequency. This is shown by the following equations:

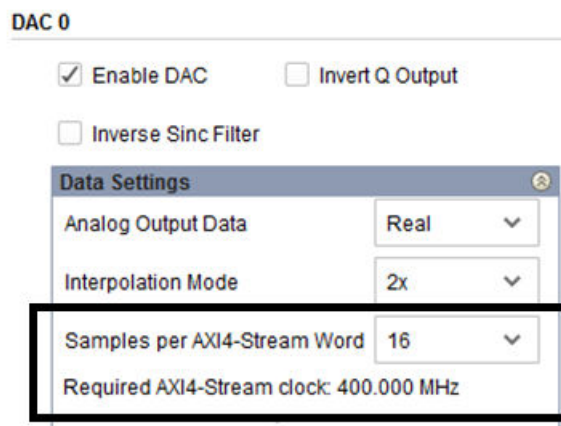
$$PL_{DataRate} = (DAC_{DataRate} \times IQMode) / InterpolationRate$$

$$PL_{Fclock} \times PL_{NumWords} = PL_{DataRate}$$

$$PL_{Fclock} = PL_{DataRate} / PL_{NumWords}$$

The IP core automatically calculates the data rates based on the RF-DAC sample rate and datapath settings. This is shown in the following figure.

Figure 60: RF-DAC Data Interface Data and Clock Configuration



Each tile has independent clocking; sample rates, clock rates, PL rates, and configurations can be specified on a per-tile basis.

### Related Information

[RF-DAC Converter Configuration](#)

## ***PL Clock Interface***

The AXI4-Stream data for all four tile streams is synchronous to a clock from the PL, which has a naming convention of `sX_axis_clk`, where X represents the RF-DAC tile number. This clock must be at the frequency specified by the Required AXI4-Stream clock displayed on the IP core configuration screen.

The RF-DAC tile also outputs a clock that can be used by the PL. This output clock is a divided version of the RF-DAC sample clock and therefore is frequency locked to it. This clock has a naming convention of `clk_dacX`, where X represents the RF-DAC tile number.

### **Related Information**

[Clocking](#)

## ***Interface FIFO Overflow***

The data rate through the interface gearbox FIFOs must be constant during runtime of the RF-DAC tile, with no frequency drift between the PL clock and RF-DAC analog sample clock domains. If there is a frequency mismatch between these domains, a FIFO overflow can occur. The interface FIFOs have a built-in feature to determine if FIFO overflow has occurred, which is flagged to the PL through the IP interrupt mechanism.

There are two types of overflow: actual and marginal. Actual overflow indicates that the FIFO read/write pointers are overlapping, which means data is not being transferred safely between domains, and action must be taken. Marginal overflow is a warning and indicates that the FIFO read/write pointers are close to overlapping. Overflow should not occur during normal operation, and if overflow is seen it is an indication that the clocking infrastructure of the PL/PCB/IP core is incorrectly configured.

### **Related Information**

[Interrupt Hierarchy](#)

## ***Synchronization***

The gearbox FIFOs provide a flexible data and clock interface for the RF-DAC tiles. However, as with all dual-clock FIFOs, latency might vary between one tile and another. While all channels within a tile have the same latency, some applications might require more than one RF-DAC tile to be used, and require the latencies to be matched across all RF-DAC channels. These applications can use the Multi-Tile Synchronization (MTS) feature to achieve this inter-tile synchronization.

### **Related Information**

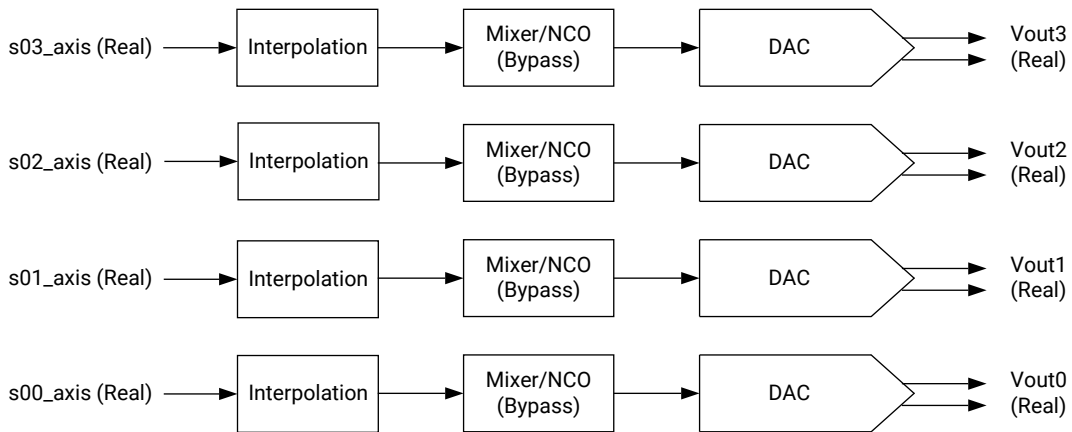
[Multi-Tile Synchronization](#)

## RF-DAC IP Configuration

The RF-DAC can be configured in several modes. The basic configuration options are available on the IP core configuration screen in the Vivado® IDE and advanced operating modes can be configured using the RFdc driver API.

### RF-DAC Real Input to Real Output

Figure 61: RF-DAC Real Input to Real Output



X18284-082719

Figure 62: RF-DAC Real Input to Real Output IP Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [0.0 - 6.554]    Clock Out (MHz)

Reference Clock (MHz)     AXI4-Stream Clock (MHz)

**Multi Tile Sync**

Enable Multi Tile Sync

**Converter Configuration**

DAC Pair 0,1

DAC Pair 2,3

**DAC 0**

Enable DAC     Invert Q Output

Inverse Sinc Filter

**Data Settings**

Analog Output Data

Interpolation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 400.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

**Analog Settings**

Nyquist Zone

Decoder Mode

**DAC 1**

Enable DAC     Invert Q Output

Inverse Sinc Filter

**Data Settings**

Analog Output Data

Interpolation Mode

Samples per AXI4-Stream Cycle

**Mixer Settings**

Mixer Type

Mixer Mode

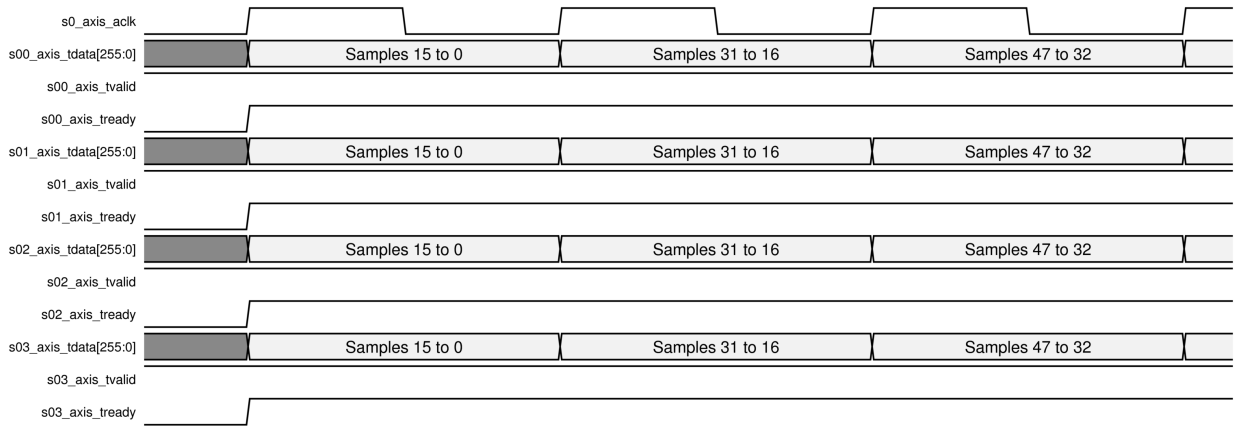
**Analog Settings**

Nyquist Zone

Decoder Mode

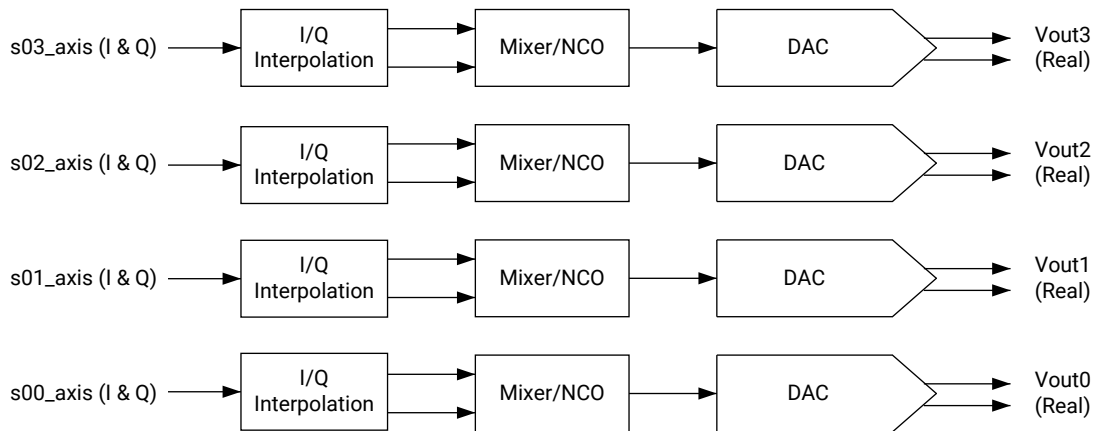
The following figure shows an RF-DAC with real input to real output, 1x interpolation, the mixer bypassed, and running at a 400 MHz AXI4-Stream clock.

Figure 63: RF-DAC Real Input to Real Output Data Timing



### RF-DAC I/Q Input to Real Output

Figure 64: RF-DAC I/Q Input to Real Output



X18279-082719

Figure 65: RF-DAC I/Q Input to Real Output IP Core Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [0.0 - 6.554]      Clock Out (MHz)

Reference Clock (MHz)       AXI4-Stream Clock (MHz)

---

**Multi Tile Sync**

Enable Multi Tile Sync

---

**Converter Configuration**

**DAC Pair 0,1**    **DAC Pair 2,3**

**DAC 0**

Enable DAC     Invert Q Output

Inverse Sinc Filter

**Data Settings**

Analog Output Data

Interpolation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 400.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

NCO Phase

**Analog Settings**

Nyquist Zone

Decoder Mode

**DAC 1**

Enable DAC     Invert Q Output

Inverse Sinc Filter

**Data Settings**

Analog Output Data

Interpolation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 400.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

NCO Phase

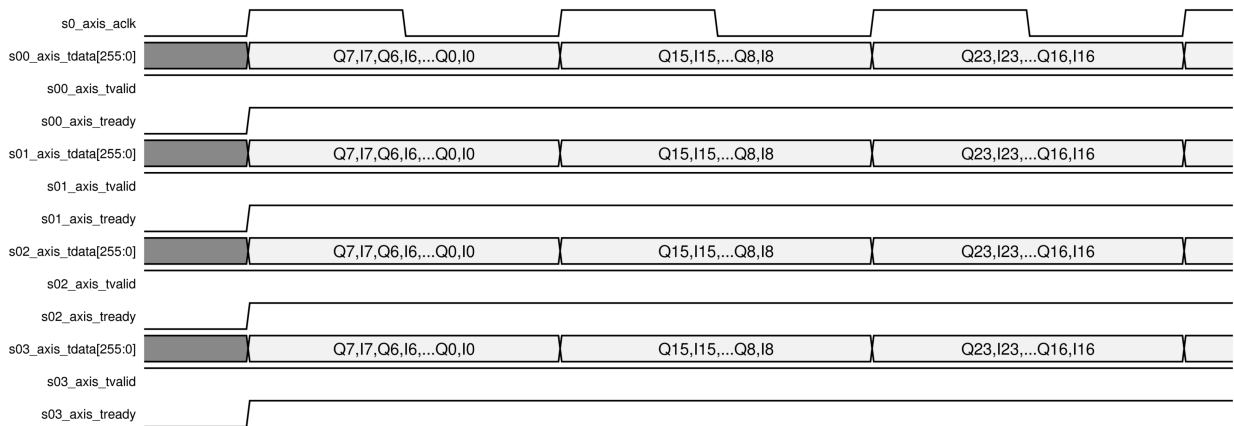
**Analog Settings**

Nyquist Zone

Decoder Mode

The following figure shows an RF-DAC with I/Q input to real output, 2x interpolation, the mixer bypassed, and running at a 400 MHz AXI4-Stream clock.

Figure 66: RF-DAC I/Q Input to Real Output Timing

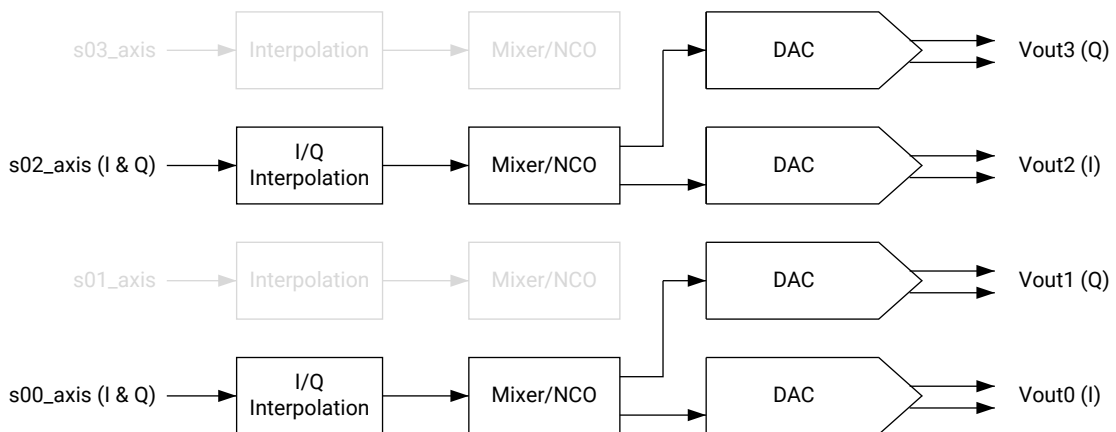


**Note:** Interpolation is x2 because the bandwidth available on the AXI4-Stream interface is limited.

### RF-DAC I/Q Input to I/Q Output

For I/Q input to I/Q output, the RF-DACs are paired. An I/Q output signal requires a pair of RF-DACs, one for I data and one for Q data. As shown in the following figure, a single DUC block outputs I and Q data and these signals are sent to the two RF-DACs.

Figure 67: RF-DAC I/Q Input to I/Q Output



X18280-082719



Figure 68: RF-DAC I/Q Input to I/Q Output IP Core Configuration

**PLL and Clocking Configuration**

Enable PLL

Sampling Rate (GSPS)  [0.0 – 6.554]      Clock Out (MHz)

Reference Clock (MHz)       AXI4-Stream Clock (MHz)

---

**Multi Tile Sync**

Enable Multi Tile Sync

---

**Converter Configuration**

**DAC Pair 0,1**    **DAC Pair 2,3**

**DAC 0**

Enable DAC     Invert Q Output

Inverse Sinc Filter

**Data Settings**

Analog Output Data

Interpolation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 400.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

NCO Phase

**Analog Settings**

Nyquist Zone

Decoder Mode

**DAC 1**

Enable DAC     Invert Q Output

Inverse Sinc Filter

**Data Settings**

Analog Output Data

Interpolation Mode

Samples per AXI4-Stream Cycle

Required AXI4-Stream clock: 400.000 MHz

**Mixer Settings**

Mixer Type

Mixer Mode

NCO Frequency (GHz)

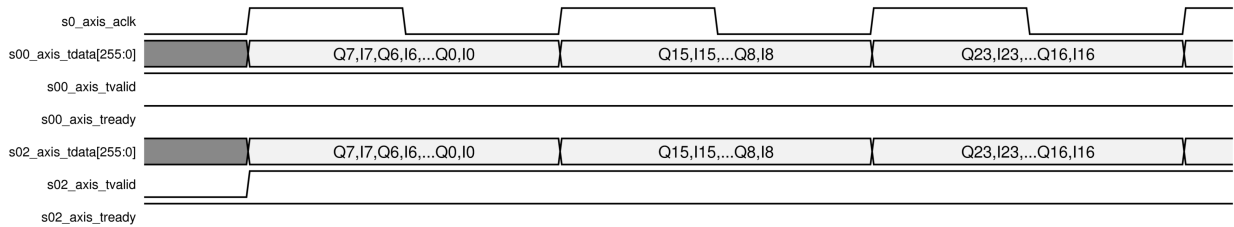
NCO Phase

**Analog Settings**

Nyquist Zone

Decoder Mode

The following figure shows an RF-DAC with I/Q input data to I/Q output data with 2x interpolation, the mixer bypassed, and with a 400 MHz AXI4-Stream clock.

**Figure 69: RF-DAC I/Q Input to I/Q Output Data Timing**


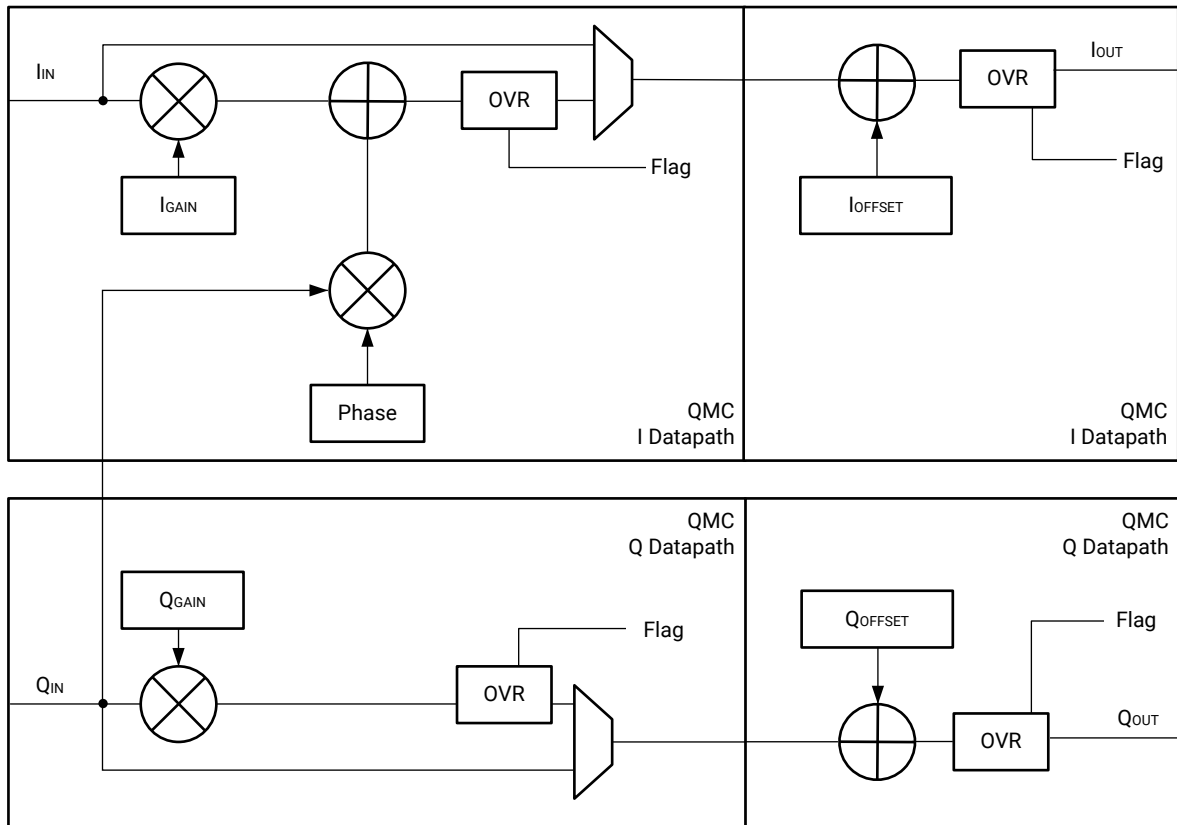
**Note:** Interpolation is x2 because the bandwidth available on the AXI4-Stream interface is limited.

## Quadrature Modulator Correction

When using an external analog quadrature mixer device, a pair of converters (either RF-ADC or RF-DAC) must be used to handle the I and Q datapaths after conversion. Due to external events or circumstances, errors or imbalances can be introduced in the analog I and Q signal paths which, if not corrected, can lead to system performance degradation. The necessary corrections to restore any system from degrading are accomplished using the quadrature modulator correction (QMC) circuit.

As shown in the following figure, the QMC circuit is for correction only. Error and/or imbalance detection must be done by application-specific code in the internal interconnect logic.

Figure 70: Converter QMC Circuitry



X18260-082819

The QMC circuit can compensate for the following:

- Gain correction is done by multiplying the signal by a gain factor. This factor has a range of 0 to 2.0, and individual factors can be applied to the I and Q datapaths. The output resolution of the block is 16 bits.
- Phase correction is achieved by adding a scaled fraction of Q to the I value. The result of this addition can result in a gain error that must be corrected by the gain error correction block. Phase correction has a range of approximately  $\pm 26$  degrees.
- Offset correction is done by adding a fixed LSB value to the sampled signal. Range is -2048 to 2047

For full QMC functionality, a pair of converters need to be set up and used in I/Q mode. When disabling the phase correction factor, the QMC block can supply gain and offset correction to a converter used in real mode. In this case, the converters do not need to be paired to make use of the QMC block functions.

## Update QMC Settings

The gain, phase, and offset correction factor values can be set using the RFdc driver API. An example of using the API is as follows.

```
// Initial Setup

XRFdc_QMC_Settings QMC_Settings_I, QMC_Settings_Q; // RF-ADC block0 is I,
RF-ADC
block1 is Q
QMC_Settings_I.EventSource = XRFDC_EVNT_SRC_TILE; // QMC Settings are
updated with a
tile event
QMC_Settings_Q.EventSource = XRFDC_EVNT_SRC_TILE;

....
// Update Gain/Phase/Offset for I/Q RF-DACs in tile0

QMC_Settings_I.GainCorrectionFactor = 0.9; // Set Gain for I
QMC_Settings_I.PhaseCorrectionFactor = -5.0; // I/Q imbalance factor
applied to I
side, approx in degrees.
QMC_Settings_I.EnableGain           = 1;
QMC_Settings_I.EnablePhase         = 1;
QMC_Settings_Q.GainCorrectionFactor = 0.95;
QMC_Settings_Q.EnableGain          = 1;
XRFdc_SetQMCSettings(ptr, XRFDC_ADC_TILE, 0, 0, &QMC_Settings_I); // Write
settings
for ADC0,0 - I ADC
XRFdc_SetQMCSettings(ptr, XRFDC_ADC_TILE, 0, 1, &QMC_Settings_Q); //Write
settings
for ADC0,1 - Q ADC
XRFdc_UpdateEvent(ptr, XRFDC_ADC_TILE, 0, 0, XRFDC_EVENT_QMC); //Generate a
Tile
Update Event - applies all QMC Settings at once
```

It is also possible to read back the QMC settings from any converter, using the `XRFdc_GetQMCSettings` RFdc driver API command. This populates the `QMC_Settings` structure with the values from the hardware.

### Related Information

[XRFdc\\_GetQMCSettings](#)

[XRFdc\\_SetQMCSettings](#)

[XRFdc\\_UpdateEvent](#)

## I/Q RF-ADC Naming Convention

When converters are used in an I/Q pair, the I channel always uses the even block numbers, and the Q channel always uses the odd block numbers. For example, in a Dual RF-ADC tile, the I RF-ADC is block0, and the Q RF-ADC is block1. For Quad RF-ADC tiles, or RF-DAC tiles, the I channels are block0 and block2, while the related Q channels are block1 and block3, respectively.

## QMC Overflow

Setting excessive gain, phase or offset correction factors relative to the input signal can cause the datapath to overflow. To aid system debug, the QMC block contains built-in overflow detection and saturation blocks. These blocks generate flags which are connected to the IP interrupt mechanism using the datapath interrupt. The related interrupt flags for the QMC block are:

XRFDC\_IXR\_QMC\_GAIN\_PHASE\_MASK

XRFDC\_IXR\_QMC\_OFFST\_MASK

These interrupts can be enabled per converter channel.

- The gain correction multiplier range is between 0 and 2.0.
- The range of the phase correction multiplier is approximately  $\pm 26$  degrees, or  $\pm 0.5$  in terms of magnitude.

### Related Information

[Interrupt Hierarchy](#)

## Coarse Delay

Coarse delay allows the adjusting of the delay in the digital datapath which can be useful to compensate for delay mismatch in a system implementation. The compensation here is limited to some period of the sampling clock. For PCB design and flight time information for correct delay adjustment see the *UltraScale Architecture PCB Design User Guide (UG583)*. The following table shows the number of periods of the sampling clock ( $T_1$  or  $T_2 = 2 * T_1$ ) for this delay tuning capability.

*Table 50: Delay Tuning Sampling Clock Periods*

Tile Type	Digital Control	Coarse Delay Step
Dual RF-ADC	0 to 7	T2
Quad RF-ADC	0 to 7	T1
RF-DAC	0 to 7	T1

## Dynamic Update Events

Certain datapath features in the RF-ADC and RF-DAC tiles can be updated during runtime. Depending on the application, it might be required to update a setting immediately or to wait for a certain event to apply the update. For example, when resetting the NCO phase, using the SYSREF event allows the phase reset action to be applied at the same time across multiple tiles. The following table shows the available events types.

Table 51: Event Types

Event Name	Function	Event Source	Comment
Immediate	Updates the settings of a given feature immediately after writing.	RFdc Driver	Does not apply for Dual RF-ADC tiles
Block	Updates when the block event register is written. Allows features of a given block to update at the same time.	RFdc Driver	Does not apply for Dual RF-ADC tiles
Tile	Updates when the tile event register is written. Allows features in all blocks in a tile to update at the same time.	RFdc Driver	
PL	Updates when the PL_event input on a tile is asserted. Allows features across multiple blocks/tiles to be updated. <b>Note:</b> This event does not guarantee synchronized updates across tiles.	dacXY_pl_event or adcXY_pl_event input	Asserted from user design in the PL <sup>1</sup>
SYSREF <sup>2</sup>	Updates when a SYSREF is received through the SYSREF_p/n input pins. Allows features across multiple blocks/tiles/devices to update at the same time. The updates are requested using a corresponding API function call, such as XRFDC_ResetNCOPhase to arm the event for all the block/tiles being requested/armed. The host processor checks the return of the API call to ensure that all of the events arming are completed before issuing the SYSREF pulse if synchronization is wanted across blocks, tiles and devices.	SYSREF_p/n input	Asserted from the SYSREF generator on the PCB (Single or Multi-device) or from API call (Single device only)
PL Marker	Updates when the user_sysref_dac signal is asserted. This signal is synchronous and aligned to the RF-DAC data. For Multi-Tile Synchronization (MTS), this trigger is not recommended because of FIFO skew adjustment between tiles and devices.	user_sysref_dac IP input	Applies only to RF-DAC tiles

**Notes:**

1. These signals are synchronous to the converter output clocks, `clk_dacX` and `clk_adcX`. These clocks are on local routing. A small amount of logic can be clocked on the output clocks and used to drive the PL event signals. This logic should be placed near the converter to facilitate timing closure. Alternatively the PL event signals can be generated on a global clock and then synchronized to the output clock of the relevant converter.
2. The SYSREF signal used here is coming from analog SYSREF in the RF-DAC tile master. When using SYSREF as a trigger event for dynamic updates to synchronize multiples devices, analog SYSREF must be DC-coupled with the ability to operate in pulse mode; that is, the SYSREF signal is a gapped pulse instead of a continuous clock. This is different from the SYSREF requirement for the MTS function which requires a continuous clock for the duration of the MTS process. In order to support both MTS and SYSREF dynamic updates, the external SYSREF clock supplied has to support both modes. See the Multi-Tile Synchronization section for further details.

Features that are intended to be dynamically updated must be programmed to be sensitive to one of these update events. The update is a one-shot event. The first trigger latches in the update after it has been armed. The event source for each feature is flexible and can be individually set. The following features support this dynamic update:

- Fine Mixer / NCO
- Quadrature Modulator Correction (QMC)
- Coarse Delay

### Related Information

[Multi-Tile Synchronization](#)

## Update Event Use

The RFdc driver API uses the `EventSource` parameter of a particular function to set up the trigger event for that function. It can also issue a subset of the events. The following code shows an example of the setup of a tile event source followed by the tile event.

```
// Initial Setup - Mixer settings for 2 DDC blocks

XRFdc_Mixer_Settings Mixer_Settings_0;
XRFdc_Mixer_Settings Mixer_Settings_1;

// Mixer Settings for both DDC blocks are updated with a tile event

Mixer_Settings_0.EventSource = XRFDC_EVNT_SRC_TILE;
Mixer_Settings_1.EventSource = XRFDC_EVNT_SRC_TILE;
...
// Make changes to mixer settings and write them to both DDCs

Mixer_Settings_0.freq = 1.23;
Mixer_Settings_1.freq = 1.43;
XRFdc_SetMixerSettings (ptr, XRFDC_ADC_TILE, 0, 0, &Mixer_Settings_0);
XRFdc_SetMixerSettings (ptr, XRFDC_ADC_TILE, 0, 1, &Mixer_Settings_1);

// Generate the event for DDC block 0
// This maps to the Tile event, which updates both DDCs at the same moment
XRFdc_UpdateEvent(ptr, XRFDC_ADC_TILE, 0, 0, XRFDC_EVENT_MIXER);
```

### Related Information

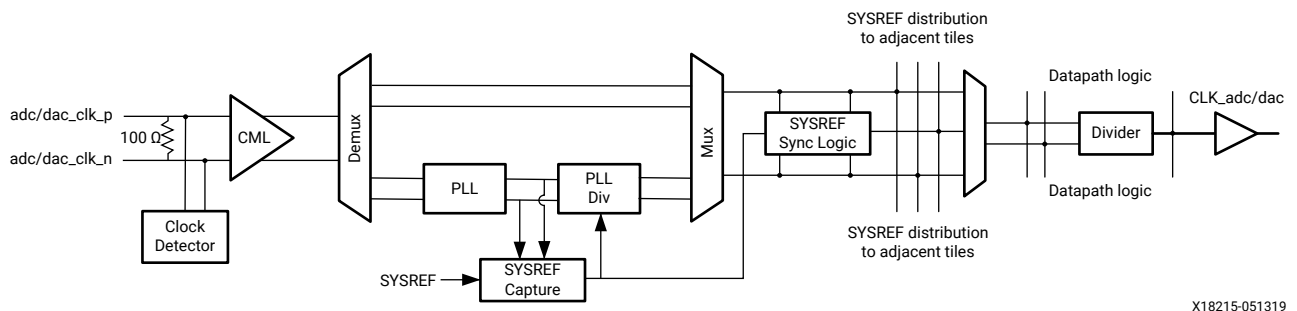
[XRFdc\\_SetMixerSettings](#)

[XRFdc\\_UpdateEvent](#)

# PLL

Each RF-ADC and RF-DAC tile includes a clocking system with an input clock divider, a PLL, and an output divider. When used with the SYSREF input, the clocking system can be synchronized in multi-tile or multi-device designs. When the PLL is used, the output sampling clock is fed to the multiplexer, which can also be used by a direct input clock before it is routed into the clock network of the tile (as shown below). The clock from the PLL is used in both the analog and digital portions of the RF-ADC or RF-DAC in a tile. Each RF-ADC or RF-DAC in a tile has a local divider to divide and distribute the very low jitter clock from the PLL or a direct input to the different functions in each RF-ADC or RF-DAC function.

Figure 71: Tile PLL Clocking Structure

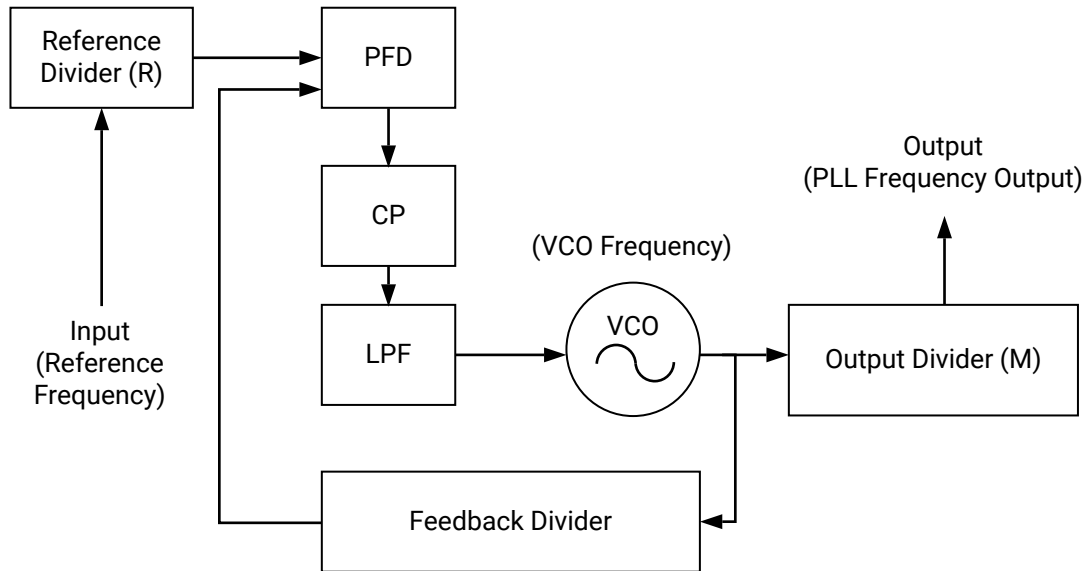


The operating parameters of the PLL are set using the Vivado® IDE to specify the default PLL configuration or by the RFdc driver API if runtime adjustment of the PLL is required. The PLL must first be enabled in the Vivado IDE if runtime adjustment using the API is required.

The internal PLL block diagram is shown in the following figure. When the internal PLL is used, the IP wizard or the API function dedicated to configuring the PLL system sets a reference divider value as an integer, a feedback divider value as an integer, and an output divider value as an integer to achieve the best performance of the PLL using the VCO in the correct range. In any frequency configuration, the best performance of phase noise is achieved when the PLL system is able to select a reference divider of 1. Allowed value for frequency ranges and integer values of dividers are specified in *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)*. For any internal PLL use, the frequency system has the formula  $F_s = (F_{in}/R) \cdot (FBDiv/M)$  as shown in the diagram below where  $F_s$  is the PLL frequency output and  $F_{in}$  is the reference frequency.



Figure 72: PLL System



X20569-050919

## PLL Parameters

The following table shows key parameters of the on-chip PLL.

Table 52: PLL Parameters

Reference Divider (R)	Feedback Divider (N)	Output Divider (M)	VCO Frequency (GHz)
1 to 4	13 to 160	2,4,6,8..., (even number $\leq 64$ )	8.5 to 13.2

## Interrupt Handling

The RF-ADC and RF-DAC tiles can generate interrupts during operation which can help debug or avoid potential issues. The interrupts are shown in the following table.

Table 53: Interrupts

Interrupt	Description	Interrupt Mask Macro	Mask Value
RF-ADC/RF-DAC datapath interrupt	IP register interrupt at register 0x208 0x210 0x218 0x220 This interrupt indicates that one of the datapath interrupts is active. To clear this interrupt, all datapath sub-interrupts must be cleared.	XRFDC_ADC_DAT_OVR_MASK	0x40000000U
Overflow in RF-DAC Interpolation Stage 0/1/2/3 I or Q datapath	Indicates that one of the interpolation stages has overflowed/saturated. Flags are per-stage and I / Q paths to indicate where the overflow has occurred. Data amplitude is too high.	XRFDC_DAC_IXR_INTP_I_STG0_MASK	0x00000010U
		XRFDC_DAC_IXR_INTP_I_STG1_MASK	0x00000020U
		XRFDC_DAC_IXR_INTP_I_STG2_MASK	0x00000040U
		XRFDC_DAC_IXR_INTP_Q_STG0_MASK	0x00000080U
		XRFDC_DAC_IXR_INTP_Q_STG1_MASK	0x00000100U
		XRFDC_DAC_IXR_INTP_Q_STG2_MASK	0x00000200U
Overflow in RF-DAC Decimation Stage 0/1/2/3 I or Q datapath	Indicates one of the RF-ADC decimation stages has overflowed/saturated. Flags are per-stage and I/Q paths to indicate where the overflow has occurred. Data amplitude is too high.	XRFDC_ADC_IXR_DMON_I_STG0_MASK	0x00000010U
		XRFDC_ADC_IXR_DMON_I_STG1_MASK	0x00000020U
		XRFDC_ADC_IXR_DMON_I_STG2_MASK	0x00000040U
		XRFDC_ADC_IXR_DMON_Q_STG0_MASK	0x00000080U
		XRFDC_ADC_IXR_DMON_Q_STG1_MASK	0x00000100U
		XRFDC_ADC_IXR_DMON_Q_STG2_MASK	0x00000200U
Overflow in RF-DAC/RF-ADC QMC Gain/Phase	Indicates the QMC gain/phase correction has overflowed/saturated. Data amplitude, or correction factors are too high and should be reduced.	XRFDC_IXR_QMC_GAIN_PHASE_MASK	0x00000400U
Overflow in RF-DAC/RF-ADC QMC Offset	Indicates the QMC offset correction has overflowed/saturated. Data amplitude, or correction factors are too high and should be reduced.	XRFDC_IXR_QMC_OFFST_MASK	0x00000800U
Overflow in RF-DAC Inverse Sinc Filter	Indicates the RF-DAC inverse Sinc filter has overflowed/saturated. Data amplitude is too high and should be reduced. See "RF DAC Inverse Sinc Filter, Overflow"	XRFDC_DAC_IXR_INVSNC_OF_MASK	0x00001000U
Sub RF-ADC0/1/2/3 Over/Under range Interrupt	Indicates the analog input is exceeding the full-scale range of the Sub-RF-ADC. The input signal amplitude should be reduced. See "Over Range"	XRFDC_SUBADC0_IXR_DCDR_OF_MASK	0x00010000U
		XRFDC_SUBADC0_IXR_DCDR_UF_MASK	0x00020000U
		XRFDC_SUBADC1_IXR_DCDR_OF_MASK	0x00040000U
		XRFDC_SUBADC1_IXR_DCDR_UF_MASK	0x00080000U
		XRFDC_SUBADC2_IXR_DCDR_OF_MASK	0x00100000U
		XRFDC_SUBADC2_IXR_DCDR_UF_MASK	0x00200000U
		XRFDC_SUBADC3_IXR_DCDR_OF_MASK	0x00400000U
		XRFDC_SUBADC3_IXR_DCDR_UF_MASK	0x00800000U

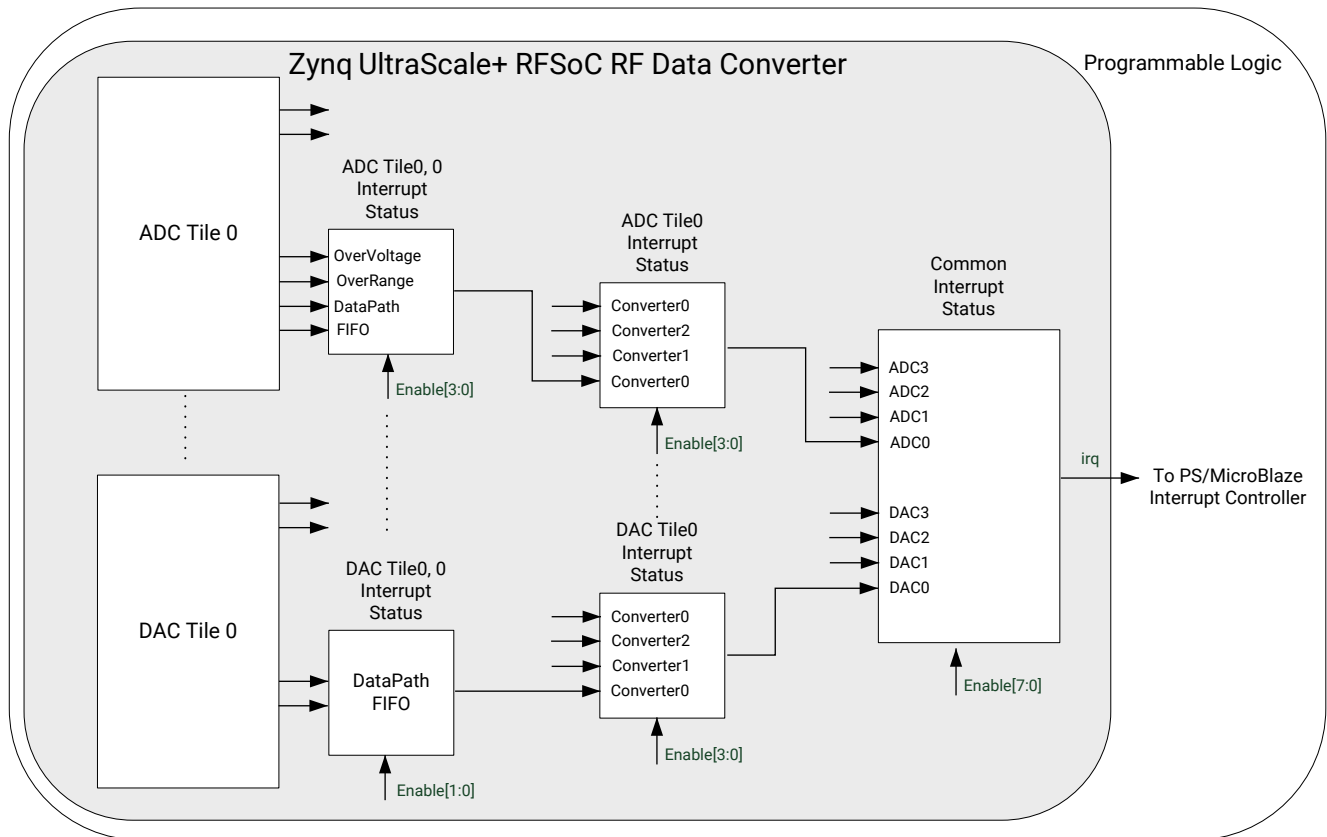
Table 53: Interrupts (cont'd)

Interrupt	Description	Interrupt Mask Macro	Mask Value
RF-ADC over range	Indicates the analog input is exceeding the full-scale range of the RF-ADC. The input signal amplitude should be reduced. To clear this interrupt, the sub-RF-ADC Over range interrupts must be cleared. See "Over Range".	XRFDC_ADC_OVR_RANGE_MASK	0x08000000U
RF-ADC over voltage	Indicates the analog input is exceeding the safe input range of the RF-ADC input buffer and the buffer has been shut down. The input signal amplitude and common mode should be brought within range. See "Over Voltage".	XRFDC_ADC_OVR_VOLTAGE_MASK	0x04000000U
RF-ADC/RF-DAC FIFO over/underflow	IP register interrupt at register 0x208 0x210 0x218 0x220 This interrupt indicate one of the datapath interrupts is active. To clear this interrupt, all datapath sub-interrupts must be cleared.	XRFDC_ADC_FIFO_OVR_MASK	0x80000000U
RF-ADC/RF-DAC Overflow	Indicates that the FIFO interface is incorrectly setup, clocks/data throughput mismatch.	XRFDC_IXR_FIFOUSRDAT_OF_MASK	0x00000001U
RF-ADC/RF-DAC Underflow		XRFDC_IXR_FIFOUSRDAT_UF_MASK	0x00000002U
RF-ADC/RF-DAC Marginal Overflow		XRFDC_IXR_FIFOMRGNIND_OF_MASK	0x00000004U
RF-ADC/RF-DAC Marginal Underflow		XRFDC_IXR_FIFOMRGNIND_UF_MASK	0x00000008U

## Interrupt Hierarchy

The following figure shows the interrupt hierarchy within the IP core.

Figure 73: Interrupt Hierarchy



X19541-050919

Datapath overflow indicates that a sub-block in the signal chain has detected that the output signal amplitude has exceeded full scale, and has been saturated. If overflow is detected it indicates that either the signal amplitude is too high, or the block settings are incorrect for the signal amplitude. For example, using a large gain correction factor on a signal that is close to full-scale causes a QMC gain overflow.

All of the interrupts shown in the previous figure are enabled and handled by the interrupt functions of the RFdc driver API. The default interrupt status handler (`XRFdc_IntrHandler`) automatically follows the interrupt hierarchy in the previous figure to determine the source of the interrupt, and then passes this information to the user-defined status handler to respond.

### Related Information

- [Quadrature Modulator Correction](#)
- [RF-ADC Decimation Filters](#)
- [RF-DAC Interpolation Filters](#)
- [Clocking](#)

## Interrupt Setup and Handling Example

The following code shows an interrupt setup and handling example.

```
//
// Handler
//
// User-Defined interrupt status handler

void RFdcHandler (void *CallBackRef, u32 Type, int Tile, u32 Block, u32
Event) {
// Check the type of interrupt event

    if (Type == XRFDC_DAC_TILE) {
        xil_printf("\nInterrupt occurred for ADC%d,%d :", Tile_Id, Block_Id);
        if(Event & (XRFDC_IXR_FIFOUSRDAT_OF_MASK |
XRFDC_IXR_FIFOUSRDAT_UF_MASK)) {
            xil_printf("FIFO Actual Overflow\r\n");
        }
        if(Event & (XRFDC_IXR_FIFOMRGNIND_OF_MASK |
XRFDC_IXR_FIFOMRGNIND_UF_MASK)){
            xil_printf("FIFO Marginal Overflow\r\n");
        }
        if(Event & XRFDC_DAC_IXR_INTP_STG_MASK) {
            xil_printf("Interpolation Stages Overflow\r\n");
        }
        // ... Other handling code ...
    } else {
        xil_printf("\nInterrupt occurred for ADC%d,%d :\r\n", Tile_Id,
Block_Id);
        // ... ADC handling code ...
    }
}

//
// Setup
//
// Register the user-defined interrupt handler

XRFdc_SetStatusHandler(ptr, ptr, (XRFdc_StatusHandler) RFdcHandler);
//
// Setup for RF-DAC tile 0, block 0 - FIFO Interrupt
//
XRFdc_IntrEnable(ptr, XRFDC_DAC_TILE, 0, 0, XRFDC_IXR_FIFOUSRDAT_MASK);
//
// Test
//
// Force a FIFO interrupt by setting the FIFO write words to a different
value from
the design requirements.
// Note: Assumes a value of 1 is invalid for this design.
XRFdc_SetFabRdVldWords(ptr, 0, 0, 1);
```

This example shows the RFdc driver-specific interrupt handling and the forcing of an interrupt by changing the FIFO parameters. The setup and registration of the RF Data Converter `irq` output with the PS or MicroBlaze™ interrupt controller is not shown. For more details, see the examples provided with the RFdc driver API (see the *Zynq UltraScale+ Device Technical Reference Manual (UG1085)*).

**Related Information**
[XRFdc\\_IntrEnable](#)
[XRFdc\\_SetFabRdVldWords](#)

# Clocking

The following table describes the clocks associated with the Zynq® UltraScale+™ RFSoc RF Data Converter IP core.

*Table 54: IP Core Clocks*

Clock	Description
s_axi_aclk	AXI4-Lite clock. This clock is common to all tiles in the core and should be driven by the system CPU used for the AXI4-Stream interconnect connected to the AXI4-Lite interface. This clock frequency range is specified in <i>Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)</i> .
dac[3:0]_clk_p/n	Sampling Clock or Reference Clock to internal PLL for the RF-DAC tiles. There are up to four differential clock inputs on the core because there is a maximum of four RF-DAC tiles available. This clock must be driven with the frequency chosen on the IP core configuration screen.
adc[3:0]_clk_p/n	Sampling Clock or Reference Clock to internal PLL for the RF-ADC tiles. There are up to four differential clock inputs on the core because there is a maximum of four RF-ADC tiles available. This clock must be driven with the frequency chosen on the IP core configuration screen.
clk_dac[3:0]	Output clock to interconnect logic from the RF-DAC tiles. This is a user-configurable output clock from the core to the user logic. The use of this clock is optional. It can be used to drive the PL design if another frequency-locked RF-DAC sample clock is not available in the PL. The frequency of this clock output can be chosen from a list of values, related to the sample rate, in the IP core configuration screen.
clk_adc[3:0]	Output clock to interconnect logic from the RF-ADC tiles. This is a user-configurable output clock from the core to the user logic. It can be used to drive the PL design if another frequency-locked RF-ADC sample clock is not available in the PL. The frequency of this clock output can be chosen from a list of values, related to the sample rate, in the IP core configuration screen.
s[3:0]_axis_aclk	Slave AXI4-Stream clocks for RF-DAC data interfaces. There are up to 16 AXI4-Stream interfaces on the core, one per RF-DAC. Each RF-DAC tile has a single clock common to up to four AXI4-Stream data interfaces. All AXI4-Stream data interfaces on a tile must operate at the same clock rate but the bandwidth of each interface can be different because each AXI4-Stream interface can have a different width as specified in the IP core configuration screen. This clock must be frequency-locked to the RF-DAC sample clock as outlined in Interface FIFO Overflow.
m[3:0]_axis_aclk	Master AXI4-Stream clocks for RF-ADC data interfaces. There are up to eight or 16 AXI4-Stream interfaces on the core depending on the chosen device, one per RF-ADC. Each RF-ADC tile has a single clock common to up to two or four AXI4-Stream data interfaces. All AXI4-Stream data interfaces on a tile must operate at the same clock rate; however the bandwidth of each interface can be different because each AXI4-Stream interface can have a different width as specified in the IP core configuration screen. This clock must be frequency-locked to the RF-ADC sample clock as outlined in Interface FIFO Overflow.

**Related Information**
[Interface FIFO Overflow](#)

## Clock Inputs

Each RF-ADC or RF-DAC tile has its own clock input. There is no need to instantiate a clock input buffer in your design because the current-mode logic (CML) clock input buffer is implemented in the tile architecture. The clock input buffer can be used as a sampling clock for the tile or as a reference for the tile PLL. The CML clock input has an on-die differential termination of 100  $\Omega$ .

The clock input buffer has a detection circuit for measuring activity at the external clock inputs. When a clock is present, the input buffer is activated. When no clock is present, the outputs of the clock buffer are forced into a steady state. If the clock is lost, the core shuts the tile down but will restart the tile without user intervention when the clock returns. The Reset Count Register increments each time the core shuts the tile down.

When the PLL is not used, the clock buffer output is passed through a multiplexer into the RF-ADC or RF-DAC tile clock network. When the PLL is used, the output of the clock buffer is driven to the PLL to serve as the reference clock.

In each Zynq® UltraScale+™ RFSoc, there is a dedicated input SYSREF pin pair per package. The SYSREF clocks the multi-tile and multi-device synchronization. For multi-tile designs, the SYSREF connects into a master tile and the signal must be distributed from inside that tile to all other tiles used in the design.

The presence of a clock at the input pins can be checked by using the RFDc software API. If no clock is detected the RF-ADC and RF-DAC blocks affected do not start up and stay in the Clock Detection state.

### Related Information

[Power-up Sequence](#)

[XRFdc\\_GetIPStatus](#)

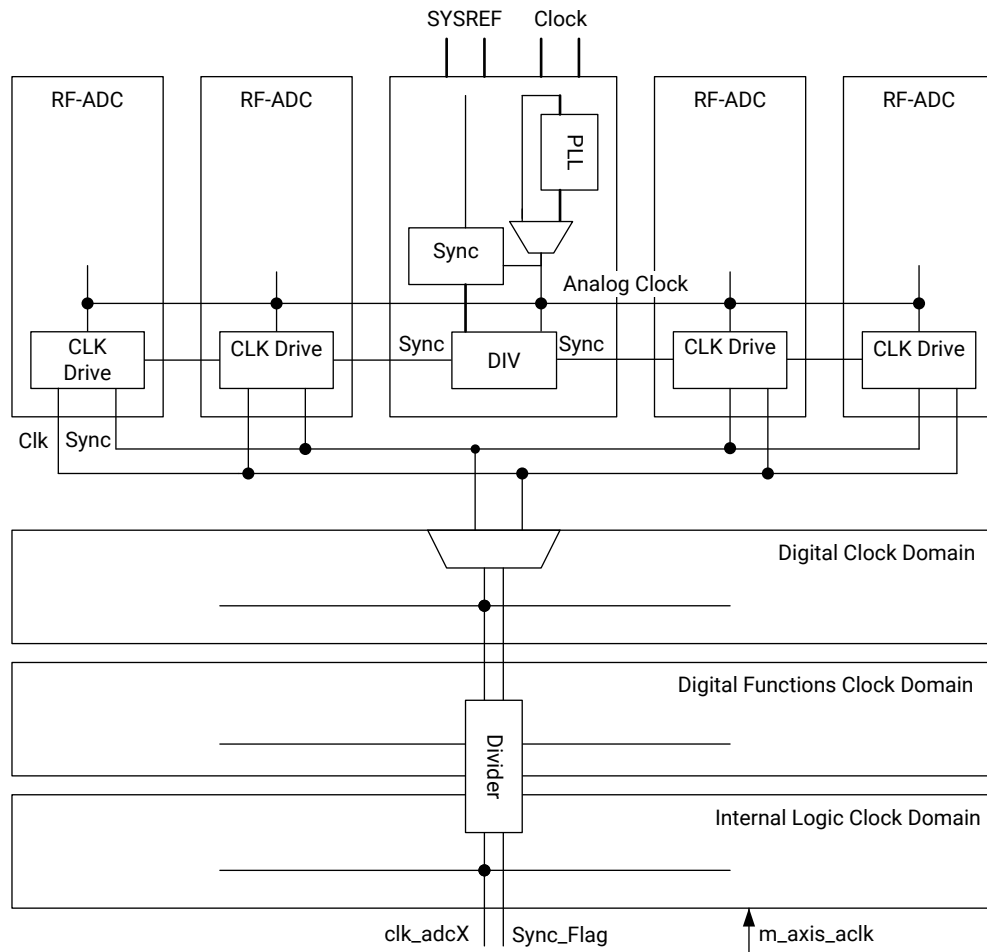
[Reset Count Register \(0x0038\)](#)

## Tile Clocking Structure

The following figures show the tile high-level clocking structure. The clocking structure in a tile starts from a single clock source coming from the tile PLL or straight from the clock input pins.

For a tile to keep a low-jitter and high-speed clocking structure, a single input clock is routed to each RF-ADC or RF-DAC in the tile and divided for the dedicated signal conditioning functions within the RF-ADC or RF-DAC.

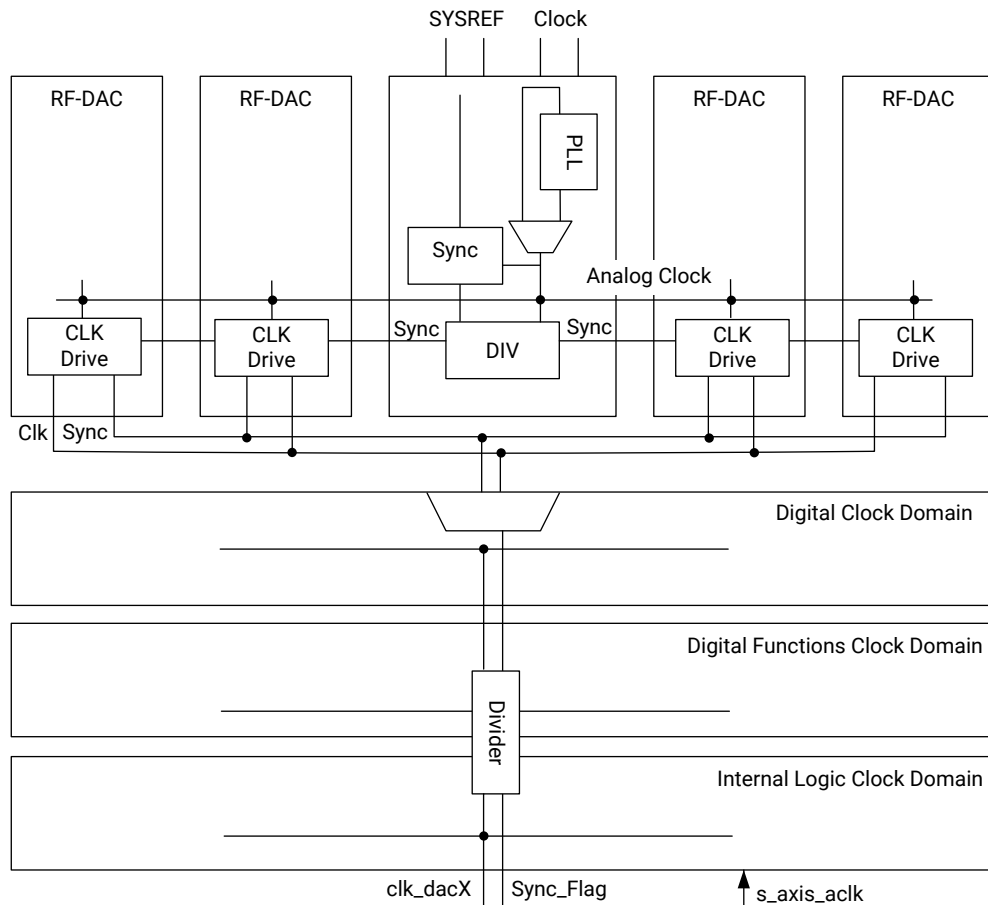
Figure 74: RF-ADC Clock Structure in a Tile



X18249-080917



Figure 75: RF-DAC Clock Structure in a Tile

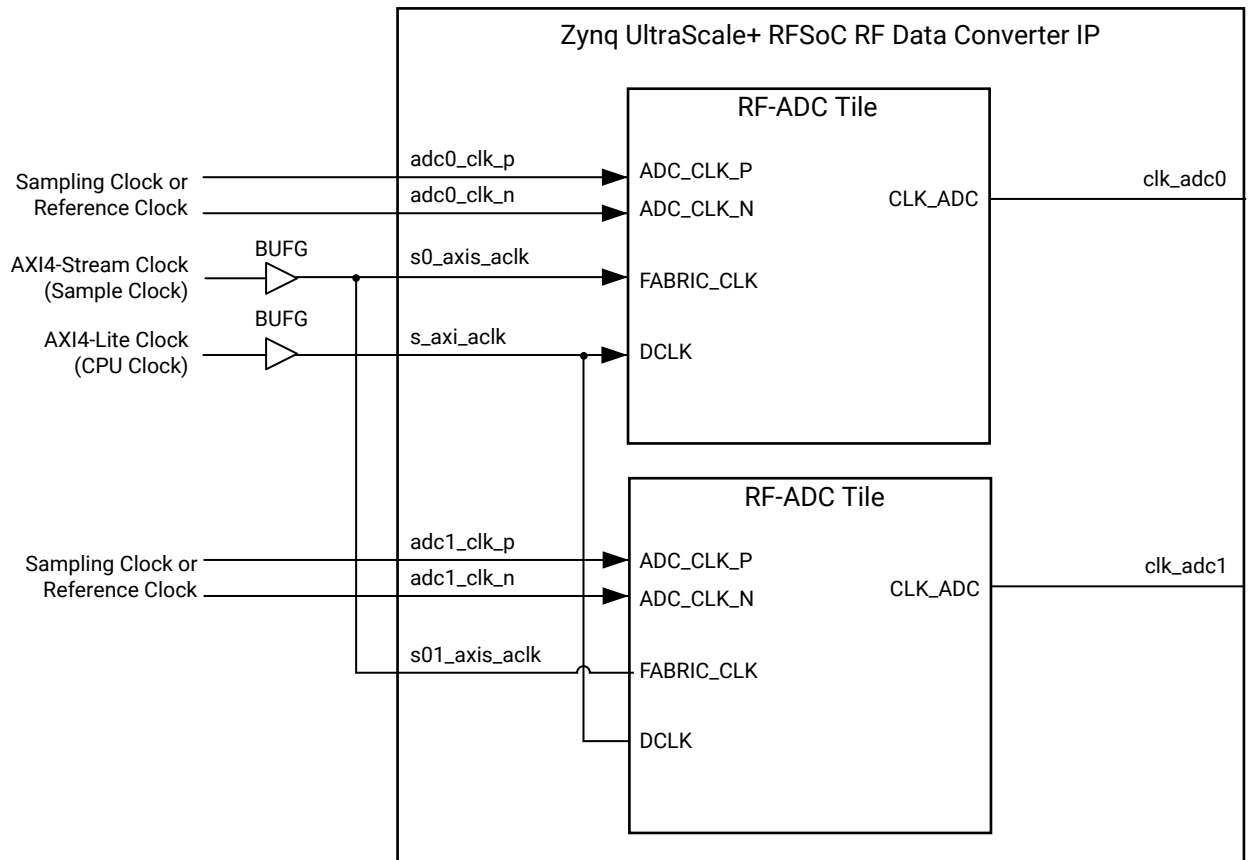


X18216-080917

The clock coming out from the divider of the master tile can be used to drive the PL. A BUFG\_GT buffer is automatically instantiated by the IP core when the output clock is used.

The following figure shows the clock connections for a typical Zynq® UltraScale+™ RFSoc RF Data Converter IP core. The IP core automatically instantiates and connects the constituent RF-ADC and RF-DAC tiles based on the selections in the Vivado® IDE.

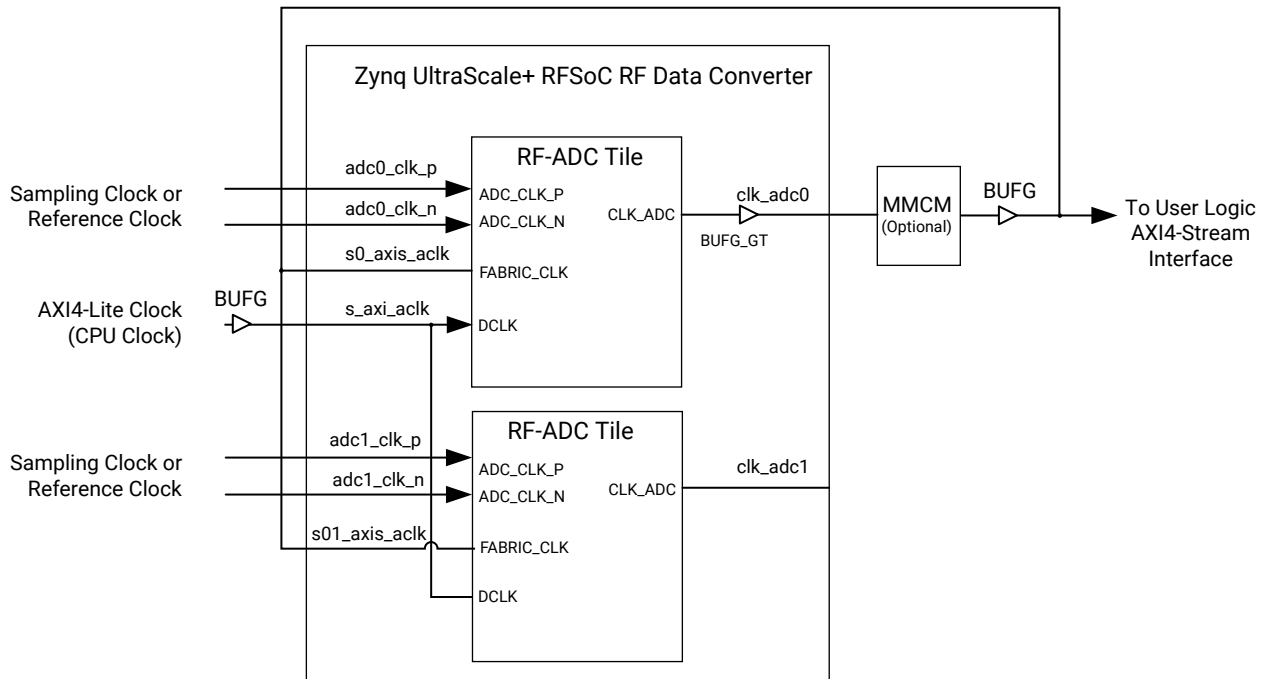
Figure 76: IP Core Clocking



X18271-081117

In most applications the AXI4-Stream clock is common to multiple converters and is supplied from an external clock as shown in the previous figure. This clock must be derived from the same master clock source as the sampling and/or reference clocks to the RF-ADC and RF-DAC tiles. Alternatively, it is possible to use the output of one of the RF-DAC or RF-ADC tiles to generate the AXI4-Stream clocks as shown in the following figure.

Figure 77: Typical Clocking Scheme Using RF-ADC Output Clock as Data Clock



X18272-081117

## Resets

Standard AXI4-Stream resets are provided for each clock domain. Reset of the RF-ADC and RF-DAC is done using the AXI4-Lite interface. A full tile reset is carried out by asserting bit 0 of the Master Reset Register. Individual tiles can be reset and disabled by writing to the Restart registers. Disabling the tile is achieved in a similar way. Follow the individual tile reset procedure to restart the tile after it has been disabled.

### Related Information

[Master Reset Register \(0x0004\)](#)

[Restart All Tiles](#)

[Restart Tile](#)

[Power-Down Tile](#)

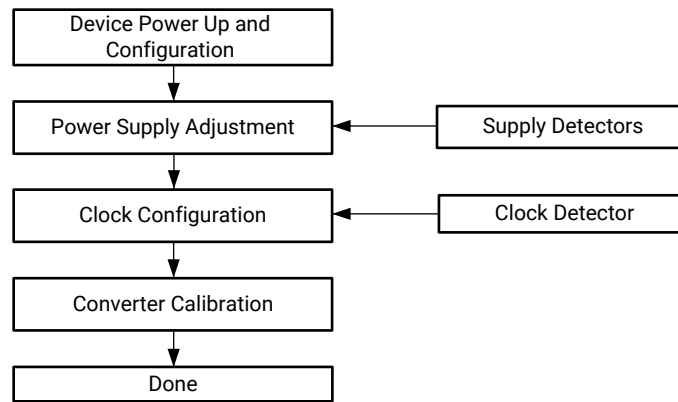
# Power-up Sequence

Tiles consist of different independent blocks, powered from different power supplies and clocked by different versions of the main clock. Tiles must be brought up in a known sequence for the converters to function correctly. The power-up state machine is run automatically when the device is configured or reconfigured with a bitstream but it can also be rerun at any time under software control.



**IMPORTANT!** The IP should only be started when all external clocks are running and stable (glitch-free). The IP AXI4-Lite reset (`s_axi_aresetn`) can be held Low if the startup needs to be delayed until the external clocks are valid.

Figure 78: POR Finite State Machine



X18500-081117

## Power-on Sequence Steps

Table 55: Power-on Sequence Numbers

Sequence Number	State	Description
0-2	Device Power-up and Configuration	The configuration parameters set in the Vivado® IDE are programmed into the converters. The state machine then waits for the external supplies to be powered up. In hardware this can take up to 25 ms. However this is reduced to 200 µs in behavioral simulations.
3-5	Power Supply Adjustment	The configuration settings are propagated to the analog sections of the converters. In addition the regulators, bias settings in the RF-DAC, and the common-mode output buffer in the RF-ADC are enabled.
6-10	Clock Configuration	The state machine first detects the presence of a good clock into the converter. Then, if the PLL is enabled, it checks for PLL lock. The clocks are then released to the digital section of the converters.
11-13	Converter Calibration (ADC only)	Calibration is carried out in the RF-ADC. In hardware this can take approximately 10 ms, however this is reduced to 10 µs in behavioral simulations.

Table 55: Power-on Sequence Numbers (cont'd)

Sequence Number	State	Description
14	Wait for deassertion of AXI4-Stream reset	The AXI4-Stream reset for the tile should be asserted until the AXI4-Stream clocks are stable. For example, if the clock is provided by a MMCM, the reset should be held until it has achieved lock. The state machine waits in this state until the reset is deasserted.
15	Done	The state machine has completed the power-up sequence.

#### Related Information

[Interrupt Hierarchy](#)

[RF-DAC/RF-ADC Tile <n> FIFO Disable Register \(0x0230\)](#)

## Restart All Tiles

To restart all tiles and perform the complete power-on sequence after initialization under software control, perform the following steps:

1. Write 0x0000\_0001 to the Master Reset register to restart all tiles (start and end states revert to their default so there is no need to write 0x0000\_000F to the individual tile Restart State register).
2. Poll the tile Restart Power-On State Machine register for each individual tile to check operation is complete. The power-on sequence is complete when this register reads all zeros.

#### Related Information

[Master Reset Register \(0x0004\)](#)

[Restart State Register \(0x0008\)](#)

[Restart Power-On State Machine Register \(0x0004\)](#)

## Restart Tile

To restart a tile and perform the complete power-on sequence after initialization under software control, perform the following steps:

1. Write 0x0000\_000F to the tile<n> Restart State register.
2. Write 0x0000\_0001 to the tile<n> Restart Power-On State Machine register to restart the tile.
3. Poll the tile<n> Restart Power-On State Machine register to check operation is complete; the power-on sequence is complete when this register reads all zeros.

#### Related Information

[Restart State Register \(0x0008\)](#)

[Restart Power-On State Machine Register \(0x0004\)](#)

## Power-Down Tile

To power down a tile, the power-on sequence is rerun in the same way as restarting a tile but with the End State set to 3 rather than F. Setting the End State to 3 makes the power-on sequencer clear all the registers without performing a full power-on sequence. Perform the following steps to power down a tile:

1. Write 0x0000\_0003 to the individual tile Restart State register.
2. Write 0x0000\_0001 to the tile<n> Restart Power-On State Machine register to restart the tile.
3. Poll the tile<n> Restart Power-On State Machine register to check operation is complete; the power-down sequence is complete when this register reads all zeros.

### Related Information

[Restart State Register \(0x0008\)](#)

[Restart Power-On State Machine Register \(0x0004\)](#)

---

## Bitstream Reconfiguration

To load a new bitstream or reload the current bitstream into the Zynq® UltraScale+™ RFSoc when there are one or more tiles in operation then the following steps must be followed to ensure safe operation.

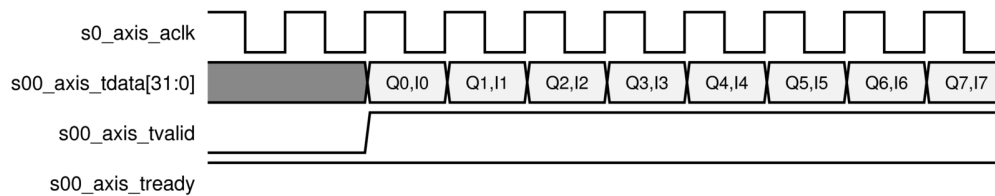
1. Mute the analog front end to avoid TX and RX signaling.
2. Individually shut down all RF-ADC and RF-DAC tiles used using the `XRFDC_Shutdown` API command or the Power-Down Tile registers.
3. Load the new bitstream or reload the existing bitstream.
4. Proceed with the normal Power-up sequencing.

---

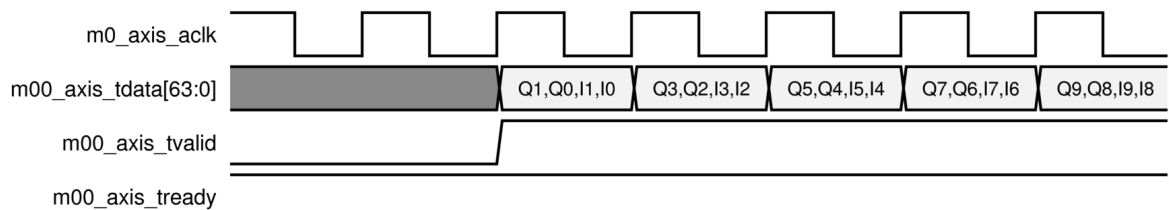
## Interfacing to the AXI4-Stream Interface

The number of AXI4-Stream interfaces and the width of each bus depends on the bandwidth required by the mode of operation selected in the Vivado® IDE. Interpolation and decimation allow the AXI4-Stream bandwidth requirements to be reduced and the width of each AXI4-Stream interface can be selected on the IP core configuration screen in conjunction with the clock rate and interpolation/decimation settings.

The following figure shows a single RF-DAC data input at 4 GSPS with 8x interpolation with I and Q on a single AXI4-Stream interface running at 500 MHz.

**Figure 79: Single RF-DAC Input - 32-bit**


The following figure shows an RF-ADC at 4 GSPS with 4x decimation with I and Q on a single AXI4-Stream interface running at 500 MHz.

**Figure 80: Single RF-ADC Output**


The following figure shows 2x32-bit RF-DACs at 4 GSPS with real data, 8x interpolation, and running at a 250 MHz AXI4-Stream clock.

**Figure 81: Two RF-DACs with Real Data on Separate Interfaces at 250 MHz**


## Applications Overview

This section gives an overview of how the Zynq® UltraScale+™ RFSoc RF Data Converter features can be used in a system.

## Multi-Tile Synchronization

The Zynq® UltraScale+™ RFSoc has a very flexible clocking and data interface to enable a multitude of different applications within the same device. Each RF-ADC/RF-DAC tile has its own independent clocking and data infrastructure, which allows each tile to be driven with individual sample rates, and PL data-word widths. The converters within a single tile share the clocking and data infrastructure, so the sample rates and latency are fixed. However, certain applications might require more than one tile, or even more than one Zynq UltraScale+ RFSoc to be used together. For these applications, matching converter latency across tiles is critical. The multi-tile synchronization (MTS) feature can be used to achieve this multi-tile and device alignment.

**Note:** In this documentation multi-tile synchronization (MTS) is synonymous with multi-converter synchronization.

### Related Information

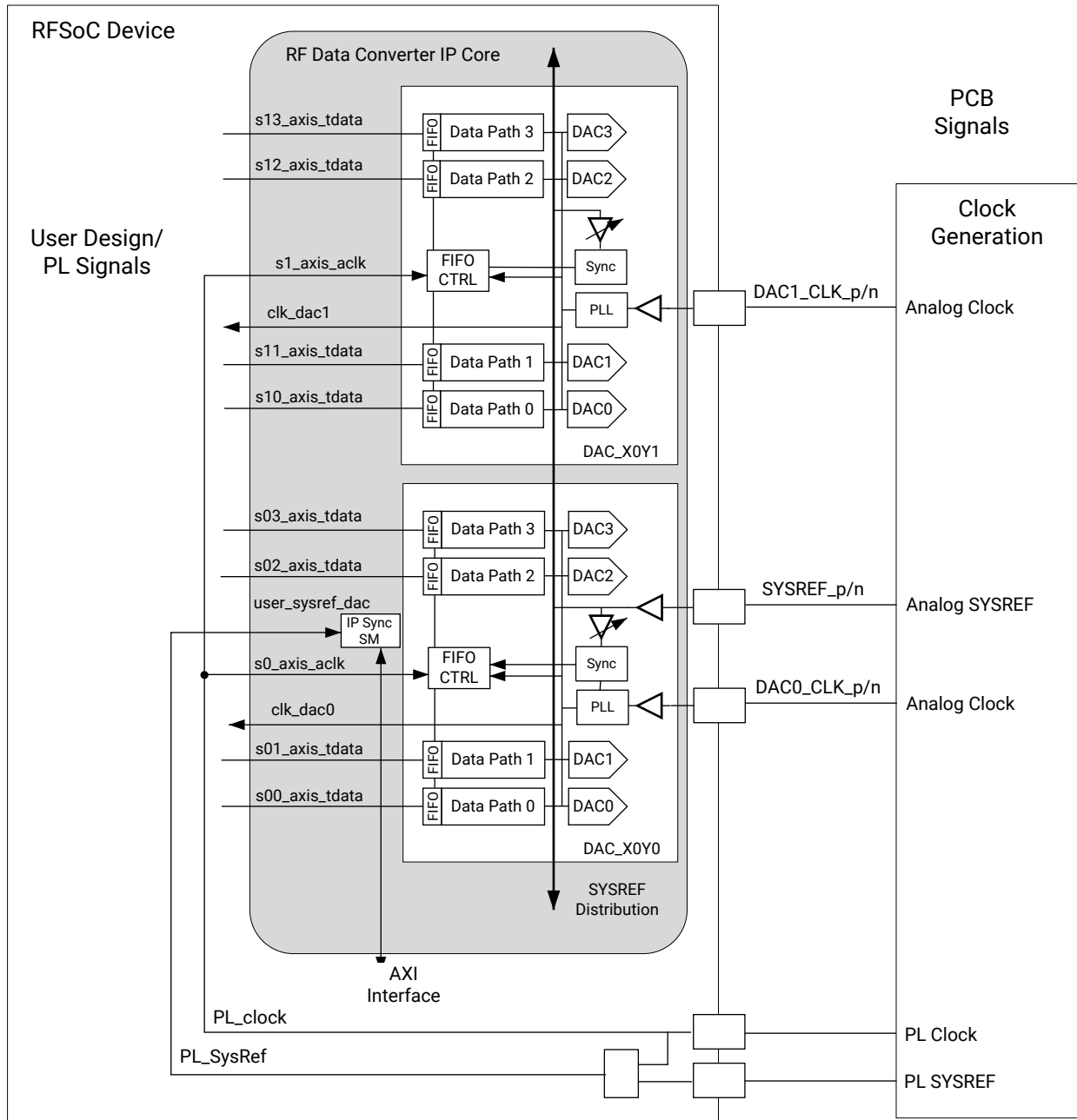
[Clocking](#)

### *Detailed Description*

In any system with multiple independent converters and clocking structures, there are several potential sources of latency uncertainty, such as the clock divider phase, NCO phase, FIFO latencies, clock skew, and data skew. Discrete converters have standardized the use of the JESD204B SYSREF scheme for synchronization, and as a result the Zynq® UltraScale+™ RFSocs have implemented a complementary, simplified scheme using SYSREF.



Figure 82: Zynq® UltraScale+™ RFSoc Multi-Tile Synchronization Example



X19517-032118

The integration of the converters in the Zynq® UltraScale+™ RFSoc results in the elimination of the serial transceiver links for data communication. However, to provide a flexible clocking and number of data words for the PL design, each RF-ADC and RF-DAC incorporates independent gearbox FIFOs. These FIFOs allow data to be transferred between the PL clock domain to the converter sample clock domain, but this results in a non-deterministic latency that must also be measured and corrected for by using SYSREF.

The Zynq® UltraScale+™ RFSoc RF Data Converter and RFdc driver API solution provide an easy to use and integrated way to synchronize each of the tiles in a given device. The previous figure shows an example system with multiple tiles which are to be synchronized. The following applies to the previous figure:

- Two RF-DAC tiles are shown for the purposes of illustration, but the method applies equally to any number of RF-ADC or RF-DAC tiles.
- A single analog SYSREF input per device is distributed internally to all RF-ADCs and RF-DACs.
- There is an internal analog SYSREF delay adjust per tile, to ease PCB requirements by allowing SYSREF setup/hold time to be optimized on-chip.
- There is an analog clock input per tile; these are shown as DAC0\_CLK and DAC1\_CLK.
- The common clock for the PL user design and tiles is shown as PL Clock (PL\_clock).
- The SYSREF for the PL is shown as PL SYSREF (PL\_SysRef). There is a separate PL SYSREF for RF-ADCs and RF-DACs if both are in the synchronization group.
- The synchronization state machine is contained within the core, which, with the RFdc driver API, implements the synchronization solution.
- For RF-DAC multi-tile synchronization (MTS), the RF-DAC tile master is always used as the reference tile for all other RF-DAC tiles to synchronize to. As a result, RF-DAC tile 0 must be an active tile in the application of MTS for the RF-DAC.
- Similarly, the RF-ADC tile master is always used as the reference tile for the synchronization of all other RF-ADC tiles. As a result, RF-ADC tile 0 must be an active tile in the application of MTS for the RF-ADC.
- If it is required to disable a tile that is present in the SYSREF chain it should be powered down to state 3 in the power-on sequence. This ensures that the SYSREF is still propagated along the chain.
- Package flight time skew for each converter should be taken into account and adjusted for during PCB design.

## Synchronization Steps

At a high level, the synchronization steps that are carried out are as follows:

1. Enable all clocks and SYSREF generators
  - a. Both analog and digital clocks must be running and locked before synchronization begins
  - b. Any change to the clocks requires resynchronization
2. SYSREF analog capture
  - a. Ensures SYSREF is safely captured by auto-adjusting the internal SYSREF programmable delay for setup/hold

- b. This is done for each tile and requires a number of periodic SYSREF pulses so that the optimal delay value can be determined. As a result, a periodic SYSREF clock is needed for the MTS process.
3. Clock divider reset
  - When all tiles are safely capturing SYSREF, a subsequent SYSREF edge is used to synchronize all divider phases
4. FIFO latency measure and adjust
  - a. Use the `PL_SysRef` signal to measure the latency through each FIFO
  - b. Use the measurements across all tiles to adjust the latencies so that they match

Steps 2, 3, and 4 are handled automatically by the IP core and RFdc driver API.

### **RFdc Driver API Use**

The RFdc driver API is used to initiate the synchronization procedure and check the status. The following example applies to the RF-DAC synchronization group. The same procedure should be run on the RF-ADC synchronization group.

```
int status_dac;
XRFdc_MultiConverter_Sync_Config DAC_Sync_Config;           // Declare DAC MTS
Settings structure

XRFdc_Multiconverter_Init (&DAC_Sync_Config, 0, 0);       // Initialize
DAC MTS Settings
DAC_Sync_Config.Tiles = 0x3;                               //
Tiles to sync bit-mask: DAC tiles 0 and 1

// Run Multi-Tile-Sync for the DAC Group
status_dac = XRFdc_Multiconverter_Sync(RFdcInstPtr, XRFDC_DAC_TILE,
&DAC_Sync_Config);
```

After synchronization, the synchronization status can be reported:

```
if(status_dac == XRFDC_MTS_OK)
    xil_printf("INFO : DAC Multi-Tile-Sync completed successfully\n");
else
    xil_printf("ERROR : DAC Multi-Tile-Sync Issue - check metal_log for
error information\n");

//
// Report Overall Latency in Sample Clocks and delay adjustments made in
terms of PL words added to each Tile/FIFO
//
xil_printf("\n\n=== Multi-Tile Sync Report ===\n");
for(i=0; i<4; i++) {
    if((1<<i)&DAC_Sync_Config.Tiles) {
        XRFdc_GetInterpolationFactor(RFdcInstPtr, i, 0, &factor);
        xil_printf("  DAC%d: Latency(T1) =%3d, Adjusted Delay (T%d) =%3d
```

```

\n", i,
DAC_Sync_Config.Latency[i], factor,
DAC_Sync_Config.Offset[i]);
    }
}
    
```

**Note:** The apparent latency value reported must not be confused with the absolute FIFO or tile data latency; it is a relative measure of the alignment between tiles.

## Advanced Multi-Tile Synchronization API Use

The multi-tile API use outlined in the previous section guarantees alignment between tiles in each group. However, the total latency through the tiles can vary by a number of words related to the FIFO read-clock period divided by the number of read-words. For applications where the total latency must be constant, the `Target_Latency` setting in the API can be enabled. This setting adjusts all delays less than the target to the target value; any measured values greater than the target results in an error. To prevent this error, the `Target_Latency` value must first be determined for the user FIFO and tile configuration by running `XRFdc_MultiConverter_Sync` with the target set to zero. The returned latency value (plus some margin) must be set as the target value for future runs.

The margin value to be applied is specified in terms of sample clocks. For the RF-ADC tiles, this value must be a multiple of the number of FIFO read-words times the decimation factor, and for RF-DAC tiles this value can be a constant of 16. An example of setting the target latency is as follows:

```

XRFdc_MultiConverter_Init (&DAC_Sync_Config, 0, 0);           // Initialize
DAC MTS Settings
DAC_Sync_Config.Tiles = 0x3;                                  //
Tiles to sync bit-mask: DAC tiles 0 and 1
DAC_Sync_Config.Target_Latency = 296;                       // Target
latency - measured value (280 + 16 margin)
    
```

## SYSREF Signal Requirements

The `SYSREF` signal is the timing reference for the system and must therefore be handled correctly to ensure it does not degrade the synchronization. This signal has the following requirements.

1. The `SYSREF` signal must be a high-quality, free-running, low-jitter square wave, to allow it to be captured consistently by the analog sample clock.
2. The `SYSREF` frequency must meet the following requirements:
  - a. If synchronizing RF-ADC and RF-DAC tiles with different sample frequencies, the frequency must be an integer submultiple of:

$$\text{GCD}(\text{DAC\_Sample\_Rate}/16, \text{ADC\_Sample\_Rate}/16)$$

- b. `SYSREF` must also be an integer submultiple of all PL clocks that sample it. This is to ensure the periodic `SYSREF` is always sampled synchronously.
  - c. Less than 10 MHz.
3. `SYSREF` must be safely captured by the PL, before passing to the core:
  - a. Setup/hold of `PL_SysRef` to `PL_clock` must be handled as part of the user design.
  - b. Clock generation on the PCB can be used to facilitate this, if supported.
4. Analog `SYSREF` and PL `SYSREF` must be the same frequency and be synchronized in phase at the Zynq® UltraScale+™ RFSoc package pin inputs.
5. For MTS synchronization, which covers both divider phase sync and FIFO latency sync, the analog `SYSREF` and PL `SYSREF` must be a continuous clock for the duration of the MTS procedure. It can be either AC-coupled or DC-coupled. The `SYSREF` signals can be shut down after the completion of the MTS procedure. Note that they should be turned on and stabilized prior to the MTS function call.
6. If analog `SYSREF` is being used to synchronize the phase of the DUC/DDC mixer NCOs across multiple Zynq® UltraScale+™ RFSocS, then it must be DC-coupled with the ability to generate single or multiple pulse waveforms.

#### Related Information

[Example SYSREF Frequency Calculation](#)

[PL SYSREF Capture](#)

#### Example SYSREF Frequency Calculation

$$\text{ADC}_{\text{SampleRate}} = 3.93216 \text{ GHz}$$

$$\text{DAC}_{\text{SampleRate}} = 4.91520 \text{ GHz}$$

$$\text{GCD}(\text{ADC}_{\text{SampleRate}}/16, \text{DAC}_{\text{SampleRate}}/16) = 61.44 \text{ MHz}$$

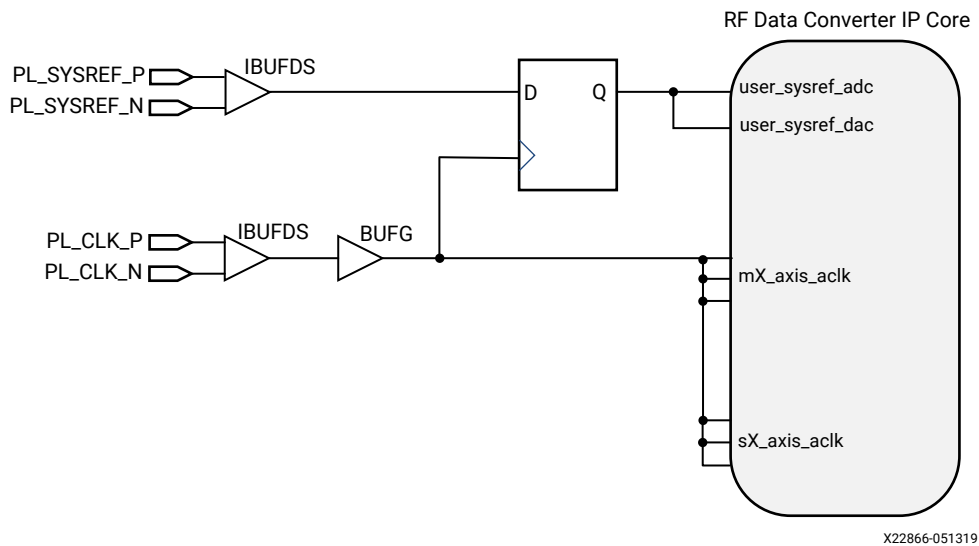
$$\text{Integer Sub - Multiple AND} < 10 \text{ MHz} = 7.68 \text{ MHz}$$

These equations generate a `SYSREF` frequency that satisfies the requirements of the RF-ADC/RF-DAC internal clocking. The `SYSREF` signal is also being captured synchronously by the PL; because of this, the `SYSREF` frequency must also be a submultiple of the PL clock driving the IP interfaces or any other clock interacting with the PL `SYSREF`.

### PL SYSREF Capture

To guarantee effective synchronization between tiles, it is important to reliably capture the PL SYSREF input in the device logic. The PL SYSREF and PL clock inputs must be differential signals that obey the clock and banking rules for the device. The PL clock input must be on a dedicated clock pin. The AXI4-Stream clocks for the Zynq® UltraScale+™ RF Data Converter core must be generated from the PL clock input and *not* from the clock outputs from the core itself. The following figure shows an example PL SYSREF capture circuit where the RF-ADC and RF-DAC are operating at the same AXI4-Stream clock frequency.

Figure 83: Example PL SYSREF Capture



The `report_datasheet` command can be used in the Vivado® IDE to calculate the required setup and hold times for the PL SYSREF input at the device pins. The following figure shows an example calculation for a -1 speed grade device with a 500 MHz PL clock.

Figure 84: Setup and Hold Report

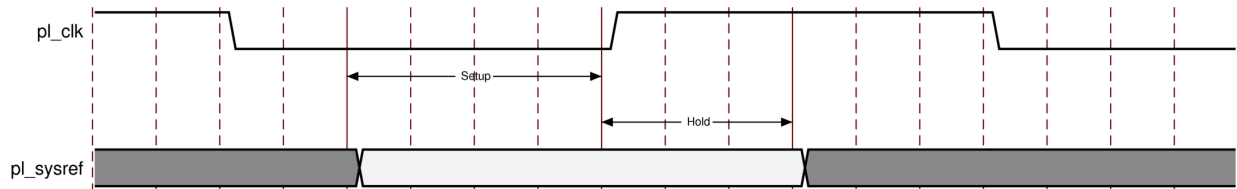
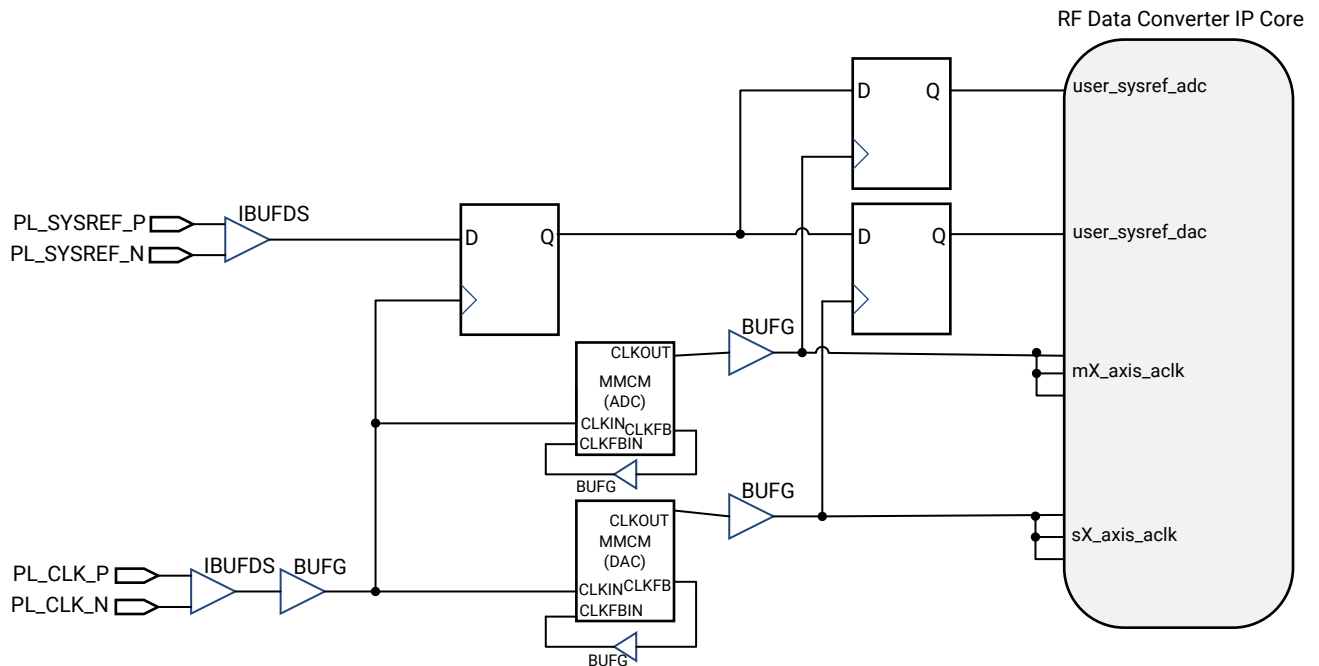


Table 56: Setup and Hold Report

Reference Clock	Input Port	Setup (ns) to Clk (Edge)	Hold (ns) to Clk (Edge)
pl_clk	pl_sysref_n	0.965 (r)	0.285 (r)
pl_clk	pl_sysref_p	0.965 (r)	0.285 (r)

In the example above, the PL SYSREF must be delayed to arrive at least 0.285 ns after the rising edge of the PL clock to ensure the hold requirement is met. The delay must be added externally to the device. For each new design, or when the design is changed, the `report_datasheet` step should be run to reevaluate the setup and hold requirements. If the RF-ADC and RF-DAC are operating at different AXI4-Stream clock frequencies, the circuit shown in the following figure can be used to capture PL SYSREF.

**Figure 85: Example PL SYSREF Capture for Different Clock Rates**



X22830-050919

Here, clock network de-skew MMCMs are used to divide down the frequency of the PL clock to the required rates. The RF-ADC and RF-DAC AXI4-Stream clocks must be common sub-multiples of the PL clock. In addition, the PL SYSREF frequency must be a common sub-multiple of all the clocks in the system.

## Digital Feature Synchronization

After the synchronization is complete, a common state for the clocking is shared between all tiles. The same synchronization infrastructure can also be used by the digital datapath features using Dynamic Update Events.

One example of this is the NCOs that make up the fine mixer features in the DDC and DUC chains. The NCO settings can be applied at the exact same moment across multiple tiles and multiple Zynq UltraScale+ RFSocS using SYSREF. Each NCO supports the dynamic setting of frequency and phase with the changes being applied on a SYSREF or other update event. Similarly, there is an option to reset the phase of the NCO on a SYSREF or other update event. For a single-tile synchronization, the tile event can be used to synchronize the NCOs within a tile. For multiple tile synchronization, SYSREF events should be used.

#### Related Information

[Dynamic Update Events](#)

## Analog Synchronization

### Analog SYSREF Capture/Receive

The receiver of the analog SYSREF in Tile228 (RF-DAC tile 0) is designed to capture a rising edge at the input pins and generate a sync pulse that gets distributed internally in a balanced way to all RF-DAC and RF-ADC tiles for the MTS function and the dynamic update event triggering function. The MTS function requires a continuous analog SYSREF clock that generates a continuous sequence of sync pulses during the duration of the MTS procedural call. The Dynamic Update event trigger function, on the other hand, only uses the first detected rising edge after the event is armed (an event update being requested). The arming, which is done using a corresponding API function call, should complete without seeing any sync pulses generated. Otherwise, some blocks will be latched in with the new update value while other blocks do not. As a result, the sequencing of events and the analog SYSREF waveform must adhere to the following guidance for the proper operation of each of the two functions. To simplify the implementation of some clocking designs, an API function call, `XRFdc_sysrefConfig`, is made available to enable and disable the analog SYSREF receiver, providing a way to gap the SYSREF sync pulses. Note that this API function is not a timed function, and therefore does not easily sync across multiple devices. As a result, the API `XRFdc_sysrefConfig` is not meant to be used solely to synchronize the first rising edge of the SYSREF across multiple devices.

### Analog SYSREF AC- and DC-Coupling

The analog SYSREF supports both AC- and DC-coupling (see the *UltraScale Architecture PCB Design User Guide (UG583)* for electrical characteristic requirements). In AC-coupled mode, the analog SYSREF receiver is sensitive to noise glitches during the OFF state of the output analog SYSREF clock and during the initial turning on of the clock. During these times the receiver unpredictability generates sync pulse due to the glitches. Therefore you must allow the clock to stabilize prior to enabling the SYSREF receiver to avoid incorrect triggering of events.



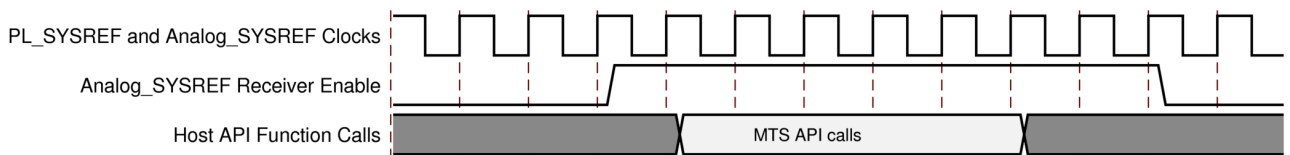
Operating in DC-coupled mode has the potential to avoid these glitch states if the analog SYSREF transmitter stays in a known logic state when it is OFF. Clock outputs that have a pulse function, such as the TI LMK04826 clock device, have the ability to predictably generate pulses on demand. This is required if a dynamic update event using the analog SYSREF is required across multiple devices; the same rising edge is required at all of the devices that are armed for an update.

## MTS Sequencing

### Sequence for AC- or DC-Coupling on Analog SYSREF for Single or Multiple Devices

- Enable PL SYSREF and analog SYSREF clocks in continuous clock mode
- Wait for clocks to stabilize, typically in the order of microseconds, depending on what clock device is used.
- Enable the analog SYSREF receiver with the API command
- Run RF-ADC and/or RF-DAC MTS API functions as described in previous example code
- Wait for completion of API return value
- Disable the analog SYSREF receiver with the API command
- Disable PL SYSREF and analog SYSREF clocks at the clock device

*Figure 86: AC- or DC-Coupling on Analog SYSREF for Single or Multiple Devices*

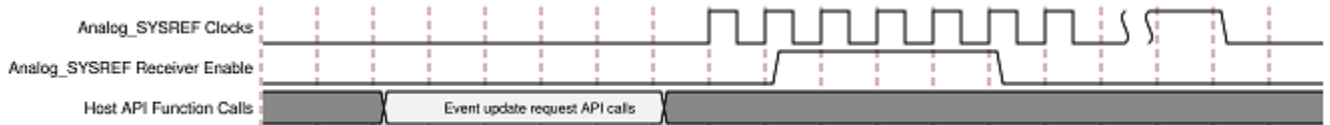


### Dynamic Update Event Trigger Using SYSREF for Single Device with AC- or DC-Coupling

- Disable the analog SYSREF receiver with the API command
- Run all API commands for event update request (such as NCO reset); this is the arming process
- Wait for successful return of API calls to make sure all register writes have been completed
- Enable the analog SYSREF clock in continuous mode
- Wait for clock to be stabilized
- Enable the analog SYSREF receiver with the API command
- Wait long enough to ensure a rising edge has been detected, at this point the update would commence.
- Disable the analog SYSREF receiver with the API command

- Disable the analog SYSREF clock

**Figure 87: Dynamic Update Event Trigger using SYSREF for Single Device with AC- or DC-Coupling**

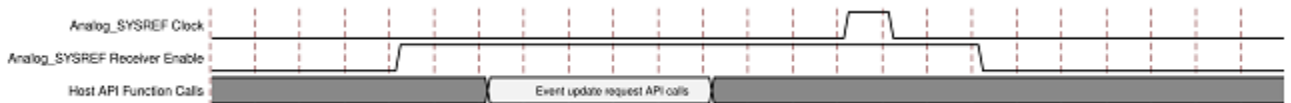


### Dynamic Update Event Trigger Using SYSREF for Multiple Devices with DC-Coupling

DC-coupling and a pulse SYSREF clock are required

- Program the clock generator device to use pulse SYSREF mode; place in idle state (no pulses being generated)
- Wait for analog coupling to be stabilized
- Enable the analog SYSREF receiver for all devices involved (there should be no pulse detected at this point)
- Run all API command for event update request (such as NCO reset) for all tiles and devices; this is the arming process
- Wait for successful return of API calls to make sure all register writes have been completed
- Issue a single synchronized pulse from the analog SYSREF to all devices involved.
- Wait long enough to ensure a rising edge has been detected; at this point the update starts.
- Disable the analog SYSREF receiver with the API command

**Figure 88: Dynamic Update Event Trigger Using SYSREF for Multiple Devices with DC-Coupling**



## Automatic Gain Control Systems

Automatic gain control (AGC) is commonly used in RF-ADC applications where the dynamic range of the input can vary considerably. It provides a way of using the input range of the RF-ADC and maximizing Signal-to-Noise ratio (SNR), while at the same time offering the flexibility to respond to varying signal amplitudes.

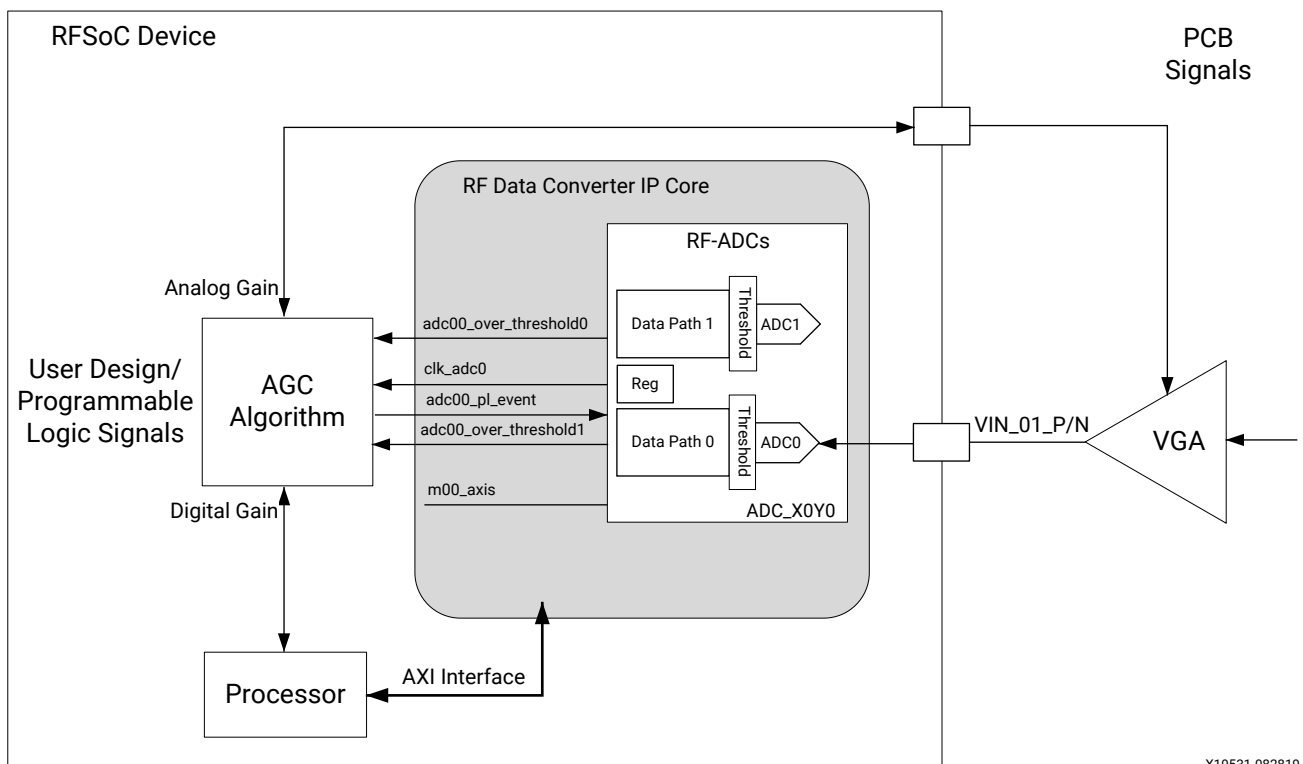
AGC systems consist of the following components:

- Variable Gain Amplifier (VGA)

- RF-ADC
- Signal amplitude monitoring
- AGC Algorithm / Decision logic
- Digital gain compensation

The Zynq® UltraScale+™ RFSoc RF-ADC channels allow the implementation of custom AGC solutions by integrating the signal amplitude monitoring and compensation features into the RF-ADC tiles. These features can be used in conjunction with an external VGA and AGC logic embedded in the FPGA PL. This is shown in the following figure.

Figure 89: Automatic Gain Control



X19531-082819

This figure shows an example AGC application. The signal amplitude monitoring is implemented within each RF-ADC channel using the threshold feature. This feature provides two thresholds that can be programmed per RF-ADC channel. When a threshold level is violated, this is indicated directly in the PL, thereby bypassing any latency within the datapath.

A sample high level operation of the AGC is as follows:

1. At system initialization, the threshold levels and modes are set by the RFdc driver API.
2. If a threshold level is violated:
  - a. The real-time over threshold output flags assert.

- b. The PL-based AGC algorithm makes a decision and computes the new VGA gain and compensation gain
- c. Gain values are programmed into the VGA and digital compensation logic
- d. `adcXY_pl_event` is asserted by the AGC logic.
- e. Thresholds are cleared.

#### Related Information

[Threshold Settings](#)

[Threshold RFdc Driver API Commands](#)

[Digital Gain Compensation](#)

[Clearing Threshold Flags](#)

### **Digital Gain Compensation**

As the VGA adjusts the analog signal level, it might be required to compensate this digitally after the RF-ADC to keep the overall signal-level constant. To minimize the disturbance on the received signal, it is possible to align the gain adjustments so that the digital compensation is applied at the same point in time relative to the analog gain being applied. This application moment must account for relative latencies between the propagation of the analog gain change through the VGA, RF-ADC and digital logic. There are a number of ways this gain adjustment and synchronized application can be achieved.

#### **Compensation Using QMC Gain**

The QMC Gain features are built into the RF-ADC tiles and offer a programmable gain between 0 and 2.0, with 16-bit output resolution. Using this, the 12-bit RF-ADC output signal can be placed to give a flexible dynamic range through the digital datapath. The application moment of the gain value in the QMC block can be controlled by the `adcXY_pl_event` signal. This signal is controlled directly from the user design in the PL and should be delayed to account for the latency required by the external VGA adjustment and RF-ADC conversion. When using this option, the sticky threshold outputs can be auto-cleared when the gain is applied.

#### Related Information

[Clearing Threshold Flags](#)

#### **Compensation Using PL Gain**

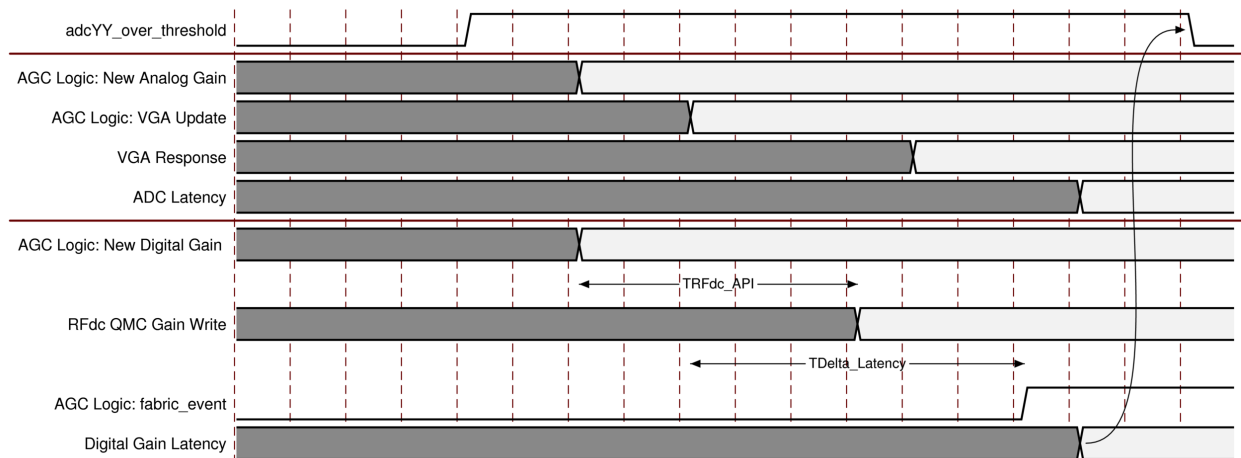
For wider dynamic range digital gain adjustment, the FPGA PL can be used. Similarly to the QMC Gain method, the latency through the VGA, RF-ADC and datapath should be used to delay the application of the gain change in the PL. After the PL gain change, the thresholds can be cleared by using the RFdc driver API, or by issuing a `adcXY_pl_event`.

## Determining Gain-Change Latency

To apply the digital gain compensation at the correct moment, the relative latency between the propagation of the gain change through the analog path must be aligned with the propagation of the digital gain update code through the digital path. Because the external VGA communication and response time is specific to the user application, these relative delays must be measured in-system. For the digital path, each tile has a dedicated input called `adcXY_pl_event` that can be used to apply a setting, such as gain, at a deterministic time. To determine the relative delay between the digital and analog paths, a test signal of constant amplitude should be applied during the design phase, and digital and analog gain adjustments applied. Looking at the output data, the delta time between the digital and analog gain application moments can be observed as a hump/through in the output signal amplitude. This delta time should be added to the `adcXY_pl_event` assertion in by counting periods of the RF-ADC output clock (`clk_adcX`). Using this predetermined delay during normal operation of the design ensures that the gain application moments are aligned.

The following figure illustrates the stages of responding to a threshold event for an example AGC application. The middle region of the diagram represents signals related to the analog/VGA path, while the lower region represents signals related to the digital path.

Figure 90: Example AGC Threshold Event Response



As shown in this figure, the AGC update is initiated by the assertion of a threshold. In response to this the AGC algorithm in the PL calculates gain adjustments to be made for both the analog and digital paths. The propagation of the delay through the analog path is shown by the VGA update write, VGA settling response, and RF-ADC latency. The propagation delay through the digital path is shown by the API write of the new gain value, denoted by `TRFdc_API`, and the `adcXY_pl_event` assertion, denoted by `TDelta_Latency`. `TDelta_Latency` is chosen by the user application to match the Digital and Analog latencies. After the digital gain update has been applied, the threshold can be automatically deasserted, if the Auto-Clear function is enabled.

It should be noted that the application of the QMC gain using the RFdc driver API might incur some driver overhead. In applications where this latency is undesirable, the PL Digital Gain Compensation method can be used.

### Related Information

[Digital Gain Compensation](#)

## NCO Frequency Hopping

When the RF-ADC or RF-DAC mixer with numerically controlled oscillator is in use the frequency of the NCO can be tuned in real-time using the RF-ADC and RF-DAC real-time NCO interfaces. This section describes the operation of these interfaces.

When carrying out frequency hopping information on the required phase and frequency of the NCO must be conveyed to the IP core. The NCO frequency is set by the 48-bit twos complement signed number `nco_freq`. The `nco_freq` signal has a range of  $-Fs/2$  to  $Fs/2$ . The maximum positive value (0x7FFF\_FFFF\_FFFF) results in a NCO frequency of  $Fs/2$  while the maximum negative value (0x8000\_0000\_0000) produces a NCO frequency of  $-Fs/2$ .

If the required NCO frequency is higher than  $Fs/2$  the frequency should be folded down into the  $-Fs/2$  to  $Fs/2$  range by subtracting the sampling frequency as required. Similarly, for large negative frequencies, the sampling frequency is added to the required NCO frequency until it is in range. For NCO frequencies in even Nyquist zones the resultant frequency should be negated. For RF-ADCs where calibration mode 1 is selected the mixer frequency should be reduced by  $Fs/2$ .

The phase information is carried by `nco_phase`. This is a phase offset added to the output of the NCO accumulator. The `nco_phase` input is an 18-bit twos complement signed number in the range -180 to 180 degrees. The maximum positive value (0x1\_FFFF) gives a phase shift of 180 degrees. The maximum negative value (0x2\_0000) gives a phase shift of -180 degrees.

The `nco_update_en` signal is used to determine what frequency and phase words are modified on an update event. The bits in the `nco_update_en` vector are set High to modify the following bits of the frequency and phase numbers.

- Bit 5: Reset the phase offset to 0
- Bit 4: `nco_phase[17:16]`
- Bit 3: `nco_phase[15:0]`
- Bit 2: `nco_freq[47:32]`
- Bit 1: `nco_freq[31:16]`
- Bit 0: `nco_freq[15:0]`

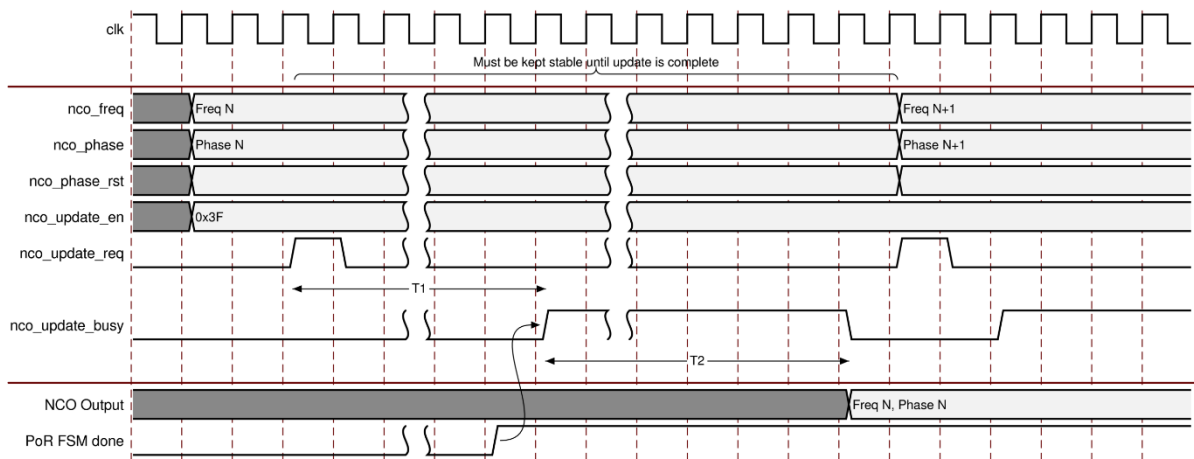
For example, if only bits 2:0 of `nco_update_en` are set High, only the frequency value is changed on an update event.

## Single Converter Mode

When multi-tile synchronization (MTS) is not supported all the converters are independent. An NCO update state machine is instantiated for each converter. Before a NCO update can be performed the event source must be set to Tile. See Dynamic Update Events for more information on setting the event source.

The process for updating the NCO settings is shown below.

**Figure 91: Single Converter NCO Update**



1. Firstly the required frequency, phase, and phase reset values are set on the real-time NCO signal interface ports.
2. The update enable port is set to indicate which converter register values are to be updated. The more values that are required to be updated increases  $T_2$  (the time to write the NCO values into the converter registers). Each register write takes five `s_axi_aclk` cycles.
3. A pulse is then sent to the update request port. If the IP state machine has completed the start up process the access to the requested NCO registers commences.  $T_1$  indicates the wait time for the IP state machine to complete.
4. The busy signal remains High until the NCO update process is complete. The frequency, phase, and phase reset values should be held until the busy signal is deasserted.

### Related Information

[Dynamic Update Events](#)

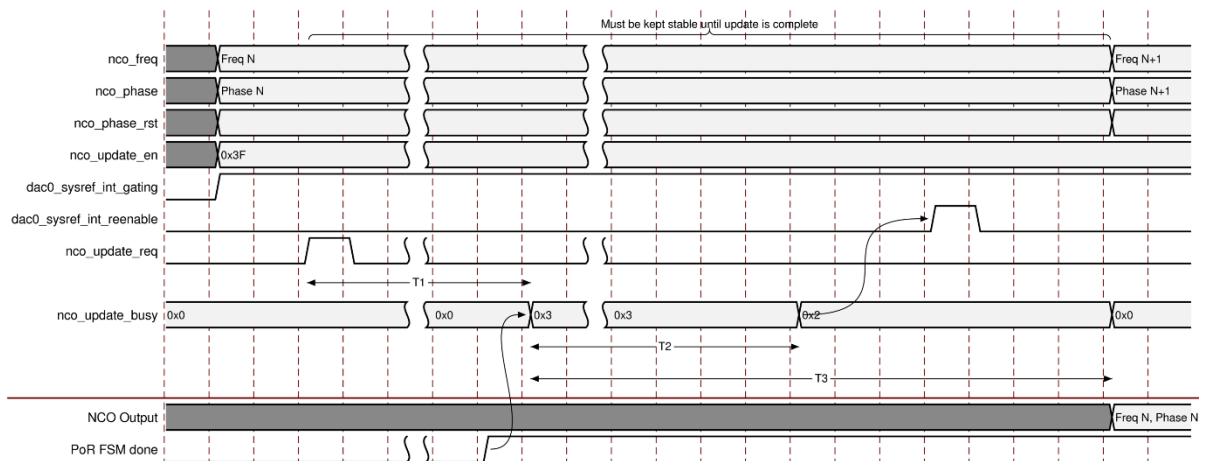
## Multi-Tile Mode

When multi-tile synchronization is selected the SYSREF is used to update the NCO settings. This ensures that the mixer updates are synchronized across all the converters. Before an NCO update can take place the event source must be set to SYSREF. See Dynamic Update Events (link below) for more information on setting the event source.

Because the SYSREF is used to trigger the NCO update it should be gated to ensure that an edge does not arrive during the update process. If the SYSREF is gated externally the IP core works in the same way as the single converter case. The SYSREF should be turned off before the NCO update process is started and re-enabled when the IP has deasserted the busy signal.

Alternatively, the IP core can gate the SYSREF signal internally. This is achieved by writing to registers in RF-DAC 228. In this case, the process for updating the NCO settings is shown below.

Figure 92: Multi-tile NCO Update



- First, the required NCO frequency, phase, and phase reset values are set on the Real-Time NCO signal interface ports for all NCO channels that are to be updated.
- `dac0_sysref_int_gating` is driven High to indicate to the IP that it should gate the SYSREF internally.
- The update enable port is set to indicate which register values are to be updated. The more values that are required to be updated increases T2 (the time to write the NCO values into the converter registers). Each register write takes  $5 s\_axi\_aclk$  cycles.
- A pulse is sent to the update request port. If the IP state machine has completed the start up process the access to the requested NCO registers starts. T1 indicates the wait time for the IP state machine to complete.
- Bit 1 of the `dac0_nco_update_busy` output is asserted when the SYSREF has been stopped. The other converters in the multi-tile synchronization group wait for this output to go High before requesting their NCO updates through the Real-Time NCO signal interface.



- When all the update busy outputs for the converter in the multi-tile synchronization group are Low, with the exception of `dac0_nco_update_busy[1]`, the NCO register writes have been completed.
- A pulse is sent to the `dac0_sysref_int_reenable` port to indicate that SYSREF can be restarted. The process is complete when `dac0_nco_update_busy[1]` goes Low. T3 indicates the time taken for the process to complete from the start of the initial register updates. This time depends on the SYSREF frequency, the number of register values to be updated, and the number of converters in the tile synchronization group.

#### Related Information

[Dynamic Update Events](#)

[Real-Time NCO Signal Interface Ports for RF-DACs](#)

[Real-Time NCO Signal Interface Ports for Quad RF-ADCs](#)

[Real-Time NCO Signal Interface Ports for Dual RF-ADCs](#)

# Design Flow Steps

This section 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](#))
- *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
- *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
- *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))

---

## 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](#)) 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, 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:

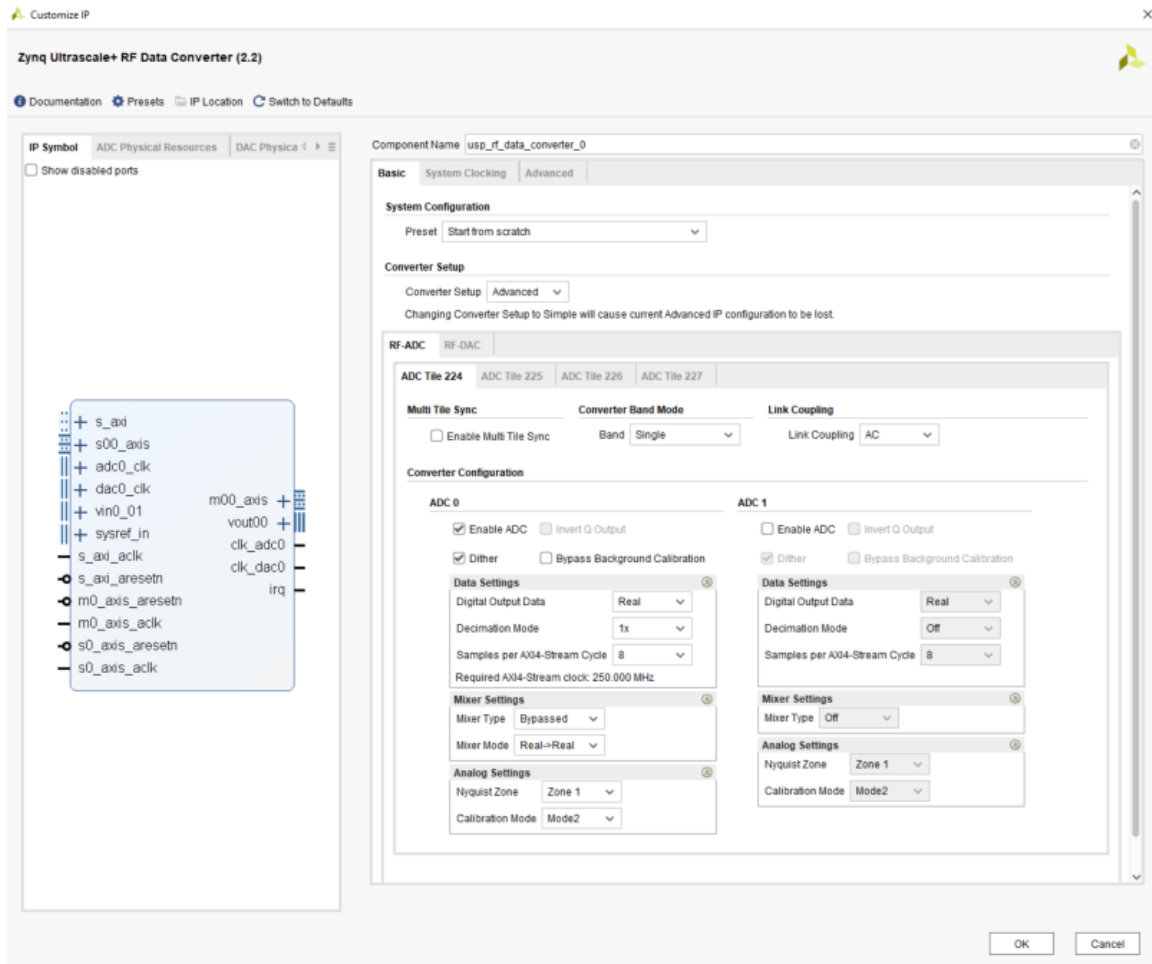
1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

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

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

## Basic Tab

Figure 93: Basic Tab



### Presets

The Vivado® IDE IP core configuration screen provides a method of saving and applying preset configurations. From the Presets menu, several fixed configurations are available.

- **4x4:** ADC:R2C DAC C2R: 4 RF-ADCs and 4 RF-DACs enabled. The RF-ADCs are configured in real to I/Q mode. The RF-DACs are configured in I/Q to real mode.
- **8x8:** ADC:R2C DAC C2R: 8 RF-ADCs and 8 RF-DACs enabled. The RF-ADCs are configured in real to I/Q mode. The RF-DACs are configured in I/Q to real mode.
- **R2C/C2R Multi x2:** ADC:R2C Multi x 2 DAC: C2R Multi x2. The RF-ADCs are configured in x2 multi-band using real to I/Q data. The RF-DACs are configured in x2 multi-band using IQ to real data.

- **C2C Multi x2:** ADC:C2C Multi x 2 DAC: C2C Multi x2. The RF-ADCs are configured in x2 multi-band using IQ to I/Q data. The RF-DACs are configured in x2 multi-band using I/Q to I/Q data.
- **C2R Multi 2x2:** DAC: C2R Multi 2x2. 2x2 multi-band on a single RF-DAC tile using I/Q to real data.
- **C2C Multi 2x2:** DAC: C2C Multi 2x2. 2x2 multi-band on a single RF-DAC tile using I/Q to I/Q data.
- **C2R Multi x4:** DAC: C2C Multi x4. 1x4 multi-band on a single RF-DAC tile using I/Q to real data.

In addition to these fixed presets, the current configuration can be saved by selecting **Save Configuration...** from the Presets menu. After the configuration is saved it can be reloaded at any time by selecting **Apply Configuration...**

## Component Name

The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed from these characters: a through z, 0 through 9 and “\_” (underscore).

## System Configuration

Configuration Preset: This menu allows the user to select from a list of predefined configurations to avoid having to enter all settings manually.

## Converter Setup

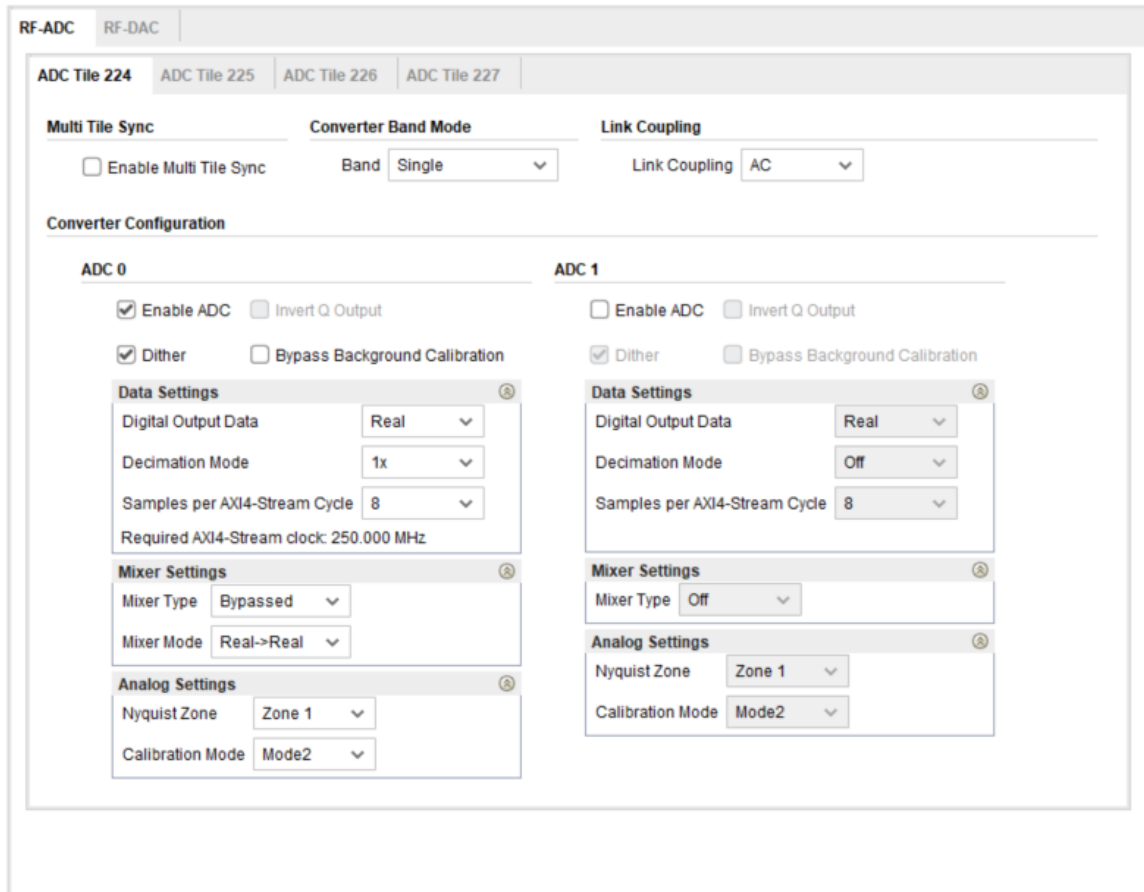
The converter setup menu provides advanced and simple options for setting up the IP configuration.

- **Advanced:** All tiles are independently configurable.
- **Simple:** The user is presented with the configuration options for a single RF-ADC and RF-DAC tile. After the tile is set up the RF-ADC configuration can be applied to any RF-ADC in the device by selecting the tile from the given tile list. Similarly the RF-DAC configuration can be applied to any RF-DAC tile.

It is possible to change from simple to advanced setup without losing configuration information. However, it is not possible to change from advanced to simple.

## RF-ADC Tab

Figure 94: RF-ADC Tab



### RF-ADC Tile Configuration

These parameters are common to every RF-ADC in a tile.

- Link Coupling:** This determines if the input signal to the RF-ADC is AC or DC-coupled. Typical applications use AC-coupling, but for certain applications such as Zero IF, DC-coupling is also supported. When DC link coupling mode is selected, the common mode voltage of the external driving circuit needs to be aligned with the RF-ADC internal common mode voltage by using the externally driven VCM pin (see RF-ADC Analog Input)). The VCM pin is only enabled for DC-coupling mode, and can be left floating if AC-coupling mode is chosen.
- Converter Band Mode:** Each RF-ADC can operate either independently (single band mode) or together with other converters in the tile (multi-band mode). The following modes are available for the RF-ADC.
  - Single:** The converter operates in single band mode.

- **Multi x2 (pair 01):** The output from ADC 0 is routed through the DDC channel of both ADC 0 and 1. Digital settings, such as mixer mode and type, are available for each DDC channel. In Quad devices the other two RF-ADCs in the tile operate in single band mode.
- **Multi x2 (pair 23):** This option is similar to the previous mode. Here the output from ADC 2 is routed to DDC channels 2 and 3. The other converters in the tile operate in single band mode. This option is only available on Quad devices.
- **Multi x2 (both):** A combination of the previous multi-band options. The data from ADC 0 provides the input for DDC channels 0 and 1. The data from ADC 2 supplies DDC channels 2 and 3. This option is only available on Quad devices.
- **Multi x4:** All DDC channels in the tile are supplied from ADC 0. This option is only available on Quad devices.
- **Multi Tile Sync:** When enabled, the tile is included in a multi-tile synchronization group. Tile 224 must be enabled and present in the group along with converter 0 of the tile being configured to enable this option. A reference clock must also be provided to tile 224. See Multi-Tile Synchronization for details.

#### Related Information

[RF-ADC Analog Input](#)

[Multi-Tile Synchronization](#)

[RF-ADC Multi-Band Operation](#)

### ***RF-ADC Converter Configuration***

- **Enable ADC:** Select if the selected converter within the selected tile is enabled. Valid values are True and False.
- **Invert Q Output:** This parameter is configurable only when I/Q output data is selected and the fine mixer is enabled. When set, the quadrature output of the mixer is negated. This allows -Q data to be generated.
- **Dither:** Selects if dither is enabled for the selected tile. Dither should be enabled unless the sample rate is under 0.75 times the maximum sampling rate for the RF-ADC.
- **Bypass Background Calibration:** If checked the background calibration logic is implemented in the IP core. The driver can be used to download a fixed set of calibration coefficients to the IP. This option is only available in Real input to Real output mode with further restrictions on the decimation mode and the number of samples per AXI4-Stream cycle.

### **Data Settings**

- **Digital Output Data:** Sets the data type of the selected converter within the selected tile. The parameter is only configurable when the converter is enabled. Valid values are Real and I/Q. When converter 0 is set to I/Q, converter 1 must also be enabled and when converter 2 is set to I/Q, converter 3 must also be enabled; otherwise the configuration is invalid.

- **Decimation Mode:** Sets the decimation values of the selected converter within the selected tile. The parameter is only configurable when the converter is enabled. Values 1x, 2x, 4x, 8x are selected from a drop-down menu. When the converter is not enabled, the value is Off.
- **Samples per AXI4-Stream Word:** Sets the number of words per cycle. This parameter is configurable when the specific converter has been enabled. Valid values are between 1 and 8 and can be selected using a drop-down list. The range of values is limited depending on the selected sample rate to keep the AXI4-Stream clock within the specification. The required AXI4-Stream clock is an input to the IP core and its value, based on the selected bus width, is displayed.

## Mixer Settings

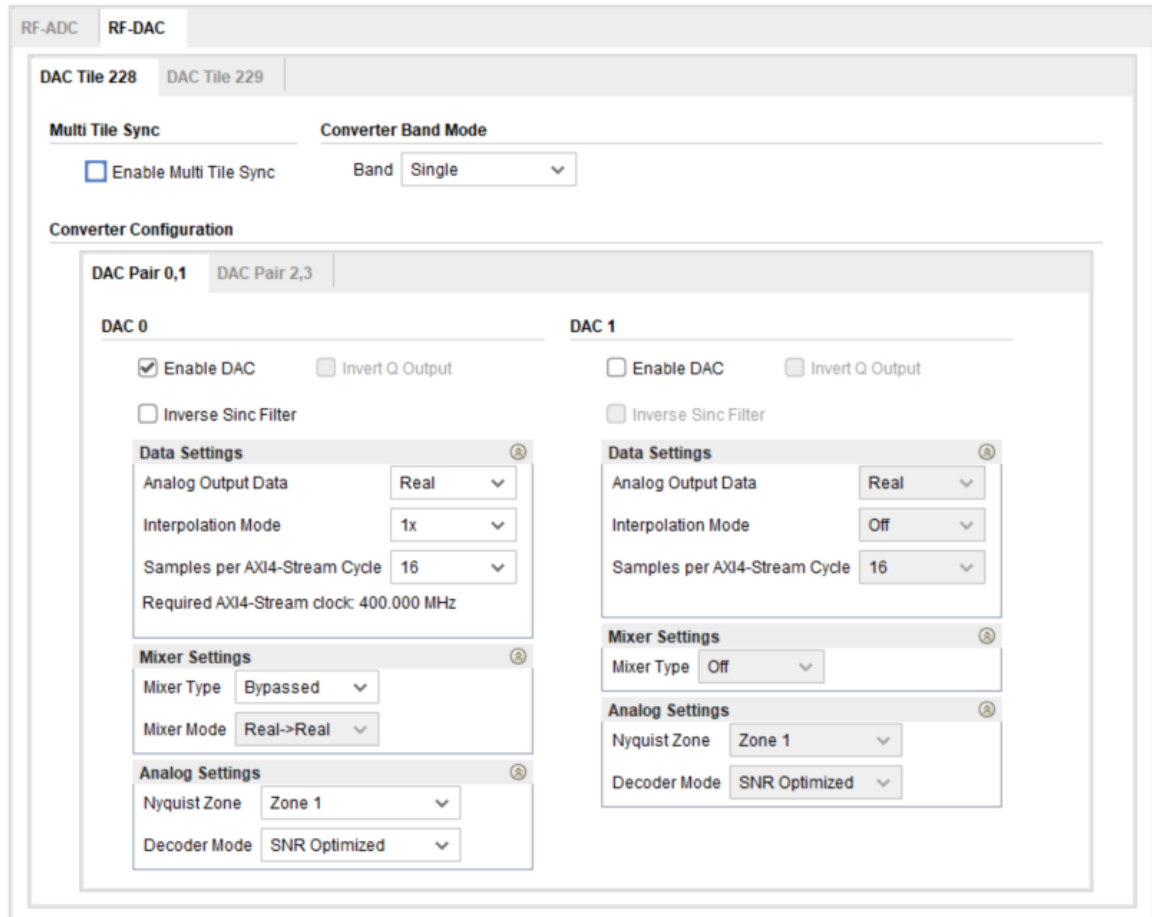
- **Mixer Type:** Set the type of Mixer to be used. Valid options are bypassed, coarse, and fine. Selectable options depend on the selection in the converter Digital Output Data field.
- **Mixer Mode:** Set the mixer mode of the selected converter within the selected tile. The parameter is only configurable when the converter is enabled. The choice of mixer modes depends on the mixer type and the format of the digital output data selected. When real data is output the mixer is bypassed. When I/Q data is output the mixer can be set to Real to I/Q or I/Q to I/Q.
- **Coarse Mixer Frequency:** Sets the frequency of the coarse mixer. The parameter is configurable only when coarse is selected as the Mixer Type. Valid options are  $F_s/2$ ,  $F_s/4$  and  $-F_s/4$ .
- **Fine Mixer Frequency:** Sets the frequency of the fine mixer. The parameter is only available when fine is selected as the mixer type. The valid range of frequencies is -10 GHz to 10 GHz.
- **Fine Mixer Phase:** Sets the phase of the fine mixer. The parameter is only available when fine is selected as the mixer type. Valid range is -180 to 180.

## Analog Settings

- **Nyquist Zone:** Selects between even and odd Nyquist zone operation.
- **Calibration Mode:** Selects between different calibration optimization schemes depending on the features of the input signals. Mode 1 is optimal for input frequencies  $F_{\text{samp}}/2(\text{Nyquist}) \pm 10\%$ . Otherwise, use Mode 2.

## RF-DAC Tab

Figure 95: RF-DAC Tab



### RF-DAC Tile Configuration

These parameters are common to every RF-DAC in a tile.

- Converter Band Mode:** Each RF-DAC can operate either independently (single band mode) or together with other converters in the tile (multi-band mode). The following modes are available for the RF-DAC. See RF-DAC Multi-Band Operation.
  - Single:** The converter operates in single band mode.
  - Multi x2 (pair 01):** DUC channels 0 and 1 drive the input of DAC0. Digital settings, such as mixer mode and type, are available for each DUC channel. The other two RF-DACs in the tile operate in single band mode.
  - Multi x2 (pair 23):** This option is similar to the previous mode. Here DAC2 is driven by DUC channels 2 and 3. The other converters in the tile operate in single band mode.



- **Multi x2 (both):** A combination of the previous multi-band options. DAC0 converts the data from DUC channels 0 and 1. DAC2 is driven by DUC channels 2 and 3.
- **Multi x4:** DAC0 accepts data from all the DUC channels in the tile.
- **Multi Tile Sync:** When enabled the tile is included in a multi-tile synchronization group. Tile 228 must be enabled and present in the group along with converter 0 of the tile being configured to enable this option. A reference clock must also be provided to tile 228. See Multi-Tile Synchronization for details.

#### Related Information

[Multi-Tile Synchronization](#)

[RF-DAC Multi-Band Operation](#)

### ***RF-DAC Converter Configuration***

- **Enable DAC:** Select if the selected converter within the selected tile is enabled. Valid values are True and False.
- **Invert Q Output:** When enabled, the Q output of the converter is inverted. This parameter is configurable only when I/Q output data is selected. A follow-on analog mixer must implement a subtraction,  $I \times \cos(f) - Q \times \sin(f)$  so the option to provide a -Q signal (negative quadrature) is provided so that the external mixer can perform the addition,  $I \times \cos(f) + (-Q)\sin(f)$
- **Inverse Sinc Filter:** Select whether the inverse sinc filter is enabled or not. The parameter is only configurable when the converter is enabled. Valid values are True and False.

#### **Data Settings**

- **Analog Output Data:** Sets the data type of the selected converter within the selected tile. The parameter is only configurable when the converter is enabled. Valid values are Real and I/Q. When converter 0 is set to I/Q, converter 1 must also be enabled and when converter 2 is set to I/Q, converter 3 must also be enabled; otherwise the configuration is invalid.
- **Interpolation Mode:** Sets the interpolation values of the current converter. The parameter is only configurable when the converter is enabled. Values 1x, 2x, 4x, 8x are selected from a drop-down menu. When the converter is not enabled, the value is Off.
- **Samples per AXI4-Stream Word:** Sets the number of words per cycle. This parameter is configurable when at least one converter in the tile is enabled. Valid values are between 1 and 16 and can be selected using a drop-down list. The range of values is limited depending on the selected sample rate in order to keep the AXI4-Stream clock within the specification. The required AXI4-Stream clock is an input to the IP core and its value, based on the selected bus width, is displayed.

#### **Mixer Settings**

- **Mixer Type:** Sets the type of Mixer to be used. Valid options are bypassed, coarse, and fine. Selectable options depend on the selection in the converter Analog Output Data field.

- **Mixer Mode:** Sets the mixer mode of the selected converter within the selected tile. The parameter is only configurable when the converter is enabled. The choice of mixer modes depends on the selected Mixer Type and the format of the analog output data selected. When real data is output the mixer can be set to I/Q -> Real or Bypass and when I/Q data is output the mixer can be set to I/Q to I/Q.
- **Coarse Mixer Frequency:** Sets the frequency of the coarse mixer. The parameter is configurable only when coarse is selected as the mixer type. Valid options are:  $F_s/2$ ,  $F_s/4$  and  $-F_s/4$ .
- **Fine Mixer Frequency:** Sets the frequency of the fine mixer. This parameter is only available when fine is selected as the mixer type. The valid range of frequencies is -10 GHz to 10 GHz.
- **Fine Mixer Phase:** Sets the phase of the fine mixer. The parameter is only available when fine is selected as the mixer type. Valid range is -180 to 180 degrees.

### Analog Settings

- **Nyquist Zone:** Selects between even and odd Nyquist zone operation.
- **Decoder Mode:** The RF-DAC decoder can be optimized for low signal-to-noise ratio or for high linearity.

## System Clocking Tab

Figure 96: System Clocking Tab

**AXI4-Lite Interface Configuration**

AXI4-Lite Clock (MHz) 100.0

**Tile Clocking Settings**

Tile	Sampling Rate (GSPS)	Max Fs (GSPS)	PLL	Reference Clock (MHz)	PLL Ref Clock (MHz)	Ref Clock Divider	Fabric Clock (MHz)	Clock Out (MHz)
ADC 224	2.0	4.096	<input type="checkbox"/>	2000.000	-	1	250.000	15.625
ADC 225	2.0	4.096	<input type="checkbox"/>	2000.000	-	1	0.0	15.625
ADC 226	2.0	4.096	<input type="checkbox"/>	2000.000	-	1	0.0	15.625
ADC 227	2.0	4.096	<input type="checkbox"/>	2000.000	-	1	0.0	15.625
DAC 228	6.4	6.554	<input type="checkbox"/>	6400.000	-	1	400.000	50.000
DAC 229	6.4	6.554	<input type="checkbox"/>	6400.000	-	1	0.0	50.000

**PLL Summary Settings**

Tile	Vco (MHz)	Fb Div	M	R
ADC 224	-	-	-	-
ADC 225	-	-	-	-
ADC 226	-	-	-	-
ADC 227	-	-	-	-
DAC 228	-	-	-	-
DAC 229	-	-	-	-

The System Clocking tab is present when the Converter Setup is set to Advanced. This screen enables the user to enter information on the system sample rates and clock frequencies. When the Converter Setup is set to Simple this information is entered on the main RF-ADC and RF-DAC tabs.

## ***AXI4-Lite Interface Configuration***

- **AXI Clock Frequency (MHz):** The core requires information on the frequency of the AXI4-Lite clock input to ensure the correct timing of the power-on sequence of the RF-ADC and RF-DAC blocks. The speed of the clock should be entered in MHz. The maximum allowed frequency is given by the DRP clock maximum frequency specified in the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics* ([DS926](#)).

## ***Tile Clock Settings***

The sampling rate and clock frequencies are set in this section. These settings are shared by each converter in a tile.

- **Sampling Rate (GSPS):** Sets the sampling rate for each tile. It is configurable when at least one of the converters in the tile is enabled. Valid values depend on the selected device and package. These values can be found in *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics* ([DS926](#)).
- **Max Fs (GSPS):** The maximum sample rate for each tile is given for reference.
- **PLL:** Selects whether the PLL within the tile is being used or bypassed. It is configurable when at least one of the converters in the tile is enabled. If it is required to dynamically reconfigure the PLL using the RFdc driver API, the PLL must be enabled in the Vivado® IDE. Valid values are True or False.
- **Reference Clock (MHz):** Sets the frequency of the clock input for the tile. It is configurable when the PLL and at least one of the converters in the tile is enabled. Its values depend on the sampling rate of the tile. A drop-down list of values based on the sample rate selected is presented. It is recommended to use the correct reference frequency at the PLL input for optimum phase noise performance. See the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics* ([DS926](#)) for the  $F_{REF}$  specification.
- **PLL Reference Clock:** Sets the frequency of the PLL input clock if the PLL is enabled for a specific tile.
- **Reference Clock Divider:** Sets the divider on the PLL reference clock input.
- **Fabric Clock (MHz):** This is the frequency of the clock to be supplied on the AXI4-Stream clock input for the selected tile. The required frequency is determined by the sample rate and the settings in the Converter Configuration. Because all AXI4-Stream ports on a tile share a common AXI4-Stream clock all converter configurations on a tile must require the same clock frequency.

- **Clock Out (MHz):** Sets the frequency of the output clock on the tile. This clock can be used to drive the AXI4-Stream clock inputs. It is configurable when at least one of the converters in the tile is enabled. The values depend on the sampling rate of the tile. A drop-down list of values based on the sample rate selected is presented.

**Related Information**

[RF-ADC Converter Configuration](#)

[RF-DAC Converter Configuration](#)

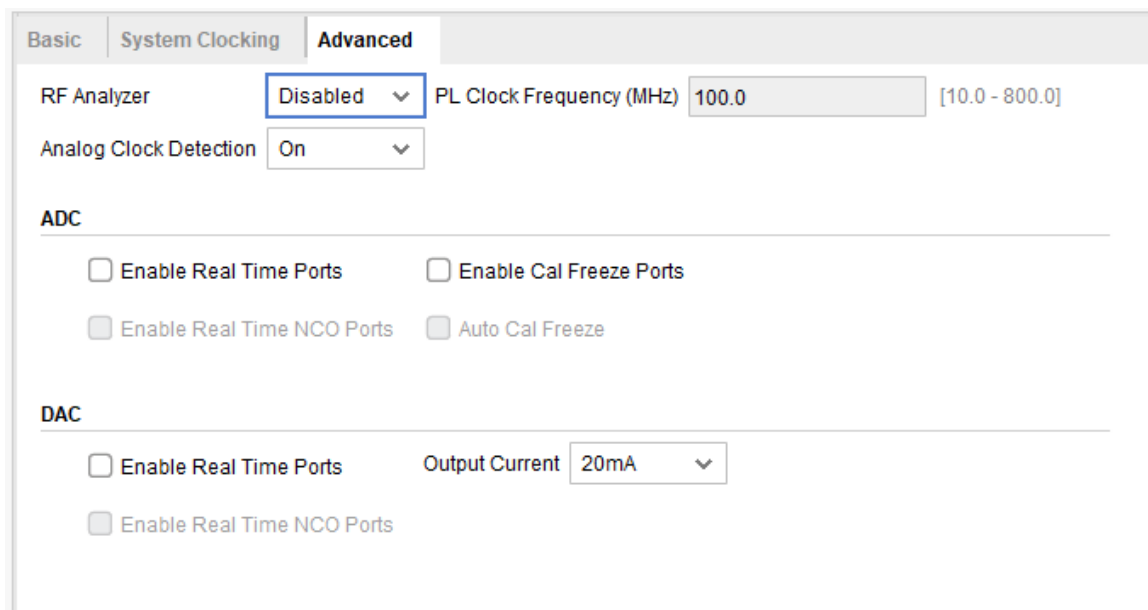
**PLL Summary Settings**

The PLL Summary gives information on the frequency and divider settings of the enabled PLLs. The following information is displayed:

- **Vco(MHz):** The frequency of the voltage controlled oscillator.
- **Fb Div:** The feedback divider setting
- **M:** The output divider setting
- **R:** The reference clock divider setting

**Advanced Tab**

Figure 97: Advanced Tab



- **RF Analyzer:** The RF Analyzer provides a hardware test system. The system contains data stimulus and capture blocks configured for this instance of the IP core.

- **PL Clock Frequency (MHz):** If multi-tile sync is selected then the user must supply a PL clock and PL SYSREF to the system. For accurate SYSREF capture the frequency must be a common integer multiple of the RF-ADC and RF-DAC AXI4-Stream clocks.
- **Analog Clock Detection:** Not available for customer use.

#### Related Information

[Multi-Tile Synchronization](#)

[RF Analyzer](#)

### RF-ADC

- **Enable Real Time Ports:** Enables the RF-ADC Over Threshold, Over Voltage, Over Range and `pl_event` ports.
- **Enable Cal Freeze Ports:** Enables the calibration freeze logic. Calibration is frozen when the analog input is over voltage or the relevant `int_cal_freeze` input is asserted.
- **Auto Cal Freeze:** If this option is selected, the calibration is frozen when the analog input is over range, in addition to the manual calibration freeze process.
- **Enable Real Time NCO Ports:** The frequency and phase of the mixer NCO in the DDC can be modified in real time. Selecting this option enables the real time NCO interface for the RF-ADC. The operation of this interface is described in NCO Frequency Hopping.

### RF-DAC

- **Enable Real Time Ports:** Enables the RF-DAC Fast Shutdown and `pl_event` ports.
- **Enable Real Time NCO Ports:** The frequency and phase of the mixer NCO in the DUC can be modified in real time. Selecting this option enables the real time NCO interface for the RF-DAC. The operation of this interface is described in NCO Frequency Hopping.
- **Output Current:** Select between 20 mA and 32 mA RF-DAC output current. In 20 mA mode, `DAC_AVTT` is equal to 2.5V. In 32 mA mode, `DAC_AVTT` is equal to 3.0V.

#### Related Information

[Real-Time Signal Interface Ports for RF-DACs](#)

[Real-Time Signal Interface Ports for Dual RF-ADCs](#)

[Real-Time Signal Interface Ports for Quad RF-ADCs](#)

[RF-DAC Output Current Mode](#)

[NCO Frequency Hopping](#)

## User Parameters

The following table shows the relationship between the fields in the Vivado® IDE and the user parameters (which can be viewed in the Tcl Console).

Table 57: User Parameters

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
<b>RF-ADC</b>		
ADC Tile n Enable <sup>1</sup>	C_ADCn_Enable	0
ADC Tile n Feedback Divider	C_ADCn_FBDIV	10
ADC Tile n Fabric Frequency	C_ADCn_Fabric_Freq	0
ADC Tile n Outclk Frequency	C_ADCn_Outclk_Freq	250
ADC Tile n PLL Enable	C_ADCn_PLL_Enable	FALSE
ADC Tile n Refclk Frequency	C_ADCn_Refclk_Freq	2000
ADC Tile n Sampling Rate	C_ADCn_Sampling_Rate	2
ADC Tile n Link Coupling	C_ADCn_Link_Coupling	0
ADC Tile n Multi Tile Sync	C_ADCn_Multi_Tile_Sync	FALSE
ADC Tile n Multiband setting	C_ADCn_Band	0
ADC Tile n Reference Clock input divider	C_ADCn_Refclk_Div	1
ADC Tile n Clock Source	C_ADCn_Clock_Source	n
ADC Tile n Clock Distribution	C_ADCn_Clock_Dist	0
ADC Tile y Converter z Data Type <sup>2</sup>	C_ADC_Data_Typeyz	0
ADC Tile y Converter z Data Width	C_ADC_Data_Widthyz	8
ADC Tile y Converter z Decimation Mode	C_ADC_Decimation_Modeyz	1
ADC Tile y Converter z Mixer Mode	C_ADC_Mixer_Modeyz	2
ADC Tile y Converter z Nyquist Zone	C_ADC_Nyquistyz	1
ADC Tile y Converter z Calibration Optimization	C_ADC_CalOpt_Modeyz	0
ADC Tile y Converter z Enable	C_ADC_Sliceyz_Enable	TRUE
ADC Tile y Converter z Mixer Type	C_ADC_Mixer_Typeyz	3
ADC Tile y Converter z Fine Mixer Frequency	C_ADC_NCO_Freqyz	0.0
ADC Tile y Converter z Fine Mixer Phase	C_ADC_NCO_Phaseyz	0
ADC Tile y Converter z Coarse Mixer Phase	C_ADC_Coarse_Mixer_Freqyz	0
ADC Tile y Converter z Invert Q Output	C_ADC_Neg_Quadratureyz	FALSE
ADC Tile y Converter z Dither	C_ADC_Ditheryz	TRUE
ADC Tile y Converter z background calibration bypass	C_ADC_Bypass_BG_Calyz	FALSE
<b>Common</b>		
AXI Clock Frequency	C_Axiclk_Freq	100
Component Name	C_COMPONENT_NAME	usp_rf_data_converter_0
Converter_Setup	C_Converter_Setup	Advanced
Enable RF-ADC Real-Time Signals	C_ADC_RTS	FALSE
Enable RF-ADC NCO Real-Time Signals	C_ADC_NCO_RTS	FALSE
Enable RF-DAC Real-Time Signals	C_DAC_RTS	FALSE

Table 57: User Parameters (cont'd)

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Enable RF-DAC NCO Real-Time Signals	C_DAC_NCO_RTS	FALSE
Enable RF-ADC Calibration Freeze	C_Calibration_Freeze	FALSE
Enable RF-ADC Auto Calibration Freeze	C_Auto_Calibration_Freeze	FALSE
DAC Output Current	C_DAC_Output_Current	0
<b>RF-DAC</b>		
DAC Tile n Enable	C_DACn_Enable	0
DAC Tile n Feedback Divider	C_DACn_FBDIV	10
DAC Tile n Fabric Frequency	C_DACn_Fabric_Freq	0
DAC Tile n Outclk Frequency	C_DACn_Outclk_Freq	400
DAC Tile n PLL Enable	C_DACn_PLL_Enable	FALSE
DAC Tile n Refclk Frequency	C_DACn_Refclk_Freq	6400
DAC Tile n reference clock input divider	C_DAC_Refclk_Div	1
DAC Tile n Sampling Rate	C_DACn_Sampling_Rate	6.5
DAC Tile n Multi Tile Sync	C_DACn_Multi_Tile_Sync	FALSE
DAC Tile n Multiband configuration mode	C_DACn_Mode	0
DAC Tile n Output Current	C_DACn_VOP	20
DAC Tile n Clock source	C_DACn_Clock_Source	n+4
DAC Tile n Clock distribution	C_DACn_Clock_Distr	0
DAC Tile y Converter z Data Type	C_DAC_Data_Typeyz	0
DAC Tile y Converter z Data Width	C_DAC_Data_Widthyz	16
DAC Tile y Converter z Interpolation Mode	C_DAC_Interpolation_Modeyz	0
DAC Tile y Converter z Decoder Mode	C_DAC_Decoder_Modeyz	0
DAC Tile y Converter z Invsinc Ctrl	C_DAC_Invsinc_Ctrlyz	FALSE
DAC Tile y Converter z Mixer Mode	C_DAC_Mixer_Modeyz	2
DAC Tile y Converter z Nyquist Zone	C_DAC_Nyquistyz	1
DAC Tile y Converter z Enable	C_DAC_Sliceyz_Enable	FALSE
DAC Tile y Converter z Mixer Type	C_DAC_Mixer_Typeyz	3
DAC Tile y Converter z Fine Mixer Frequency	C_DAC_NCO_Freqyz	0.0
DAC Tile y Converter z Fine Mixer Phase	C_DAC_NCO_Phaseyz	0
DAC Tile y Converter z Coarse Mixer Phase	C_DAC_Coarse_Mixer_Freqyz	0
DAC Tile y Converter z Invert Q Output	C_DAC_Neg_Quadratureyz	FALSE

**Notes:**

- n = 0 to 3.
- y = 0 to 3, z = 0 to 3.

## Output Generation

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

---

## Simulation

Simulation is supported using Mentor Graphics Questa Advanced Simulator in the Vivado® Design Suite. 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](#)). The RF-ADC and RF-DAC blocks have bit-accurate real number simulation models. This means that each of the datapath features can be simulated and the related analog signals observed.

### Related Information

[Example Design](#)

---

## Synthesis and Implementation

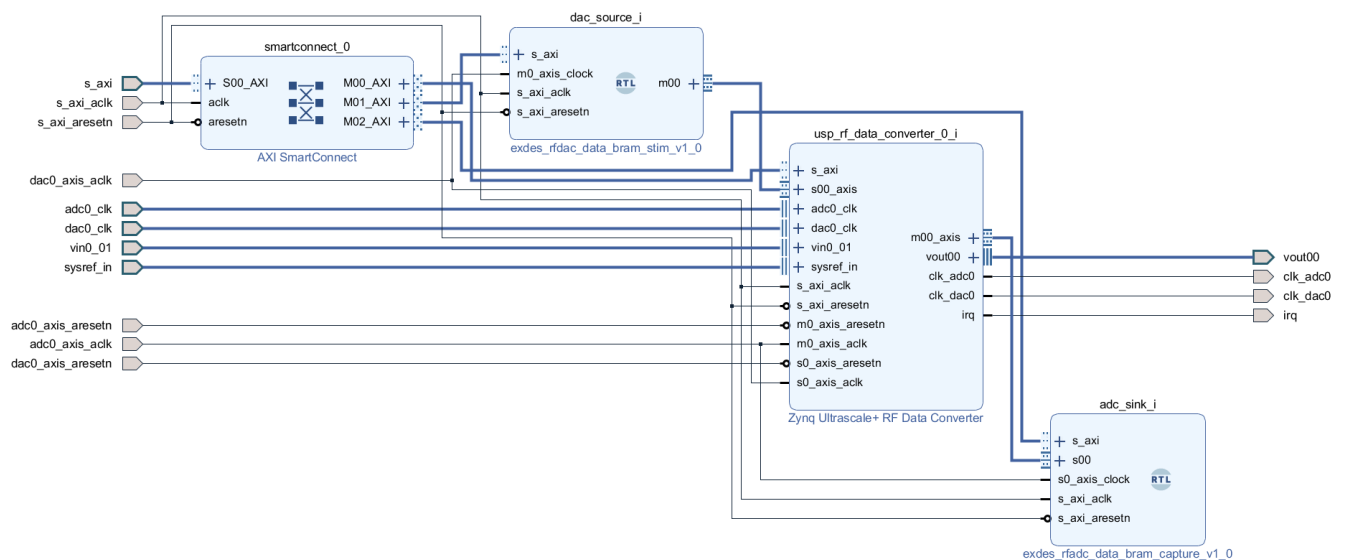
For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* ([UG896](#)).



# Example Design

The following figure shows the example design that is provided with the core.

Figure 98: Example Design



The complete example design can be opened as a separate project by right-clicking the core in the project hierarchy after it has been customized using the IP catalog. Right-click the <component\_name>.xci file in the Design Sources hierarchy in the Sources window and select **Open IP Example Design**. This opens a new Vivado® IP integrator project in a new window with a complete RF Data Converter IP example design.

The Zynq® UltraScale+™ RFSoc RF Data Converter example design consists of the following:

- An instance of the Zynq® UltraScale+™ RF Data Converter IP core
- Data stimulus block for RF-DAC input
- Data capture block for RF-ADC output
- Smartconnect for AXI4-Lite addressing of the design
- An example device-level wrapper containing BUFGs

## RF-ADC Data Capture Block

The RF-ADC data capture block enables the capture of the data converted by the RF-ADC(s) in the Zynq® UltraScale+™ RF Data Converter core.

The data from each enabled AXI4-Stream output on the Zynq® UltraScale+™ RF Data Converter is stored in a separate channel in the data capture block. As each converter can have up to four AXI4-Stream interfaces, there are a maximum of 16 channels present in the data capture block. The enabled AXI4-Stream interfaces are mapped into consecutive channels starting with ADC0. The storage in each channel consists of 128 kbits of block RAM.

The address map for the data capture block is shown below.

*Table 58: RF-ADC Data Capture Address Map*

Address	Register	Access Type	Description
0x0000_0000	Memory ID code	RO	Memory Information Bit 31. High. Signaling a data capture block. Bits 30:24. Number of channels in the data capture block. Bits 23:0. Total memory size of the data capture block.
0x0000_0004	Start Data	R/W	Starts the data capture. Bits 31:1. Reserved Bit 0. Writing a 1 to this register starts the storage of data in to the block RAM. This register remains high until the capture has been completed.
0x0000_0008	Channel Enable	R/W	Enables each channel. Bits 31:16. Reserved Bits 15:0. Channel enable. Each bit controls one channel (bit n controlling channel n). The channel should be enabled before starting a capture. When the capture is complete the channel should be disabled before the next capture can start.
0x0000_0010 - 0x0000_004C	Samples to capture (channel 0 to 15)	R/W	The number of samples to capture from the RF-ADC. One address per channel.
0x0000_4000 - 0x0000_7FFC	Channel 0 memory	R/W	Channel 0 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0000_8000 - 0x0000_BFFC	Channel 1 memory	R/W	Channel 1 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0000_C000 - 0x0000_FFFC	Channel 2 memory	R/W	Channel 2 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample

Table 58: RF-ADC Data Capture Address Map (cont'd)

Address	Register	Access Type	Description
0x0001_0000 - 0x0001_3FFC	Channel 3 memory	R/W	Channel 3 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0001_4000 - 0x0001_7FFC	Channel 4 memory	R/W	Channel 4 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0001_8000 - 0x0001_BFFC	Channel 5 memory	R/W	Channel 5 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0001_C000 - 0x0001_FFFC	Channel 6 memory	R/W	Channel 6 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0002_0000 - 0x0002_3FFC	Channel 7 memory	R/W	Channel 7 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0002_4000 - 0x0002_7FFC	Channel 8 memory	R/W	Channel 8 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0002_8000 - 0x0002_BFFC	Channel 9 memory	R/W	Channel 9 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0002_C000 - 0x0002_FFFC	Channel 10 memory	R/W	Channel 10 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0003_0000 - 0x0003_3FFC	Channel 11 memory	R/W	Channel 11 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0003_4000 - 0x0003_7FFC	Channel 12 memory	R/W	Channel 12 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0003_8000 - 0x0003_BFFC	Channel 13 memory	R/W	Channel 13 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0003_C000 - 0x0003_FFFC	Channel 14 memory	R/W	Channel 14 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0004_0000 - 0x0004_3FFC	Channel 15 memory	R/W	Channel 15 data capture Bits 31:16. Odd data sample Bits 15:0. Even data sample

## RF-DAC Data Stimulus Block

The RF-DAC data stimulus block consists of a 128 kbits block RAM that can be loaded with samples which are then sent to the RF-DAC (s) in the Zynq® UltraScale+™ RF Data Converter core.

Each channel in the stimulus block drives an AXI4-Stream on the Zynq® UltraScale+™ RF Data Converter IP core. As each converter can have up to four AXI4-Stream interfaces, there are a maximum of 16 channels in the data stimulus block. Each enabled AXI4-Stream interface is mapped into consecutive channels starting with DAC0.

The address map for the data stimulus block is shown below.

**Table 59: RF-DAC Data Stimulus Address Map**

Address	Register	Access Type	Description
0x0000_0000	Memory ID code	RO	Memory Information Bit 31. Low. Signaling a data stimulus block. Bits 30:24. Number of channels in the data stimulus block. Bits 23:0. Total memory size of the data stimulus block.
0x0000_0004	Start data	R/W	Starts the transmission of data Bits 31:1. Reserved Bit 0. Start data (self clearing). Setting this bit high starts the transmission of the data stored in memory to the RF-DAC. To stop transmission the channel is disabled using the channel enable register.
0x0000_0008	Channel enable	R/W	Enables each channel. Bits 31:16. Reserved Bits 15:0. Channel enable. Each bit controls one channel (bit n controlling channel n). The channel should be enabled before issuing a start data command. To stop the test the channel should be disabled.
0x0000_0010 - 0x0000_004C	Samples to generate	R/W	The number of samples to transmit before wrapping around to the beginning of the sequence. One address per channel.
0x0000_4000 - 0x0000_7FFC	Channel 0 memory	R/W	Channel 0 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0000_8000 - 0x0000_BFFC	Channel 1 memory	R/W	Channel 1 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample

Table 59: RF-DAC Data Stimulus Address Map (cont'd)

Address	Register	Access Type	Description
0x0000_C000 - 0x0000_FFFC	Channel 2 memory	R/W	Channel 2 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0001_0000 - 0x0001_3FFC	Channel 3 memory	R/W	Channel 3 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0001_4000 - 0x0001_7FFC	Channel 4 memory	R/W	Channel 4 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0001_8000 - 0x0001_BFFC	Channel 5 memory	R/W	Channel 5 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0001_C000 - 0x0001_FFFC	Channel 6 memory	R/W	Channel 6 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0002_0000 - 0x0002_3FFC	Channel 7 memory	R/W	Channel 7 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0002_4000 - 0x0002_7FFC	Channel 8 memory	R/W	Channel 8 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0002_8000 - 0x0002_BFFC	Channel 9 memory	R/W	Channel 9 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0002_C000 - 0x0002_FFFC	Channel 10 memory	R/W	Channel 10 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0003_0000 - 0x0003_3FFC	Channel 11 memory	R/W	Channel 11 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0003_4000 - 0x0003_7FFC	Channel 12 memory	R/W	Channel 12 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0003_8000 - 0x0003_BFFC	Channel 13 memory	R/W	Channel 13 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0003_C000 - 0x0003_FFFC	Channel 14 memory	R/W	Channel 14 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample
0x0004_0000 - 0x0004_3FFC	Channel 15 memory	R/W	Channel 15 data stimulus Bits 31:16. Odd data sample Bits 15:0. Even data sample

## Digital Data Format

Each word interface to the RF-DAC and RF-ADC is 16 bits wide, even though the RF-DAC resolution is 14 bits and the RF-ADC 12 bits. The word is in twos complement format, giving the range shown in the following table.

The RF-ADC and RF-DAC hex values are displayed appropriately left shifted to align to the input most significant bit.

*Table 60: RF-ADC and RF-DAC Word Interface*

	AXI4-Stream I/F		RF-ADC Resolution (<<4)		RF-DAC Resolution (<<2)	
	Hex	Dec	Hex	Dec	Hex	Dec
Max positive	0x7FFF	32767	0x7FF0	2047	0x7FFC	8191
	0x0001	1	0x0010	1	0x0004	1
Zero	0x0000	0	0x0000	0	0x0000	0
	0xFFFF	-1	0xFFFF0	-1	0xFFFFC	-1
Min negative	0x8000	-32768	0x8000	-2048	0x8000	-8192

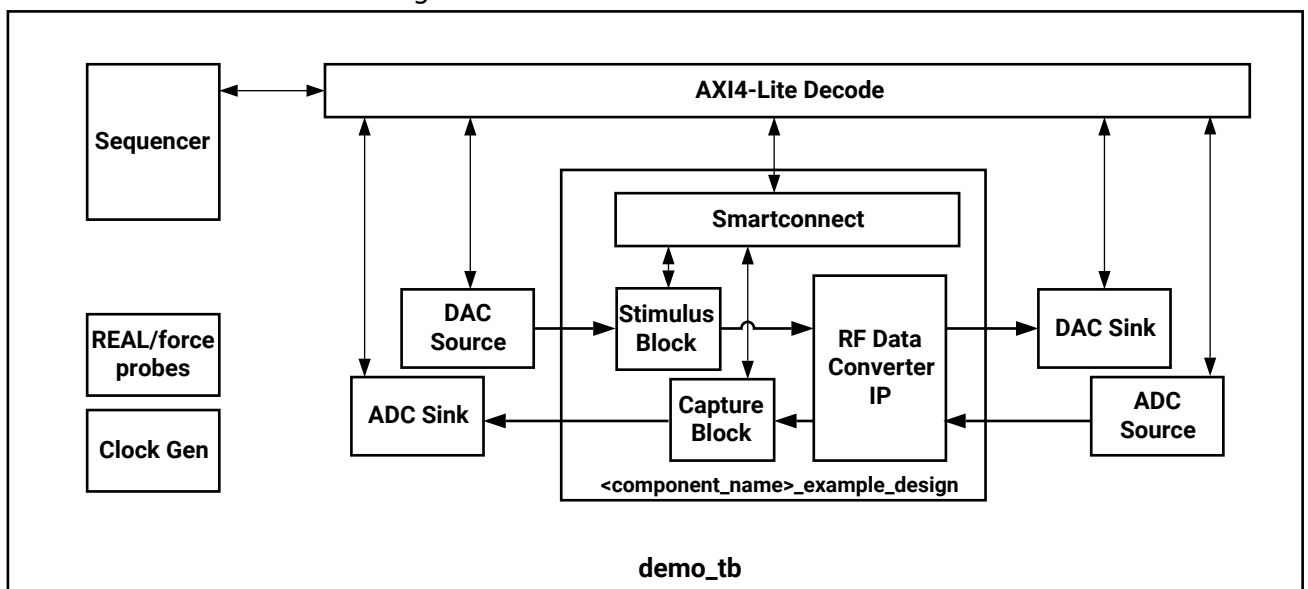
# Test Bench

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

For information on setting up and running simulations in the Vivado Design Suite, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)*. Zynq® UltraScale+™ RFSoc RF Data Converter core supports behavioral simulation along with post-implementation functional simulation.

The following figure shows the structure of the Zynq® UltraScale+™ RFSoc RF Data Converter test bench.

Figure 99: Demonstration Test Bench



X21611-092519

The following blocks are present in the demonstration test bench.

- **Clock Generator:** This block produces the required AXI4-Lite and AXI4-Stream clocks for the design.
- **DAC Source:** This block outputs a series of samples representing a tone at a set frequency. These samples are written in to the data stimulus block in the example design.
- **DAC Sink:** This block performs a FFT on the RF-DAC analog output. It checks that the input tone appears at the expected frequency.

- **ADC Source:** The RF-ADC analog inputs are driven by a tone at a set frequency. If the mixers in the RF-ADC are enabled the tone is mixed with a sine wave before being input to the RF-ADC. The mixer in the ADC source block is set to run at the negative of the mixer frequency in the converter.
- **ADC Sink:** This block performs a FFT on the RF-ADC digital output. It checks that the input tone appears at the expected frequency.
- **Sequencer:** This block is responsible for managing accesses to the test bench components through an AXI4-Lite interface. It contains examples of how to access and configure the basic test harness.

The sequencer in the demonstration test bench runs through the following stages. At some stages the Power-up sequence state machine is stopped despite the sequence not being complete. This is in order to speed up the simulation. It is not essential to follow the steps in the demonstration test bench for simulation of the converters. The state machine can be allowed to run from start to finish without being restarted.

1. The test bench components are set up. The sources and sinks are set up with the configuration information from the IP core. The power-up sequence state machine is started and the initial configuration stage is completed.
2. Sine waves from the DAC Source block are written into the memory in the stimulus block for each enabled channel.
3. Some of the configuration registers are then changed to help speed up the simulation. For example, in the RF-ADC the dither is disabled. This allows the calibration step in the power-up sequence to be skipped, saving simulation time. When the fine mixers are enabled they can also be reprogrammed at this point to minimize any spectral leakage in the FFT.
4. The RF-DACs and RF-ADCs are then run to the stage before the converter clocks are released to the digital section of the converters. The simulation is slowed when the converter clocks are enabled and stopping the state machine at this stage helps to reduce run time.
5. The RF-DACs are fully powered up and the transmission of data from the stimulus block is started.
6. The sequencer waits for the DAC Sink blocks for each RF-DAC output to report their FFT results. The RF-DACs are then powered down.
7. The RF-ADCs are powered up to the start of the foreground calibration stage and data capture is started for each of the enabled channels.
8. When data capture is complete the RF-ADCs are powered down. The sequencer then reads the captured data back and sends it to the ADC Sink block.
9. The sequencer waits for the ADC Sink blocks for each enabled channel to report their FFT results. The simulation then completes.

When  $F_s/2$  mixing is enabled output similar to that in the following figures should be visible on the analog I/O.



Figure 100: RF-ADC Analog I/O

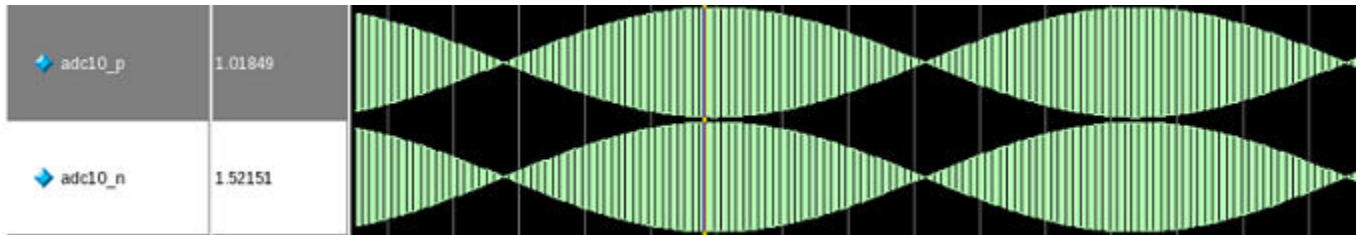
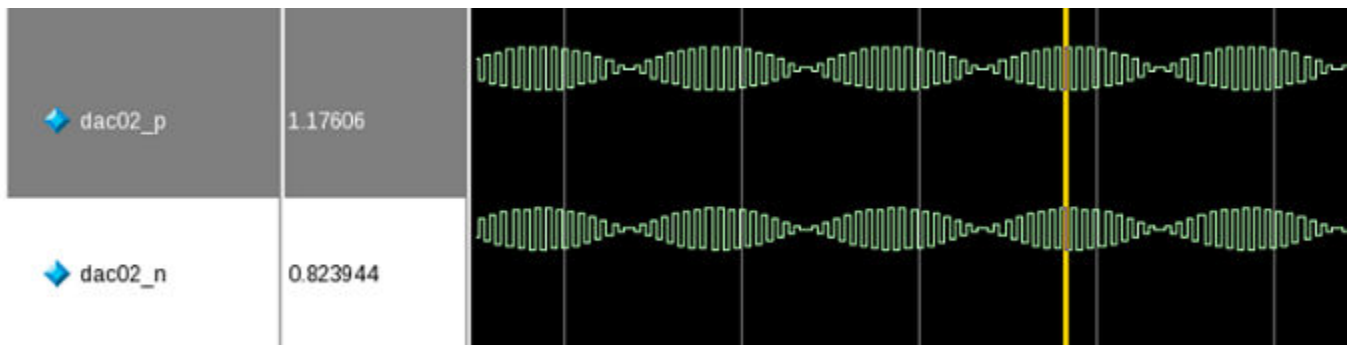


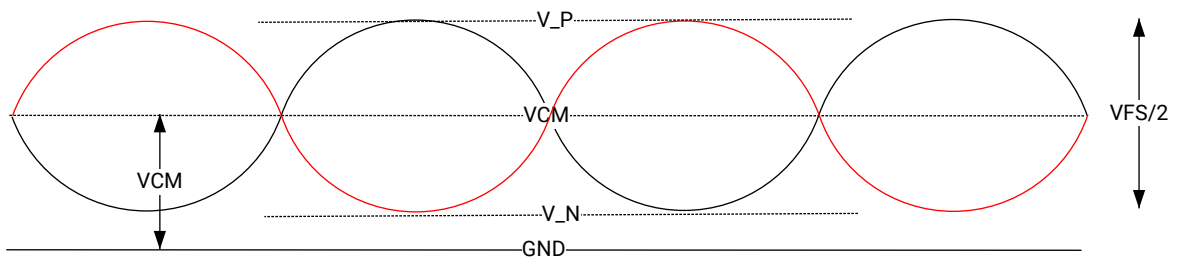
Figure 101: RF-DAC Analog I/O



## Analog Signaling

The RF-DAC and RF-ADC analog I/O are differential and are driven as shown in the following figure in the test bench.

Figure 102: Test Bench Analog Signaling



X18809-072817

The internal RF-DAC and RF-ADC model analog functions using Real Number models. In the Verilog example design these must be passed as 64-bit wires, using *force* and *hierarchical assignments* to set and read the values.

The simulated transfer function uses a full-scale voltage swing (VFS) of 1.14V and a common mode voltage (VCM) of 1.25V for the RF-ADC. For the actual full-scale input voltage range see the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)*.

A code excerpt from the `demo_tb.v` file is shown below. This is generated when you generate the example design. This can be used as a guide to show how to drive and read data into RF-DAC and RF-ADC primitives.

```
//-----
// Force the analog signals.
//-----
`ifndef DO_NOT_USE_RFAMS_REAL_SIGNAL_FORCE
// RF-ADC
real adc00_p;
real adc00_n;

always @ (*) begin
    // Map the RF-ADC signals to top level
    adc00_p = $bitstoreal(adc_source.vout_00_p);
    adc00_n = $bitstoreal(adc_source.vout_00_n);

    // Force the RF-ADC analog input
    force
DUT.i_rf_dut_block.inst.rf_dut_rf_wrapper.i.rx0_u_adc.SIP_HSADC_INST.VIN0_P
= adc00_p;
    force
DUT.i_rf_dut_block.inst.rf_dut_rf_wrapper.i.rx0_u_adc.SIP_HSADC_INST.VIN0_N
= adc00_n;

end

// RF-DAC

real dac00_p;
real dac00_n;

always @ (*) begin
    // Map the RF-DAC signals to the top level
    dac00_p =
DUT.i_rf_dut_block.inst.rf_dut_rf_wrapper.i.tx0_u_dac.SIP_HSDAC_INST.VOUT0_P
;
    dac00_n =
DUT.i_rf_dut_block.inst.rf_dut_rf_wrapper.i.tx0_u_dac.SIP_HSDAC_INST.VOUT0_N
;

    // force the RF-DAC output onto the RF-DAC sink
    force dac_sink.vin_00_p = $realtobits(dac00_p);
    force dac_sink.vin_00_n = $realtobits(dac00_n);

end
`endif
```

The code excerpt below shows analog data loopback from the RF-DAC to the RF-ADC on a Gen 1 device. Care must be taken to map the RF-DAC analog data output to within the correct range for the RF-ADC data input.

```

real dac00_p;
real dac00_n;
always @ (*) begin

    dac00_p =
DUT.usp_rf_data_converter_0_ex_i.usp_rf_data_converter_0.inst.rfdc_ex_usp_rf
_data_converter_0_0_rf_wrapper_i.tx0_u_dac.SIP_HSDAC_INST.VOUT0_P;
    dac00_n =
DUT.usp_rf_data_converter_0_ex_i.usp_rf_data_converter_0.inst.rfdc_ex_usp_rf
_data_converter_0_0_rf_wrapper_i.tx0_u_dac.SIP_HSDAC_INST.VOUT0_N;

    // Force the ADC analog input
    force
DUT.usp_rf_data_converter_0_ex_i.usp_rf_data_converter_0.inst.rfdc_ex_usp_rf
_data_converter_0_0_rf_wrapper_i.rx0_u_adc.SIP_HSADC_INST.VIN_I01_P = 1.25
+ (dac00_p-dac00_n); // ADC VCM = 1.25V
    force
DUT.usp_rf_data_converter_0_ex_i.usp_rf_data_converter_0.inst.rfdc_ex_usp_rf
_data_converter_0_0_rf_wrapper_i.rx0_u_adc.SIP_HSADC_INST.VIN_I01_N = 1.25
- (dac00_p-dac00_n); // ADC VCM = 1.25V
end

```

# Upgrading

## Changes from V2.1 to V2.2

### Parameter Changes

The RF-ADC and RF-DAC output divide parameters have been removed from version 2.2 of the IP. These are set internally by the IP.

*Table 61: Parameter Changes*

Parameter <sup>1</sup>	Upgrade Action
ADCX_Outdiv	Do not set manually. Allow the IP to set this parameter.
DACX_Outdiv	Do not set manually. Allow the IP to set this parameter.

**Notes:**

1. X refers to the location of the tile in the converter column.

## Changes from V2.0 to V2.1

### Port Changes

In version 2.1 a new port has been added in the Debug group. The powerup\_state output will be asserted when the POR Finite State Machine detects an error in the power-up sequence.

*Table 62: Port Changes*

Port <sup>1</sup>	Direction	Upgrade Action
adcX_powerup_state	Out	Leave open
dacX_powerup_state	Out	Leave open

**Notes:**

1. X refers to the location of the tile in the converter column

## Parameter Changes

The mapping of the decimation and interpolation settings into the C\_ADC\_Decimation\_Mode and C\_DAC\_Interpolation\_Mode IP variables has been changed in version 2.1. The changes are shown below.

**Table 63: Decimation and Interpolation Parameter Mode Changes**

Parameter <sup>1</sup>	Version 2.0	Version 2.1
C_ADC_Decimation_ModeXY	<ul style="list-style-type: none"> <li>0 = Decimation off</li> <li>1 = Decimation x1</li> <li>2 = Decimation x2</li> <li>3 = Decimation x4</li> <li>4 = Decimation x8</li> </ul>	<ul style="list-style-type: none"> <li>0 = Decimation off</li> <li>1 = Decimation x1</li> <li>2 = Decimation x2</li> <li>4 = Decimation x4</li> <li>8 = Decimation x8</li> </ul>
C_DAC_Interpolation_ModeXY	<ul style="list-style-type: none"> <li>0 = Interpolation off</li> <li>1 = Interpolation x1</li> <li>2 = Interpolation x2</li> <li>3 = Interpolation x4</li> <li>4 = Interpolation x8</li> </ul>	<ul style="list-style-type: none"> <li>0 = Interpolation off</li> <li>1 = Interpolation x1</li> <li>2 = Interpolation x2</li> <li>4 = Interpolation x4</li> <li>8 = Interpolation x8</li> </ul>

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DUC/DDC block in the tile (0 to 3).

A new option has been added to the C\_ADC\_Mixer\_Type and C\_DAC\_Mixer\_Type parameters, The mixers can now be set to "Off" in addition to the previously defined settings.

**Table 64: Mixer Type Parameter Changes**

Parameter <sup>1</sup>	Version 2.0	Version 2.1
C_ADC_Mixer_TypeXY	<ul style="list-style-type: none"> <li>0 = Bypassed</li> <li>1 = Coarse</li> <li>2 = Fine</li> </ul>	<ul style="list-style-type: none"> <li>0 = Bypassed</li> <li>1 = Coarse</li> <li>2 = Fine</li> <li>3 = Off</li> </ul>
C_DAC_Mixer_TypeXY	<ul style="list-style-type: none"> <li>0 = Bypassed</li> <li>1 = Coarse</li> <li>2 = Fine</li> </ul>	<ul style="list-style-type: none"> <li>0 = Bypassed</li> <li>1 = Coarse</li> <li>2 = Fine</li> <li>3 = Off</li> </ul>

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DUC/DDC block in the tile (0 to 3).

## Changes from V1.2 to V2.0

### Port Changes

Separate RF-ADC and RF-DAC versions of the `user_sysref` port have been added. This change affects cores with multi-tile synchronization enabled.

*Table 65: Port Changes in Version 2.0*

Port	Direction	Upgrade Action
<code>user_sysref_adc</code>	In	Connect to <code>user_sysref</code> , if required
<code>user_sysref_dac</code>	In	Connect to <code>user_sysref</code> , if required

## Changes from V1.1 to V1.2

### Port Changes

When the RF-ADC real-time signals are enabled the ports in the following table have been added.

*Table 66: Port Changes in Version 1.2*

Port	Direction	Upgrade Action
<b>2 GSPS RF-ADC<sup>1</sup></b>		
<code>adcXZ_over_range</code>	Out	Leave open
<code>adcXZ_over_voltage</code>	Out	Leave open
<b>4 GSPS RF-ADC<sup>2</sup></b>		
<code>adcX_ZZ_over_range</code>	Out	Leave open
<code>adcX_ZZ_over_voltage</code>	Out	Leave open

**Notes:**

1. X refers to the location of the tile in the converter column. Z refers to the location of the RF-ADC in the tile (0 to 3).
2. X refers to the location of the tile in the converter column. ZZ is either 01 (the lower RF-ADC in the tile) or 23 (the upper RF-ADC in the tile)

When the RF-ADC real-time signals are enabled the ports in the following table have been renamed.

Table 67: Ports Renamed in Version 1.2

Port	Previous Name (v1.1)	Direction	Upgrade Action
<b>2 GSPS RF-ADC<sup>1</sup></b>			
adcXZ_over_threshold1	adcXY_over_threshold1	Out	Leave open
adcXZ_over_threshold2	adcXY_over_threshold2	Out	Leave open
<b>4 GSPS RF-ADC<sup>2</sup></b>			
adcX_ZZ_over_threshold1	adcXY_over_threshold1	Out	Leave open
adcX_ZZ_over_threshold2	adcXY_over_threshold2	Out	Leave open

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DDC block in the tile (0 to 3). Z refers to the location of the RF-ADC in the tile (0 to 3).
2. X refers to the location of the tile in the converter column. Y refers to the location of the DDC block in the tile (0 to 3). ZZ is either 01 (the lower RF-ADC in the tile) or 23 (the upper RF-ADC in the tile).

## Parameter Changes

There are no parameter changes.

# 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 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. The [Xilinx Community Forums](#) are also available where members can learn, participate, share, and ask questions about Xilinx solutions.

### Documentation

This product guide is the main document associated with the 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<sup>®</sup> 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 Record for the Core***

AR [69907](#).

## **Technical Support**

Xilinx provides technical support on the [Xilinx Community Forums](#) 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 ask questions, navigate to the [Xilinx Community Forums](#).

---

## **Debug Tools**

There are many tools available to address Zynq® UltraScale+™ RF Data Converter design issues. It is important to know which tools are useful for debugging various situations.

### **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 to interact with the logic debug LogiCORE 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](#)).

---

## Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado® debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems.

### General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.
- If your outputs go to 0, check your licensing.

---

## Interface Debug

### AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk` and `aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.
- If the simulation has been run, verify in simulation and/or a debug feature capture that the waveform is correct for accessing the AXI4-Lite interface.

### AXI4-Stream Interfaces

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 `ac1k` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed.
- Check core configuration.

**Related Information**[Interfacing to the AXI4-Stream Interface](#)

# Zynq UltraScale+ RFSoc RF Data Converter Bare-metal/Linux Driver

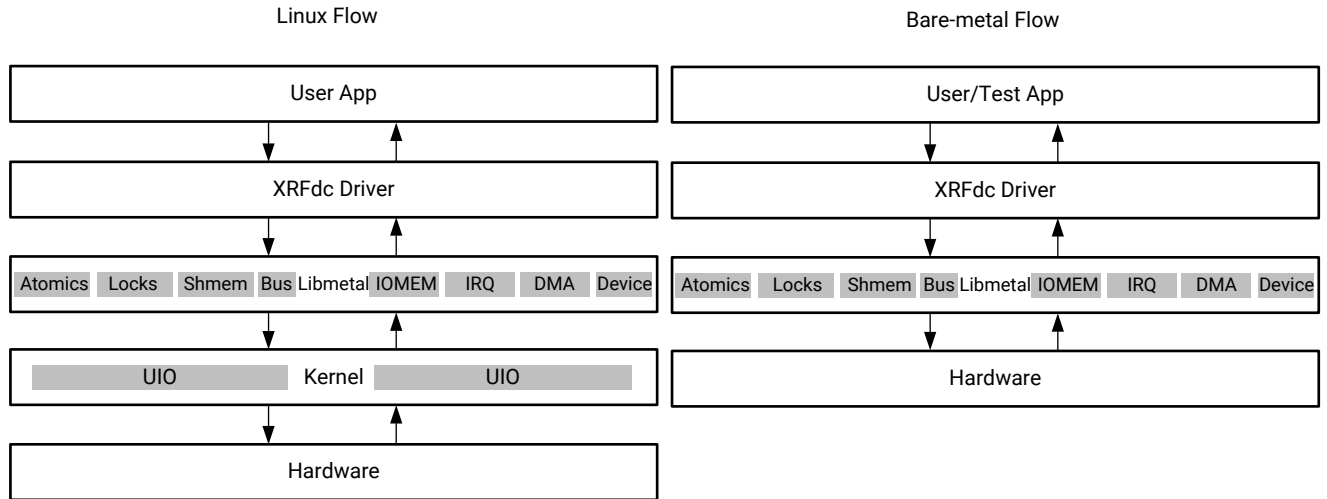
---

## Overview

The RFdc driver API functions for the Zynq<sup>®</sup> UltraScale+<sup>™</sup> RFSoc RF Data Converter are described in this chapter. The same driver is used for both bare metal and Linux. The driver for both software platforms runs on the libmetal software layer provided by Xilinx. The driver is composed of the following files:

- API
  - `xrfdc.c`: The user interface API functions are implemented in this file.
  - `xrfdc.h`: The user interface API prototypes are provided in this file. The file provides prototypes of the driver instance structure and all other structures used across the APIs. The file implements utility in-line functions to access various data in the driver and IP.
  - `xrfdc_mts.c`: The multi-tile synchronization API functions are implemented in this file.
  - `xrfdc_mts.h`: The multi-tile synchronization API prototypes are provided in this file.
  - `xrfdc_mixer.c`: The mixer API functions are implemented in this file.
  - `xrfdc_clock.c`: The clock related API functions are implemented in this file.
  - `xrfdc_mb.c`: The multi-band API functions are implemented in this file.
- Hardware Register Map
  - `xrfdc_hw.h`: Definitions for the hardware register maps are provided in this file. The file also provides masks for various relevant fields for the Zynq UltraScale+ RFSoc register interface.
- Interrupt Handling
  - `xrfdc_intr.c`: Implements functions for the handling of various interrupts and errors from the IP core.

Figure 103: Linux and Bare-metal Software Flow Block Diagram



X18676-072817

## About Libmetal

Libmetal is a Xilinx developed open source software stack that provides common user APIs to access devices, handle device interrupts, and request memory across the following operating environments:

- Linux user space (based on Userspace IO and Virtual Function IO support in the kernel)
- RTOS (with and without virtual memory)
- Bare-metal environments

The libmetal I/O region abstraction provides access to memory mapped I/O and shared memory regions. This includes primitives to read and write memory with ordering constraints and the ability to translate between physical and virtual addressing on systems that support virtual memory.

---

## Data Structures

All the data structures used in the driver are defined in the `xrfdc.h` and `xrfdc_mts.h` files.

## struct XRFdc

This is the instance structure for internal use.

```
XRFdc_Config RFdc_Config;
u32 IsReady;
u32 ADC4GSPS;
metal_phys_addr_t BaseAddr;
struct metal_io_region *io;
struct metal_device *device;
XRFdc_DAC_Tile DAC_Tile[4];
XRFdc_ADC_Tile ADC_Tile[4];
XRFds_StatusHandler StatusHandler;
void *CallBackRef;
u8 UpdateMixerScale;
```

### Description

- **XRFdc\_Config RFdc\_Config:** Driver structure configuration.
- **u32 IsReady:** This flag is used to indicate that the driver is ready.
- **u32 ADC4GSPS:** This flag is used to indicate if RF-ADC is 4 GSPS.
- **metal\_phys\_addr\_t BaseAddr:** Base address.
- **struct metal\_io\_region:** Libmetal IO structure.
- **struct metal\_device \*device:** Libmetal device structure.
- **XRFdc\_DAC\_Tile DAC\_Tile[4]:** RF-DAC tile structure for four tiles.
- **XRFdc\_ADC\_Tile ADC\_Tile[4]:** RF-ADC tile structure for four tiles.
- **XRFds\_StatusHandler StatusHandler:** Event handler function.
- **void \*CallBackRef:** Callback reference for event handler.
- **u8 UpdateMixerScale:** Set to 1 to overwrite mixer scale.

## struct XRFdc\_Config

This structure is for internal driver use (`metal_phys_addr_t` represents unsigned long).

```
u32 DeviceId;
metal_phys_addr_t BaseAddr;
u32 ADCType;
u32 MasterADCTile;
u32 MasterDACTile;
u32 ADCSysRefSource;
u32 DACSysRefSource;
XRFdc_DACTile_Config DACTile_Config[4];
XRFdc_ADCTile_Config ADCTile_Config[4];
```

## Description

- **u32 ADCType:** Quad RF-ADC part or Dual RF-ADC part.
- **u32 MasterADCTile:** Corresponds to the `C_ADC_Master` IP parameter.
- **u32 MasterDACTile :** Corresponds to the `C_DAC_Master` IP parameter.
- **u32 ADCSysRefSource :** Corresponds to the `C_ADC_Sysref_Source` IP parameter.
- **u32 DACSysRefSource:** Corresponds to the `C_DAC_Sysref_Source` IP parameter.

## struct XRFdc\_PLL\_Settings

This structure is for internal driver use.

```

u32 Enabled;
double RefClkFreq;
double SampleRate;
u32 RefClkDivider;
u32 FeedbackDivider;
u32 OutputDivider;
u32 FractionalMode;
u64 FractionalData;
u32 FractWidth;
  
```

## Description

- **u32 Enabled:** Indicates if the PLL is enabled (1) or disabled (0).
- **double RefClkFreq:** Reference clock frequency (MHz).
- **double SampleRate:** Sampling rate (GHz).
- **u32 RefClkDivider:** Reference clock divider.
- **u32 FeedbackDivider:** Feedback divider.
- **u32 OutputDivider:** Output divider.
- **u32 FractionalMode:** Fractional mode. Currently not supported.
- **u64 FractionalData:** Fractional part of the feedback divider. Currently not supported.
- **u32 FractWidth:** Fractional data width. Currently not supported.

## struct XRFdc\_QMC\_Settings

This structure is used to set or get the QMC settings.

```
u32 EnablePhase;
u32 EnableGain;
double GainCorrectionFactor;
double PhaseCorrectionFactor;
s32 OffsetCorrectionFactor;
u32 EventSource;
```

### Description

- **u32 EnablePhase:** Indicates if phase is enabled (1) or disabled (0).
- **u32 EnableGain:** Indicates if gain is enabled(1) or disabled (0).
- **double GainCorrectionFactor:** Gain correction factor. Range: 0 to 2.0 (Exclusive).
- **double PhaseCorrectionFactor:** Phase correction factor. Range: +/- 26.5 degrees (Exclusive).
- **s32 OffsetCorrectionFactor:** Offset correction factor is adding a fixed LSB value to the sampled signal.
- **u32 EventSource:** Event source for QMC settings. XRFDC\_EVNT\_SRC\_\* represents valid values.

Table 68: Valid Macros for EventSource

Macro	Description
XRFDC_EVNT_SRC_IMMEDIATE	Update after register write
XRFDC_EVNT_SRC_SLICE	Update using SLICE event source
XRFDC_EVNT_SRC_TILE	Update using TILE event source
XRFDC_EVNT_SRC_SYSREF	Update using SYSREF event source
XRFDC_EVNT_SRC_MARKER	Update using MARKER event source
XRFDC_EVNT_SRC_PL	Update using PL event source

## struct XRFdc\_CoarseDelay\_Settings

This structure is used to set or get coarse delay settings.

```
u32 CoarseDelay;
u32 EventSource;
```

### Description

- **u32 CoarseDelay:** Coarse delay in the number of samples. Range: 0 to 7.



- **u32 EventSource:** Event source for coarse delay settings. XRFDC\_EVNT\_SRC\_\* represents valid values.

**Note:** See QMC\_Settings structure for valid EventSource macros.

### Related Information

[struct XRFdc\\_QMC\\_Settings](#)

## struct XRFdc\_Mixer\_Settings

This structure is used to set or get mixer settings.

```
double Freq;
double PhaseOffset;
u32 EventSource;
u32 CoarseMixFreq;
u32 MixerMode;
u8 FineMixerScale;
u8 MixerType;
```

### Description

- **double Freq:** NCO frequency. Range: -Fs to Fs (MHz).
- **double PhaseOffset:** NCO phase offset. Range: -180 to 180 (Exclusive).
- **u32 EventSource:** Event source for mixer settings. XRFDC\_EVNT\_SRC\_\* represents valid values.

**Note:** See QMC\_Settings structure for valid EventSource macros

- **u32 CoarseMixFreq:** Coarse mixer frequency. XRFDC\_COARSE\_MIX\_\* represents valid values.

*Table 69: Valid Macros for CoarseMixFreq*

Macro	Description
XRFDC_COARSE_MIX_OFF	Coarse Mixer frequency is OFF
XRFDC_COARSE_MIX_SAMPLE_FREQ_BY_TWO	Coarse Mixer frequency is Fs/2
XRFDC_COARSE_MIX_SAMPLE_FREQ_BY_FOUR	Coarse Mixer frequency is Fs/4
XRFDC_COARSE_MIX_MIN_SAMPLE_FREQ_BY_FOUR	Coarse Mixer frequency is -Fs/4
XRFDC_COARSE_MIX_BYPASS	Coarse Mixer frequency is BYPASS

- **u32 MixerMode:** Mixer mode for fine or coarse mixer. XRFDC\_MIXER\_MODE\_\* represents valid values.

Table 70: Valid Macros for MixerMode

Macro	Description
XRFDC_MIXER_MODE_OFF	Mixer mode is OFF (only for Fine Mixer)
XRFDC_MIXER_MODE_C2C	Mixer mode is Complex to Complex
XRFDC_MIXER_MODE_C2R	Mixer mode is Complex to Real
XRFDC_MIXER_MODE_R2C	Mixer mode is Real to Complex

- **u8 FineMixerScale:** NCO output scale. XRFDC\_MIXER\_SCALE\_\* represents valid values.

Table 71: Valid Macros for FineMixerScale

Macro	Description
XRFDC_MIXER_SCALE_AUTO	Fine Mixer scale will be auto updated
XRFDC_MIXER_SCALE_1P0	Fine Mixer Scale is set to 1.0
XRFDC_MIXER_SCALE_0P7	Fine Mixer Scale is set to 0.7

- **u8 MixerType:** Mixer Type indicates coarse or fine mixer. XRFDC\_MIXER\_TYPE\_\* represents valid values.

Table 72: Valid Macros for MixerType

Macro	Description
XRFDC_MIXER_TYPE_COARSE	Mixer Type is Coarse Mixer
XRFDC_MIXER_TYPE_FINE	Mixer Type is Fine Mixer
XRFDC_MIXER_TYPE_OFF	The Mixer is Off
XRFDC_MIXER_TYPE_DISABLED	The Mixer is permanently off (from hardware design)

### Related Information

[struct XRFdc\\_QMC\\_Settings](#)

## struct XRFdc\_Threshold\_Settings

This structure is used to set or get RF-ADC threshold settings.

```

u32 UpdateThreshold;
u32 ThresholdMode[2];
u32 ThresholdAvgVal[2];
u32 ThresholdUnderVal[2];
u32 ThresholdOverVal[2];
  
```

### Description

- **u32 UpdateThreshold:** Selects which threshold to update. XRFDC\_UPDATE\_THRESHOLD\_\* represents valid values.

Table 73: Valid Macros for UpdateThreshold

Macro	Description
XRFDC_UPDATE_THRESHOLD_0	Update for Threshold0
XRFDC_UPDATE_THRESHOLD_1	Update for Threshold1
XRFDC_UPDATE_THRESHOLD_BOTH	Update for Threshold0 and Threshold1

- **u32 ThresholdMode[2]:** Entry 0 is for Threshold0 and 1 for Threshold1. Range: 0 to 3 (0-OFF, 1-sticky-over, 2-sticky-under and 3-hysteresis)
- **u32 ThresholdAvgVal[2]:** Threshold average value. Entry 0 is for Threshold0 and 1 for Threshold1.
- **u32 ThresholdUnderVal[2]:** Under threshold value. Entry 0 is for Threshold0 and 1 for Threshold1.
- **u32 ThresholdOverVal[2]:** Over threshold value. Entry 0 is for Threshold0 and 1 for Threshold1.

## struct XRFdc\_TileStatus

This structure is for internal driver use.

```
u32 IsEnabled;
u32 TileState;
u8 BlockStatusMask;
u32 PowerUpState;
u32 PLLState;
```

### Description

- **u32 IsEnabled:** Indicates tile is enabled (1) or disabled (0).
- **u32 TileState:** Indicates current tile state.
- **u8 BlockStatusMask:** Bit mask for converter status. 1 indicates converter enable.
- **u32 PowerUpState:** Indicates power-up status.
- **u32 PLLState:** Indicates if PLL is locked or unlocked.

## struct XRFdc\_IPStatus

This structure is used to get the IP core status.

```
XRFdc_TileStatus DACTileStatus[4];
XRFdc_TileStatus ADCTileStatus[4];
u32 State;
```

### Description

- **XRFdc\_TileStatus DACTileStatus[4]:** Tile status for four RF-DAC tiles.
- **XRFdc\_TileStatus ADCTileStatus[4]:** Tile status for four RF-ADC tiles.
- **u32 State:** Not currently supported.

## struct XRFdc\_BlockStatus

This structure is used to get the status of RF-DACs or RF-ADCs.

```
double SamplingFreq;
u32 AnalogDataPathStatus;
u32 DigitalDataPathStatus;
u8 DataPathClocksStatus;
u8 IsFIFOFlagsEnabled;
u8 IsFIFOFlagsAsserted;
```

### Description

- **double SamplingFreq:** Sampling frequency.
- **u32 AnalogDataPathStatus:**
  - RF-ADC
    - bit[0] Converter enable/disable.
  - RF-DAC
    - [3:0] Inverse sinc enable/disable.
    - [7:4] Decoder mode.
- **u32 DigitalDataPathStatus:**
  - RF-ADC
    - [3:0] FIFO status (enable/disable).
    - [7:4] Decimation factor.
    - [11:8] Mixer mode.
  - RF-DAC
    - [3:0] FIFO status.
    - [7:4] Interpolation factor.
    - [11:8] Adder status.
    - [15:12] Mixer mode.

- **u8 DataPathClocksStatus:** Indicates if all required datapath clocks are enabled; 1 if all clocks enabled, 0 otherwise.
- **u8 IsFIFOFlagsEnabled:** FIFO flags enabled mask; 1 is enabled, otherwise 0.
- **u8 IsFIFOFlagsAsserted:** FIFO flags asserted mask; 1 is enabled, otherwise 0.

## struct XRFdc\_DACBlock\_AnalogDataPath\_Config

This structure is for internal driver use.

```
u32 BlockAvailable;
u32 InvSyncEnable;
u32 MixMode;
u32 DecoderMode;
```

### Description

- **u32 BlockAvailable:** Corresponds to the `C_DAC_Slice{xy}_Enable` IP parameter.
- **u32 InvSyncEnable:** Corresponds to the `C_DAC_Invsync_Ctrl{xy}` IP parameter.
- **u32 MixMode:** Corresponds to the `C_DAC_Mixer_Mode{xy}` IP parameter.
- **u32 DecoderMode:** Corresponds to the `C_DAC_Decoder_Mode{xy}` IP parameter.

## struct XRFdc\_DACBlock\_DigitalDataPath\_Config

This structure is for internal driver use.

```
u32 DataType;
u32 DataWidth;
u32 InterpolationMode;
u32 FifoEnable;
u32 AdderEnable;
u32 MixerType;
```

### Description

- **u32 DataType:** Corresponds to the `C_DAC_Data_Type{xy}` IP parameter.
- **u32 DataWidth:** Corresponds to the `C_DAC_Data_Width{xy}` IP parameter.
- **u32 InterpolationMode:** Corresponds to the `C_DAC_Interpolation_Mode{xy}` IP parameter.
- **u32 FifoEnable:** Corresponds to the `C_DAC_Fifo{xy}` IP parameter.
- **u32 AdderEnable:** Corresponds to the `C_DAC_Adder{xy}` IP parameter.
- **u32 MixerType:** Corresponds to the `C_DAC_Mixer_Type{xy}` IP parameter.

## struct XRFdc\_ADCBlock\_AnalogDataPath\_Config

This structure is for internal driver use.

```
u32 BlockAvailable;
u32 MixMode;
```

### Description

- **u32 BlockAvailable:** Corresponds to the C\_ADC\_Slice{xy}\_Enable IP parameter.
- **u32 MixMode:** Corresponds to the C\_ADC\_Mixer\_Mode{xy} IP parameter.

## struct XRFdc\_ADCBlock\_DigitalDataPath\_Config

This structure is for internal driver use.

```
u32 DataType;
u32 DataWidth;
u32 DecimationMode;
u32 FifoEnable;
u32 MixerType;
```

### Description

- **u32 DataType:** Corresponds to the C\_ADC\_Data\_Type{xy} IP parameter.
- **u32 DataWidth:** Corresponds to the C\_ADC\_Data\_Width{xy} IP parameter.
- **u32 DecimationMode:** Corresponds to the C\_ADC\_Decimation\_Mode{xy} IP parameter.
- **u32 FifoEnable:** Corresponds to the C\_ADC\_Fifo{xy} IP parameter.
- **u32 MixerType:** Corresponds to the C\_ADC\_Mixer\_Type{xy} IP parameter.

## struct XRFdc\_DACTile\_Config

This structure is for internal driver use.

```
u32 Enable;
u32 PLLEnable;
double SamplingRate;
double RefClkFreq;
double FabClkFreq;
u32 FeedbackDiv;
u32 OutputDiv;
u32 RefClkDiv;
u32 MultibandConfig;
XRFdc_DACBlock_AnalogDataPath_Config DACBlock_Analog_Config[4];
XRFdc_DACBlock_DigitalDataPath_Config DACBlock_Digital_Config[4];
```

## Description

- **u32 Enable:** Corresponds to the `C_DAC{x}_Enable` IP parameter.
- **u32 PLLEnable:** Corresponds to the `C_DAC{x}_PLL_Enable` IP parameter.
- **double SamplingRate:** Corresponds to the `C_DAC{x}_Sampling_Rate` IP parameter.
- **double RefClkFreq:** Corresponds to the `C_DAC{x}_RefClk_Freq` IP parameter.
- **double FabClkFreq:** Corresponds to the `C_DAC{x}_FabClk_Freq` IP parameter.
- **u32 FeedbackDiv:** Corresponds to the `C_DAC{x}_FBDIV` IP parameter.
- **u32 OutputDiv:** Corresponds to the `C_DAC{x}_OutDiv` IP parameter.
- **u32 RefClkDiv:** Corresponds to the `C_DAC{x}_Refclk_Div` IP parameter.
- **u32 MultibandConfig:** Corresponds to the `C_DAC{x}_Multiband` IP parameter.

## struct XRFdc\_ADCTile\_Config

This structure is for internal driver use.

```

u32 Enable;
u32 PLLEnable;
double SamplingRate
double RefClkFreq;
double FabClkFreq;
u32 FeedbackDiv;
u32 OutputDiv;
u32 RefClkDiv;
u32 MultibandConfig;
XRFdc_ADCBlock_AnalogDataPath_Config ADCBlock_AnalogConfig[4];
XRFdc_ADCBlock_DigitalDataPath_Config ADCBlock_Digital_Config[4];
  
```

## Description

- **u32 Enable:** Corresponds to the `C_ADC{x}_Enable` IP parameter.
- **u32 PLLEnable:** Corresponds to the `C_ADC{x}_PLL_Enable` IP parameter.
- **double SamplingRate:** Corresponds to the `C_ADC{x}_Sampling_Rate` IP parameter.
- **double RefClkFreq:** Corresponds to the `C_ADC{x}_RefClk_Freq` IP parameter.
- **double FabClkFreq:** Corresponds to the `C_ADC{x}_FabClk_Freq` IP parameter.
- **u32 FeedbackDiv:** Corresponds to the `C_ADC{x}_FBDIV` IP parameter.
- **u32 OutputDiv:** Corresponds to the `C_ADC{x}_OutDiv` IP parameter.
- **u32 RefClkDiv:** Corresponds to the `C_ADC{x}_Refclk_Div` IP parameter.

- **u32 MultibandConfig:** Corresponds to the `C_ADC{x}_Multiband` IP parameter.

## struct XRFdc\_DACBlock\_AnalogDataPath

This structure is for internal driver use.

```
u32 Enabled;
u32 MixedMode;
double TerminationVoltage;
double OutputCurrent;
u32 InverseSincFilterEnable;
u32 DecoderMode;
void *FuncHandler;
u32 NyquistZone;
u8 AnalogPathEnabled;
u8 AnalogPathAvailable;
XRFdc_QMC_Settings QMC_Settings;
XRFdc_CoarseDelay_Settings CoarseDelay_Settings;
```

### Description

- **u32 Enabled:** RF-DAC enable (1) or disable (0).
- **u32 MixedMode:** Mixer mode.
- **double TerminationVoltage:** Termination voltage.
- **double OutputCurrent:** Output current.
- **u32 InverseSincFilterEnable:** Inverse sinc filter enable (1) or disable (0).
- **u32 DecoderMode:** Decoder mode.
- **void \*FuncHandler:** Function handler (currently not used in the driver).
- **u32 NyquistZone:** RF-DAC Nyquist Zone.
- **u8 AnalogPathEnabled:** Flag to indicate AnalogPath is enabled(1) or disabled(0).
- **u8 AnalogPathAvailable:** Flag to indicate AnalogPath is available or not.
- **XRFdc\_QMC\_Settings QMC\_Settings:** QMC settings structure.
- **XRFdc\_CoarseDelay\_Settings CoarseDelay\_Settings:** CoarseDelay settings structure.



## struct XRFdc\_DACBlock\_DigitalDataPath

This structure is for internal driver use.

```

u32 DataType;
u32 DataWidth;
int ConnectedIData;
int ConnectedQData;
u32 InterpolationFactor;
u8 DigitalPathEnabled;
u8 DigitalPathAvailable;
XRFdc_Mixer_Settings Mixer_Settings;
  
```

### Description

- **u32 DataType:** Digital input datatype.
- **u32 DataWidth:** Data width (samples per AXI4-Stream word).
- **int ConnectedIData:** Data converter connected for I datapath. Valid values are 0-3 and -1.
- **int ConnectedQData:** Data converter connected for Q datapath. Valid values are 0-3 and -1.
- **u32 InterpolationFactor:** Interpolation factor.
- **u8 DigitalPathEnabled:** Flag to indicate DataPath is enabled(1) or disabled(0).
- **u8 DigitalPathAvailable:** Flag to indicate DigitalPath is available or not.
- **XRFdc\_Mixer\_Settings Mixer\_Settings:** Mixer Settings structure

## struct XRFdc\_ADCBlock\_AnalogDataPath

This structure is for internal driver use.

```

u32 Enabled;
XRFdc_QMC_Settings QMC_Settings;
XRFdc_CoarseDelay_Settings CoarseDelay_Settings;
XRFdc_Threshold_Settings Threshold_Settings;
u32 NyquistZone;
u8 CalibrationMode;
u8 AnalogPathEnabled;
u8 AnalogPathAvailable;
  
```

### Description

- **u32 Enabled:** RF-ADC Enable (1) or Disable (0).
- **XRFdc\_QMC\_Settings QMC\_Settings:** QMC settings structure.
- **XRFdc\_CoarseDelay\_Settings CoarseDelay\_Settings:** Coarse delay settings structure.
- **XRFdc\_Threshold\_Settings Threshold\_Settings:** Threshold settings structure.

- **u32 NyquistZone:** Nyquist zone.
- **u8 CalibrationMode:** Calibration mode, set by the `XRFdc_SetCalibrationMode` API function.
- **u8 AnalogPathEnabled:** Flag to indicate AnalogPath is enabled(1) or disabled(0).
- **u8 AnalogPathAvailable:** Flag to indicate AnalogPath is available or not.

## struct XRFdc\_ADCBlock\_DigitalDataPath

This structure is for internal driver use.

```
u32 DataType;
u32 DataWidth;
u32 DecimationFactor;
int ConnectedIData;
int ConnectedQData;
u8 DigitalPathEnabled;
u8 DigitalPathAvailable;
XRFdc_Mixer_Settings Mixer_Settings;
```

### Description

- **u32 DataType:** Analog input data type.
- **u32 DataWidth:** Data width (samples per AXI4-Stream word).
- **u32 DecimationFactor:** Decimation factor.
- **int ConnectedIData:** Data converter connected for I datapath. Valid values are 0-3 and -1.
- **int ConnectedQData:** Data converter connected for Q datapath. Valid values are 0-3 and -1.
- **u8 DigitalPathEnabled:** Flag to indicate DataPath is enabled(1) or disabled(0).
- **u8 DigitalPathAvailable:** Flag to indicate DigitalPath is available or not.
- **XRFdc\_Mixer\_Settings Mixer\_Settings:** Mixer settings structure.

## struct XRFdc\_DAC\_Tile

This structure is for internal driver use.

```
u32 TileBaseAddr;
u32 NumOfDACBlocks;
XRFdc_PLL_Settings PLL_Settings;
u8 MultibandConfig;
XRFdc_DACBlock_AnalogDataPath DACBlock_Analog_Datapath[4];
XRFdc_DACBlock_DigitalDataPath DACBlock_Digital_Datapath[4];
```

## Description

- **u32 TileBaseAddr:** Tile base address.
- **u32 NumOfDACBlocks:** Number of RF-DACs enabled.
- **XRFdc\_PLL\_Settings PLL\_Settings:** PLL settings structure.
- **u8 MultibandConfig:** Multi-band configuration for RF-DAC tile.
- **XRFdc\_DACBlock\_AnalogDataPath DACBlock\_Analog\_Datapath[4]:**  
DACBlock\_AnalogDataPath structure for four RF-DACs.
- **XRFdc\_DACBlock\_DigitalDataPath DACBlock\_Digital\_Datapath[4]:**  
DACBlock\_DigitalDataPath structure for four RF-DACs.

## struct XRFdc\_ADC\_Tile

This structure is for internal driver use.

```
u32 TileBaseAddr;
u32 NumOfADCBlocks;
XRFdc_PLL_Settings PLL_Settings;
u8 MultibandConfig;
XRFdc_ADCBlock_AnalogDataPath ADCBlock_Analog_Datapath[4];
XRFdc_ADCBlock_DigitalDataPath ADCBlock_Digital_Datapath[4];
```

## Description

- **u32 TileBaseAddr:** Tile base address.
- **u32 NumOfADCBlocks:** Number of RF-ADCs enabled.
- **XRFdc\_PLL\_Settings PLL\_Settings:** PLL settings structure.
- **u8 MultibandConfig:** Multiband configuration for ADC Tile.
- **XRFdc\_ADCBlock\_AnalogDataPath ADCBlock\_Analog\_Datapath[4]:**  
ADCBlock\_AnalogDataPath structure for four RF-ADCs.
- **XRFdc\_ADCBlock\_DigitalDataPath ADCBlock\_Digital\_Datapath[4]:**  
ADCBlock\_DigitalDataPath structure for four RF-ADCs.

## struct XRFdc\_MultiConverter\_Sync\_Config

This structure is used to configure the MTS algorithm. Indicated files can be configured in the code.

```

u32 RefTile;
u32 Tiles;
int Target_Latency;
int Offset[4];
int Latency[4];
int Marker_Delay;
int SysRef_Enable;
XRFdc_MTS_DTC_Settings DTC_Set_PLL;
XRFdc_MTS_DTC_Settings DTC_Set_T1;
  
```

### Description

- **u32 RefTile:** Reference tile (must be 0).
- **u32 Tiles:** Bitmask indicating which tiles to align. BitX enables MTS for TileX. Tile0 must always be enabled.
- **int Target\_Latency:** Sets the target relative latency. This is required to be set for multi-device alignment, or deterministic latency use-cases. It is not required to be set for single-device alignment.
- **int Offset[4]:** Status – indicates the value the interface data was delayed by to achieve alignment, per tile.
- **int Latency[4]:** Status – indicates the measured relative-latency value of each tile.
- **int Marker\_Delay:** Marker delay is for internal use.
- **int SysRef\_Enable:** Set to 1 (default) to keep SYSREF capture enabled after MTS runs. Set to 0 to disable SYSREF capture.

## struct XRFdc\_MTS\_Marker

This structure is for internal driver use. There are no user-configurable fields.

```

u32 Count[4];
u32 Loc[4];
  
```

## struct XRFdc\_MTS\_DTC\_Settings

This structure is for internal driver use. There are no user-configurable fields.

```
u32 RefTile;
u32 IsPLL;
int Target[4];
int Scan_Mode;
int DTC_Code[4];
int Num_Windows[4];
int Max_Gap[4];
int Min_Gap[4];
int Max_Overlap[4];
```

## struct XRFdc\_Calibration\_Coefficients

Generic calibration coefficient structure.

```
u32 Coeff0;
u32 Coeff1;
u32 Coeff2;
u32 Coeff3;
u32 Coeff4;
u32 Coeff5;
u32 Coeff6;
u32 Coeff7;
```

### Description

- **u32 Coeff{N}**: Nth Coefficient

**Note:** Coeff{4-7} applies when chopping is active and is only relevant to the time skew calibration block.

## struct XRFdc\_Cal\_Freeze\_Settings

Generic calibration freezing settings structure

```
u32 CalFrozen;
u32 DisableFreezePin;
u32 FreezeCalibration;
```

### Description

- **u32 CalFrozen**: Status that indicates that the calibration has been frozen
- **u32 DisableFreezePin**: Disables the calibration freeze pin
- **u32 FreezeCalibration**: Freezes the calibration using the freeze port

## User API Functions

All user API functions are implemented in the source file `xrfdc.c`. The prototypes for these are provided in header file `xrfdc.h`.

All driver API functions have `InstancePtr` as an argument. It is a pointer to the `XRFdc` structure that contains information about the base address, the structure pointing to `RFdc` default configurations, and structures for RF-ADC and RF-DAC tile configurations. The `InstancePtr` argument is initialized in the `XRFdc_CfgInitialize` API function. Call the `XRFdc_CfgInitialize` API function to initialize `InstancePtr` before calling any other functions.

### XRFdc\_CfgInitialize

#### Function Prototype

```
u32 XRFdc_CfgInitialize(XRFdc *InstancePtr, XRFdc_Config *ConfigPtr);
```

#### Arguments

- `XRFdc *InstancePtr`: Pointer to the driver instance.
- `XRFdc_Config *ConfigPtr`: Pointer to the configuration structure.

#### Description

This API function populates appropriate entries in the driver instance by copying relevant entries from the configuration structure. This function is required for any software interaction with the `RFdc` driver API and must be called first before using any other API functions.

#### Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE`

### XRFdc\_LookupConfig

#### Function Prototype

```
XRFdc_Config *XRFdc_LookupConfig(u16 DeviceId);
```

### Arguments

- **u16 DeviceId:** ID of the device whose configuration information is required.

### Description

This API function looks up the device configuration based on the unique ID of the device. A table contains the configuration information for each device in the system.

### Return Value

The function returns a pointer to the configuration found, or `NULL` if the specified device ID was not found.

## XRFdc\_RegisterMetal

### Function Prototype

```
u32 XRFdc_RegisterMetal(XRFdc *InstancePtr, u16 DeviceId, struct
metal_device **DevicePtr);
```

### Arguments

- **XRFdc \* InstancePtr:** Pointer to the driver instance.
- **u16 Device\_Id:** The device id for the driver.
- **struct metal\_device \*\*DevicePtr:** Pointer to the metal device.

### Description

This API function registers RFDC with Libmetal

### Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE`

## XRFdc\_StartUp

### Function Prototype

```
u32 XRFdc_StartUp(XRFdc *InstancePtr, u32 Type, int Tile_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **int Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3 and -1 (for all tiles).

### Description

This API function restarts the tile as requested through `Tile_Id`. If -1 is passed as `Tile_Id`, the function restarts all the enabled tiles. Existing register settings are not lost or altered in the process.

### Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE` (returned when the tile is not enabled or available)

## XRFdc\_Shutdown

### Function Prototype

```
u32 XRFdc_Shutdown(XRFdc *InstancePtr, u32 Type, int Tile_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **int Tile\_Id:** Valid values are 0-3 and -1 (for all tiles).

### Description

This API function stops the tile as requested through `Tile_Id`. If -1 is passed as `Tile_Id`, the function stops all the enabled tiles. The existing register settings are not cleared.

**Note:** This is a common API function for RF-ADC and RF-DAC tiles.

### Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE` (returned when the tile is not enabled or available)



## XRFdc\_Reset

### Function Prototype

```
u32 XRFdc_Reset(XRFdc *InstancePtr, u32 Type, int Tile_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **int Tile\_Id:** Valid values are 0-3 and -1 (for all tiles).

### Description

This API function resets the tile as requested through `Tile_Id`. If `-1` is passed as `Tile_Id`, it resets all the enabled tiles. All existing register settings are cleared and are replaced with the settings initially configured.

**Note:** This is a common API function for RF-ADC and RF-DAC tiles.

### Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE` (returned when the tile is not enabled or available)

## XRFdc\_GetIPStatus

### Function Prototype

```
u32 XRFdc_GetIPStatus(XRFdc *InstancePtr, XRFdc_IPStatus *IPStatusPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **XRFdc\_IPStatus \*IPStatusPtr:** Pointer to the `XRFdc_IPStatus` structure through which the status is returned.

### Description

This API function returns the IP status.

## Return Value

XRFDC\_SUCCESS

# XRFdc\_GetBlockStatus

## Function Prototype

```
u32 XRFdc_GetBlockStatus(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id, XRFdc_BlockStatus *BlockStatusPtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC block number inside the tile. Valid values are 0-3.
- **XRFdc\_BlockStatus \*BlockStatusPtr:** Pointer to the XRFdc\_BlockStatus structure through which the RF-ADC/RF-DAC block status is returned.

## Description

This API function returns the requested block status.

**Note:** This is a common API function for RF-ADCs/RF-DACs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE (Converter not enabled)

# XRFdc\_SetMixerSettings

## Function Prototype

```
u32 XRFdc_SetMixerSettings(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id, XRFdc_Mixer_Settings *MixerSettingsPtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.

- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **XRFdc\_Mixer\_Settings \*MixerSettingsPtr:** Pointer to the `XRFdc_Mixer_Settings` structure that passes the mixer/NCO settings.

### Description

Mixer/NCO settings passed are used to update the corresponding block level registers. The driver structure is updated with the new values.

**Note:** This is a common API function for RF-ADCs/RF-DACs.

### Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE`

## XRFdc\_GetMixerSettings

### Function Prototype

```
u32 XRFdc_GetMixerSettings(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, XRFdc_Mixer_Settings *MixerSettingsPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **XRFdc\_Mixer\_Settings \*MixerSettingsPtr:** Pointer to the `XRFdc_Mixer_Settings` structure in which the existing mixer/NCO settings are returned.

### Description

Mixer/NCO settings from corresponding registers are returned to the caller.

**Note:** This is a common API function for RF-ADCs/RF-DACs.

### Return Value

`XRFDC_SUCCESS`

XRFDC\_FAILURE

## XRFdc\_SetQMCSettings

### Function Prototype

```
u32 XRFdc_SetQMCSettings(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, XRFdc_QMC_Settings *QMCSettingsPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **XRFdc\_QMC\_Settings \*QMCSettingsPtr:** Pointer to the XRFdc\_QMC\_Settings structure used to update the corresponding registers.

### Description

QMC settings from the passed structure are populated into the corresponding registers. The driver structure is updated with the new values.

**Note:** This is a common API function for RF-ADCs/RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetQMCSettings

### Function Prototype

```
u32 XRFdc_GetQMCSettings(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, XRFdc_QMC_Settings *QMCSettingsPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.

- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **XRFdc\_QMC\_Settings \*QMCSettingsPtr:** Pointer to the `XRFdc_QMC_Settings` structure that returns the corresponding register values.

### Description

QMC settings from the relevant registers are returned back to the caller.

**Note:** This is a common API function for RF-ADCs/RF-DACs.

### Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE`

## XRFdc\_GetCoarseDelaySettings

### Function Prototype

```
u32 XRFdc_GetCoarseDelaySettings(XRFdc *InstancePtr, u32 Type, u32 Tile_Id,
u32 Block_Id, XRFdc_CoarseDelay_Settings *CoarseDelaySettingsPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **XRFdc\_CoarseDelay\_Settings \*CoarseDelaySettingsPtr:** Pointer to the `XRFdc_CoarseDelay_Settings` structure which returns the corresponding register values.

### Description

Coarse delay settings from the relevant registers are returned back to the caller.

**Note:** This is a common API function for RF-ADCs/RF-DACs.

### Return Value

`XRFDC_SUCCESS`

XRFDC\_FAILURE

## XRFdc\_SetCoarseDelaySettings

### Function Prototype

```
u32 XRFdc_SetCoarseDelaySettings(XRFdc *InstancePtr, u32 Type, u32 Tile_Id,
u32 Block_Id, XRFdc_CoarseDelay_Settings *CoarseDelaySettingsPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **XRFdc\_CoarseDelay\_Settings \*CoarseDelaySettingsPtr:** Pointer to the `XRFdc_CoarseDelay_Settings` structure through which the coarse delay settings are passed.

### Description

Coarse delay settings passed from the caller are updated in the relevant registers. The driver structure is updated with the new values.

**Note:** This is a common API function for RF-ADCs/RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_UpdateEvent

### Function Prototype

```
u32 XRFdc_UpdateEvent(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, u32 Event);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.

- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **u32 Event:** Event can be mixer, QMC, or coarse delay.

**Table 74: Valid Macros for Event Argument**

Macro	Description
XRFDC_EVENT_MIXER	Mixer event
XRFDC_EVENT_CRSE_DLY	Coarse delay event
XRFDC_EVENT_QMC	QMC event

### Description

Use this function to trigger the update event for an event if the event source is Slice or Tile.

**Note:** This is a common API function for RF-ADCs/RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetInterpolationFactor

### Function Prototype

```
u32 XRFdc_SetInterpolationFactor(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u32 InterpolationFactor);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number. Valid values are 0-3.
- **u32 InterpolationFactor:** Interpolation factor to be set for the RF-DAC.

**Table 75: Valid Macros for InterpolationFactor Argument**

Macro	Description
XRFDC_INTERP_DECIM_OFF	Interpolation is OFF
XRFDC_INTERP_DECIM_1X	1X interpolation factor
XRFDC_INTERP_DECIM_2X	2X interpolation factor

Table 75: Valid Macros for InterpolationFactor Argument (cont'd)

Macro	Description
XRFDC_INTERP_DECIM_4X	4X interpolation factor
XRFDC_INTERP_DECIM_8X	8X interpolation factor

### Description

This API function sets the interpolation factor for the requested RF-DAC and also updates the FIFO read width based on the interpolation factor.

Dynamic change in interpolation has an impact on the block throughput. The AXI4-Stream clock rate can be dynamically changed to account for this change of throughput. In non-MTS mode, the recommended procedure is to turn off the FIFO (`Xrfdc_setupfifo`), change the clock rate (`Xrfdc_SetfabClkOutDiv`), clear the FIFO interrupt, and restart the FIFO (`Xrfdc_SetupFifo`).

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetInterpolationFactor

### Function Prototype

```
u32 XRFdc_GetInterpolationFactor(XRFdc *InstancePtr, u32 Tile_Id, u32
Block_Id, u32 *InterpolationFactorPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number inside the tile. Valid values are 0-3.
- **u32 \*InterpolationFactorPtr:** Pointer to return the interpolation factor for the RF-DAC.

### Description

The interpolation factor for the requested RF-DAC is returned back to the caller.

**Note:** This API function is only applicable for RF-DACs.



## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_SetDecimationFactor

## Function Prototype

```
u32 XRFdc_SetDecimationFactor(XRFdc *InstancePtr, u32 Tile_Id, u32
Block_Id, u32 DecimationFactor);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC number. Valid values are 0-3.
- **u32 DecimationFactor:** Decimation factor to be set for the RF-ADC.

*Table 76: Valid Macros for DecimationFactor Argument*

Macro	Description
XRFDC_INTERP_DECIM_OFF	Decimation is OFF
XRFDC_INTERP_DECIM_1X	1X Decimation factor
XRFDC_INTERP_DECIM_2X	2X Decimation factor
XRFDC_INTERP_DECIM_4X	4X Decimation factor
XRFDC_INTERP_DECIM_8X	8X Decimation factor

## Description

This API function sets the decimation factor for the requested RF-ADC and also updates the FIFO write width based on the decimation factor.

Dynamic change in decimation has an impact on the block throughput. The AXI4-Stream clock rate can be dynamically changed to account for this change of throughput. In non-MTS mode, the recommended procedure is to turn off the FIFO (`Xrfdc_setupfifo`), change the clock rate (`Xrfdc_SetfabClkOutDiv`), clear the FIFO interrupt, and restart the FIFO (`Xrfdc_SetupFifo`).

**Note:** This function is only applicable for RF-ADCs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_GetDecimationFactor

## Function Prototype

```
u32 XRFdc_GetDecimationFactor(XRFdc *InstancePtr, u32 Tile_Id, u32
Block_Id, u32 *DecimationFactorPtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC number inside the tile. Valid values are 0-3.
- **u32 \*DecimationFactorPtr:** Pointer to return the decimation factor for the RF-ADC.

## Description

Decimation factor for the requested RF-ADC is returned back to the caller.

**Note:** This API function is only applicable for RF-ADCs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_SetFabClkOutDiv

## Function Prototype

```
u32 XRFdc_SetFabClkOutDiv(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u16
FabClkDiv);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.

- **u16 FabClkDiv:** PL clock divider to be set for a tile.

Table 77: Valid Macros for FabClkDiv Argument

Macro	Description
XRFDC_FAB_CLK_DIV1	Divided by 1
XRFDC_FAB_CLK_DIV2	Divided by 2
XRFDC_FAB_CLK_DIV4	Divided by 4
XRFDC_FAB_CLK_DIV8	Divided by 8
XRFDC_FAB_CLK_DIV16	Divided by 16

### Description

Use this function to set the divider for PL clock out.

**Note:** This is a common function for RF-ADCs and RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetFabClkOutDiv

### Function Prototype

```
u32 XRFdc_GetFabClkOutDiv(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u16 *FabClkDivPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u16 \*FabClkDivPtr:** Pointer to get the PL clock divider for a tile.

### Description

Use this function to get the clock divider for the PL.

**Note:** This is a common function for the RF-ADC and the RF-DAC.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetFabWrVldWords

### Function Prototype

```
u32 XRFdc_SetFabWrVldWords(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,
u32 FabricWrVldWords);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number inside the tile. Valid values are 0-3.
- **u32 FabricWrVldWords:** Write fabric data rate to be set for RF-DAC.

### Description

This API function sets the write fabric data rate for the requested RF-DAC by writing to the corresponding register.

**Note:** This API function is only applicable for RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetFabWrVldWords

### Function Prototype

```
u32 XRFdc_GetFabWrVldWords(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, u32 *FabricWrVldWordsPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.

- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **u32 \*FabricWrVldWordsPtr:** Pointer to return the write PL data rate for the RF-ADCs/RF-DACs.

### Description

Write PL data rate for the requested RF-ADC/RF-DAC is returned back to the caller.

**Note:** This is a common API function for RF-ADCs and RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetFabRdVldWords

### Function Prototype

```
u32 XRFdc_SetFabRdVldWords(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,
u32 FabricRdVldWords);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC number inside the tile. Valid values are 0-3.
- **u32 FabricRdVldWords:** Read PL data rate to be set for RF-ADC.

### Description

This API function sets the read PL data rate for the requested RF-ADC by writing to the corresponding register.

**Note:** This API function is only applicable for RF-ADCs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetFabRdVldWords

### Function Prototype

```
u32 XRFdc_GetFabRdVldWords(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, u32 *FabricRdVldWordsPtr);
```

### Arguments

- **XRFdc\* InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.
- **u32 \*FabricRdVldWordsPtr:** Pointer to return the read PL data rate for the RF-ADC/RF-DACs.

### Description

Read PL data rate for the requested RF-ADC/RF-DAC is returned back to the caller.

**Note:** This is a common API function for RF-ADCs and RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_ThresholdStickyClear

### Function Prototype

```
u32 XRFdc_ThresholdStickyClear(XRFdc *InstancePtr, u32 Tile_Id, u32
Block_Id, u32 ThresholdToUpdate);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC number inside the tile. Valid values are 0-3.
- **u32 ThresholdToUpdate:** Select which Threshold (Threshold0 or Threshold1 or both) to update.

**Note:** See `Threshold_Settings` structure for valid macros.

### Description

This API function clears the sticky bit in threshold configuration registers based on the `ThresholdToUpdate` parameter.

**Note:** This API function is applicable only for RF-ADCs.

### Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE`

### Related Information

[struct XRFdc\\_Threshold\\_Settings](#)

## XRFdc\_SetThresholdClrMode

### Function Prototype

```
u32 XRFdc_SetThresholdClrMode(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u32 ThresholdToUpdate, u32 ClrMode);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC number inside the tile. Valid values are 0-3.
- **u32 ThresholdToUpdate:** Select which Threshold (`Threshold0` or `Threshold1` or both) to update.

**Note:** See `Threshold_Settings` structure for valid macros.

- **u32 ClrMode:** Clear mode can be manual (register write) or auto clear (QMC gain update event).

**Table 78: Valid Macros for ClrMode Argument**

Macro	Description
<code>XRFDC_THRESHOLD_CLRMD_MANUAL_CLR</code>	Manual clear
<code>XRFDC_THRESHOLD_CLRMD_AUTO_CLR</code>	Auto clear

## Description

This API function sets the threshold clear mode.

**Note:** This API function is only applicable for RF-ADCs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## Related Information

[struct XRFdc\\_Threshold\\_Settings](#)

# XRFdc\_GetThresholdSettings

## Function Prototype

```
u32 XRFdc_GetThresholdSettings(XRFdc *InstancePtr, u32 Tile_Id, u32
Block_Id, XRFdc_Threshold_Settings *ThresholdSettingsPtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC number inside the tile. Valid values are 0-3.
- **XRFdc\_Threshold\_Settings \*ThresholdSettingsPtr:** Pointer through which the register settings for thresholds are passed back.

## Description

This API function reads threshold settings from the corresponding registers.

**Note:** This API function is only applicable for RF-ADCs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE



## XRFdc\_SetThresholdSettings

### Function Prototype

```
u32 XRFdc_SetThresholdSettings(XRFdc *InstancePtr, u32 Tile_Id, u32
Block_Id, XRFdc_Threshold_Settings *Threshold_Settings);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC number inside the tile. Valid values are 0-3.
- **XRFdc\_Threshold\_Settings \*Threshold\_Settings:** Pointer through which the register settings for thresholds are passed to the API.

### Description

This API function writes the threshold settings to the relevant registers. The driver structure is updated with the new values.

**Note:** This API function is only applicable for RF-ADCs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetDecoderMode

### Function Prototype

```
u32 XRFdc_SetDecoderMode(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u32
DecoderMode);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number inside the tile. Valid values are 0-3.
- **u32 DecoderMode:** Decoder mode settings. Valid values are:
  - Maximum linearity, for randomized decoder

- Maximum SNR, for non-randomized decoder

Table 79: Valid Macros for DecoderMode Arguments

Macro	Description
XRFDC_DECODER_MAX_SNR_MODE	Maximum SNR mode
XRFDC_DECODER_MAX_LINEARITY_MODE	Maximum Linearity mode

### Description

This API function writes the decoder mode to the relevant registers. The driver structure is updated with the new values.

**Note:** This API function is used only for the RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetDecoderMode

### Function Prototype

```
u32 XRFdc_GetDecoderMode(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u32 *DecoderModePtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number inside the tile. Valid values are 0-3.
- **u32 \*DecoderModePtr:** Pointer in which decoder mode settings to be returned back to the caller. Valid values are:
  - Maximum linearity, for randomized decoder
  - Maximum SNR, for non-randomized decoder

### Description

This API function reads the decoder mode from the relevant registers.

**Note:** This API function is used only for the RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_ResetNCOPhase

### Function Prototype

```
u32 XRFdc_ResetNCOPhase(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number inside the tile. Valid values are 0-3.

### Description

This API function resets the NCO phase of the current block phase accumulator.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetupFIFO

### Function Prototype

```
u32 XRFdc_SetupFIFO(XRFdc *InstancePtr, u32 Type, int Tile_Id, u8 Enable);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **int Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3 and -1 (for all tiles).
- **u8 Enable:** Valid values are 1 (FIFO enable) and 0 (FIFO Disable).

## Description

This API function enables and disables the RF-ADC/RF-DAC FIFO.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_GetOutputCurr

## Function Prototype

```
u32 XRFdc_GetOutputCurr(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u32 *OutputCurrPtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number inside the tile. Valid values are 0-3.
- **u32 \*OutputCurrPtr:** OutputCurr pointer to return the output current.

## Description

This API function gets the output current.

**Note:** This API function is used only for the RF-DACs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_GetFIFOStatus

## Function Prototype

```
u32 XRFdc_GetFIFOStatus(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u8 *EnablePtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u8 \*EnablePtr:** Valid values are 1 (FIFO enable) and 0 (FIFO Disable).

### Description

This API function gets the current status of the RF-ADC/RF-DAC FIFO.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_DumpRegs

### Function Prototype

```
void XRFdc_DumpRegs (XRFdc *InstancePtr, u32 Type, int Tile_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **int Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3 and -1.

### Description

This function is intended for debug purposes. It prints the contents of registers to the console for the passed `Tile_Id`. If -1 is passed, it prints the contents of the registers for all the tiles for the respective RF-ADC or RF-DAC. It prints the offset of the register as well as the content.

### Return Value

None.

## XRFdc\_SetNyquistZone

### Function Prototype

```
u32 XRFdc_SetNyquistZone(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id, u32 NyquistZone);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number inside the tile. Valid values are 0-3.
- **u32 NyquistZone:** Valid values are 1 (Odd), 2 (Even).

### Description

This API function sets the Nyquist zone for the RF-ADC/RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetNyquistZone

### Function Prototype

```
u32 XRFdc_GetNyquistZone(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id, u32 *NyquistZonePtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number inside the tile. Valid values are 0-3.
- **u32 \*NyquistZonePtr:** Pointer to return Nyquist Zone.

## Description

This API function gets the Nyquist zone for the RF-ADCs/RF-DACs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_SetInvSincFIR

## Function Prototype

```
u32 XRFdc_SetInvSincFIR(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u16 Mode);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC number. Valid values are 0-3.
- **u16 Mode:** Valid values are 0 (disable) and 1 (First Nyquist Zone).

## Description

This API function is used to enable or disable the inverse sinc filter.

**Note:** This API function is used only for the RF-DACs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_GetInvSincFIR

## Function Prototype

```
u32 XRFdc_GetInvSincFIR(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u16 *ModePtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u16 \*ModePtr:** Enable pointer is used to get the status. Valid values are 0 (disable) and 1 (first Nyquist zone).

### Description

This API function is used to get the inverse sinc filter status.

**Note:** This API function is used only for the RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetCalibrationMode

### Function Prototype

```
u32 XRFdc_SetCalibrationMode(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,
u8 CalibrationMode);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC number. Valid values are 0-3.
- **u8 CalibrationMode:** Valid values are 1 (Mode1) and 2 (Mode2).

### Description

This API function sets the calibration mode for the RF-ADC. After calling this API, you have to restart the entire tile for the new calibration mode to operate correctly.

**Note:** This API function is used only for the RF-ADCs. After changing the calibration mode of a slice, the commands `Xrfdc_Shutdown` and `Xrfdc_startup` must be issued.



## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_GetCalibrationMode

## Function Prototype

```
u32 XRFdc_GetCalibrationMode(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,
u8 *CalibrationModePtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC block number. Valid values are 0-3.
- **u8 \*CalibrationModePtr:** Pointer to get the calibration mode.

## Description

This API function sets the calibration mode for the RF-ADCs.

**Note:** This API function is used only for RF-ADCs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_DisableCoefficientsOverride

## Function Prototype

```
u32 XRFdc_DisableCoefficientsOverride(XRFdc *InstancePtr, u32 Tile_Id, u32
Block_Id, u32 CalibrationBlock);
```

## Arguments

- **XRFdc \* InstancePtr:** Pointer to the driver instance
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3
- **u32 Block\_Id:** RF-ADC block number inside the tile. Valid values are 0-3

- **u32 CalibrationBlock:** The calibration block whose override is to be disabled. Valid values are 1-3 representing the OCB2, GCB, and TSCB respectively.

*Table 80: Valid Macros for CalibrationBlock Argument*

Macro	Description
XRFDC_CAL_BLOCK_OCB2	Offset Calibration Block (Foreground)
XRFDC_CAL_BLOCK_GCB	Gain Calibration Block (Background)
XRFDC_CAL_BLOCK_TSCB	Time Skew Calibration Block (Background)

## Description

This API function disables the coefficient override for the selected block.

The clock is selected using the CalibrationBlock parameter.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_SetCalCoefficients

## Function Prototype

```
u32 XRFdc_SetCalCoefficients(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,
u32 CalibrationBlock, XRFdc_Calibration_Coefficients *CoeffPtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC block number inside the tile. Valid values are 0-3.
- **u32 CalibrationBlock:** The calibration block whose override is to be disabled. Valid values are 1-3 representing the OCB2, GCB, and TSCB respectively.
- **XRFdc\_Calibration\_Coefficients \*CoeffPtr:** Pointer to generic calibration coefficients structure

*Table 81: Valid Macros for CalibrationBlock Argument*

Macro	Description
XRFDC_CAL_BLOCK_OCB2	Offset Calibration Block (Foreground)
XRFDC_CAL_BLOCK_GCB	Gain Calibration Block (Background)

Table 81: Valid Macros for CalibrationBlock Argument (cont'd)

Macro	Description
XRFDC_CAL_BLOCK_TSCB	Time Skew Calibration Block (Background)

### Description

This API function enables the coefficient override and programs the provided coefficients for the selected block.

The block is selected using the CalibrationBlock parameter.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetCalCoefficients

```
u32 XRFdc_GetCalCoefficients(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,
u32 CalibrationBlock, XRFdc_Calibration_Coefficients *CoeffPtr);
```

### Arguments

- **XRFdc \* InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC block number inside the tile. Valid values are 0-3.
- **u32 CalibrationBlock:** The calibration block whose override is to be disabled. Valid values are 1-3 representing the OCB2, GCB, and TSCB respectively.
- **XRFdc\_Calibration\_Coefficients \*CoeffPtr:** Pointer to generic calibration coefficients structure

Table 82: Valid Macros for Calibration Block Argument

Macro	Description
XRFDC_CAL_BLOCK_OCB2	Offset Calibration Block (Foreground)
XRFDC_CAL_BLOCK_GCB	Gain Calibration Block (Background)
XRFDC_CAL_BLOCK_TSCB	Time Skew Calibration Block (Background)

### Description

This API function populates the provided coefficients structure for the selected block.

The block is selected using the Calibration Block parameter.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetCalFreeze

### Function Prototype

```
u32 XRFdc_SetCalFreeze(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,  
XRFdc_Cal_Freeze_Settings *CalFreezePtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC block number inside the tile. Valid values are 0-3.
- **XRFdc\_Cal\_Freeze\_Settings \*CalFreezePtr:** Pointer to generic calibration freeze settings structure

### Description

This API function freezes/unfreezes the calibration via the calibration port and disables/enables the calibration freeze pin.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetCalFreeze

### Function Prototype

```
u32 XRFdc_GetCalFreeze(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,  
XRFdc_Cal_Freeze_Settings *CalFreezePtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.

- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC block number inside the tile. Valid values are 0-3.
- **XRFdc\_Cal\_Freeze\_Settings \*CalFreezePtr:** Pointer to generic calibration freeze settings structure

### Description

This API function populates the provided settings structure with the calibration freeze settings/status.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetDither

### Function Prototype

```
u32 XRFdc_SetDither(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u32 Mode);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC block number inside the tile. Valid values are 0-3.
- **u32 Mode:** Disable (0) or enable (1) dither

### Description

This API function enables/disables the dither.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetDither

### Function Prototype

```
u32 XRFdc_GetDither(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id, u32 *ModePtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC block number inside the tile. Valid values are 0-3.
- **u32 \*ModePtr:** Pointer to be populated with mode

### Description

This API function populates the pointer provided with the dither mode.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_MultiBand

### Function Prototype

```
u32 XRFdc_MultiBand(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u8 DigitalDataPathMask, u32 DataType, u32 DataConverterMask);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u8 DigitalDataPathMask:** Represents the datapath mask for multi-band. The first four bits represent four datapaths; 1 is enabled and 0 is disabled.
- **u32 DataType:** DataType is the mixer data type; valid values are XRFDC\_MB\_DATATYPE\_\*

Table 83: Valid Macros for DataType Argument

Macro	Description
XRFDC_MB_DATATYPE_C2C	DataType is Complex to Complex
XRFDC_MB_DATATYPE_R2C	DataType is Real to Complex
XRFDC_MB_DATATYPE_C2R	DataType is Complex to Real

- **u32 DataConverterMask:** DataConverterMask is a block-enabled mask (input/output driving blocks); 1 means enabled and 0 means disabled.

### Description

This API function is used to set up single-band and multi-band configuration.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_MultiConverter\_Init

### Function Prototype

```
void XRFdc_MultiConverter_Init(XRFdc_MultiConverter_Sync_Config *ConfigPtr,
int *PLL_CodesPtr, int *T1_CodesPtr);
```

### Arguments

- **XRFdc\_MultiConverter\_Sync\_Config \*ConfigPtr:** Multi-tile synchronization configuration structure.
- **int \*PLL\_CodesPtr:** Optional target codes for PLL analog SYSREF capture. Set to 0 (NULL) when not used.
- **int \*T1\_CodesPtr:** Optional target codes for T1 analog SYSREF capture. Set to 0 (NULL) when not used.

### Description

This API function initializes the multi-tile synchronization configuration structures; it must be called before `XRFdc_MultiConverter_Sync`. Optionally, it allows previously determined target codes to be provided for the PLL/T1 analog SYSREF capture.

### Return Value

None.

## XRFdc\_MultiConverter\_Sync

### Function Prototype

```
u32 XRFdc_MultiConverter_Sync(XRFdc *InstancePtr, u32 Type,
XRFdc_MultiConverter_Sync_Config *ConfigPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **XRFdc\_MultiConverter\_Sync\_Config \*ConfigPtr:** Multi-tile sync config structure.

### Description

This is the top-level API function used for multi-tile synchronization.

### Return Value

XRFDC\_MTS\_OK

XRFDC\_MTS\_TIMEOUT

XRFDC\_MTS\_MARKER\_RUN

XRFDC\_MTS\_MARKER\_MISM

XRFDC\_MTS\_NOT\_SUPPORTED

## XRFdc\_DynamicPLLConfig

### Function Prototype

```
u32 XRFdc_DynamicPLLConfig(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u8
Source, double RefClkFreq, double SamplingRate);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u8 Source:** Internal PLL or external clock source.



Table 84: Valid Macros for Source Argument

Macro	Description
XRFDC_EXTERNAL_CLK	For external clock
XRFDC_INTERNAL_PLL_CLK	For internal PLL

- **double RefClkFreq:** Reference clock frequency in MHz ( $F_{REF}$  min to  $F_{REF}$  max).
- **double SamplingRate:** Sampling rate frequency in MHz ( $F_s$  min to  $F_s$  max).



**IMPORTANT!** For the  $F_{REF}$  and  $F_s$  specifications see the Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics ([DS926](#)).

## Description

This API function is used for dynamic switching between the internal PLL and the external clock sources, and for configuring the internal PLL for the RF-ADCs/RF-DACs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# XRFdc\_GetClockSource

## Function Prototype

```
u32 XRFdc_GetClockSource(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 *ClockSourcePtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 \*ClockSourcePtr:** Pointer to return the clock source.

## Description

This API function gets the clock source for the RF-ADCs/RF-DACs.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetPLLConfig

### Function Prototype

```
u32 XRFdc_GetPLLConfig(XRFdc *InstancePtr, u32 Type, u32 Tile_Id,
XRFdc_PLL_Settings *PLLSettings);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** Valid values are 0-3 and -1 (for all tiles).
- **XRFdc\_PLL\_Settings \*PLLSettings:** Pointer to the `XRFdc_PLL_Settings` to be populated with the PLL settings.

### Description

This API function reads the PLL settings from the registers and populates the `XRFdc_PLL_Settings` structure.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetPLLLockStatus

### Function Prototype

```
u32 XRFdc_GetPLLLockStatus(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
*LockStatusPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 \*LockStatusPtr:** Pointer to return the PLL lock status.

### Description

This API function gets the PLL lock status for the RF-ADCs/RF-DACs.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetLinkCoupling

### Function Prototype

```
u32 XRFdc_GetLinkCoupling(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id,
u32 *ModePtr)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC block number. Valid values are 0-3.
- **u32 \*ModePtr:** Pointer to get the Link coupling mode.

### Description

This API function gets the Link Coupling mode for the RF-ADC block.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_MTS\_Sysref\_Config

### Function Prototype

```
u32 XRFdc_MTS_Sysref_Config(XRFdc *InstancePtr,
XRFdc_MultiConverter_Sync_Config *DACSyncConfigPtr,
XRFdc_MultiConverter_Sync_Config *ADCSyncConfigPtr, u32 SysRefEnable)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.

- **XRFdc\_MultiConverter\_Sync\_Config \*DACSyncConfigPtr:** Pointer to RF-DAC Multi-Tile Sync config structure.
- **XRFdc\_MultiConverter\_Sync\_Config \*ADCSyncConfigPtr:** Pointer to ADC Multi-Tile Sync config structure.
- **u32 SysRefEnable:** 0 for disable and 1 for enable.

### Description

This API is used to enable and disable the sysref.

### Return Value

XRFDC\_MTS\_OK

## Interrupt Handling

All interrupt handling functions are implemented in the `xrfdc_intr.c` file. The prototypes for these are provided through `xrfdc.h`.

### XRFdc\_IntrEnable

#### Function Prototype

```
u32 XRFdc_IntrEnable(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, u32 IntrMask);
```

#### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC or RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC or RF-DAC number inside the tile. Valid values are 0-3.
- **u32 IntrMask:** Interrupts to be enabled. The valid masks are:
  - XRFDC\_IXR\_FIFOUSRDAT\_MASK 0x0000000FU
  - XRFDC\_IXR\_FIFOUSRDAT\_OF\_MASK 0x00000001U
  - XRFDC\_IXR\_FIFOUSRDAT\_UF\_MASK 0x00000002U
  - XRFDC\_IXR\_FIFOMRGNIND\_OF\_MASK 0x00000004U

- XRFDC\_IXR\_FIFOMRGNIND\_UF\_MASK 0x00000008U
- XRFDC\_ADC\_IXR\_DATAPATH\_MASK 0x00000FF0U
- XRFDC\_ADC\_IXR\_DMON\_STG\_MASK 0x000003F0U
- XRFDC\_DAC\_IXR\_DATAPATH\_MASK 0x00001FF0U
- XRFDC\_DAC\_IXR\_INTP\_STG\_MASK 0x000003F0U
- XRFDC\_DAC\_IXR\_INTP\_I\_STG0\_MASK 0x00000010U
- XRFDC\_DAC\_IXR\_INTP\_I\_STG1\_MASK 0x00000020U
- XRFDC\_DAC\_IXR\_INTP\_I\_STG2\_MASK 0x00000040U
- XRFDC\_DAC\_IXR\_INTP\_Q\_STG0\_MASK 0x00000080U
- XRFDC\_DAC\_IXR\_INTP\_Q\_STG1\_MASK 0x00000100U
- XRFDC\_DAC\_IXR\_INTP\_Q\_STG2\_MASK 0x00000200U
- XRFDC\_ADC\_IXR\_DMON\_I\_STG0\_MASK 0x00000010U
- XRFDC\_ADC\_IXR\_DMON\_I\_STG1\_MASK 0x00000020U
- XRFDC\_ADC\_IXR\_DMON\_I\_STG2\_MASK 0x00000040U
- XRFDC\_ADC\_IXR\_DMON\_Q\_STG0\_MASK 0x00000080U
- XRFDC\_ADC\_IXR\_DMON\_Q\_STG1\_MASK 0x00000100U
- XRFDC\_ADC\_IXR\_DMON\_Q\_STG2\_MASK 0x00000200U
- XRFDC\_IXR\_QMC\_GAIN\_PHASE\_MASK 0x00000400U
- XRFDC\_IXR\_QMC\_OFFST\_MASK 0x00000800U
- XRFDC\_DAC\_IXR\_INVSNC\_OF\_MASK 0x00001000U
- XRFDC\_SUBADC\_IXR\_DCDR\_MASK 0x00FF0000U
- XRFDC\_SUBADC0\_IXR\_DCDR\_OF\_MASK 0x00010000U
- XRFDC\_SUBADC0\_IXR\_DCDR\_UF\_MASK 0x00020000U
- XRFDC\_SUBADC1\_IXR\_DCDR\_OF\_MASK 0x00040000U
- XRFDC\_SUBADC1\_IXR\_DCDR\_UF\_MASK 0x00080000U
- XRFDC\_SUBADC2\_IXR\_DCDR\_OF\_MASK 0x00100000U
- XRFDC\_SUBADC2\_IXR\_DCDR\_UF\_MASK 0x00200000U
- XRFDC\_SUBADC3\_IXR\_DCDR\_OF\_MASK 0x00400000U
- XRFDC\_SUBADC3\_IXR\_DCDR\_UF\_MASK 0x00800000U
- XRFDC\_ADC\_OVR\_VOLTAGE\_MASK 0x04000000U
- XRFDC\_ADC\_OVR\_RANGE\_MASK 0x08000000U

- XRFDC\_ADC\_DAT\_OVR\_MASK 0x40000000U
- XRFDC\_ADC\_FIFO\_OVR\_MASK 0x80000000U

### Description

This API function enables the interrupt for the corresponding converter by taking the `IntrMask` as an input and writing to the corresponding register bit.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_IntrDisable

### Function Prototype

```
u32 XRFdc_IntrDisable(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, u32 IntrMask);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC or RF-DAC tile number. Valid values are 0-3
- **u32 Block\_Id:** RF-ADC or RF-DAC number inside the tile. Valid values are 0-3.
- **u32 IntrMask:** Interrupts to be disabled. The valid masks are described in the API for `XRFdc_IntrEnable`.

### Description

This function disables the interrupt for the corresponding converter by taking the `IntrMask` as an input and writing to the corresponding register bit.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

### Related Information

[XRFdc\\_IntrEnable](#)

## XRFdc\_SetStatusHandler

### Function Prototype

```
void XRFdc_SetStatusHandler(XRFdc *InstancePtr, void *CallbackRefPtr,
XRFdc_StatusHandler FunctionPtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **void \*CallbackRefPtr:** Upper layer callback reference passed back when the callback function is invoked.
- **XRFdc\_StatusHandler FunctionPtr:** Pointer to the callback function.

### Description

This function sets the status callback function, the status handler, which the driver calls when it encounters conditions that must be reported to the higher layer software. The handler executes in an interrupt context to minimize the amount of processing.

### Return Value

None.

## XRFdc\_IntrClr

### Function Prototype

```
u32 XRFdc_IntrClr(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id,
u32 IntrMask);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC or RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC or RF-DAC number inside the tile. Valid values are 0-3
- **u32 IntrMask:** Interrupts to be cleared. The valid masks are described in the API for `XRFdc_IntrEnable`.

## Description

This function clears the interrupt for the corresponding converter by taking the `IntrMask` as an input and writing to the corresponding register bit.

## Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE`

# XRFdc\_GetIntrStatus

## Function Prototype

```
u32 XRFdc_GetIntrStatus(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32
Block_Id, u32 *IntrStsPtr);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC or RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC or RF-DAC number inside the tile. Valid values are 0-3.
- **u32 \*IntrStsPtr:** The interrupt status mask

## Description

This function returns the status of the interrupts through the masks. Valid masks are described in the API for `XRFdc_IntrEnable`.

## Return Value

`XRFDC_SUCCESS`

`XRFDC_FAILURE`

## Related Information

[XRFdc\\_IntrEnable](#)



## XRFdc\_IntrHandler

### Function Prototype

```
u32 XRFdc_IntrHandler(u32 Vector, void *XRFdcPtr);
```

### Arguments

- **u32 Vector:** Interrupt Vector number.
- **void \*XRFdcPtr:** Pointer to the driver instance that caused the interrupt.

### Description

This function is used internally by the driver. This is the master interrupt handler for the Zynq® UltraScale+™ RF Data Converter driver. This routine must be connected to an interrupt controller using OS/BSP-specific APIs. It clears the source of the interrupt and prints the cause of the interrupt.

### Return Value

METAL\_IRQ\_HANDLED (to inform Libmetal library, IRQ is handled).

## XRFdc\_GetEnabledInterrupts

### Function Prototype

```
u32 XRFdc_GetEnabledInterrupts(XRFdc *InstancePtr, u32 Type, u32 Tile_Id,
u32 Block_Id, u32 *IntrMask)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC or RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC or RF-DAC number inside the tile. Valid values are 0-3.
- **u32 \*IntrMask:** IntrMask is a pointer to the mask of enabled interrupts. 1 denotes an enabled interrupt, and 0 denotes a disabled interrupt.:

### Description

This API function populates `IntrMask` with the enabled interrupts for a given block.

## Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

# In-line Functions

All in-line functions are defined in the `xrfdc.h` file. These are available to programmers and are also used internally by other driver API functions. These in-line functions provide a set of utility APIs.

## XRFdc\_Get\_IPBaseAddr

### Function Prototype

```
static inline u32 XRFdc_Get_IPBaseAddr(XRFdc *InstancePtr);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.

### Description

None.

### Return Value

Base address of the IP.

## XRFdc\_Get\_TileBaseAddr

### Function Prototype

```
static inline u32 XRFdc_Get_TileBaseAddr(XRFdc *InstancePtr, u32 Type, u32 Tile_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** Valid values, 0-3.

### Description

None.

### Return Value

Base address of the requested tile.

## XRFdc\_Get\_BlockBaseAddr

### Function Prototype

```
static inline u32 XRFdc_Get_BlockBaseAddr(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** Valid values are 0-3.
- **u32 Block\_Id:** Valid values are 0-3.

### Description

None.

### Return Value

Base address of the requested converter.

## XRFdc\_GetNoOfDACBlock

### Function Prototype

```
static inline u32 XRFdc_GetNoOfDACBlock(XRFdc *InstancePtr, u32 Tile_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** Valid values are 0-3.

### Description

None.

## Return Value

This function returns the number of RF-DACs enabled in the tile.

# XRFdc\_GetNoOfADCBlocks

## Function Prototype

```
static inline u32 XRFdc_GetNoOfADCBlocks(XRFdc *InstancePtr, u32 Tile_Id);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** Valid values are 0-3.

## Description

None.

## Return Value

This function returns the number of RF-ADCs enabled in the tile.

# XRFdc\_IsDACBlockEnabled

## Function Prototype

```
static inline u32 XRFdc_IsDACBlockEnabled(XRFdc *InstancePtr, u32 Tile_Id,  
u32 Block_Id);
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** Valid values are 0-3.
- **u32 Block\_Id:** Valid values are 0-3.

## Description

None.

## Return Value

If the requested RF-DAC is enabled, the function returns 1; otherwise, it returns 0.

## XRFdc\_IsADCBlockEnabled

### Function Prototype

```
static inline u32 XRFdc_IsADCBlockEnabled(XRFdc *InstancePtr, u32 Tile_Id,
u32 Block_Id);
```

### Arguments

- **XRFdc\* InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** Valid values are 0-3.
- **u32 Block\_Id:** Valid values are 0-3.

### Description

None.

### Return Value

If the requested RF-ADC is enabled, the function returns 1; otherwise, it returns 0.

## XRFdc\_IsHighSpeedADC

### Function Prototype

```
static inline u32 XRFdc_IsHighSpeedADC(XRFdc *InstancePtr, int Tile)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **int Tile:** RF-ADC/RF-DAC tile number. Valid values are 0-3.

### Description

This returns whether the tile is high speed or not.

### Return Value

1 if high speed, otherwise 0.

## XRFdc\_GetDataType

### Function Prototype

```
static inline u32 XRFdc_GetDataType(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** Valid values are 0-3.
- **u32 Block\_Id:** Valid values are 0-3.

### Description

None.

### Return Value

If the data type is real, the function returns 0; otherwise, it returns 1.

## XRFdc\_GetDataWidth

### Function Prototype

```
static inline u32 XRFdc_GetDataWidth(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** Valid values are 0-3.
- **u32 Block\_Id:** Valid values are 0-3.

### Description

None.

### Return Value

This function returns the programmed data width for the RF-ADC.

## XRFdc\_GetInverseSincFilter

### Function Prototype

```
static inline u32 XRFdc_GetInverseSincFilter(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** Valid values are 0-3.
- **u32 Block\_Id:** Valid values are 0-3.

### Description

None.

### Return Value

If the inverse sinc filter is enabled for the RF-DAC, the function returns 1; otherwise, it returns 0.

## XRFdc\_GetMixedMode

### Function Prototype

```
static inline u32 XRFdc_GetMixedMode(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** Valid values are 0-3.
- **u32 Block\_Id:** Valid values are 0-3.

### Description

None.

### Return Value

This function returns the mixed mode setting for the RF-DAC.

## XRFdc\_GetMasterTile

### Function Prototype

```
static inline u32 XRFdc_GetMasterTile(XRFdc *InstancePtr, u32 Type);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.

### Description

None.

### Return Value

The function returns the master tile ID.

## XRFdc\_GetSysRefSource

### Function Prototype

```
static inline u32 XRFdc_GetSysRefSource(XRFdc *InstancePtr, u32 Type);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.

### Description

None.

### Return Value

The function returns the source of the SYSREF (internal or external).

## XRFdc\_GetFabClkFreq

### Function Prototype

```
static inline double XRFdc_GetFabClkFreq(XRFdc *InstancePtr, u32 Type, u32 Tile_Id);
```



### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** Valid values are 0-3.

### Description

None.

### Return Value

The function returns the programmed PL clock frequency.

## XRFdc\_IsFifoEnabled

### Function Prototype

```
static inline u32 XRFdc_IsFifoEnabled(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** Valid values are 0-3.
- **u32 Block\_Id:** Valid values are 0-3.

### Description

None.

### Return Value

If the FIFO is enabled, the function returns 1; otherwise, it returns 0.

## XRFdc\_GetDriverVersion

### Function Prototype

```
static inline double XRFdc_GetDriverVersion();
```

### Arguments

None.

### Description

Gets the driver version.

### Return Value

Driver version number.

## XRFdc\_GetConnectedIData

### Function Prototype

```
static inline int XRFdc_GetConnectedIData(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number. Valid values are 0-3.

### Description

Get converter connected for I digital data path.

### Return Value

Converter number.

## XRFdc\_GetConnectedQData

### Function Prototype

```
static inline int XRFdc_GetConnectedQData(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id);
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.

- **u32 Type:** RF-ADC or RF-DAC; 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC number. Valid values are 0-3.

### Description

Get converter connected for Q digital data path.

### Return Value

Converter number.

## XRFdc\_ClrSetReg

### Function Prototype

```
static inline void XRFdc_ClrSetReg(XRFdc *InstancePtr, u32 BaseAddr, u32
RegAddr, u16 Mask, u16 Data)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 BaseAddr:** Address for a Block.
- **u32 RegAddr:** Register Offset value.
- **u16 Mask:** Bit mask value.
- **u16 Data:** Value to be written to the register.

### Description

This API performs Read Modify Write.

### Return Value

None

## XRFdc\_ClrReg

### Function Prototype

```
static inline void XRFdc_ClrReg(XRFdc *InstancePtr, u32 BaseAddr, u32
RegAddr, u16 Mask)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 BaseAddr:** Address for a Block.
- **u32 RegAddr:** Register Offset value.
- **u16 Mask:** Bit mask value.

### Description

This API performs Read and Clear.

### Return Value

None

## XRFdc\_RDReg

### Function Prototype

```
static inline u16 XRFdc_RDReg(XRFdc *InstancePtr, u32 BaseAddr, u32  
RegAddr, u16 Mask)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 BaseAddr:** Address for a Block.
- **u32 RegAddr:** Register Offset value.
- **u16 Mask:** Bit mask value.

### Description

This API performs Read and Mask with the value.

### Return Value

None

## XRFdc\_IsDACDigitalPathEnabled

### Function Prototype

```
static inline u32 XRFdc_IsDACDigitalPathEnabled(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-DAC block number. Valid values are 0-3.

### Description

This API checks whether RF-DAC digital path is enabled or not.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_IsADCDigitalPathEnabled

### Function Prototype

```
static inline u32 XRFdc_IsADCDigitalPathEnabled(XRFdc *InstancePtr, u32 Tile_Id, u32 Block_Id)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Tile\_Id:** RF-ADC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC block number. Valid values are 0-3.

### Description

This API checks whether ADC Digital path is enabled or disabled.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_CheckDigitalPathEnabled

### Function Prototype

```
static inline u32 XRFdc_CheckDigitalPathEnabled(XRFdc *InstancePtr, u32
Type, u32 Tile_Id, u32 Block_Id)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC block number. Valid values are 0-3.

### Description

This API checks whether RF-ADC/RF-DAC digital path is enabled or not.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_SetConnectedIQData

### Function Prototype

```
static inline void XRFdc_SetConnectedIQData(XRFdc *InstancePtr, u32 Type,
u32 Tile_Id, u32 Block_Id, int ConnectedIData, int ConnectedQData)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC block number. Valid values are 0-3
- **int ConnectedIData:** Converter Id to which DigitalPathI connected
- **int ConnectedQData:** Converter Id to which DigitalPathQ connected

## Description

This API set Data Converter connected for digital data path I and Q.

## Return Value

None

# XRFdc\_GetMultibandConfig

## Function Prototype

```
static inline u32 XRFdc_GetMultibandConfig(XRFdc *InstancePtr, u32 Type, u32 Tile_Id)
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.

## Description

This API is to get Multiband Config data

## Return Value

This API returns Multiband configuration.

# XRFdc\_CheckBlockEnabled

## Function Prototype

```
static inline u32 XRFdc_CheckBlockEnabled(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, u32 Block_Id)
```

## Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **u32 Block\_Id:** RF-ADC/RF-DAC block number. Valid values are 0-3.

### Description

This API checks whether RF-ADC/RF-DAC block is enabled or disabled.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_CheckTileEnabled

### Function Prototype

```
static inline u32 XRFdc_CheckTileEnabled(XRFdc *InstancePtr, u32 Type, u32 Tile_Id)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.

### Description

This API checks whether RF-ADC/RF-DAC tile is enabled or disabled.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetMaxSampleRate

### Function Prototype

```
static inline u32 XRFdc_GetMaxSampleRate(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, double *MaxSampleRatePtr)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** 0 for RF-ADC and 1 for RF-DAC.



- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **double \*MaxSampleRatePtr:** Pointer to be populated with maximum sampling rate for tile

### Description

This API is used to get the tile's maximum sampling rate.

### Return Value

XRFDC\_SUCCESS

XRFDC\_FAILURE

## XRFdc\_GetMinSampleRate

### Function Prototype

```
static inline u32 XRFdc_GetMinSampleRate(XRFdc *InstancePtr, u32 Type, u32 Tile_Id, double *MinSampleRatePtr)
```

### Arguments

- **XRFdc \*InstancePtr:** Pointer to the driver instance.
- **u32 Type:** 0 for RF-ADC and 1 for RF-DAC.
- **u32 Tile\_Id:** RF-ADC/RF-DAC tile number. Valid values are 0-3.
- **double \*MinSampleRatePtr:** Pointer to be populated with minimum sampling rate for tile

### Description

This API is used to get the tile's minimum sampling rate.

### Return Value

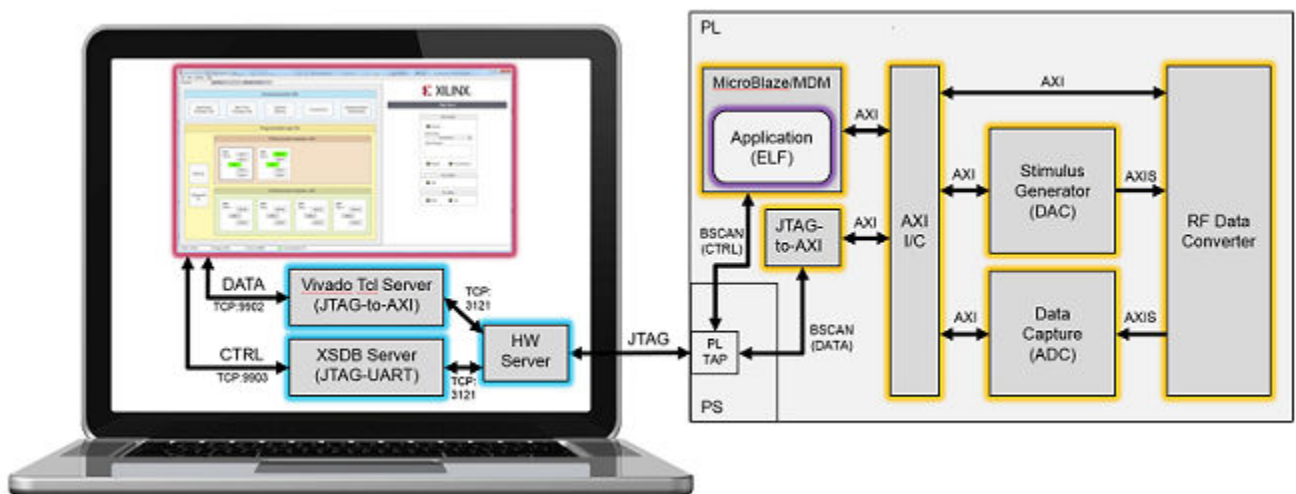
XRFDC\_SUCCESS

XRFDC\_FAILURE

# RF Analyzer

The RF Analyzer provides a method of evaluating and monitoring the converters in hardware. An overview of the system is given below. The IP example design blocks are outlined in yellow.

Figure 104: RF Analyzer



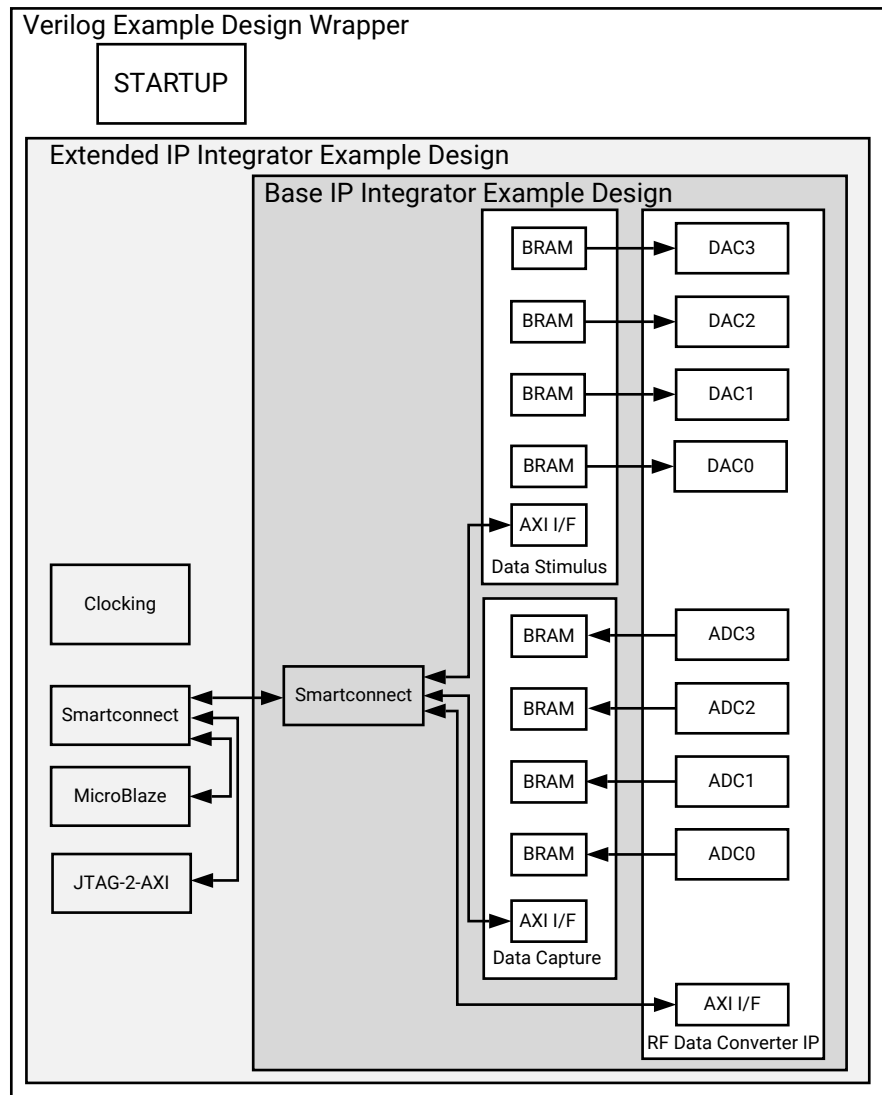
The RF Analyzer GUI provides access to the data stimulus and capture blocks together with data analysis tools for processing the data. The MicroBlaze™ provides system configuration and control.

The major changes to the existing example design are:

- The addition of a MicroBlaze to facilitate the configuration and control of the IP subsystem.
- The inclusion of a JTAG to AXI IP core to enable communication between the RF Analyzer GUI and the IP subsystem.
- Clocking logic to generate the AXI4-Stream clocks from the converter output clocks.
- Clocking logic to generate the AXI4-Lite clock from the FPGA startup component.

A block diagram of the expanded example design is shown below.

Figure 105: RF Analyzer Example Design



X21660-051319

The base IP integrator example design is as described in the Example Design Chapter. The extended IP integrator example design includes the MicroBlaze™, JTAG to AXI, and clocking blocks to support the RF Analyzer.

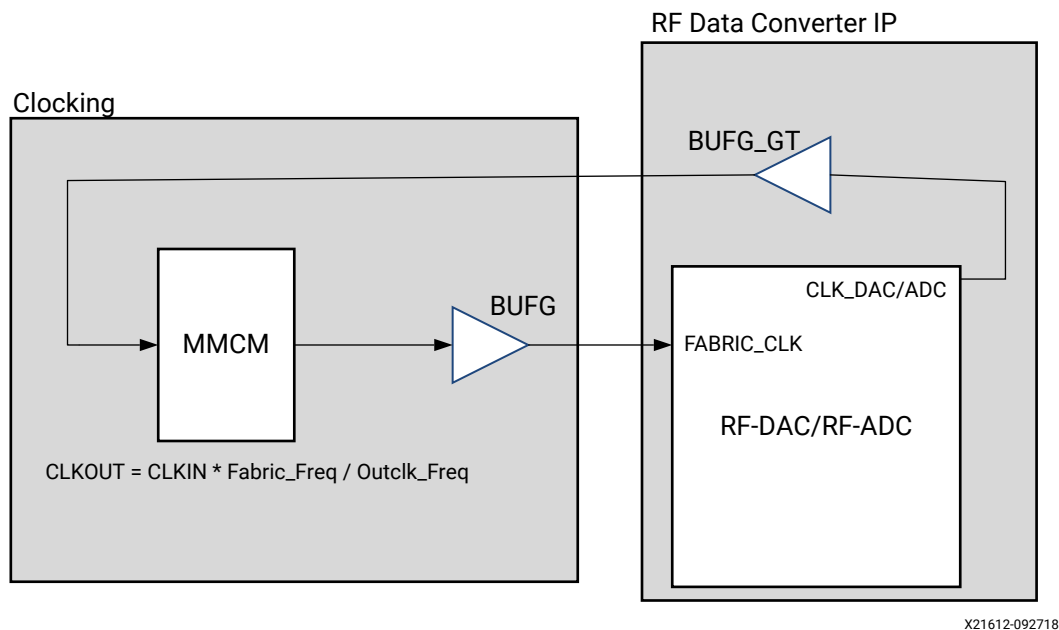
**Related Information**

[Example Design](#)

# Clocking

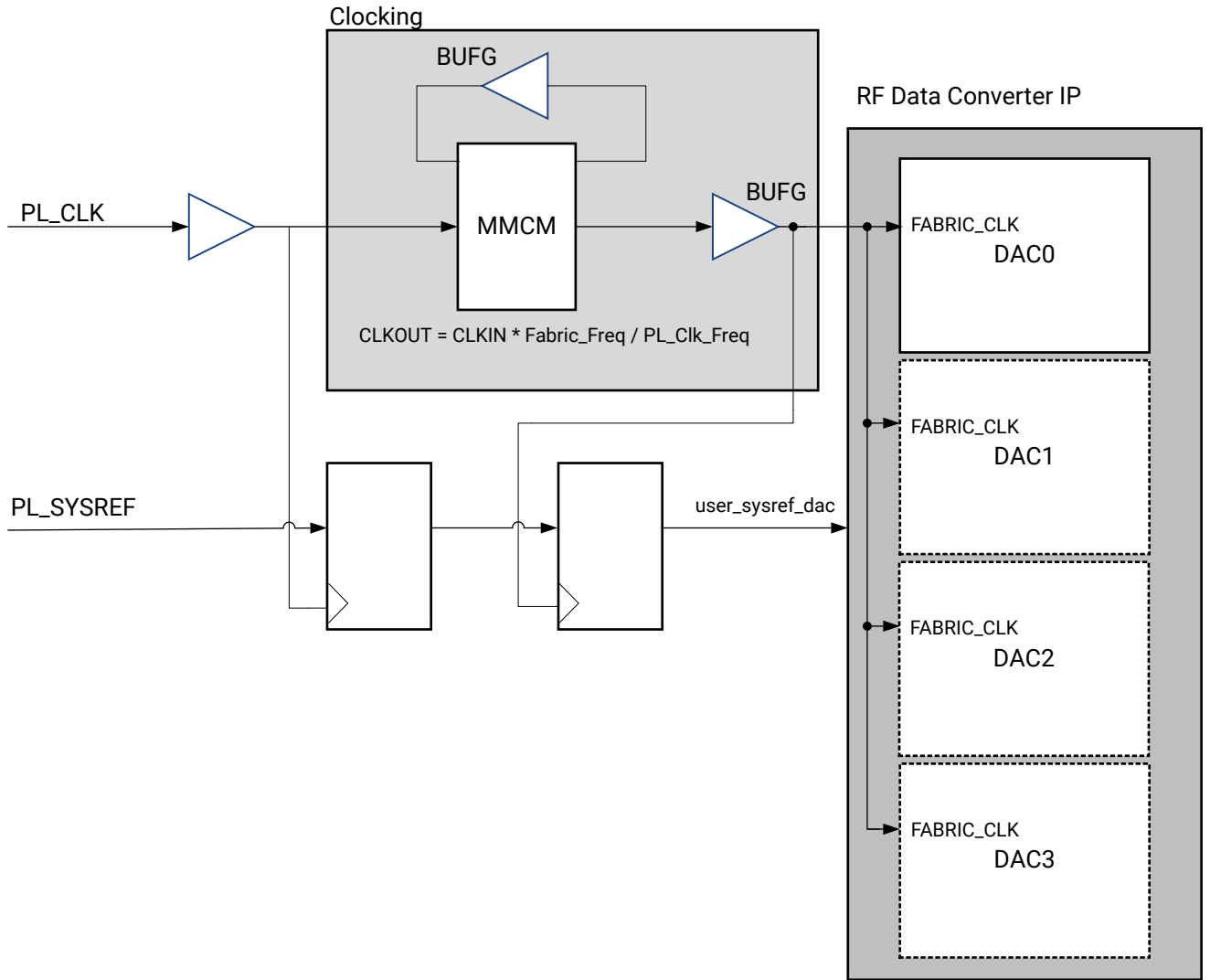
When multi-tile synchronization (MTS) is not enabled the AXI4-Stream clocks for each RF-ADC and RF-DAC are generated from the respective converter output clock. Any combination of output clock frequency and AXI4-Stream clock frequency is supported. The clock wizard IP core is used to generate a MMCM to provide the correct AXI4-Stream clock. The MMCM is programmable using the AXI4-Lite interface, allowing changes to the decimation and interpolation factors of the converters. The default clocking is shown below.

Figure 106: RF Analyzer Clocking



When the user selects multi-tile sync all the fabric clocks for the converters in the multi-tile sync group are driven from the same source. The programmable logic clock and user SYSREF are inputs. The user will input a PL clock frequency from the GUI. The pin placement of the PL clock and SYSREF is left to the user. The clocking for the DAC when multi-tile sync has been selected is shown below.

Figure 107: Clocking for Multi-Tile Synchronization



X22865-050919

For accurate SYSREF capture the PL clock frequency must be a common integer multiple of the RF-ADC and RF-DAC AXI4-Stream clock frequencies.

## Address Space

The RF Analyzer address space is shown in the following table.

**Table 85: RF Analyzer Address Space**

Address	Size	Component
0xE000_0000	1024kbytes	RF-ADC Data Capture
0xC000_0000	1024kbytes	RF-DAC Data Stimulus
0x44B0_0000	256kbytes	Converter IP
0x44C0_0000	64kbytes	DAC0 MMCM
0x44C1_0000	64kbytes	DAC1 MMCM
0x44C2_0000	64kbytes	DAC2 MMCM
0x44C3_0000	64kbytes	DAC3 MMCM
0x44C4_0000	64kbytes	ADC0 MMCM
0x44C5_0000	64kbytes	ADC1 MMCM
0x44C6_0000	64kbytes	ADC2 MMCM
0x44C7_0000	64kbytes	ADC3 MMCM

The register maps for the data capture and data stimulus blocks are described in the Example Design Chapter. The converter IP register maps are described in the Product Specification Chapter.

### Related Information

[Example Design](#)

[Register Space](#)

## User Interface

The RF Analyzer IP integrator design can be integrated into user designs. Access to the RF-DAC AXI4-Stream interface is provided through the following ports.

**Table 86: RF-DAC User Interface**

Port Name <sup>1</sup>	I/O	Clock	Description
sX_axis_aclk	Out	N/A	AXI4-Stream clock output from the MMCM in the clocking block
sXY_0_tdata	In	sX_axis_aclk	AXI4-Stream data input
sXY_0_tvalid	In	sX_axis_aclk	AXI4-Stream valid
sXY_0_tready	Out	sX_axis_aclk	AXI4-Stream ready

Table 86: RF-DAC User Interface (cont'd)

Port Name <sup>1</sup>	I/O	Clock	Description
user_select_XY_0	In	sX_axis_aclk	When this input is driven High the RF-DAC is driven from the user interface. When Low the RF-DAC is driven by the data stimulus block in the RF Analyzer.

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DUC block in the tile (0 to 3).

The AXI4-Stream interface is routed out to the user design through the ports described in the following table.

Table 87: RF-ADC User Interface

Port Name <sup>1</sup>	I/O	Clock	Description
mX_axis_aclk	Out	N/A	AXI4-Stream clock output from the MMCM in the clocking block
mXY_0_tdata	Out	mX_axis_aclk	AXI4-Stream data output
mXY_0_tvalid	Out	mX_axis_aclk	AXI4-Stream valid
mXY_0_tready	In	mX_axis_aclk	AXI4-Stream ready. Tie Low when the user interface is not in use. The RF Analyzer data capture block should be disabled when the user interface is in use.

**Notes:**

1. X refers to the location of the tile in the converter column. Y refers to the location of the DUC block in the tile (0 to 3).

## Hardware Trigger

Hardware trigger inputs are available. When enabled, asserting the hardware trigger will start the data transmission or capture. This overrides the writes to the Start Data register.

Table 88: RF-DAC Hardware Trigger Interface

Port Name <sup>1</sup>	I/O	Clock	Description
dacX_hw_trigger	In	s_axi_aclk	Hardware trigger. Assert to enable data transmission. Deassert before starting any subsequent data transmission.
dacX_hw_trigger_en	In	s_axi_aclk	Hardware trigger enable. Assert to enable the hardware trigger

**Notes:**

1. X refers to the location of the tile in the converter column.

**Table 89: RF-ADC Hardware Trigger Interface**

Port Name <sup>1</sup>	I/O		
adcX_hw_trigger	In	s_axi_aclk	Hardware trigger. Assert to enable data capture. The input should be deasserted before any subsequent data capture.
adcX_hw_trigger_en	In	s_axi_aclk	Hardware trigger enable. Assert to enable the hardware trigger

**Notes:**

1. X refers to the location of the tile in the converter column.



# Additional Resources and Legal Notices

---

## Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

---

## Documentation Navigator and Design Hubs

Xilinx<sup>®</sup> Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado<sup>®</sup> IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, select **Start** → **All Programs** → **Xilinx Design Tools** → **DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

**Note:** For more information on DocNav, see the [Documentation Navigator](#) page on the Xilinx website.

---

## References

These documents provide supplemental material useful with this product guide:

1. *Zynq UltraScale+ RFSoc Data Sheet: Overview* ([DS889](#))
2. *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics* ([DS926](#))
3. *Zynq UltraScale+ Device Technical Reference Manual* ([UG1085](#))
4. *UltraScale Architecture PCB Design User Guide* ([UG583](#))
5. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#))
6. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
7. *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
8. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
9. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
10. *Zynq UltraScale+ Device Packaging and Pinouts Product Specification User Guide* ([UG1075](#))
11. *RF Data Converter Interface User Guide* ([UG1309](#))

## Revision History

The following table shows the revision history for this document.

Section	Revision Summary
<b>10/30/2019 v2.2</b>	
All	Terminology updated to Quad and Dual converter types
<a href="#">Background Calibration Process</a>	Signal name update
<a href="#">PLL Parameters</a>	New topic
<a href="#">Detailed Description</a>	Updated
<a href="#">NCO Frequency Hopping</a>	Updated
<a href="#">Multi-Tile Mode</a>	Updated
<a href="#">User Parameters</a>	Updated
<a href="#">Chapter 7: Test Bench</a>	Updated
<a href="#">Analog Signaling</a>	Updated
<a href="#">struct XRFdc_Mixer_Settings</a>	Valid macros for MixerType updated
<a href="#">User API Functions</a> <a href="#">XRFdc_RegisterMetal</a> <a href="#">XRFdc_GetPLLConfig</a> <a href="#">Interrupt Handling</a> <a href="#">XRFdc_GetEnabledInterrupts</a>	New
<a href="#">struct XRFdc</a>	Updated driver title
<b>05/22/2019 v2.1</b>	
<a href="#">NCO Frequency Hopping</a>	Updated content
<a href="#">Over Voltage</a>	Updated content

Section	Revision Summary
<a href="#">Interrupt Handling</a>	Interrupt handling updated
<a href="#">System Clocking Tab</a>	New tab in Vivado® IDE
<a href="#">Clocking</a>	Updated content
<a href="#">Address Space</a>	Address space updated
<a href="#">Hardware Trigger</a>	New topic
<a href="#">Overview</a>	Updated driver files list
<a href="#">Summary Tab</a>	Removed tab in Vivado IDE
<a href="#">struct XRFdc_Calibration_Coefficients</a> <a href="#">struct XRFdc_Cal_Freeze_Settings</a> <a href="#">XRFdc_DisableCoefficientsOverride</a> <a href="#">XRFdc_SetCalCoefficients</a> <a href="#">XRFdc_GetCalCoefficients</a> <a href="#">XRFdc_SetCalFreeze</a> <a href="#">XRFdc_GetCalFreeze</a> <a href="#">XRFdc_SetDither</a> <a href="#">XRFdc_GetDither</a> <a href="#">XRFdc_GetPLLConfig</a> <a href="#">XRFdc_GetMaxSampleRate</a> <a href="#">XRFdc_GetMinSampleRate</a>	New driver topics
<a href="#">XRFdc_IsHighSpeedADC</a>	Replaces XRFdc_IsADC4GSPS
<a href="#">XRFdc_SetInvSincFIR</a> <a href="#">XRFdc_GetInvSincFIR</a> <a href="#">XRFdc_IntrEnable</a> <a href="#">XRFdc_IntrDisable</a> <a href="#">XRFdc_IntrClr</a> <a href="#">XRFdc_GetIntrStatus</a> <a href="#">XRFdc_IntrHandler</a>	Updated driver information
<b>12/05/2018 v2.1</b>	
<a href="#">Appendix D: RF Analyzer</a>	Added RF Analyzer appendix.
<a href="#">Reset Count Register (0x0038)</a>	Added register.
<a href="#">Over Voltage</a>	Updated section.
<a href="#">PLL</a>	Updated section.
<a href="#">Summary Tab</a>	New tab documented.
<a href="#">Chapter 6: Example Design</a>	Example Design chapter updated.
<a href="#">Chapter 7: Test Bench</a>	Test Bench chapter updated.
<a href="#">User API Functions</a>	Added to and updated section.
<a href="#">In-line Functions</a>	Added in-line functions.
<b>04/17/2018 v2.0</b>	
<a href="#">Data Structures</a> <a href="#">User API Functions</a> <a href="#">In-line Functions</a>	Added and updated driver commands.
<b>04/04/2018 v2.0</b>	
<a href="#">Port Descriptions</a>	Added clarification to the notes in the Port Description tables.

Section	Revision Summary
<a href="#">Multi-Tile Synchronization Ports</a>	Added Multi Converter Synchronization Ports table.
<a href="#">Calibration Freeze Ports for Quad RF-ADC Tiles</a> <a href="#">Calibration Freeze Ports for Dual RF-ADC Tiles</a>	Added Calibration Freeze Ports tables.
<a href="#">Basic Tab</a> <a href="#">Advanced Tab</a>	Added Basic and Advanced Vivado IDE tab descriptions.
<a href="#">RF-ADC Tab</a> <a href="#">RF-DAC Tab</a>	Improved description of RF-ADC and RF-DAC Vivado IDE tabs.
<a href="#">Chapter 6: Example Design</a>	Added information on IP Integrator example design.
<a href="#">RF-DAC Data Stimulus Block</a>	Added information on RF-DAC source register map.
<b>10/04/2017 v1.1</b>	
Major content addition and IP core updates.	Major content addition and IP core updates.
<b>04/05/2017 v1.0</b>	
Initial release.	

## Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby **DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE**; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

## **AUTOMOTIVE APPLICATIONS DISCLAIMER**

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

## **Copyright**

© Copyright 2017-2019 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. Arm is a registered trademark of Arm Limited in the EU and other countries. All other trademarks are the property of their respective owners.