ADRV903x System Development User Guide

INTRODUCTION

This document is the main source of information for system engineers and software developers using the Analog Devices ADRV903x software-defined radio transceiver family. This document is organized to make it easy to find the relevant information. These sections include:

- System Overview – explains the capability of the part and an introduction to all the subsystems and functions, including block diagrams and interfaces.
- Software Architecture Description – explains software design approach using APIs and all details needed to develop code to operate the device.
- Software Integration – explains the structure of the Analog Devices API and how to integrate into the customer's code.
- Serial Peripheral Interface (SPI) Control – main control interface between the baseband processor (BBP – also referred to as BBIC) and the device.
- System Initialization – sequence of steps needed at startup.
- JESD204B/C Interface – describes the high-speed digital interface that transfers data to/from a baseband processor.
- Synthesizer Configuration – describes the design, control, and versatility of the synthesizer subsystem.
- ARM Processor and Device Calibration – explains how the ARM schedules and controls the calibrations.
- Stream Processor and System Control – explains the stream processor functions and how they are implemented.
- Tx Overview and Path Control – describes operation of the Tx attenuation settings and available software API used for control.
- Tx Power Amplifier Protection – describes the protection circuitry and how it works in conjunction with the general purpose interrupt feature to enable Tx attenuation and notify the BBIC that such an event has taken place.
- Rx Gain Control and Gain Compensation – describes automatic and manual gain control options.
- Digital Filter Configuration – describes the digital processing portion of each receiver and transmitter and provides details on configuration options.
- General Purpose Input/Output Configuration – describes the different GPIO capabilities and how to configure them for different functions.
- General Purpose Interrupt – describes the various interrupt options that can be routed to the GPINT pins for monitoring purposes.
- RF Port Interface Overview – describes the RF port impedance matching process and explains different topologies that can be used to achieve proper impedance matching.
- Power Management Considerations – explains how to connect power supplies to the device, what inputs supply which blocks, and what precautions to take when completing a schematic and layout for power routing implementation.
- PCB Layout Considerations – provides guidelines for proper printed circuit board (PCB) layout and techniques for maximizing performance and minimizing channel-to-channel interference.
- ACE Software – explains setup and control or the device using the graphical user interface (GUI) software.

Additional information will be added to this document as new features and functions are implemented in the ADRV903x device family.

TABLE OF CONTENTS

TABLE OF CONTENTS

REVISION HISTORY

9/2025—Rev. A to Rev. B

Changes to External LO Section.....104
Added Table 48; Renumbered Sequentially.....104
Added External LO Setup Section..... 105
Changes to Table 104.....207

2/2025—Revision A: Initial Version

.....

SYSTEM OVERVIEW

The ADRV903x is a highly integrated RF agile transceiver designed for use in a wide variety of high-performance applications. The ADRV903x device contains up to eight independently controlled transmitters and receivers, two observation receivers for monitoring transmitter channel outputs, integrated synthesizers including two RF LO's, and digital signal processing functions to provide a complete transceiver solution. The device provides the advanced radio performance and low power consumption for various applications such as instrumentation, aerospace and defense and general-purpose secure communications. This document includes descriptions of all superset functions available in the ADRV903x platform. Some ADRV903x variants may be developed for specific design targets that do not include all available functions, so refer to the data sheet for the specific device to determine which features are included. To avoid confusion, the term device or ADRV903x is used throughout this document to refer to any variant that employs a specific function. Products within the ADRV903x platform are listed in Table 1.

The ADRV903x family is designed to operate over the wide frequency ranges, from 450 MHz to 7.1 GHz, using either internal phase locked loop (PLL) synthesizers or external local oscillator (LO) sources. The receiver channels support bandwidths up to 660 MHz with data transfer across (up to) eight JESD204B/JESD204C serializer-deserializer (SERDES) lanes at rates up to 32.44 Gbps. Internal LO routing allows some receivers to operate on a different LO frequency than others, allowing use in multichannel systems. The transmit channels operate over the same frequency range as the receivers. Each transmitter channel supports up to 800 MHz synthesis bandwidth with data input across (up to) eight SERDES lanes at rates up to 32.44 Gbps. Different transmitter channels can operate using the same LO frequency or a different frequency. In addition, LO routing allows the transmitters to operate at different frequencies than the receivers for additional flexibility. Two observation receiver channels provide the capability to monitor feedback from the transmitter outputs. The feedback loops can be used to implement error correction, calibration, and signal enhancing algorithms. These receivers operate in the same frequency range as the transmitter channels, and they support up to 800 MHz channel bandwidth to match the output synthesis bandwidth of the transmitter channels. These channels provide digital datapaths to the internal ARM processor for use in calibration and signal enhancement algorithms, and they can also be muxed with the Rx channels to output data to a BBIC over the Rx SERDES output lanes.

Multiple fully integrated PLLs provide a high level of flexibility and performance. Two are high performance, low power fractional-N RF synthesizers that can be configured to supply the transmitters and receivers in different configurations. A clock and serdes PLL are included to generate the converter and digital clocks for signal processing and communication interfaces respectively.

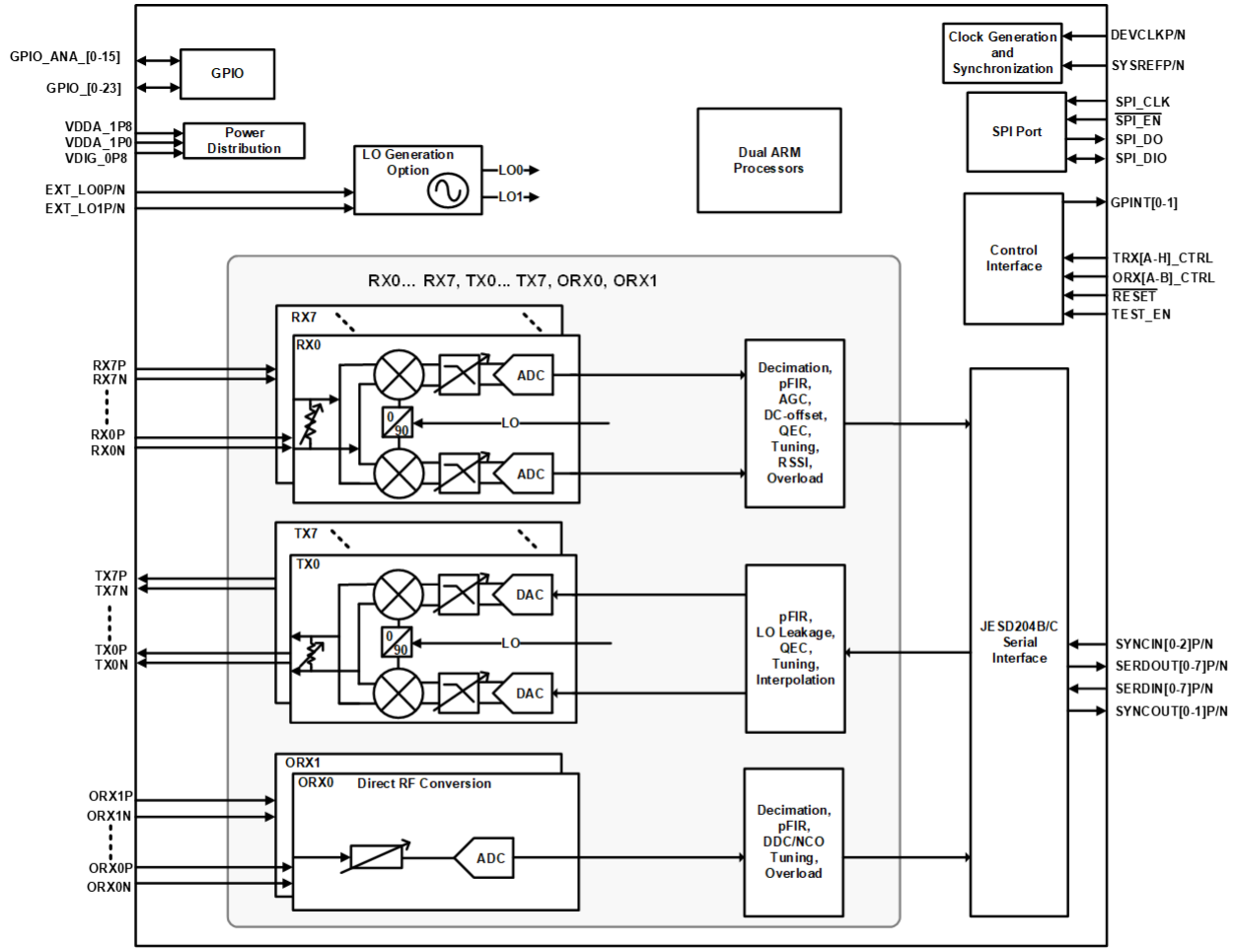
Power supplies for the device are: 1.8 V, 1.0 V, and 0.8 V. These supplies are connected directly to the power inputs for some blocks and buffered by internal LDO regulators for other functions for maximum performance. The 0.8 V supplies the digital processing blocks. The 1.8 V supplies all the GPIO and interface ports that connect with the BBIC.

Table 1. Product Subset Features

Feature	ADRV9032R
Tx	2
Rx	2
ORx	2
Tunable LO Range	450 MHz to 7125 MHz
RF Range	350 MHz to 7225 MHz
Tx Large Signal BW or Instantaneous BW	200 MHz
Tx Synthesis BW	450 MHz
Rx BW	200 MHz
ORx BW	450 MHz
JESD204B/C Lane Rate	16.5 Gbps

Figure 1 is a high-level view of the functions in the ADRV903x. Descriptions of each block with setup and control details are provided in subsequent sections of this document.

SYSTEM OVERVIEW



VDDA_1P8 REPRESENTS VVCO0_1P8, VVCO1_1P8, VANA0_1P8, VANA1_1P8, VSYS_1P8, VCONV0_1P8, VCONV1_1P8, VCONV2_1P8, VCONV3_1P8, VORX0_1P8, VORX1_1P8, VIF_1P8, VSERVCO_1P8, VCLKVCO_1P8, VTX0_1P8, AND VTX1_1P8.

VDDA_1P0 REPRESENTS VRXLO0_1P0, VTXLO0_1P0, VLO0_1P0, VLO1_1P0, VTXLO1_1P0, VRXLO1_1P0, VBB0_1P0, VBB1_1P0, VCONV0_1P0, VCONV1_1P0, VCONV2_1P0, VCONV3_1P0, VSCLK0_1P0, VSCLK1_1P0, VORX0_1P0, VORX1_1P0, VCLKSYN_1P0, VCLKGEN_1P0, VSERSYN_1P0, VSYN0_1P0, VSYN1_1P0, VDEV_1P0, VSER_1P0, and VDES_1P0.

Figure 1. ADRV903x Functional Block Diagram

SOFTWARE INTEGRATION

This section provides information about the software deliverables from ADI, including the ADI developed Application Programming Interface (API) and resource files essential for the functioning of the transceiver. This section outlines the overall architecture, folder structure, and methods for using API software on a platform. This document does not explain the API library functions. Detailed information regarding the API functions is in the doxygen document included with the API software.

SOFTWARE DELIVERABLES

The software package consists of three main folders

- api – Contains the API 'C' source code for controlling the ADRV903x family of transceiver devices which can be integrated into a user application running on a baseband processor.
- firmware – Contains pre-compiled binaries for the dual-core embedded ARM processor in the ADRV903x family of devices.
- gain_tables – Contains the programmable gain table for the receivers. The transmitter attenuation table is hard coded into the ADRV903x and is not programmable.

NOTE: The stream processor binary and the profile binary resource files are required for programming the ADRV903x. The stream binary and the profile binary resource files targeting a specific use case are generated with an ADI provided evaluation software. Therefore, the stream and profile binary resource files are not delivered as part of the software package.

SOFTWARE INTEGRATION PROCESS OVERVIEW

An overview of the software integration process is in Figure 2. A more detailed explanation of the integration process is provided in the sections to follow.

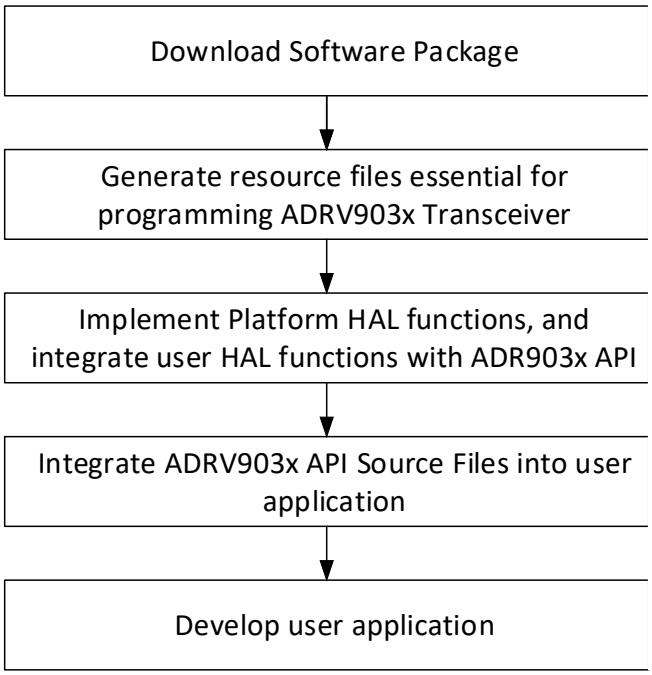


Figure 2. Software Integration Process Overview

SOFTWARE ARCHITECTURE

The ADRV903x contains dedicated signal processing blocks, ADCs, DACs, two ARM processor cores, and a co-processor called the stream processor. A simplified logical partitioning of a typical software architecture designed with the ADRV903x firmware and API is in Figure 3. The firmware for the ARM cores is a pre-compiled binary. The stream co-processor binary is user generated. The process to generate a stream binary is described in the Resource Files.

SOFTWARE INTEGRATION

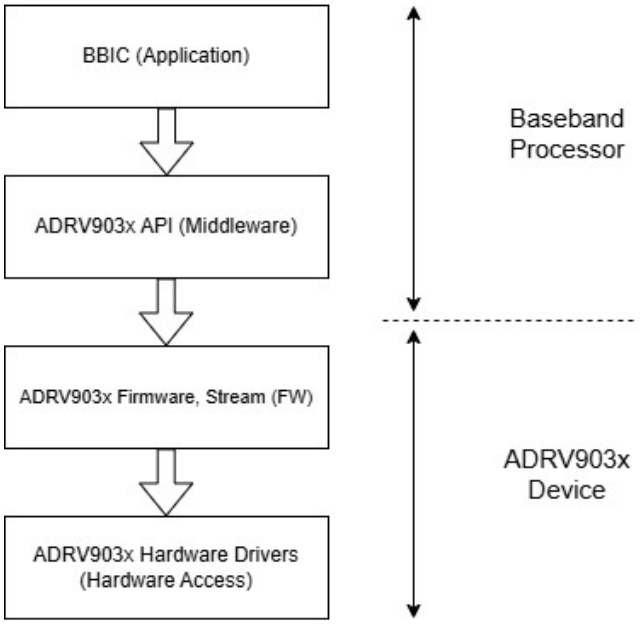


Figure 3. Logical Partitioning/Layering of the ADRV903x Software Application

An ADRV903x user application commands the ADRV903x through the API. The API C-source code is delivered as part of the software package. The API is processor and operating system agnostic and can be deployed on a bare-metal processor as well as a processor running an operating system. ADI recommends using a platform running an operating system such as Linux, which provides a sufficiently large memory footprint, and can take advantage of ADI utility functions. Refer to the API Integration for information regarding integrating the ADRV903x API into a user application.

Figure 4 shows a more detailed ADRV903x based system software architecture. A baseband processor running an ADRV903x based application controls the transceiver through the ADRV903x API. The API integrated with the user application relies on a SPI interface in the baseband processor to interact with the ADRV903x. The ARM firmware running on a dual core embedded ARM processor consists of the algorithms for transceiver calibrations, controlled through the ADRV903x API in the baseband processor. The API transacts with the firmware through a well-defined set of commands via common memory (mailbox interface). The firmware contains drivers to interact with the transceiver hardware. The transceiver hardware is memory mapped to the dual core embedded ARM processor through an AHB interface.

The stream co-processor sets up and manages the transmit/receive chains of the ADRV903x on occurrence of certain events (such as Tx/Rx/ORx enable). A stream command tied to an event is invoked by the ARM processor or directly by the baseband processor via GPIO inputs. The stream processor accesses the ADRV903x hardware resources through AHB memory mapped registers. There is one core stream processor and eighteen slice stream processors, one each for the eight Tx and Rx datapaths, and two for the ORx datapaths. The existence of individual slice stream processors for each datapath enables true real-time parallel operation of all unique Tx and Rx datapaths.

The control commands issued to the embedded ARM processor and the stream co-processor in the ADRV903x are accompanied by corresponding status responses from the firmware. The user application retrieves the command statuses through the API. The user application also monitors critical system parameters through the GP interrupt status pins. Detailed Software Architectural View of an ADRV903x Software Application.

SOFTWARE INTEGRATION

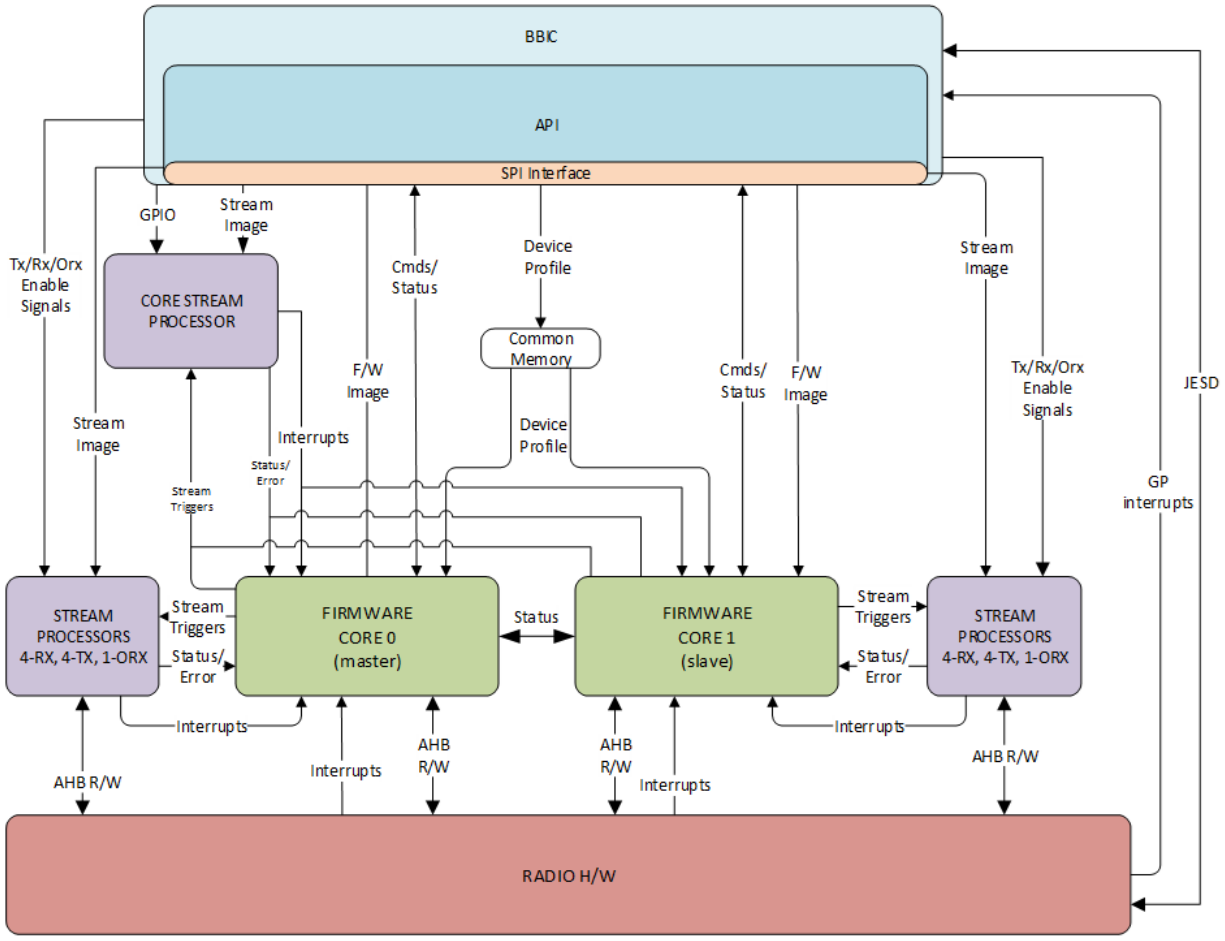


Figure 4. A Detailed Software Architectural View of an ADRV903x Software Application

ADI Evaluation System Software Architecture

The software architecture of the ADRV903x ADI evaluation system is in Figure 5. The software architecture of the ADI evaluation system broadly follows a client-server model, with the server residing in the baseband processor. The client application software can be a PC based application such as MATLAB or an application running on the baseband processor itself. A PC based application client interacts with the server through a transport layer interface link called Enhanced Remote Procedure Calling interface (ERPC) over a TCP/IP protocol.

The baseband processor consists of an SOC-FPGA host integrated with an embedded processor running a Linux operating system. The ADRV903x API is integrated with the baseband processor software to control the ADRV903x. The platform Hardware Abstraction Layer (HAL) interface (SPI, timer, etc.) is implemented for the ADI ADRV903x evaluation system, which in turn is used by the ADRV903x's API for interacting with the ADRV903x.

In a typical use case of the ADI evaluation system, the command flows from a PC based application client to the server running on the baseband processor through the ERPC transport layer interface. The command gets decoded into an API call in the server. The API call interacts with the ADRV903x through a SPI interface, returns the status to the calling function in the baseband API, which is sent back to the application layer through the ERPC transport layer link.

SOFTWARE INTEGRATION

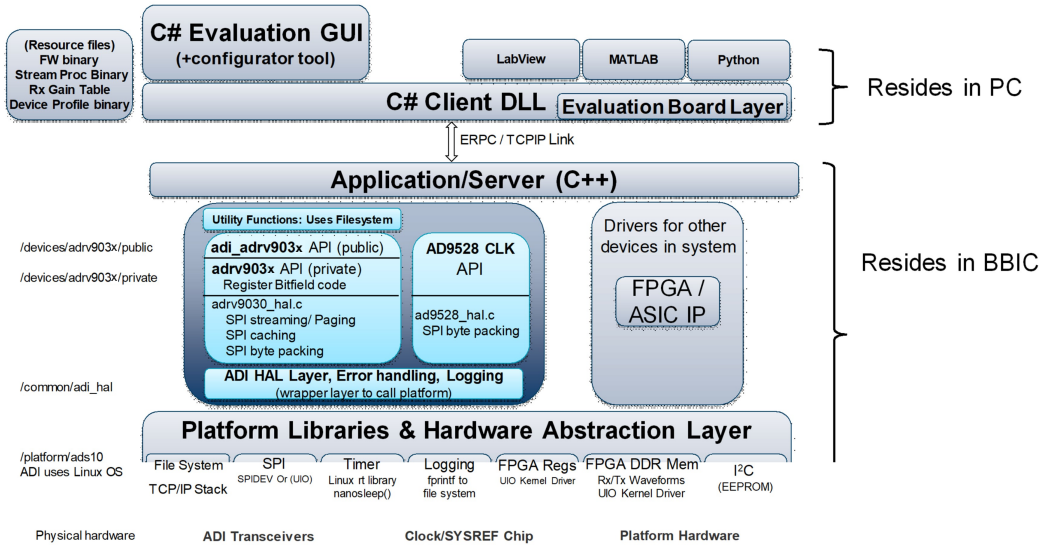


Figure 5. ADRV903x Based ADI Evaluation System Software Architecture

RESOURCE FILES

The following are the resources required for an ADRV903x software package:

- ADRV903x firmware binaries
- Stream co-processor binary
- Profile binary
- Rx gain tables

This section goes through the steps to generate resource files, which are essential for the ADRV903x to function. The resource files are programmed into the transceiver during the initialization phase. A brief description of the resource files are in Table 2. The resource files require approximately ~730 kB of memory.

Table 2. ADRV903x Platform Files

ADRV903x Platform File	Purpose	Generation Mechanism	Size
ADRV903x_FW.bin	The pre-compiled firmware binary for the embedded dual core ARM processors in the ADRV903x, which mainly consists of ADI proprietary algorithms used to calibrate the transceiver.	Delivered as part of the ADRV903x software package	641 KiloBytes total
stream_image.bin	The binary file for the stream co-processor in the ADRV903x, which is mainly used for setting up and managing the transmit/receive chains on occurrence of certain events, such as transmit/receive enable.	User generated with ADI evaluation software	88 KiloBytes
profile.bin	The ADRV903x configuration for a particular use case is programmed through the profile binary. The profile consists of the filter coefficients, clock rates, signal processing resources to enable/disable for a particular use case.	User generated with ADI evaluation software	3176 Bytes
RxGainTable.csv	The front end gain look up tables for the ADRV903x receiver.	Default table delivered as part of the ADRV903x software package. User can generate custom gain tables	Less than 10 KiloBytes

NOTE: The resource file versions must match with the API software version delivered in the same package. Using mismatched versions of resource files and the API is not supported.

SOFTWARE INTEGRATION

Firmware Binaries

The firmware for dual core embedded ARM processors in the ADRV903x is delivered in the form of pre-compiled binary files. The firmware in the ADRV903x mainly consists of ADI proprietary algorithms used for calibration, and drivers to access the ADRV903x hardware resources.

The firmware binaries are delivered in the software package under the folder firmware. It is required to program both the firmware binaries as part of the ADRV903x initialization. Refer to the programming section for information on initializing ADRV903x.

Stream Binary

The ADRV903x consists of a stream co-processor which helps setup/initialize the hardware on occurrence of certain events such as transmitter/receiver enable. The stream processor manages programming of registers across different hardware blocks in the processing chain and offers a simple command interface to perform the task of setting up the hardware on occurrence of certain events. There is one core stream processor and eighteen slice stream processors, one each for the eight Tx, Rx datapaths, and two for the ORx datapaths. The existence of individual slice stream processors for each datapath enables true real-time parallel operation of all unique Tx and Rx datapaths.

Please refer to the “Resource File Generation Using ACE” section of the ADRV903x Evaluation System User Guide for a procedure to generate a stream binary.

Profile Binary

The profile consists of the ADRV903x configuration generated for a particular use case in binary format. The profile consists of the filter coefficients, clock rates, signal processing resources to enable/disable in the transceiver for a particular use case, and the JESD configuration. The profile binary is programmed into the ADRV903x during initialization.

The user is required to supply the parameters listed in Table 3 as input to an ADI provided tool in order to generate a programmable profile binary. Refer to the Software Resource Files section in the Evaluation System User Guide document for information on how to generate these.

Table 3. User Provided Inputs to Generate an ADRV903x Profile Binary

Parameter	Description
Tx/Rx CHANNELS ENABLE	The ADRV903x consists of eight Tx, eight Rx and two ORx channels. The user can choose the Tx, Rx and ORx channels to initialize through this parameter. The Tx and Rx enables are 8-bit masks, while the ORx is a 2-bit mask.
ENABLE JESD 204C MODE	The ADRV903x supports JESD204B and JESD204C protocols for datapath interface with the baseband processor. The user can configure the part to use either JESD204B/C depending on the lane rates required.
TRANSMITTER CONFIGURATION	
Tx LO frequency	RF frequency setting of the Local Oscillator driving the transmitter channel
Tx LO frequency select	User can choose between LO1 and LO2 to drive the mixer on the transmit side
Tx Center frequency	The transmit RF carrier frequency
Tx Bandwidth	The primary signal bandwidth of the transmitter in which a carrier can be placed. The bandwidth value cannot exceed 80% of sampling rate at the JESD deframer input of the ADRV903x.
Tx Sampling Rate	Transmit signal IQ sample rate at the JESD deframer input of the ADRV903x
Tx Synthesis Bandwidth	The transmit side analysis bandwidth for DPD implemented in the baseband processor
Tx Synthesis Freq upper edge	Upper frequency edge of the synthesis bandwidth
Tx Synthesis Freq lower edge	Lower frequency edge of the synthesis bandwidth
ORx Sampling Rate	Observation receiver signal IQ sample rate at the JESD framer output of the ADRV903x
RECEIVER CONFIGURATION	
Rx LO frequency	RF frequency setting of the Local Oscillator driving the receiver channel
Rx LO frequency select	User can choose between LO1 and LO2 to drive the mixer on the receive side
Rx Center frequency – Band0	The receiver RF carrier frequency on Band0
Rx Bandwidth – Band0	Large signal receiver bandwidth on Band0
Rx Sampling Rate- Band0	The Rx IQ sample rate at the output of the JESD framer in the ADRV903x for Band0
Rx Center frequency – Band1	The receiver RF carrier frequency on Band1
Rx Bandwidth – Band1	Large signal receiver bandwidth on Band1
Rx Sampling Rate- Band1	The Rx IQ sample rate at the output of the JESD framer in the ADRV903x for Band1

SOFTWARE INTEGRATION

Table 3. User Provided Inputs to Generate an ADRV903x Profile Binary (Continued)

Parameter	Description
JESD DEFRAMER (Tx Side) CONFIGURATION	
Deframer Lane Xbar	Lane crossbar settings between the deserializer and the deframer in the Tx path.
DAC Sample Xbar	Sample crossbar settings between the deframer output and the DAC input.
M	Number of Digital to Analog converters on the transmitter side
Np	Digital to Analog converter sample bit-width
L	Number of input lanes at the ADRV903x de-serializer input
F	Number of DAC bytes per frame of data
K	Number of frames per multi-frame
S	Number of samples per clock per Digital to Analog Converter
E(JESD 204C)	Extended multi-block setting for JESD 204C use case, usually set to 1
Serializer Lanes Enabled	8-bit mask which indicates the serializer lanes enabled on the ADRV903x deframer input
JESD FRAMER(Rx Side) CONFIGURATION	
Framer Lane Xbar	Lane crossbars settings between the framer and the serializer in the Rx path
ADC Sample Xbar	Sample crossbar settings between the ADC output and framer input
M	Number of Analog to Digital converters on the receiver side
Np	Analog to Digital converter sample bit-width
L	Number of output lanes at the ADRV903x serializer output
F	Number of ADC bytes per frame of data
K	Number of frames per multi-frame
S	Number of samples per clock per Digital to Analog Converter
E(JESD204C)	Extended multi-block setting for JESD 204C use case, usually set to 1
Serializer Lanes Enabled	8-bit mask which indicates the serializer lanes enabled on theADRV903x framer input

Consider an example shown in the section below for a use case with 100 MHz Tx/Rx primary signal bandwidth, 245.76 MSPS sample rate for the transmitter data at the ADRV903x deframer input, and 245.76 MSPS sample rate for the receiver data at the ADRV903x framer output. In this example, the system supports a JESD lane rate of 9.8 Gbps, and JESD 204B is used as the link protocol for the baseband processor to the ADRV903x data interface. The example derives the parameters to input to an ADI provided tools to generate a programmable profile binary.

Transmitter Configuration for Profile Generation

In this example, we derive the transmitter side profile configuration for a 100 MHz carrier IBW signal transmitted on transmit channels Tx0−Tx3 at +50 MHz offset with respect to the LO, and on transmit channels Tx4−7 at −50 MHz offset with respect to the LO as shown in Figure 6.

SOFTWARE INTEGRATION

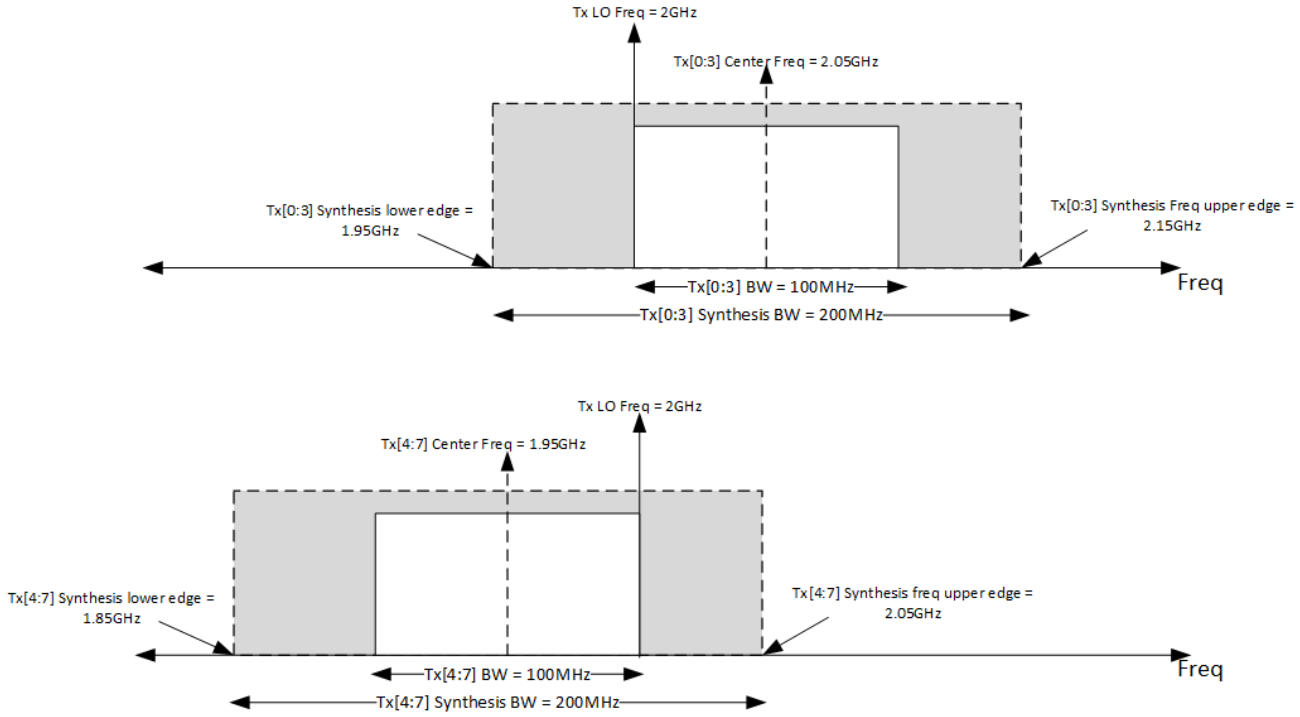


Figure 6. Example Transmit Carrier Configuration for Generating a Profile Binary

The Tx signal sample rate at the JESD deframer input is 245.76 MSPS in this example. The DPD analysis bandwidth is 200 MHz. This information is used in the transmitter configuration table (Table 4).

Table 4. Example Transmitter Configuration to Generate an ADRV903x Profile

TRANSMITTER PROFILE CONFIGURATION	
Tx LO frequency	2 GHz
Tx LO Select	LO1
Tx Channel Enable Mask	0xFF
TRANSMITTER CHANNELS[0:3] CONFIGURATION	
Tx[0:3] center frequency	2.05 GHz
Tx[0:3] Bandwidth	100 MHz
Tx[0:3] Sampling Rate	245.76 MSPS
Tx[0:3] Synthesis Bandwidth	200 MHz
Tx[0:3] Synthesis Bandwidth upper edge	2.15 GHz
Tx[0:3] Synthesis Bandwidth lower edge	1.95 GHz
TRANSMITTER CHANNELS[4:7] CONFIGURATION	
Tx[4:7] center frequency	1.95 GHz
Tx[4:7] Bandwidth	100 MHz
Tx[4:7] Sampling Rate	245.76 MSPS
Tx[4:7] Synthesis Bandwidth	200 MHz
Tx[4:7] Synthesis Bandwidth upper edge	2.05 GHz
Tx[4:7] Synthesis Bandwidth lower edge	1.85 GHz

To configure the JESD settings in the transmit path, assume that the system supports a lane rate of 9.8 Gbps in JESD 204B mode. The ADRV903x supports up to eight deserializer input lanes, and 16 bits per DAC sample (Np = 16). We have all eight transmit channels enabled in this example, therefore a total of 16 digital to analog converters (one converter per I sample and one converter per Q sample) are active (M = 16). For this example, assume an 8b/10b encoding scheme.

The number of deserializer lanes input to ADRV903x is calculated as:

SOFTWARE INTEGRATION

$$L = \frac{M \times S \times Np \times \left(\frac{10}{8}\right) \times TxSamplingRate}{\text{Lane Rate}} = \frac{16 \times 1 \times 16 \times \left(\frac{10}{8}\right) \times 245.76M}{9830.4M} = 8 \quad (1)$$

We can now proceed to calculate the number of DAC bytes per frame as follows:

$$F = \frac{M \times S \times Np}{8 \text{ x L}} = \frac{16 \times 1 \times 16}{8 \text{ x } 8} = 4 \quad (2)$$

The transmitter side JESD parameters for profile generation can be plugged into the configuration table (Table 5).

Table 5. JESD Deframer (Tx side) Profile Configuration Example

JESD DEFRAMER (Tx Side) CONFIGURATION	
Deframer Lane Xbar	Default (Please refer to the JESD204 Standard section)
DAC Sample Xbar	Default (Please refer to the JESD204 Standard section)
M	16
Np	16
L	8
F	4
K	32
S	1
Serializer Lanes Enabled	0xFF (SERDIN0-SERDIN7)

The JESD configuration for this profile is captured in Figure 7.

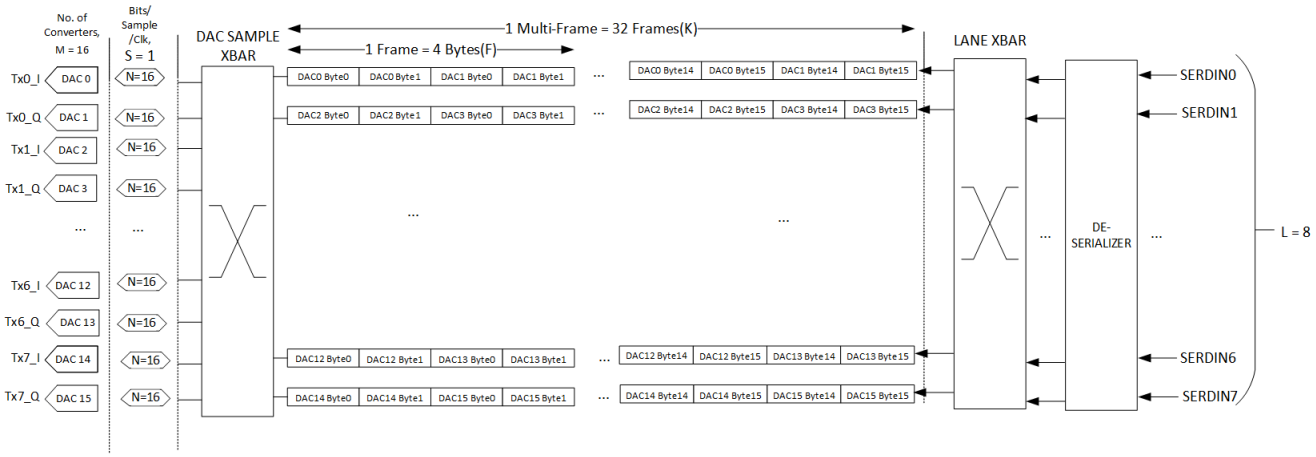


Figure 7. JESD Deframer(Tx Side) Configuration for the Example Profile

Receiver Configuration for Profile Generation

In this example, we derive the receiver side profile configuration for a 400 MHz carrier IBW signal in Figure 8 which has

- Band0 centered at 1.85 GHz with 100 MHz carrier bandwidth.
- Band1 centered at 2.15 GHz with 100 MHz carrier bandwidth.

The two bands are down-converted separately and serialized in the ADRV903x before sending it across to the baseband processor. The received signal output from the ADRV903x framer is sampled at 122.88 MSPS. These values are included for the receiver configuration in Table 6.

SOFTWARE INTEGRATION

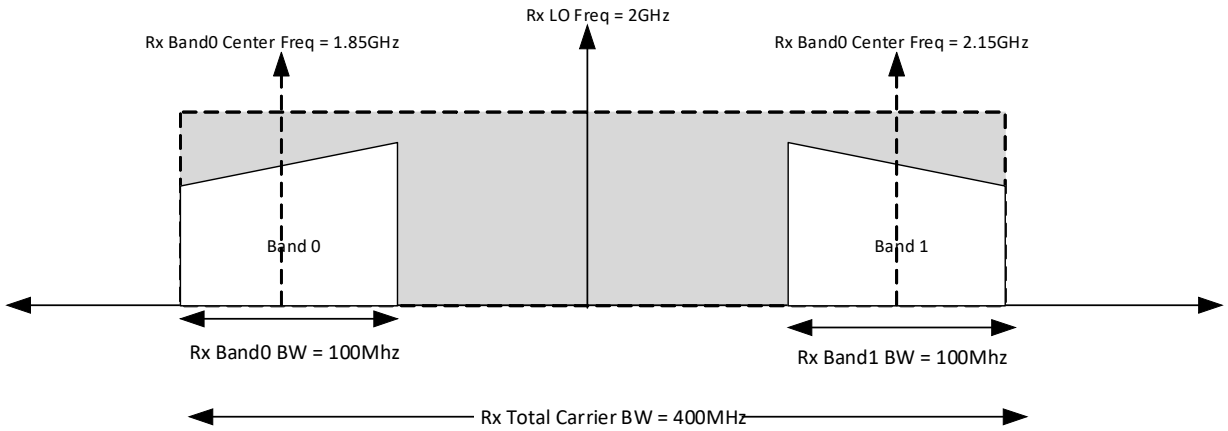


Figure 8. Wideband Receiver Carrier Configuration for an Example ADRV903x Profile Generation

Table 6. Receiver Configuration Example for the ADRV903x Profile Generation

RECEIVER CONFIGURATION	
Rx LO frequency	2 GHz
Rx LO frequency select	LO2
Rx Center frequency – Band0	1.85 GHz
Rx Bandwidth – Band0	100 MHz
Rx Sampling Rate- Band0	122.88 MSPS
Rx Center frequency – Band1	2.15 GHz
Rx Bandwidth – Band1	100 MHz
Rx Sampling Rate – Band1	122.88 MSPS

To configure the JESD settings in the receive path, assume that the system supports a lane rate of 9.8 Gbps at the serializer output lanes from the ADRV903x. The ADRV903x supports up to eight serializer output lanes, and 16 bits per ADC sample (Np = 16). We have all eight receiver channels enabled in this example, therefore a total of 16 Analog to Digital converters (one converter per I and one converter per Q sample) are active (M = 16). For this example, assume an 8b/10b encoding scheme.

There are two digital down converters present in the ADRV903x receiver path. In this case, since the two bands are digitally down converted individually in the ADRV903x receiver path before the data is framed, serialized and sent to the baseband through the ADRV903x framer and JRx module. For each individual band, the digital down converter in the ADRV903x outputs 16 channels of data corresponding to the 16 ADCs. The framer in the ADRV903x receives data from a total of 2 × 16 down converted channels corresponding to the two individual bands in this example for framing and serialization. Therefore, the number of converters (M) must be multiplied by a factor of two in this example since the framer receives data from two individual channels.

The number of serializer output lanes from the ADRV903x is calculated as:

$$L = \frac{NumBands\ x\ M\ \times\ S\ \times Np \times \left(\frac{10}{8}\right) \times RxSamplingRate}{\text{Lane Rate}}$$

$$\frac{2\ \times\ 16\ \times\ 1\ \times 16 \times \left(\frac{10}{8}\right) \times 122.88M}{9830.4\text{M}} = 8 \quad (3)$$

We can now proceed to calculate the number of ADC bytes per frame as follows:

$$F = \frac{M\ \times\ S\ \times Np}{8\ \text{x}\ \text{L}} = \frac{16\ \times\ 1\ \times 16}{8\ \text{x}\ 8} = 4 \quad (4)$$

The receiver side JESD parameters for profile generation can be plugged into the configuration table (Table 7).

Table 7. JESD Framer (Rx side) Configuration Example for the ADRV903x

JESD FRAMER (Rx Side) CONFIGURATION	
Framer Lane Xbar	Default (Please refer to the JESD204 Standard section)

SOFTWARE INTEGRATION

Table 7. JESD Framer (Rx side) Configuration Example for the ADRV903x (Continued)

DAC Sample Xbar	Default (Please refer to the JESD204 Standard section)
M	16
Np	16
L	8
F	4
K	32
S	1
Serializer Lanes Enabled	0xFF (SERDOUT0-SERDOUT7)

The JESD configuration for this profile is captured in Figure 9.

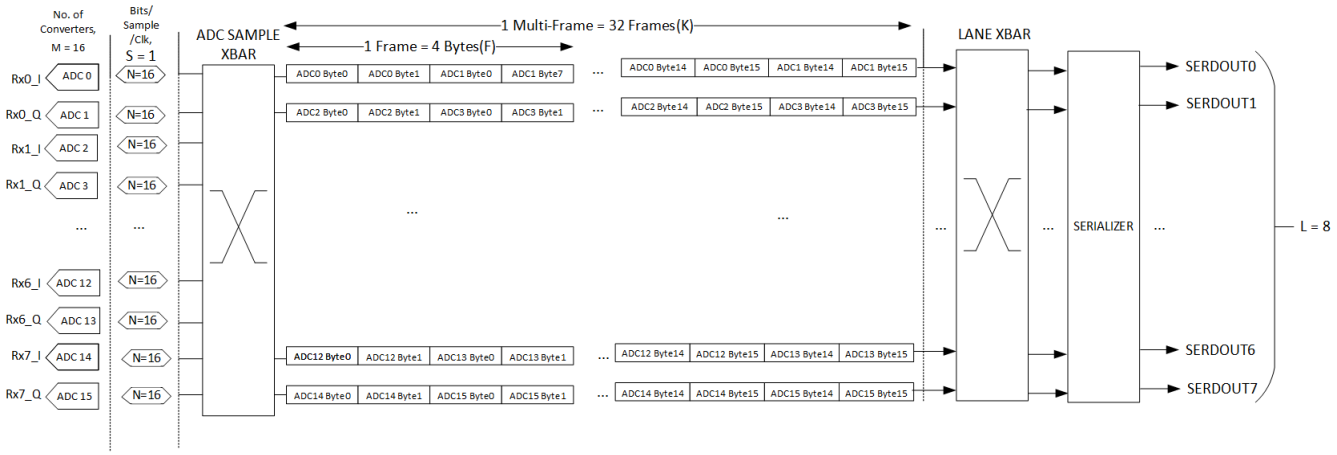


Figure 9. JESD Framer (Rx side) Configuration Example for the ADRV903x Profile Generation

Once the settings for the binary file have generated, these settings can be plugged into an ADI command line interface (CLI) tool to generate a programmable profile in a binary file format. The CLI tool is provided as part of the SW package.

Receiver Gain Table

The ADRV903x provides a 32 dB dynamic range for receiver gain control. The receiver gain is applied on the receiver front end through a programmable look up table programmed into the ADRV903x during initialization. The receiver gain can be controlled through an automatic gain control (AGC) or a manual gain control (MGC) mechanism by selecting an appropriate gain index. The receiver gain look up table consists of a maximum of 256 entries corresponding to 256 steps of resolution for gain change. Each row in the gain table provides a combination of front end attenuation and digital attenuation as shown in Figure 10.

Gain Index	FE Control Word	Ext Control	Phase Offset	Digital Gain
238	166	0	0	-1
239	161	0	0	0
240	155	0	0	0
241	149	0	0	0
242	142	0	0	0
243	135	0	0	0
244	127	0	0	-1
245	119	0	0	0
246	111	0	0	0
247	101	0	0	0
248	91	0	0	0
249	81	0	0	0
250	70	0	0	0
251	57	0	0	0
252	45	0	0	0
253	31	0	0	0
254	16	0	0	0
255	0	0	0	0

Figure 10. Example Rx Gain Table Entries

The front end attenuation is given by:

SOFTWARE INTEGRATION

$$Front\ end\ gain(dB) = 20\log_{10}\frac{256 - FE\ Control\ Word}{256} \quad (5)$$

ADI provides a default gain table in .csv format as part of the software package that can be programmed into the ADRV903x during initialization. The default gain table provides 0.5 dB gain steps per gain index and uses gain indices from 255 to 192 providing a total of 32 dB dynamic range.

The resource files (Firmware binary, Stream binary, Profile binary, Receiver gain tables) are provided as input to the API adi_ADRV903x_PreMcsInit( )while programming the device through the data structure adi_ADRV903x_TrxFileInfo_t defined in the file api/src/c_src/devices/ADRV903x/public/include/adi_ADRV903x_utilities_types.h. Please refer to the section Programming the Device for the ADRV903x programming sequence. Shown below is the listing.

```
typedef struct adi_ADRV903x_TrxFileInfo
{
      adi_ADRV903x_streamBinaryInfo_t     stream;
      /*!< Stream File Settings and Path to stream binary*/
      adi_ADRV903x_cpuBinaryInfo_t        cpu;
      /*!< CPU File Settings and Path to Firmware Binary */
      adi_ADRV903x_CpuProfileBinaryInfo_t cpuProfile;
      /*!< CPU Profile File Settings and Path to Profile Binary */
      adi_ADRV903x_RxGainTableInfo_t      rxGainTable[ADI_ADRV903X_RX_GAIN_TABLE_ARR_MAX];
      /*!< Rx Gain Table Settings */
} adi_ADRV903x_TrxFileInfo_t;
```

API INTEGRATION

The ADRV903x control commands are implemented through the ADRV903x API. This section explains the steps needed to integrate the ADRV903x API with a user application. The ADRV903x API architecture is in Figure 11.

SOFTWARE INTEGRATION

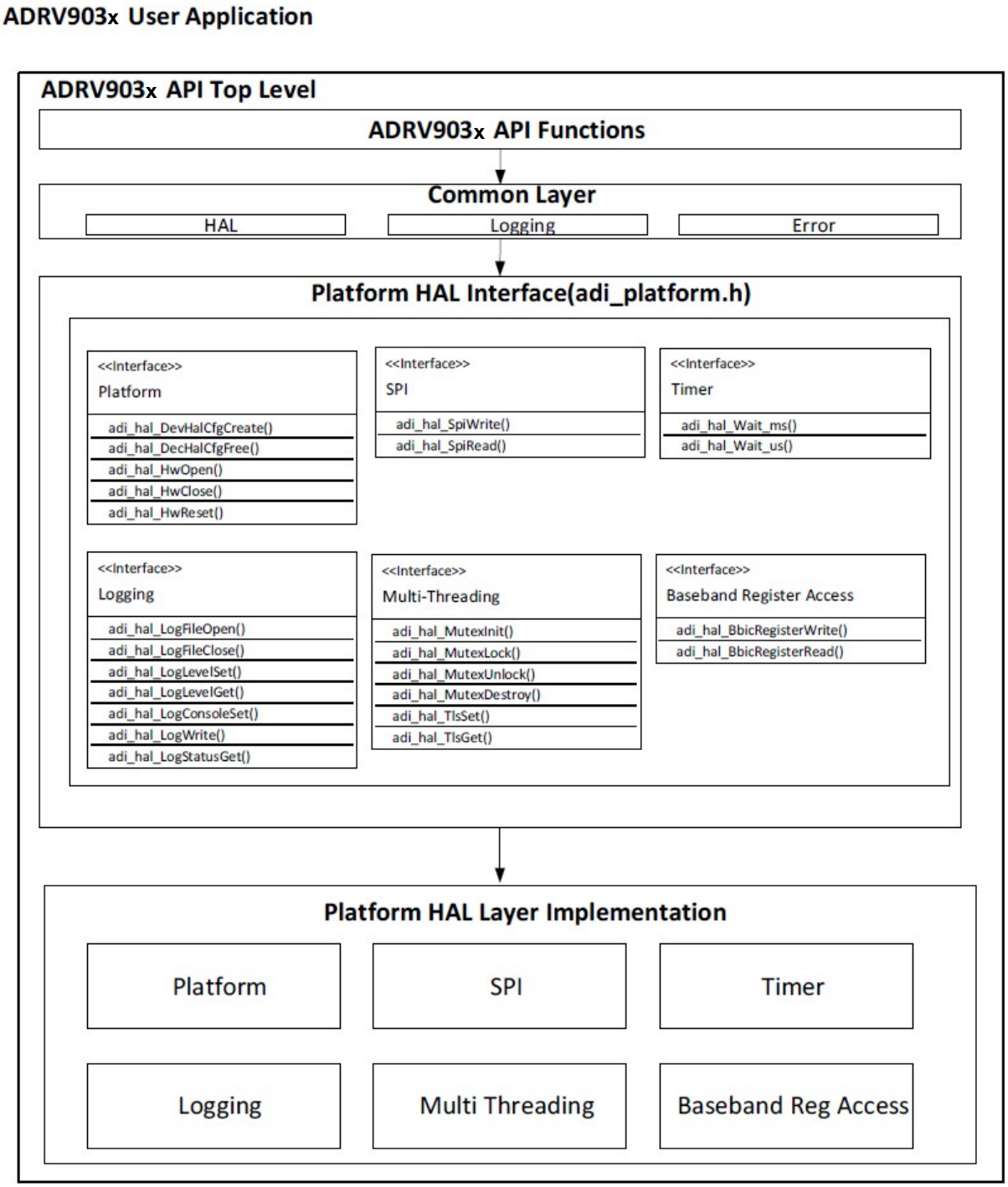


Figure 11. ADRV903x API Architecture

A brief description of each layer in the ADRV903x API is provided below.

- ADRV903x API Functions – This is the top-level interface to API functions called by the user application to configure and control the ADRV903x functionality.
- Common Layer – Service layer software functions such as hardware resource access, logging and error reporting are grouped into a common service layer. Platform specific implementations of hardware layer functions are invoked by the API through this layer.

SOFTWARE INTEGRATION

- Platform HAL Interface – Platform refers to a radio system that includes the baseband processor and the ADRV903x. Platform Hardware Abstraction Layer (HAL) refers to low level hardware abstraction layer/device drivers in the platform that are essential for the ADRV903x to function.

  An abstract interface of the platform/hardware specific functions (such as SPI drivers) are defined in the platform HAL Interface layer. The platform HAL interface defines function pointers that need to be assigned to user defined concrete implementations that are platform or hardware specific. Refer to the ADRV903x API Folder Structure section for an explanation on actions that a user needs to take in order to integrate the platform HAL interface.
- Platform HAL Implementation – This layer consists of concrete implementation of the platform HAL interface functions that are specific to a customer platform/hardware, such as SPI drivers, general purpose timers, logging, platform hardware initialization and baseband register access. These functions are assigned to the platform HAL interface function pointers during initialization.

API Integration Checklist

The following are the steps needed to implement the API into system software:

1. Discern file API file structure and copy the ADRV903x API files into a user application project
2. Implement platform HAL layer functions
3. Implement multi-threading HAL layer functions

ADRV903x API Folder Structure

The top-level folder structure of the API software delivered in the ADRV903x software package is in Figure 12. Each subfolder is explained in the following sections. ADI understands that the developer may desire to use a different folder structure. While ADI will provide API source code releases for ADRV903x family of devices in the folder structure shown below, the developer may organize the ADRV903x API into a custom folder organization if required. Modifying the content of each ADRV903x API source file prevents easy updates to future ADRV903x API releases.

SOFTWARE INTEGRATION

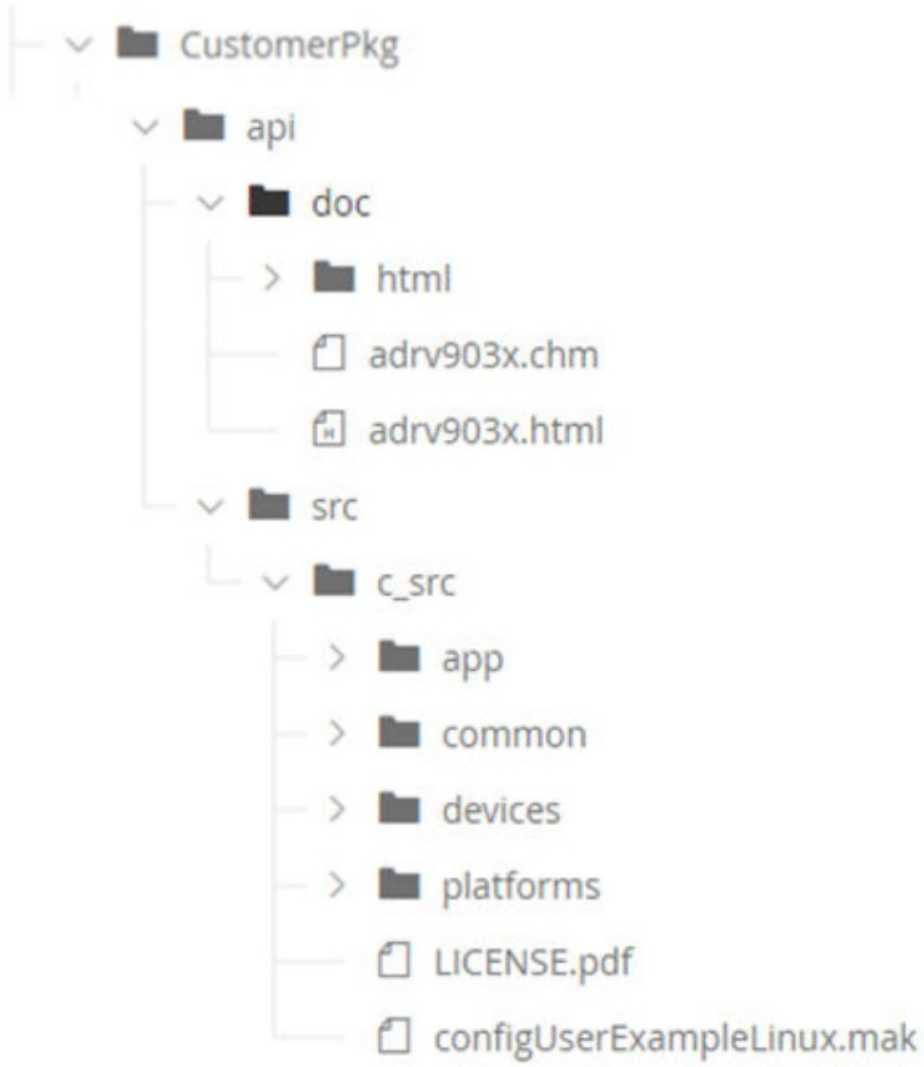


Figure 12. ADRV903x API Folder Structure

The user can follow the same folder structure described above to integrate the API into a user application. Shown below is the ADI evaluation system project where the API source code is integrated into the ADI Evaluation System software project. The API folder structure is marked in red in Figure 13.

SOFTWARE INTEGRATION

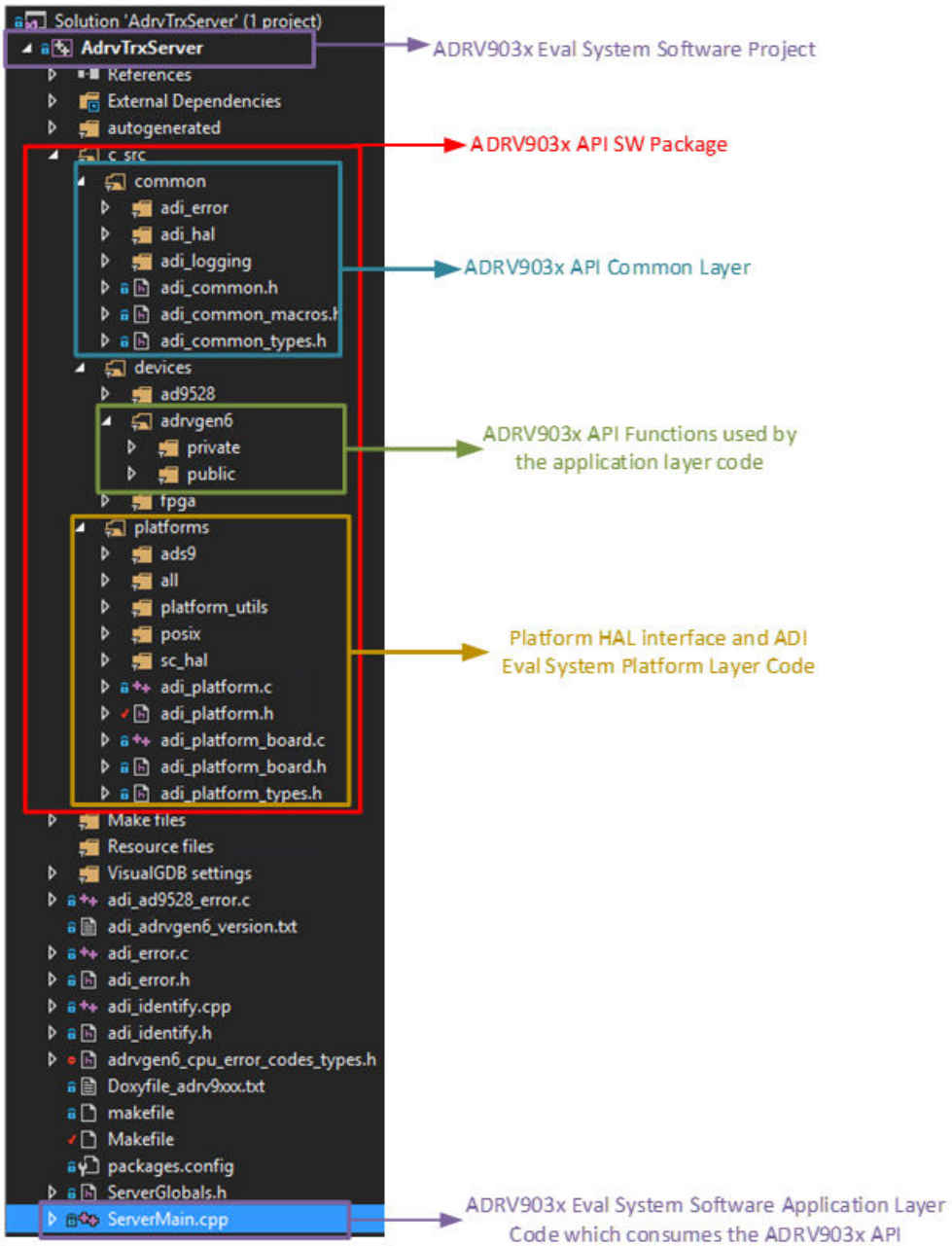


Figure 13. ADRV903x Based ADI Evaluation Software Project Integrated with the ADRV903x API

Devices Directory − Adi.ADRV903x.CustomerPkg/public/api/src/c_src/devices

The devices directory in the API folder structure contains the ADRV903x API that can be called by the user application for configuring and controlling the ADRV903x. An expanded view of the ADRV903x API folder is shown in Figure 14.

SOFTWARE INTEGRATION

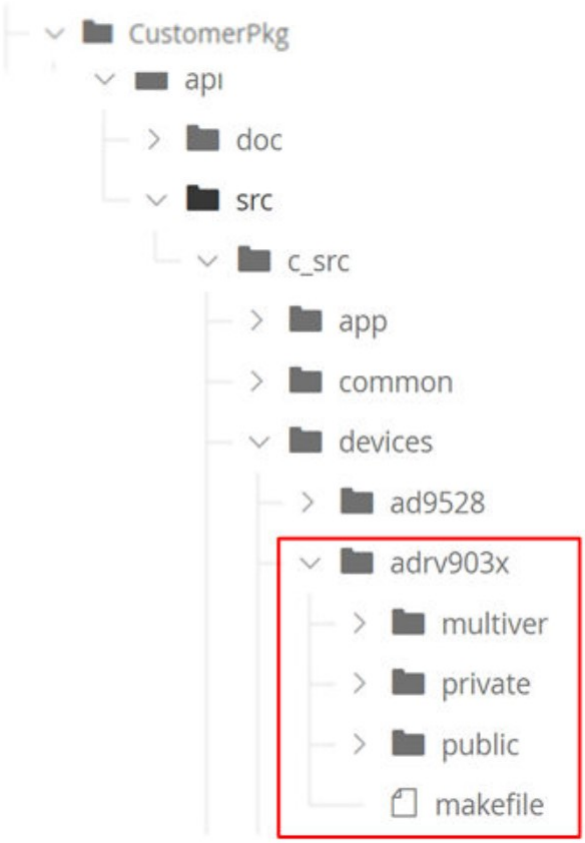


Figure 14. ADRV903x Device API Folder Structure Expanded View

The /include folder inside private and public folders consists of C-API function prototypes in the adi_ADRV903x_<API Files>.h header files, and type definitions consisting of user defined structures and enumerations in adi_ADRV903x_<API File>_types.h files. The /src folder consists of the API function definitions of the prototypes declared in /include/adi_ADRV903x_<API File>.h header files.

The API files are logically partitioned, each file corresponds to one specific functionality of the ADRV903x. Table 8 consists of the list of API files and their corresponding functionality.

Table 8. List of Top Level ADRV903x API Files (Files with Functions) Delivered in the ADRV903x Software Package

API File	Description
adi_ADRV903x_cals.h	API functions that control the ADRV903x calibrations. The user can enable/disable and retrieve calibration status through the API functions provided in this file.
adi_ADRV903x_core.h	API functions used to initialize the ADRV903x.
adi_ADRV903x_cpu.h	Low level API Functions for the dual core embedded ARM processor in the ADRV903x that are consumed by other API functions.
adi_ADRV903x_error.h	Error reporting functions for the ADRV903x.
adi_ADRV903x_hal.h	API functions that cover the ADRV903x SPI interface features.
adi_ADRV903x_radioctrl.h	API functions for the ADRV903x radio control functions such as transmit/receive enable and LO frequency setting.
adi_ADRV903x_rx.h	API functions to control the ADRV903x receiver chain functionality such as gain control and formatter.
adi_ADRV903x_tx.h	API functions to control the ADRV903x transmitter chain functionality such as attenuation control and PA protection.
adi_ADRV903x_utilities.h	Utility API functions that can be used to program the ADRV903x.
adi_ADRV903x_user.h	Compile time constants/macros for the ADRV903x API.
adi_ADRV903x_agc.h	API functions for the ADRV903x Automatic Gain Control functionality
adi_ADRV903x_datainterface.h	API functions to control the ADRV903x JESD data interface
adi_ADRV903x_gpio.h	API functions to setup GPIO pins on the ADRV903x. This also includes the GP Interrupt functions that provide the ADRV903x diagnostic data to the user.

SOFTWARE INTEGRATION

NOTE: The user is strictly forbidden from modifying the contents in the devices folder. The only file that a user can edit is the adi_ADRV903x_user.h which contains compile time macros to adjust intended functionality of certain features, such as timeout values for commands.

Common Layer − Adi.ADRV903x.CustomerPkg/api/src/c_src/common

The /c_src/common/ folder contains common layer functions highlighted in the API architecture (Figure 11). The common layer functions are the set of service layer calls used by all API functions to abstract away low level functions such as Hardware Abstraction Layer calls. These service layer functions include error reporting, logging and hardware abstraction layer access. An expanded view of the common layer code delivered as part of the ADRV903x API package is shown in Figure 15.

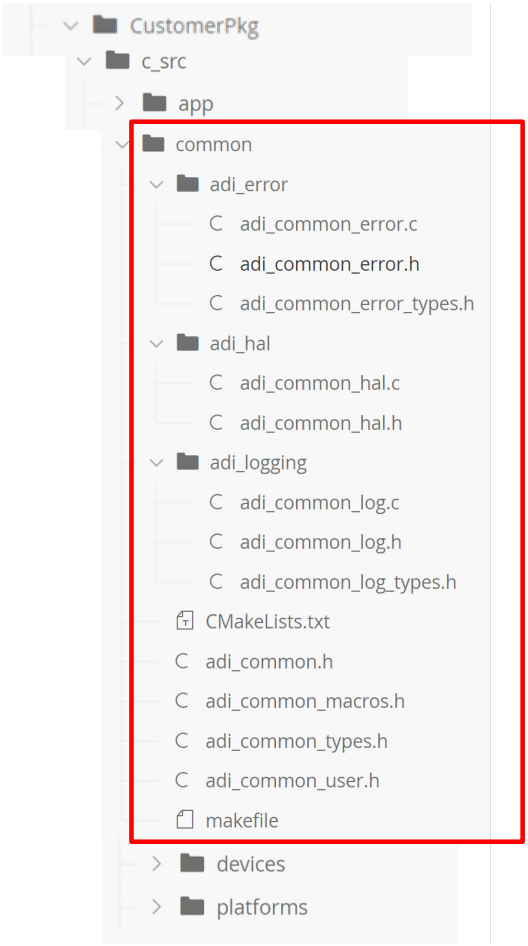


Figure 15. Expanded View of the ADRV903x Common Layer Code Delivered in ADRV903x Software Package

Common Layer Logging Functions

The API provides a simple logging feature function that may be enabled for debugging purposes. The logging functions are contained in /common/adi_logging folder. This feature requires a user implementation of the adi_hal_LogWrite interface function in the platform HAL implementation. The adi_hal_LogWrite() platform layer function can found under api/src/c_src/platfrom/adi_platform.h. Directions to implement the ADI platform HAL functions are described in the next section. The APIs will optionally call to send debug information to the system via the HAL. The function adi_hal_LogLevelSet may be used to configure HAL flags to configure how the HAL processes the various message types from the API layer. ADI transceiver open-hardware function adi_hal_HwOpen will call this function to set the desired logging level. Available logging levels are given by the enum adi_common_LogLevel_e defined in /common/adi_common_log_types.h as shown in Table 9.

SOFTWARE INTEGRATION

Table 9. ADRV903x Logging Levels

adi_common_LogLevel_e Enum value	Description
ADI_COMMON_LOG_NONE	All types of log messages not selected
ADI_COMMON_LOG_MSG	Log message type
ADI_COMMON_LOG_WARN	Warning message type
ADI_COMMON_LOG_ERR	Error message type
ADI_COMMON_LOG_API	API function entry for logging purposes
ADI_COMMON_LOG_API_PRIV	Private API function entry for logging purposes
ADI_COMMON_LOG_BF	BF function entry for logging purposes
ADI_COMMON_LOG_HAL	ADI HAL function entry for logging purposes
ADI_COMMON_LOG_SPI	SPI transaction type
ADI_COMMON_LOG_ALL	All types of log messages selected

The hierarchy of logging function calls are shown in Figure 16. The logging is initiated in the API, and it propagates through the common layer and ultimately gets logged through a user implemented **adi_hal_LogWrite()** function.

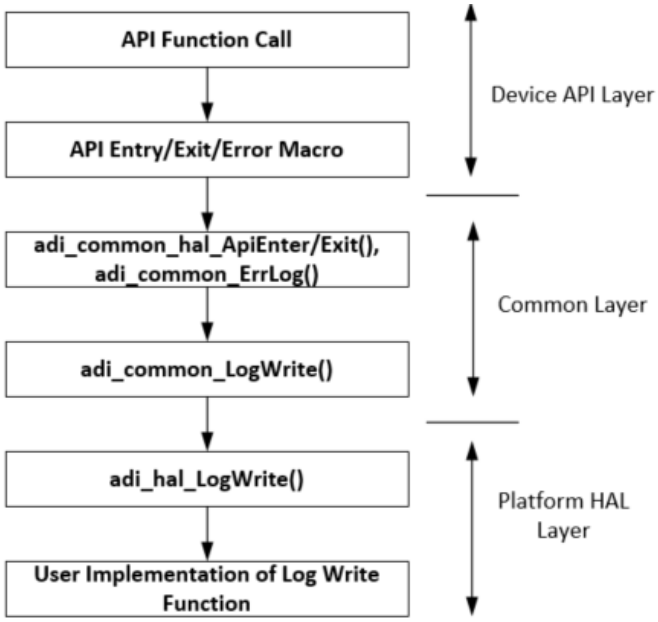


Figure 16. Hierarchy of the ADRV903x Logging Function Calls

An illustration of logging function being invoked in the top level API function is shown in Figure 17. In this illustration, the RxGainSet() API calls the macro ADI_API_ENTER_RTN() to log the function entry, and ADI_ERROR_REPORT() macro to log an error. The common layer logging functions are invoked through the logging macros, and the log message is finally propagated to the user implemented HAL layer logging function.

SOFTWARE INTEGRATION

```
_API adi_adrv903x_ErrAction_e adi_adrv903x_RxTxEnableSet(adi_adrv903x_Device_t* const   device,
                                                         const uint32_t                orxChannelMask,
                                                         const uint32_t                orxChannelEnable,
                                                         const uint32_t                rxChannelMask,
                                                         const uint32_t                rxChannelEnable,
                                                         const uint32_t                txChannelMask,
                                                         const uint32_t                txChannelEnable)


 adi_adrv903x_ErrAction_e recoveryAction = ADI_ADRV903X_ERR_ACT_CHECK_PARAM;

 ADI_ADRV903X_NULL_DEVICE_PTR_RETURN(device);
 ADI_ADRV903X_API_ENTRY(&device->common);

 ADI_ADRV903X_RADIOCTRL_RANGE_CHECK > 0
 recoveryAction = adrv903x_RxTxEnableSetRangeCheck(device, orxChannelMask, orxChannelEnable, rxChannelMask, rxChannelEnable, txChannelMask, txChannelEnable);
 if (recoveryAction != ADI_ADRV903X_ERR_ACT_NONE)
 {
     ADI_API_ERROR_REPORT(&device->common, recoveryAction, "RxTxEnableSetRangeCheck Issue");
     goto cleanup;
 }
dif

 /*Enable requested Rx Channel signal chains*/
 recoveryAction = adrv903x_RxEnableSet(device, rxChannelMask, rxChannelEnable);
 if (recoveryAction != ADI_ADRV903X_ERR_ACT_NONE)
 {
     ADI_API_ERROR_REPORT(&device->common, recoveryAction, "RxEnableSet Issue");
     goto cleanup;
 }

 /* Enable requested ORx Channel signal chains */
 recoveryAction = adrv903x_OrxEnableSet(device, orxChannelMask, orxChannelEnable);
 if (recoveryAction != ADI_ADRV903X_ERR_ACT_NONE)
 {
     ADI_API_ERROR_REPORT(&device->common, recoveryAction, "OrxEnableSet Issue");
     goto cleanup;
 }

 /* Enable requested Tx Channel signal chains */
 recoveryAction = adrv903x_TxEnableSet(device, txChannelMask, txChannelEnable);
 if (recoveryAction != ADI_ADRV903X_ERR_ACT_NONE)
 {
     ADI_API_ERROR_REPORT(&device->common, recoveryAction, "TxEnableSet Issue");
     goto cleanup;
 }

anup:
 ADI_ADRV903X_API_EXIT(&device->common, recoveryAction);
```

Figure 17. Illustration of Logging Functions Invoked from the ADRV903x Top Level API

Error Handling

The error handling functions are found in the /common/adi_error folder of the ADRV903x API software package. Each ADRV903x API function returns an enum adi_ADRV903x_ErrorAction_e value representing a recovery action. The user is expected to handle the errors returned from the API in the application layer code. Error handling flow in a user application can be divided into two phases

- Development/Debug phase: In this phase, we would expect the user to debug errors through manual intervention by taking advantage of the Error Handbook provided by ADI as part of the software package and other tools such as logging to develop error handling code in the application software.
- Deployment phase: In this phase, the error handling code is built into the application software that is production ready and the errors occurring during runtime are handled programmatically.

The general strategy to handle errors during the development and deployment phases is shown in Figure 18.

SOFTWARE INTEGRATION

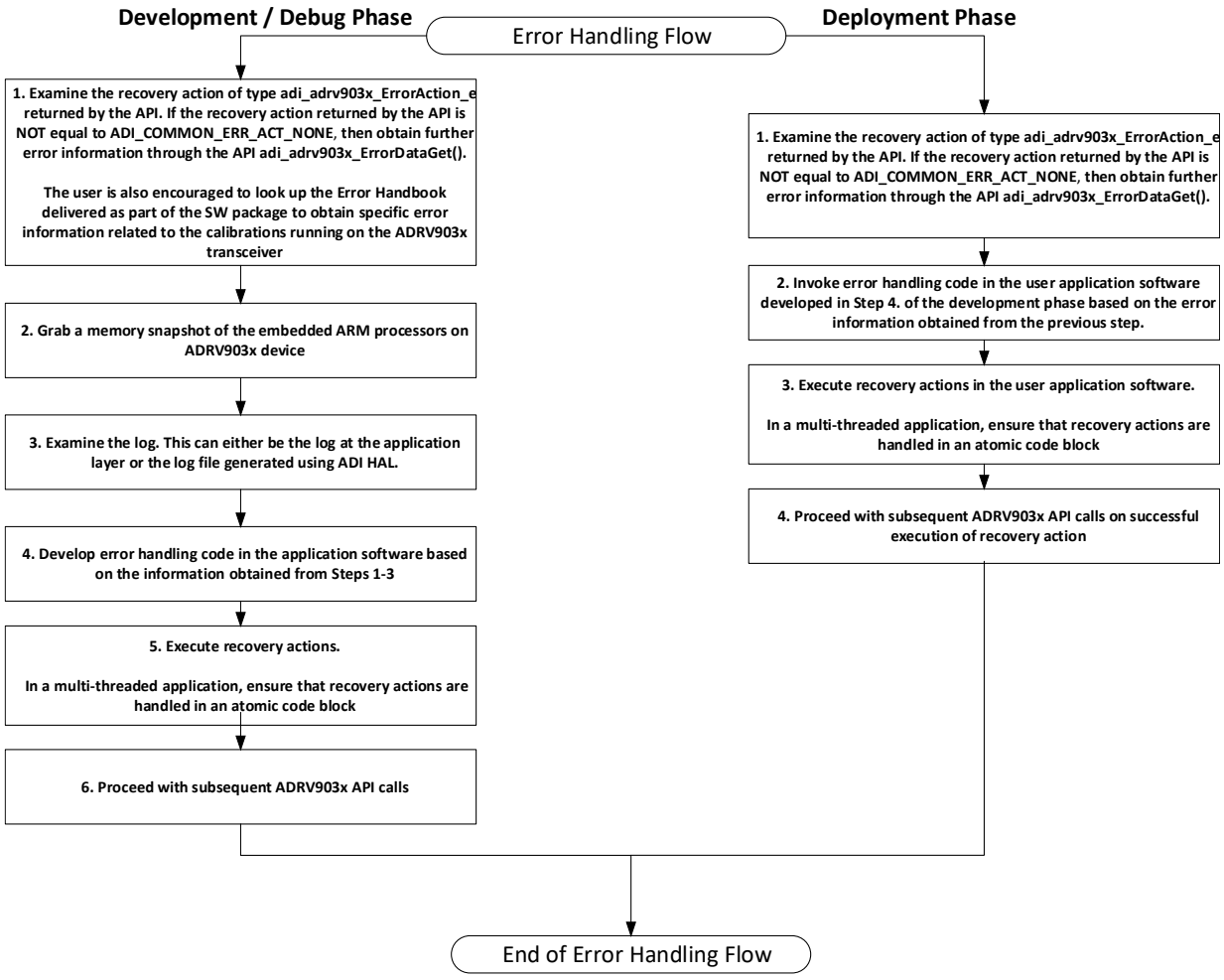


Figure 18. General Error Handling Strategy for ADRV903x

The set of error recovery actions returned by each API are in Table 10. Each ADRV903x API function call responds to the application layer with information on action that needs to be taken due to a possible error in the API function call. The error structure will contain further information in order to take the adequate action.

Table 10. ADRV903x API Error Recovery Actions

Recovery Action Name	Value	Description
ADI_COMMON_ERR_ACT_NONE	0	API function completed successfully – no error handling action is required.
ADI_COMMON_ERR_ACT_CHECK_PARAM	−1	API OK – Invalid parameter detected in API
ADI_COMMON_ERR_ACT_OPEN_DEVICE	−10	API OK – Device Not Open
ADI_COMMON_ERR_ACT_CHECK_INTERFACE	−100	API OK – Interface is reporting an error (SPI/timer/Data Interface)
ADI_COMMON_ERR_ACT_CHECK_FEATURE	−200	API OK – Feature is reporting an error. Feature refers to a logical partition of ADRV903x functionality such as GPIO, PA Protection, GP Interrupt, gain control etc.
ADI_COMMON_ERR_ACT_RESET_INTERFACE	−300	API NOT OK – Interface Not Working, device cannot be program or access timer/I2C/SPI/Data interface
ADI_COMMON_ERR_ACT_RESET_FEATURE	−400	API NOT OK – Reset device feature (for example ARM init cals)
ADI_COMMON_ERR_ACT_RESET_DEVICE	−500	API NOT OK – Full system Reset required

The API error structure is accessed via adi_ADRV903x_Device_t->adi_common_Device_t-> adi_common_ErrData_t data structure. The adi_common_ErrData_t consists of a stack of error frames (adi_common_ErrFrame_t) that can be traced to the lowest level function in

SOFTWARE INTEGRATION

which the error was first encountered. An example context diagram of an error stack frame is shown in Figure 19. In this example, an error encountered in the ADRV903x bitfield access low level function is propagated to the top level API function call, and the error from all three levels of function calls are captured in the error stack frame accessible to the user via device data structure (adi_ADRV903x_Device_t).

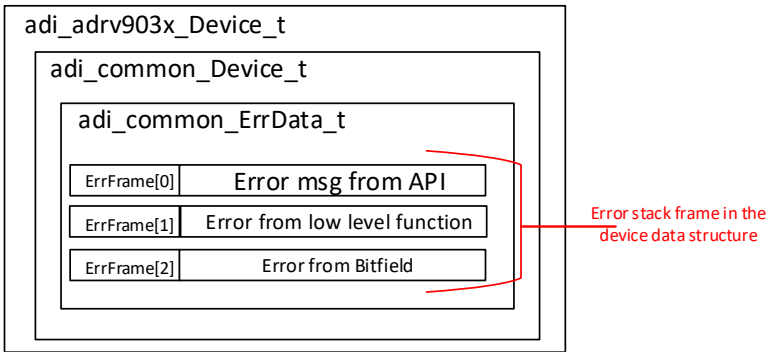


Figure 19. Example Context Diagram of Error Stack Frame Returned from the ADRV903x

Each error frame in the error stack trace contains members described below that can be used to narrow the action to be taken.

- errSrc: Current source of error detected, indicating the source file where the error.
- errCode: Current error code.
- line: Line of the source code where the error was returned.
- function: Function name where the error occurred.
- file: File name where the error occurred.
- varName: Variable name which has the error.
- varData: Variable data which has the error.
- errMsg: Error message to describe the error.
- action: Recovery action.

Debug Support

Error Handbook

ADI delivers a firmware error handbook as part of the software package under the firmware folder. The error handbook contains descriptions of all possible error codes reported by the ADRV903x firmware and is intended to be used as reference documentation during development/debug phase.

The entries in the firmware error handbook are organized as shown in Figure 20 where the user can look up the description and recovery action for each error code. The error table contained in the error handbook documentation can also be accessed through the API function adi_ADRV903x_ErrorDataGet() contained in the file /api/src/c_src/devices/ADRV903x/public/include/adi_ADRV903x_error.h. The API adi_ADRV903x_ErrorDataGet() is a service function to lookup the error table entry for handling error programmatically.

SOFTWARE INTEGRATION

Error Code	Description	Error Cause	Recovery Action
0x0000	No error		
0x0100	DCOFFSET CALIBRATION		
0x0101	DCOffset: Calibration was aborted while collecting data		ADI_COMMON_ERR_ACT_RERUN_FEATURE
0x0102	DCOffset: Calibration Timed out due to error in data capture	Hardware was unable to capture enough data within time limit	ADI_COMMON_ERR_ACT_SOFT_RESET

Figure 20. Error Handbook Snapshot

Each error code entry in the error handbook can have multiple recovery actions and causes associated with it. The recovery actions are organized in the descending order of priority. For example, the entry corresponding to error code 0x0103 has two recovery actions and causes associated with it as shown in Figure 21. The highest priority must be given for handling the recovery action ADI_COMMON_ERR_ACT_SOFT_RESET followed by ADI_COMMON_ERR_ACT_RESET_DEVICE. The two error causes associated with error code 0x0103 are also arranged in descending order of their likelihood.

0x0103	DCOffset: Internal Test to verify calibration failed due to error in data capture	FSC was unable to capture data require to confirm calibration completion	ADI_COMMON_ERR_ACT_SOFT_RESET
		FSC data capture was aborted	ADI_COMMON_ERR_ACT_RESET_DEVICE

Figure 21. Example Entry in the Error Handbook with Multiple Recovery Actions Associated with It

Debugging Commonly Occurring Recovery Actions

The following section describes some debug steps that a user can take on occurrence of errors described in Table 10.

API Recovery Action: ADI_COMMON_ACT_NO_ACTION

The ADI_COMMON_ACT_NO_ACTION API recovery action is returned when an ADRV903x API function completes successfully. There is no recovery action to be performed by the calling function in the application layer.

API Recovery Action: ADI_COMMON_ACT_ERR_CHECK_PARAM

The ADI_COMMON_ACT_ERR_CHECK_PARAM API recovery action is returned if an ADRV903x API parameter range check or NULL parameter check failed. If this recovery action is returned, the ADRV903x API function did not complete. It is expected that this recovery action would only be found during the debug phase of development. During application software development, this recovery action informs the developer to double check the value passed into the ADRV903x API function parameters. Once the parameters are corrected to be in the valid range, or NULL pointers are corrected, recalling the function should allow the ADRV903x API function to complete.

For debug, the developer may access further information located in the error structure, like error code, file name, function name or variable name, a message is stored in the error message variable describing the error in more detail.

If the application SW passes development test but this recovery action is returned in the field, a bug in the application layer is highly possible causing an out of range or NULL pointer error.

SOFTWARE INTEGRATION

API Recovery Action: ADI_COMMON_ACT_ERR_RESET_INTERFACE

The ADI_COMMON_ACT_ERR_RESET_INTERFACE API recovery action is returned if the ADIHAL layer reports a HAL error while attempting a SPI read or write transaction. If the ADIHAL function returns a timeout error due to SPI hardware being busy or used by another thread, the ADRV903x API will attempt to retry the SPI operation once. If the SPI transaction fails again, the ADRV903x API reports this recovery action. This action is also returned if an ADIHAL error is returned due to inability to access the driver.

Suggested application layer action:

- Call to determine the specific ADIHAL error code and verify that ADIHAL is the error source.
- Log error code and source.
- If the ADIHAL error is a timeout, the ADRV903x API function may be retried.
- If the ADIHAL error is not a timeout, application should try resetting the SPI driver and retrying the function call.
- If recovery action persists, verify SPI communication with other SPI devices and assess the need for a BBIC system reset.

If an ADRV903x API function has detected a condition that only the BBIC can determine if it is a true error or not. An example would be a data interface error counter threshold overflow. If a data interface counter were to overflow once an hour or once a month, only the BBIC would be able to determine if the counter overflow constituted an actual error condition.

Suggested application layer action:

- Record the error.
- Perform any BBIC determined recovery actions.

If an ADRV903x API function has detected a condition in where the data interface (JESD) then further information can be attained by the error structure.

Suggested application layer action:

- Record the error.
- Perform any BBIC determined recovery actions to reset the data interface.

API Recovery Action: ADI_COMMON_ACT_ERR_RESET_FEATURE

The ADI_COMMON_ACT_ERR_RESET_FEATURE API recovery action is returned by the API when an error has been detected that requires the reset of a feature of the device. As well as resetting the feature it must also be reconfigured to the state needed.

API Recovery Action: ADI_COMMON_ACT_ERR_RESET_MODULE

The ADI_COMMON_ACT_ERR_RESET_MODULE API recovery action is returned if the ADRV903x API detects an issue any of the modules:

ARM processor module, that requires a complete reset and reload of the ARM firmware. This type of action might be required if the communication interface to the ADRV903x ARM processor fails or the ARM watchdog timer reports an error. These events are not expected in production code but are failsafe mechanisms in the event of a catastrophic error.

- Issue adi_ADRV903x_RxTxEnableSet() to disable transmitter to keep hardware in a safe state. If this fails, a full ADRV903x reset is required.
- Set PA and other RF front-end components in powered down / init state.
- Call adi_common_ErrorInfoGet() to determine the specific error code and verify the error source. Log error code and source.
- Dump ADRV903x ARM memory if necessary for debug.
- Dump ADRV903x SPI registers if necessary for debug.
- Reload the ADRV903x stream processor and ARM binary firmware files.
- Continue with normal init sequence to run init calibrations and enable tracking calibrations.

Platform Layer − Adi.Adrv903x.CustomerPkg/public/api/src/c_src/platforms

The folder structure of ADRV903x platform layer code is shown in Figure 22. Here, the platform refers to the radio system which includes the baseband processor and the ADRV903x. The platform Hardware Abstraction Layer (HAL) refers to low level device drivers in the baseband processor essential for the transceiver API to function.

SOFTWARE INTEGRATION

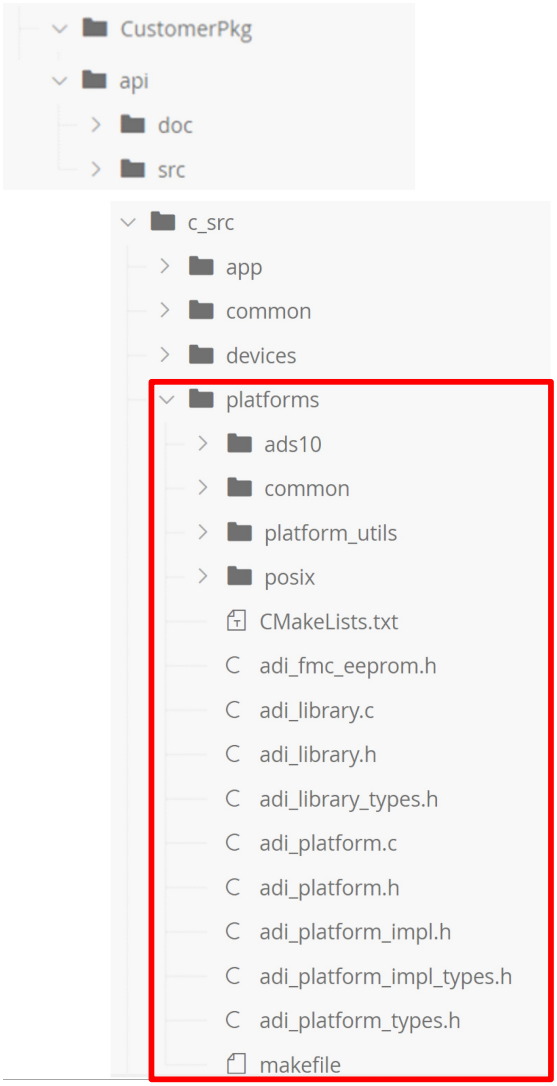


Figure 22. Expanded View of the ADRV903x Platform Layer Code Delivered in ADRV903x Software Package

The user is required to take the following actions to integrate the Platform Hardware Abstraction Layer with the ADRV903x API.

1. Examining the Platform HAL interface used by ADRV903x API
2. Configuring the Platform HAL interface
3. Implementing the Platform HAL interface

The following sections will explain the three steps mentioned above.

Platform HAL Interface

The platform HAL interface is a set of abstract hardware interface functions defined in /platforms/adi_platform.h that is accessed by the common layer code to provide services such as logging, platform SPI access, and timer to the ADRV903x API functions.

The platform HAL interface is implemented as C function pointers, that can be initialized with concrete implementations of the functions that are specific to a platform. The platform HAL interface function pointers shipped as part of the ADRV903x software package are shown in Figure 23.

SOFTWARE INTEGRATION

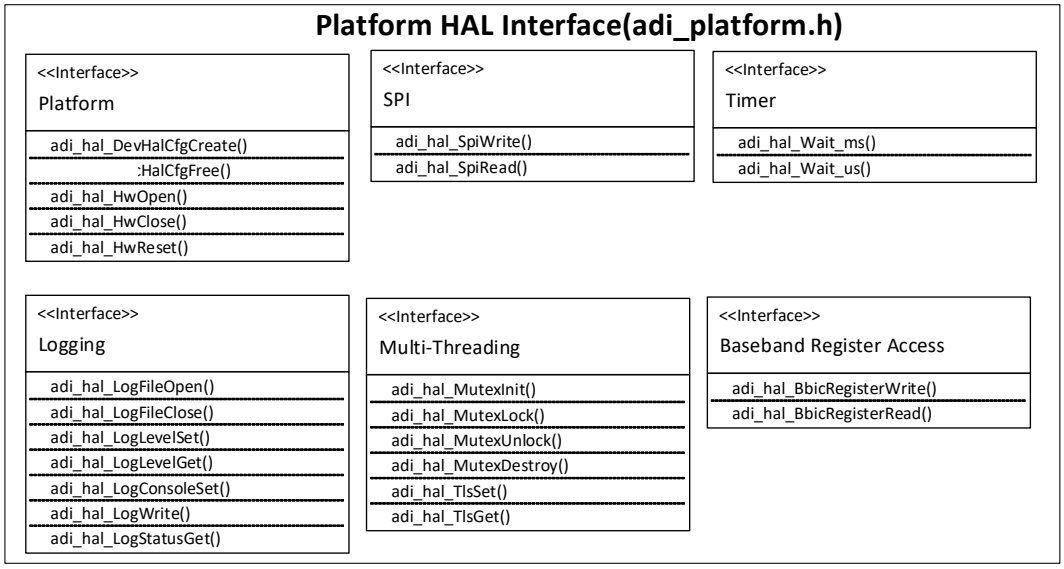


Figure 23. ADRV903x Platform HAL Interface Functions

A brief description of the platform HAL interface functions is provided in Table 11.

Table 11. ADRV903x Platform HAL Interface Function Description

Function Name	Purpose
adi_hal_HwOpen()	Open and initialize all platform drivers/resources and peripherals required to control the ADRV903x (SPI, timer, logging, etc.)
adi_hal_HwClose()	Close any resources opened by adi_hal_HwOpen
adi_hal_HwReset()	Toggle the hardware reset signal for the ADRV903x
adi_hal_SpiWrite()	Write an array of data bytes on a targeted SPI device (Address bytes are packed into the byte array before calling this function)
adi_hal_SpiRead()	Read an array of data bytes from a targeted SPI device. (Address bytes are provided by a txData array which are packed into the byte array before calling this function).
adi_hal_Wait_us()	Perform a wait/thread sleep in units of microseconds
adi_hal_Wait_ms()	Perform a wait/thread sleep in units of milliseconds
adi_hal_LogFileOpen()	Open a file for logging
adi_hal_LogLevelSet()	Mask to set the severity of information to write to the log (Error/Warning/Message)
adi_hal_LogLevelGet()	Get the current log level setting
adi_hal_LogWrite()	Log a debug message (message, warning, error) from the API to the platform log
adi_hal_LogFileClose()	Function to close the log file
adi_hal_DevHalCfgCreate()	This function allows the platform to allocate and configure the devHalCfg structure. The devHalCfg structure is described in the next section.
adi_hal_DevHalCfgFree()	This function allows the platform to deallocate the devHalCfg structure
adi_hal_BbicRegisterRead() adi_hal_BbicRegisterWrite() adi_hal_BbicRegistersRead() adi_hal_BbicRegistersWrite()	These functions are used to communicate with the Baseband processor (typically FPGA).
adi_hal_MutexInit()	Initializes the mutex into an unlocked state for multithreading applications
adi_hal_MutexDestroy()	Signal to the HAL that a mutex is no longer required.
adi_hal_MutexLock()	Acquires the mutex for a shared resource in a multithreaded application.
adi_hal_MutexUnlock()	Releases a previously acquired mutex in a multithreaded application
adi_hal_TlsSet()	Stores a value in the calling thread's thread local storage slot for the specified index

SOFTWARE INTEGRATION

Table 11. ADRV903x Platform HAL Interface Function Description (Continued)

Function Name	Purpose
adi_hal_TlsGet()	Retrieves the value in the calling thread's thread local storage slot for the specified index.

Configuring the Platform HAL

The platform HAL configuration is contained in adi_hal_Cfg_t data structure defined in /platform/adi_platform_types.h. The definition of platform HAL configuration data structure for the ADI evaluation platform is shown below. All the substructures used in adi_hal_Cfg_t are defined in /platform/adi_platform_types.h.

```
typedef struct adi_hal_Cfg
{
       uint32_t               interfacemask; /*!< Interface Mask Requested */
       uint8_t                openFlag; /*!< Device Open Status Flag */
       char                   typeName[ADI_HAL_STRING_LENGTH];   /*!< Type Name */
       adi_hal_SpiCfg_t       spiCfg; /*!< SPI Configuration */
       adi_hal_LogCfg_t       logCfg; /*!< LOG Configuration */
       adi_hal_BbicCfg_t      bbicCfg; /*!< BBIC Configuration */
       adi_hal_HwResetCfg_t   hwResetCfg; /*!< HW Reset Configuration */
       adi_hal_I2cCfg_t       i2cCfg; /*!< I2C Configuration */
       adi_hal_TimerCfg_t     timerCfg; /*!< Timer Configuration */
       adi_hal_EepromCfg_t    eepromCfg; /*!< Eeprom Configuration */
       int32_t                error;  /*!< Operating System Error Code */
} adi_hal_Cfg_t;
```

An instance of adi_hal_Cfg_t should be created and initialized to default values using adi_hal_DevHalCfgCreate() function during initialization. Memory is allocated on the heap for each instance of adi_hal_Cfg_t created. To destroy the platform HAL configuration and de-allocate the memory adi_hal_DevHalCfgDestroy() function can be called. Since the HAL configuration is opaque to ADI devices, the instance of adi_hal_Cfg_t created via adi_hal_DevHalCfgCreate() is returned as a void pointer (void*). This pointer must be multiple instances of adi_hal_Cfg_t can be created independently per platform. The lifetime of a platform HAL configuration instance is listed below.

- adi_hal_DevHalCfgCreate() -> Interfaces Defined are Hardware/Feature Specific
- adi_hal_HwOpen() -> Enable all Selected HAL Interfaces & Features
- Use Device -> Hardware Ready for Use
- adi_hal_HwClose() -> Release all Resources (clean-up task)
- adi_hal_DevHalCfgFree() -> Free Memory, that is, (clean-up task)

The platform HAL configuration stored in an instance of adi_hal_Cfg_t structure is supplied as an argument to each platform HAL function described in Table 11 as (void* devHalCfg). For example, the platform HAL SPI write function can determine if the SPI transaction needs to be in 3-wire or 4-wire mode by examining the devHalCfg.spiCfg.fourWireMode member. The flow of platform HAL configuration from a user application to the platformHAL function is shown in Figure 24.

SOFTWARE INTEGRATION

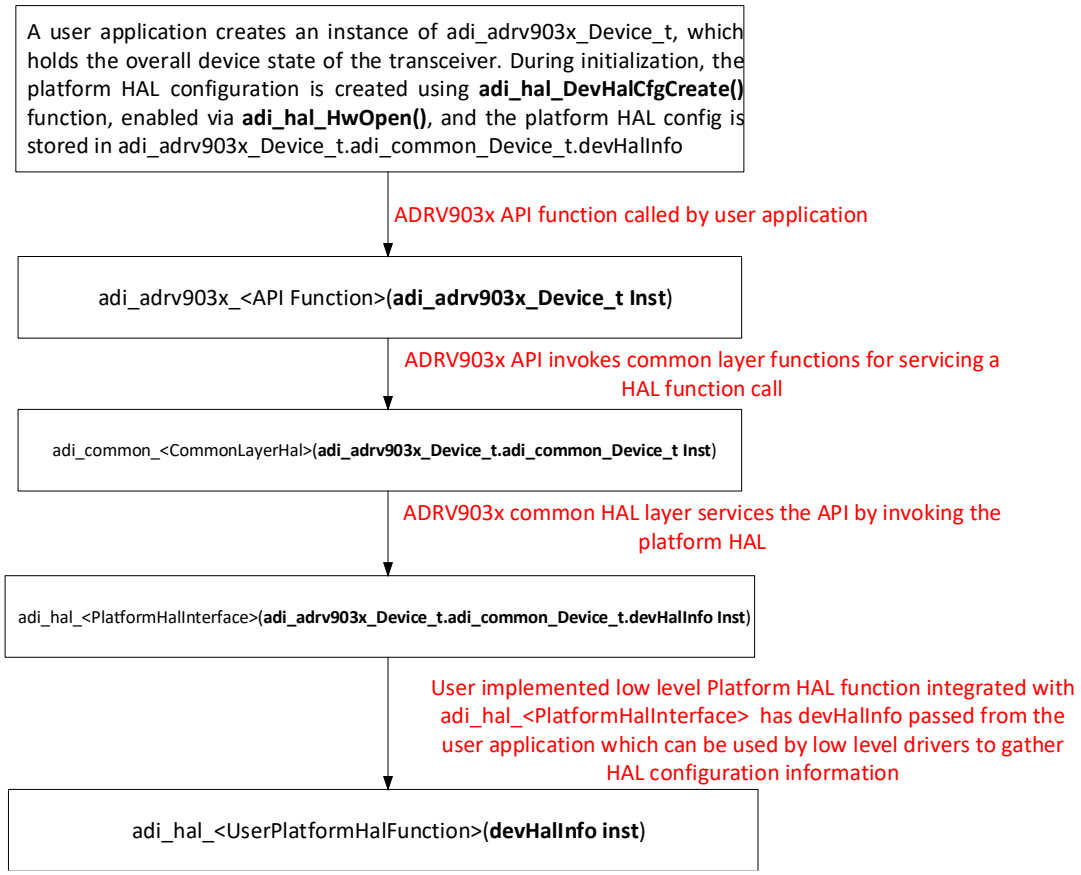


Figure 24. Flow of Platform HAL Configuration from a User Application to the Platform HAL Layer Function

Implementing the Platform HAL

Users who develop code to target custom hardware platforms will use different drivers for the peripherals such as the SPI and timer specific to their platform. Users can refer to the drivers developed for the ADI evaluation platform as reference. Implementation of platform HAL must conform the function prototypes defined in /platforms/adi_platform.h. A user implementation of all the functions defined in Table 11 is mandatory to guarantee the functionality of the ADRV903x.

The ADI HAL interface is a library of functions that the ADRV903x API uses when it needs to access the target platform hardware as described in the previous section. The software package delivered by ADI contains drivers specific to ADI evaluation platform (ADS10). The ADS10 platform HAL functions can be found in /platforms/ads10 folder shown in Figure 25.

SOFTWARE INTEGRATION

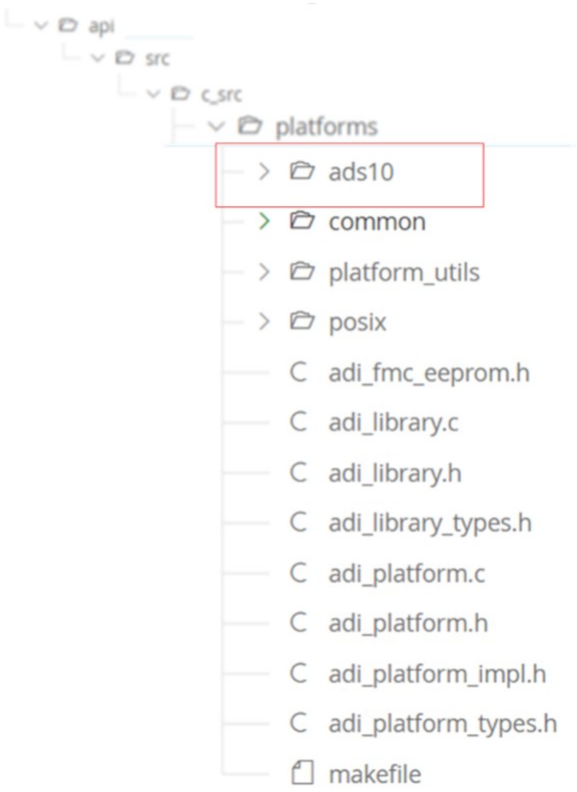


Figure 25. ADS10 Folder Containing Platform HAL Functions for ADI Evaluation Platform

The user implemented functions need to be assigned to the platform HAL interface functions during device initialization. The user can refer to /platforms/adi_platform.c/adi_hal_PlatformSetup() function for reference code where the ADI evaluation system platform drivers are assigned to the platform HAL interface functions listed in Table 10.

The code listing below is a snippet from adi_hal_PlatformSetup() function where the ADI evaluation platform HAL implementation functions are assigned to the platform HAL interface functions.

```
#ifdef _ADI_ADS10_PLATFORM
            adi_hal_HwOpen = ads10_HwOpen;
            adi_hal_HwClose = ads10_HwClose;
            adi_hal_HwReset = ads10_HwReset;
            adi_hal_DevHalCfgCreate = ads10_DevHalCfgCreate;
            adi_hal_DevHalCfgFree = ads10_DevHalCfgFree;
            adi_hal_SpiWrite = ads10_SpiWrite;
            adi_hal_SpiRead = ads10_SpiRead;
            adi_hal_LogFileOpen = ads10_LogFileOpen;
            adi_hal_LogLevelSet = ads10_LogLevelSet;
            adi_hal_LogLevelGet = ads10_LogLevelGet;
            adi_hal_LogStatusGet = ads10_LogStatusGet;
            adi_hal_LogConsoleSet = ads10_LogConsoleSet;
            adi_hal_LogWrite = ads10_LogWrite;
            adi_hal_LogFileClose = ads10_LogFileClose;
            adi_hal_Wait_us = ads10_TimerWait_us;
            adi_hal_Wait_ms = ads10_TimerWait_ms;
            /* only required to support the ADI FPGA*/
            adi_hal_BbicRegisterRead   = ads10_BbicRegisterRead;
            adi_hal_BbicRegisterWrite  = ads10_BbicRegisterWrite;
            adi_hal_BbicRegistersRead  = ads10_BbicRegistersRead;
```

SOFTWARE INTEGRATION

```
            adi_hal_BbicRegistersWrite = ads10_BbicRegistersWrite;
            adi_hal_ThreadSelf = posix_ThreadSelf;
            adi_hal_TlsGet = all_TlsGet;
            adi_hal_TlsSet = all_TlsSet;
            adi_hal_MutexInit = posix_MutexInit;
            adi_hal_MutexLock = posix_MutexLock;
            adi_hal_MutexUnlock = posix_MutexUnlock;
            adi_hal_MutexDestroy = posix_MutexDestroy;
            error = all_TlsInit();
#else
            error = ADI_HAL_ERR_NOT_IMPLEMENTED;
#endif
```

Multi-Threading

The purpose of the multi-threading feature is to support multi-threaded applications using the ADRV903x API. It means that use of the API by a multi-threaded application should be safe and function the same as a single-threaded application. Invoking API calls from several different threads rather than just one must not lead to inconsistent or faulty operation.

The multi-threading feature is independent of the transceiver itself so is not configured at Init-time or run-time as such. If an application is single-threaded and therefore does not require the multi-threading feature it should be possible to disable it to minimize resource usage and any run-time overhead. The API is intended to support multiple threads NOT multiple processes as shown in Figure 26. Multi-thread support allows a single customer application process to divide tasks logically among several threads without having consider how the threads will safely contend for API devices. Typically, a customer can use this feature to have some threads monitoring device status while other threads can deal with device functionality.

NOTE: If a customer has their own multi-threading management environment and don't wish to use ADI multi-threading HAL functions, then the multi-threading HAL functions in adi_hal_PlatformSetup() can be implemented as empty function stubs.

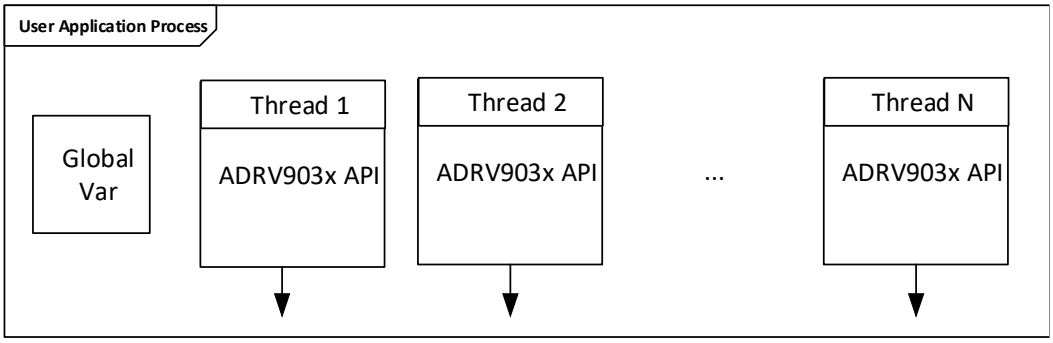


Figure 26. Multi-threaded User Application Process Using ADRV903x API

The multi-threading feature support depends on a user-supplied HAL layer being able to provide recursive mutexes. A recursive mutex means a thread can lock the mutex multiple times without first unlocking it. It requires the same number of unlocks to release the mutex before another thread can acquire the mutex.

A thread safe API functionality requires concrete implementations to the platform HAL interface functions described in Table 11.

Each ADRV903x will have its own mutex supplied by the HAL. This mutex is used to ensure that only one thread at a time is accessing the device (both physically and its associated state information held in the adi_ADRV903x_Device_t device structure). When a thread calls the API, the API function immediately locks the mutex associated with the device before proceeding with its operations, thus preventing any other thread invoking the API from accessing the device while the operation is in progress. Each API call will release the mutex acquired at the beginning of the API call as it's last act before exiting.

SOFTWARE INTEGRATION

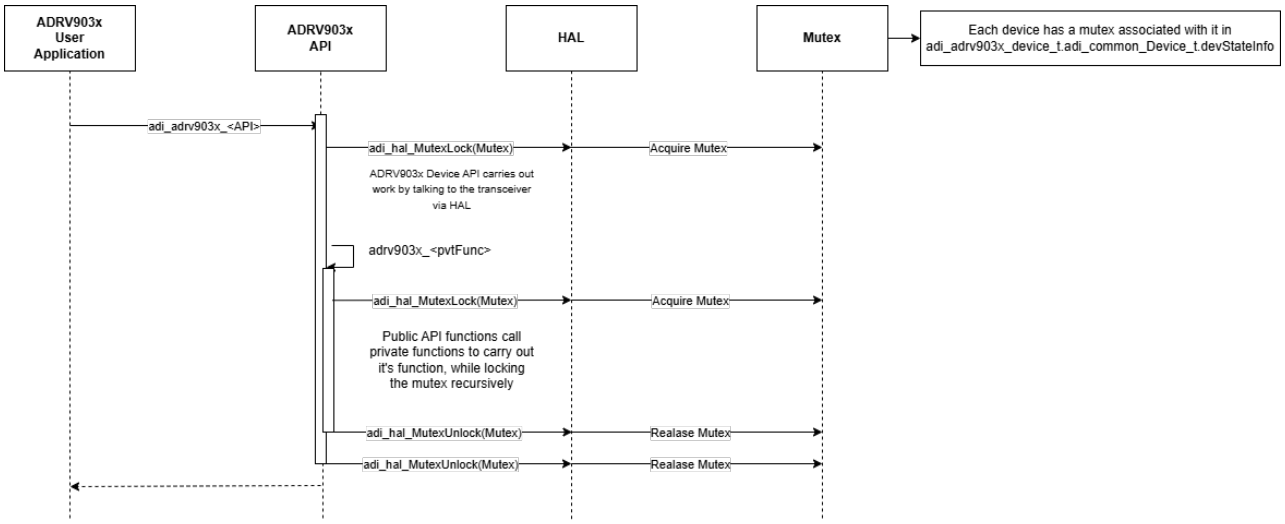


Figure 27. ADRV903x Mutex Operation Sequence

The following steps summarize the actions required to be taken by the user to implement thread safe API:

- Examine the multi-threading platform HAL functions in Table 11.
- Implement a recursive mutex that can be used to lock shared state information recursively.
- Integrate the user implemented platform HAL with ADI multi-threading functions in /platforms/adi_hal_platforms.c.
- Integrate the thread safe API functions into multiple threads in a user application. The API will invoke the user implemented platform HAL mutex functions to acquire and release shared state information.

Error Handling Memory

Error handling was introduced previously in the Error Handling section of this document. In the context of a multi-threading application, each thread must associate an error key with error handling memory on each thread. The user must implement the platform HAL functions adi_hal_TlsGet() and adi_hal_TlsSet() functions for thread safe error handling. In the context of a multi-threading application, each thread must associate an error key with error handling memory on each thread. The user must implement the platform HAL functions adi_hal_TlsGet() and adi_hal_TlsSet() functions in order to implement thread safe error handling functions. This is illustrated in Figure 28.

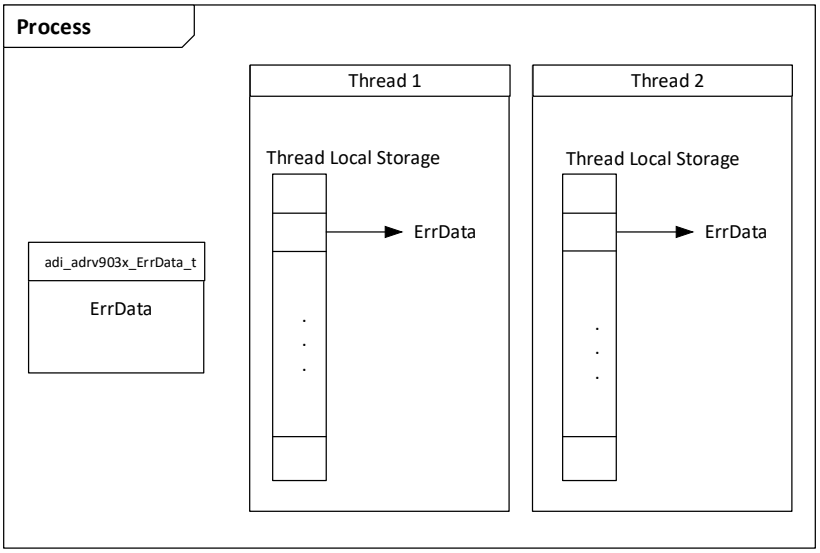


Figure 28. Illustration of Thread Local Storage

SOFTWARE INTEGRATION

In a typical use case, the user would call adi_hal_TlsSet() function in a user application to associate the error handling memory (adi_ADRV903x_Device_t.adi_common_Device_t.error) with a specific thread. When an error occurs, the API function retrieves the error handling memory associated with the thread via adi_hal_TlsGet() function, and updates the error value from the thread in which the error has occurred.

Shown below is an example pseudo-code snippet in which the error handling code and error data retrieval in a thread is shown.

```
void* tlsThread(void* args) {
    thread_args_t *thread_args = (thread_args_t*) args;

    while (time(NULL) < thread_args->end_time)
    {
        adi_hal_TlsSet(HAL_TLS_ERR, value_err);

        retVal = adi_adrv903x_API();

        if(retVal = ADI_ADRV903x_COMMON_ACT_NO_ACTION)
        {
            /* If an error has occured, retrieve error data corresponding to this thread */
            errorFromThread = adi_hal_TlsGet(HAL_TLS_ERR)

            /* Handle recovery action in an atomic code block */
            adi_Hal_MutexLock(adrv903x_Device)

            HandleError( errorFromThread );

            adi_Hal_MutexUnlock(adrv903x_Device);
        }
    }

    /* At the end of the thread, release all the thread local storage */
    adi_hal_TlsSet(HAL_TLS_END, NULL);

    return;
}
```

Figure 29.

DEVELOPING AN APPLICATION

Once the ADRV903x API is integrated into a user software application project, the next step is to develop an application. This section goes through some mandatory steps that a user needs to take to bring up the ADRV903x in order to run an application.

Instantiating the Device Data Structure

Device data structure holds all the state information of the ADRV903x. The device data structure is defined in /c_src/api/public/adi_adrv903x_types.h. The device data structure instance is passed to each ADRV903x API function call. An overview of the device data structure content is shown in Figure 30.

SOFTWARE INTEGRATION

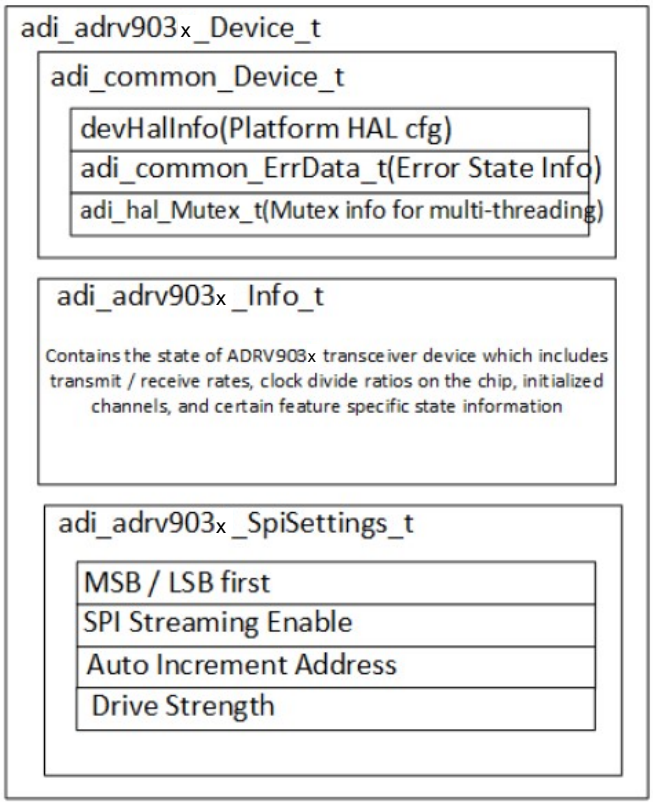


Figure 30. ADRV903x Device Data Structure

To support multiple devices the application layer code would need to instantiate multiple adi_ADRV903x_Device_t structures to describe each physical ADRV903x.

The devHalInfo is defined as a void pointer and allows the user to define and pass any platform hardware settings to the platform HAL layer functions (Refer to the section on configuring the platform HAL). For example, devHalInfo might contain information such as the SPI chip select to be used for the physical ADRV903x SPI interface. Note that the API functions are shared across all instances of physical devices. The devHalInfo structure defined by the developer identifies which physical ADRV903x is targeted (SPI chip select) when a particular ADRV903x API function is called. The developer may need to store other hardware information unique to a particular ADRV903x in this structure such as timer instances, log file information to allow for multithreading.

The devStateInfo member is of type adi_adrv903x_Info_t and is a runtime state container for the ADRV903x API. The application layer must allocate memory for this structure, but only the ADRV903x API writes to the structure. The application layer should allocate the devStateInfo structure with all zeroes. The API uses the devStateInfo structure to keep up with the current state of the API (has it been initialized, ARM loaded, etc.), as well as a debug store for any run-time data, such as error codes, error sources, etc. It is not intended for the application layer to access the devStateInfo member directly, as API functions will be provided to access the last error code and source information.

The adi_common_ErrData_t structure is used to contain the error information returned from the ADRV903x. The error structure is defined in Error Handling Memory section. The user is also expected to initialize the ADRV903x SPI settings in the sub-structure adi_ADRV903x_SpiConfigSettings_t in the device data structure.

Programming the Device

The summary of pre-requisites before attempting to program the device is listed below

1. Examine and integrate ADRV903x API source code with user application, implement platform HAL functions as described in the Platform HAL Interface section. Associate the ADI platform HAL interface functions in adi_hal_PlatformSetup( ) defined in the source file api/src/c_src/platforms/adi_platform.c.

SOFTWARE INTEGRATION

2. Generate ADRV903x resource files as described in the Resource Files section, instantiate a variable of type adi_ADRV903x_TrxFileInfo_t and initialize it with path to resource files and resource file settings.
3. Instantiate an ADRV903x device data structure as described in Instantiating the Device Data Structure section.
4. Architect error handling in the user application as described in the Error Handling section.
5. Instantiate an error data structure of type adi_common_ErrData_t and associate it with a thread using the API adi_hal_TlsSet( ) as described in the Error Handling Memory section.
6. Instantiate a HAL configuration structure of type adi_hal_Cfg_t and initialize HAL configurations as described in the Configuring the Platform HAL section.
7. Configure the BBIC to ADRV903x SPI bridge through the structure adi_ADRV903x_SpiConfigSettings_t as described in the SPI API Functions section.
8. Ensure that a valid clock is being supplied to the ADRV903x transceiver.

After instantiating the device data structure, the next essential step is to program the ADRV903x. To program the device, the resource files described in Error Handling Memory are necessary. The programming mainly involves five stages.

1. Creating the ADRV903x data structure, creating platform HAL configuration and enabling the platform HAL
2. Pre MCS initialization – In this stage, the ADRV903x hardware blocks are initialized, the resource files (Firmware binary, Stream binary, Profile, Rx gain tables) are loaded and the embedded ARM processor boot up sequence is triggered.
3. MCS – In this stage, the device requests SYSREF pulses for synchronization from a clock device in the platform.
4. Post MCS initialization – In this stage, the ADRV903x init time calibrations are performed, the LO frequencies are setup and the radio control mode is programmed.
5. JESD Bring up – In this stage, the JESD link between the baseband processor and the ADRV903x is initialized and the link is reset.

The sequence of events involved in programming the ADRV903x is shown in Figure 31. The JESD bring-up is further broken out in Figure 32.

SOFTWARE INTEGRATION

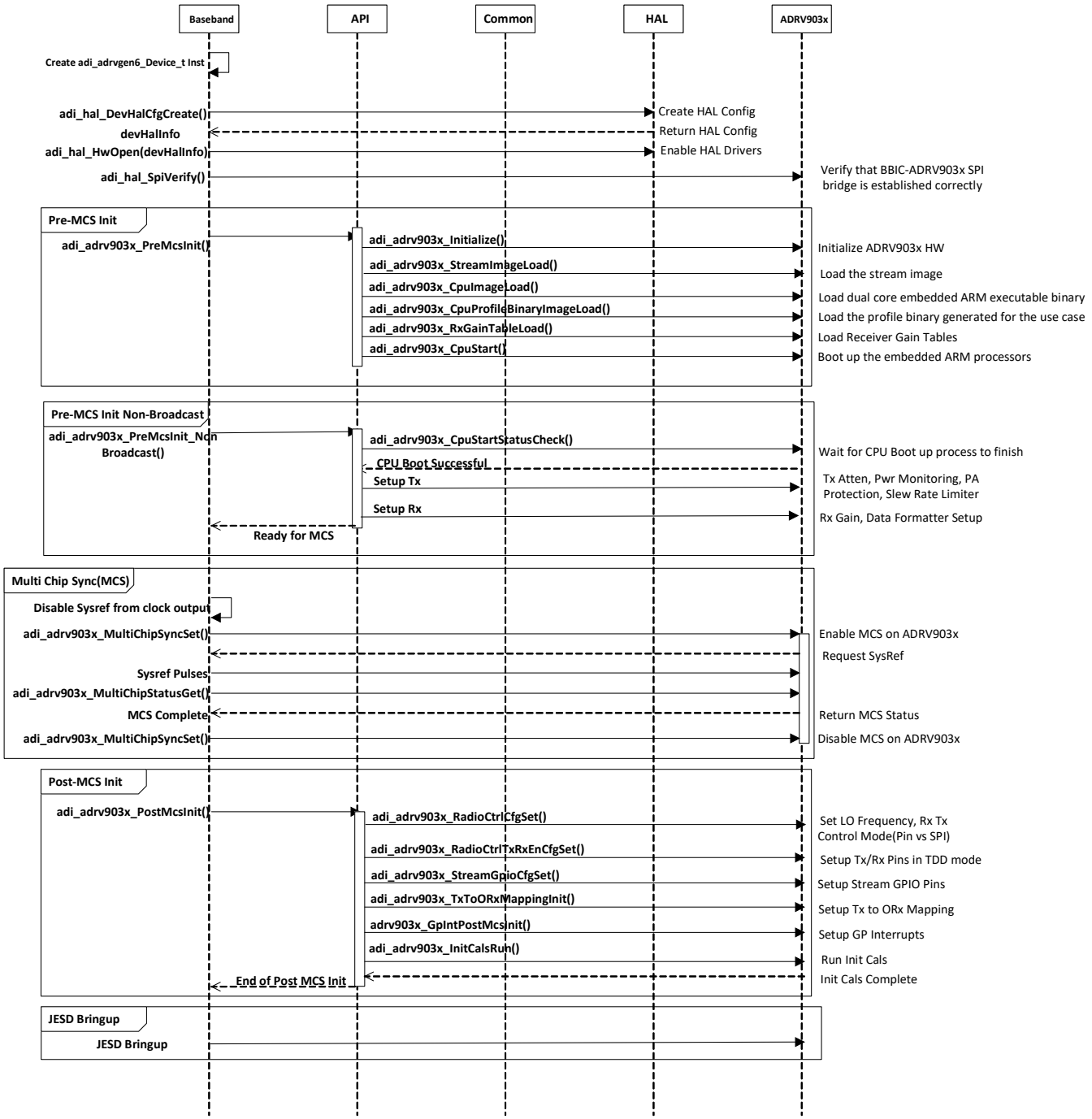


Figure 31. ADRV903x Programming Sequence

SOFTWARE INTEGRATION

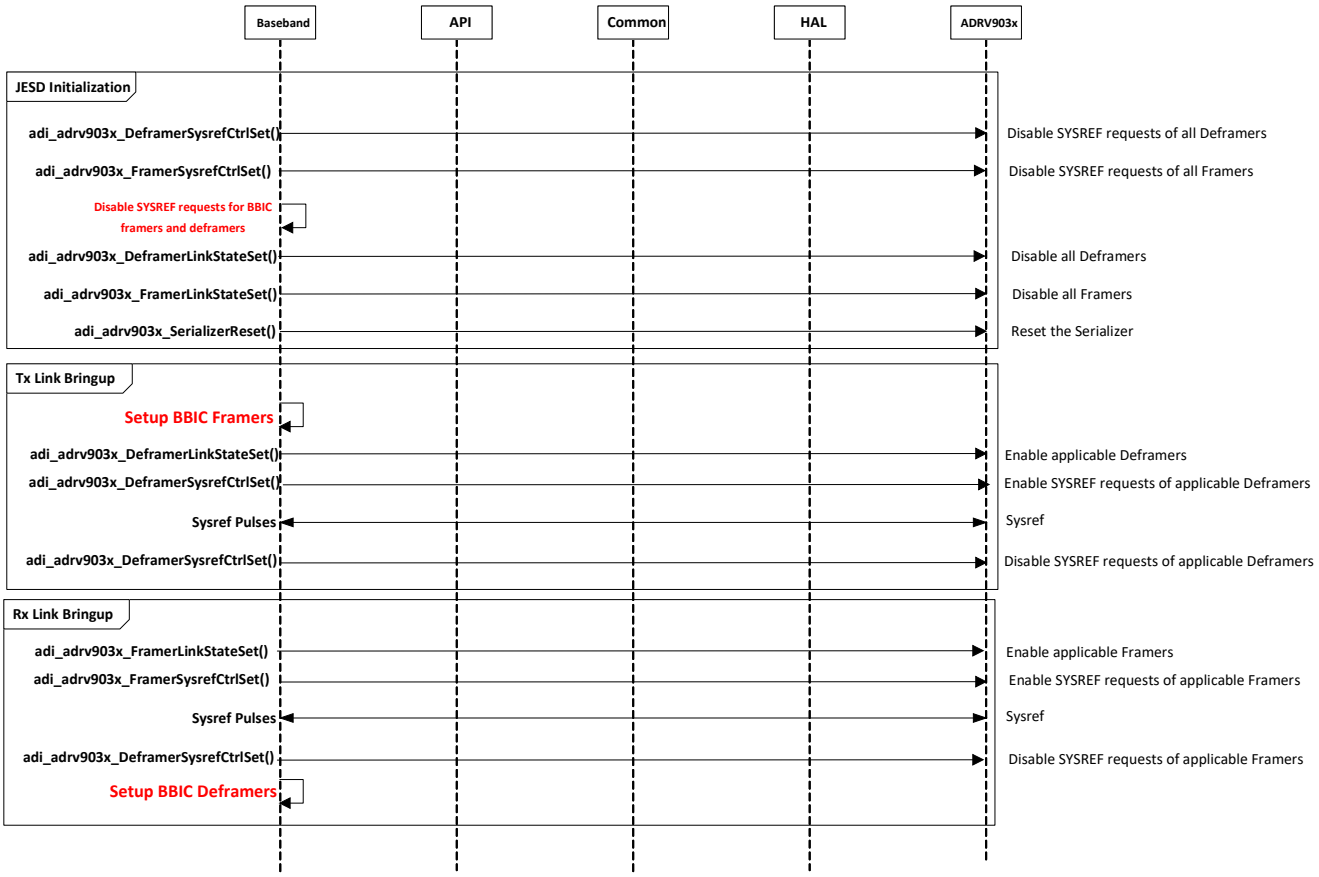


Figure 32. JESD Bring-Up Sequence with ADRV903x

GP Interrupts Setup

An important step that a user needs to take before proceeding to develop application code is to setup GP interrupts. The GP interrupts helps a user to acquire ADRV903x diagnostic data, such as PLL unlock, which can be monitored on a dedicated GP interrupt pin output from ADRV903x.

The user can create an interface that detects the GP interrupts in the hardware, invoke GP interrupt handlers provided by ADI that reside in /devices/adrv903x/adi_ADRV903x_gpio.h. Please refer to the General Purpose Interrupt section of this user guide for more details on setting up the GP Interrupts.

COMPILATION

Makefiles are delivered as part the ADI software package to assist in compilation of the API source files. The user can optionally compile the API source files into static libraries that can then be consumed in an ADRV903x application project. A typical compilation flow of an ADRV903x application using ADI provided makefiles is shown below.

SOFTWARE INTEGRATION

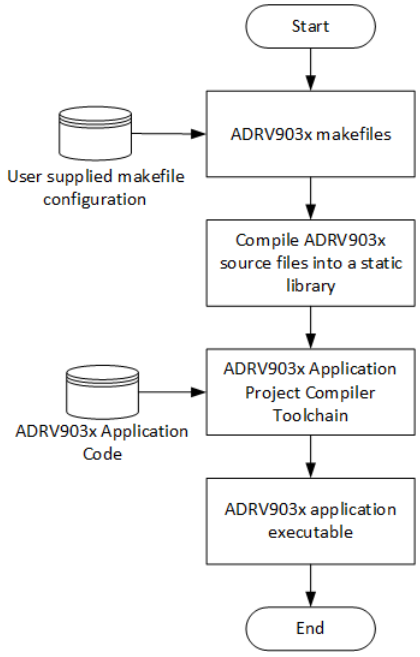


Figure 33. Compilation flow using ADRV903x Makefiles

Makefiles

There are four makefiles that are delivered as part of ADI software package shown in Table 12 which can be used to compile the ADRV903x source files into a static library for consumption in an ADRV903x application project. The static libraries are generated in an output directory whose name is specified by the user in the makefile configuration.

Table 12. ADRV903x Makefiles for Compilation

Target ADRV903x Source File	Makefile Path Relative to API Directory	Static Library Output File
ADRV903x Device API	/src/c_src/devices/adi_ADRV903x	/src/c_src/devices/adi_ADRV903x/<OutputDir>/libadi_ADRV903x.a
ADRV903x Device Common Layer	/src/c_src/common	/src/c_src/common/<OutputDir>/libadi_common.a
ADRV903x Device Platform Layer	/src/c_src/platform	/src/c_src/platform/<OutputDir>/libadi_platform.a
ADRV903x Bitfield Layer	/src/c_src/devices/adi_ADRV903x/private/bf	/src/c_src/devices/adi_ADRV903x/private/bf<OutputDir>/libadi_bf.a

NOTE: <OutputDir> is specified by the user in the makefile configuration. The makefile configuration can be found in Configuring the Makefiles section

In order to compile the makefile, the user can use the following command:

make all CONFIGFILE =<Absolute Path To Makefile> / <makefile configuration file>

Configuring the Makefiles

The user is required to setup the following toolchain configurations and pass them as input to the make commands in order to generate a static library from the ADRV903x source files. The mandatory configurations required by ADRV903x makefiles are listed in Table 13. The ADRV903x makefiles use C Compilers by default. As part of the SW package, and example makefile configuration file /api/src/c_src/configUserExample.mak. The example configuration file is based on the GCC compiler and the configuration values used in the example configuration file can be found in Table 13.

SOFTWARE INTEGRATION

Table 13. Configurations Required by ADRV903x Makefile

Configuration	Description	Value Used in ADI Example Config Files
BINARYDIR	Output folder name for the static library to be created in the same directory as the location of the makefile	–
USE_ADS_10	This configuration is only applicable for compiling source codes for ADI eval platform. A value of 1 enables compilation for ADI evaluation platform	0
CC	C – Compiler	gcc
CXX	C++ Compiler	g++
LD	Linker	$(CXX)
AR	Binary utility for archiving	ar.exe
OBJCOPY	Utility for reading / writing object files	objcopy.exe
CFLAGS	C Compiler Flags	-ggdb -Wall -Wpedantic -Werror -ffunction-sections -O0 -x c++ -fPIC
CXXFLAGS	C++ Compiler Flags	-Wall -Werror -Wpedantic -std=c++11 -ggdb -ffunction-sections -O0 -fPIC
ASFLAGS	Assembler Flags	-
LDFLAGS	Linker Flags	-Wl,-gc-sections -lgcov

In addition to the mandatory configurations required by the ADRV903x makefiles, the user can include additional makefile configurations specific to the user's project. The complete example makefile configuration listing is shown in Figure 34. The additional configurations used in the ADRV903x example project are included in the red box shown in Figure 34.

```
#Output folder for binary files
BINARYDIR := Debug

#Set this to 1, if ADS10 platform will be used
USE_ADS_10 := 0

#Toolchain configuration
CC := gcc.exe
CXX := g++.exe
LD := $(CXX)
AR := ar.exe
OBJCOPY := objcopy.exe
CFLAGS := -ggdb -Wall -Wpedantic -Werror -ffunction-sections -O0 -x c++ -fPIC
CXXFLAGS := -Wall -Werror -Wpedantic -std=c++11 -ggdb -ffunction-sections -O0 -fPIC
ASFLAGS :=
LDFLAGS := -Wl,-gc-sections -lgcov

#Additional flags
PREPROCESSOR_MACROS := PRODUCT_ADRV903x TPG_PRODUCT_NAME=adrv9xxx TPG_PRODUCT_MACRO_NAME=ADRV9xxx TPG_PRODUCT_INCLUDE_NAME=adrvgen6
ifeq ($(USE_ADS_10),1)
PREPROCESSOR_MACROS += _ADI_ADS10_PLATFORM
endif

#Additional options detected from testing the toolchain
USE_DEL_TO_CLEAN := 0
START_GROUP := -Wl,--start-group
END_GROUP := -Wl,--end-group
```

Figure 34. ADRV903x Makefile Configuration Example

SERIAL PERIPHERAL INTERFACE (SPI)

The SPI bus allows a BBIC to control the ADRV903x. The SPI bus is the primary interface to setup and configure the device. SPI registers contain control bits, status monitors, or other settings that control all functions of the device.

SPI register transactions can occur in 8-bit or 32-bit modes. Depending on the type of transaction, it may be necessary for an 8-bit read/write operation or a 32-bit operation. The API determines what transaction size is necessary based on the information it needs to read and write.

This section is an information-only section to give the user an understanding of the hardware interface the BBIC uses to control the device. All control functions are implemented using the API detailed within this document. The following sections explain the specifics of this interface.

SPI BUS SIGNALS

The SPI bus consists of the following signals:

- $\overline{\text{SPI\_EN}}$
- SPI_CLK
- SPI_DIO
- SPI_DO

$\overline{\text{SPI\_EN}}$

$\overline{\text{SPI\_EN}}$ is the active-low chip select that functions as the bus enable signal driven from the BBIC to the device. This signal is an input to the $\overline{\text{SPI\_EN}}$ pin. $\overline{\text{SPI\_EN}}$ is driven low before the first SPI_CLK rising edge and is normally driven high again after the last SPI_CLK falling edge. The device ignores the clock and data signals while $\overline{\text{SPI\_EN}}$ is high. $\overline{\text{SPI\_EN}}$ also frames communication to and from the device and returns the SPI interface to the ready state when it is driven high.

Forcing $\overline{\text{SPI\_EN}}$ high in the middle of a transaction aborts part, or all, of the transaction. If the transaction is aborted before the instruction is complete or in the middle of the first data word, the state machine returns to the ready state. Any complete data byte transfers prior to $\overline{\text{SPI\_EN}}$ de-asserting are valid, but all subsequent transfers in a continuous SPI transaction are aborted.

SPI_CLK

SPI_CLK is the serial interface reference clock driven by the BBIC. This signal is an input to the SPI_CLK pin. It is only active while $\overline{\text{SPI\_EN}}$ is low. The maximum SPI_CLK frequency is 50 MHz. These limits are determined based on the practical timing requirements and the physical limitations of the device.

SPI_DIO and SPI_DO

The SPI interface supports both 4-wire and 3-wire bus modes. When configured as a 4-wire bus, the SPI utilizes two data signals: SPI_DIO and SPI_DO. SPI_DIO is the data input line driven from the BBIC. The signal is input to the device on the SPI_DIO pin. SPI_DO is the data output from the device to the BBIC and is driven by the SPI_DO pin. When configured as a 3-wire bus, SPI_DIO is used as a bidirectional pin that both receives and transmits serial data. The SPI_DO pin is disabled in this mode. Per ADI standards, the device defaults into 3-wire mode after hardware/software reset and power cycling.

The data signals are launched on the falling edge of SPI_CLK and sampled on the rising edge of SPI_CLK by both the BBIC and the device. SPI_DIO carries the control field from the BBIC to the device during all transactions, and it carries the write data fields during a write transaction. In a 3-wire SPI configuration, SPI_DIO carries the returning read data fields from the device to the BBIC during a read transaction. In a 4-wire SPI configuration, SPI_DO carries the returning data fields to the BBIC.

The SPI_DO and SPI_DIO pins transition to a high impedance state when the $\overline{\text{SPI\_EN}}$ input is high. The device does not provide any weak pull-ups or pull-downs on these pins. When SPI_DO is inactive, it is floated in a high impedance state. If a valid logic state on SPI_DO is required, an external weak pull-up/down (10 kΩ value) should be added on the PCB.

SPI DATA TRANSFER PROTOCOL

The SPI is a flexible, synchronous serial communication bus allowing seamless interfacing to many industry standard microcontrollers and microprocessors. The serial I/O is compatible with most synchronous transfer formats, including both the Motorola SPI and Intel SSR protocols. The control field width for this device is limited to 16 bits, and multi-byte IO operation is allowed. This device cannot be used to control other devices on the bus – it only operates as a target.

SERIAL PERIPHERAL INTERFACE (SPI)

There are two phases to a communication cycle:

- Phase 1 is the control cycle, which is the writing of a control word into the device. The control word provides the serial port controller with information regarding the data field transfer cycle, which is Phase 2 of the communication cycle. The Phase 1 control field defines whether the upcoming data transfer is read or write. It also defines the register address being accessed.
- Phase 2 can be either a single byte transfer to or from the device (depending on the Phase 1 instruction) or it can be a multi-byte data transfer.

Phase 1 Instruction Format

The 16-bit control field contains the following information:

Table 14. SPI Transaction Phase 1 Instruction Format

Bit Position	Short hand Interpretation
D15	R/Wb (Read/Write-Bar)
D14:D0	A<14:0> (Address word)

These are explained further below:

- D15 — Bit 15 of the instruction word determines whether a read or write data transfer occurs after the instruction byte write. Logic high indicates a read operation; logic low indicates a write operation.
- D14:D0 – Bits A<14:0> specify the starting byte address for the data transfer during Phase 2 of the IO operation.

All byte addresses, both explicitly specified and internally generated by address auto increment/decrement are assumed to be valid. That is, if an invalid address (undefined register) is accessed, the IO operation continues as if the address space were valid. For write operations, the written bits are discarded, and read operations result in logic zeros at the output.

Single-Byte Data Transfer

When enSpiStreaming = 0, a single-byte data transfer is chosen. In this mode, $\overline{\text{SPI\_EN}}$ goes active-low, the SPI_CLK signal activates, and the address is transferred from the BBIC to the device. This mode is always used in direct communication between the BBIC and the device.

The API parameter msbFirst indicates the bit packing order of Phase 1 and Phase 2 data. Phase 1 is always transmitted first.

- If msbFirst = 0, then this is LSB first mode. In this mode, the LSB of the address is the first bit transmitted from the BBIC, followed by the next 14 bits in order from next LSB to MSB. The final bit in the phase1 instruction signifies if the operation is read (set) or write (clear). If the operation is a write, the BBIC will transmit the next 8 bits LSB to MSB. If the operation is a read, the device will transmit the next 8 bits LSB to MSB.
- If msbFirst = 1, then this is MSB first mode. In this mode, the first bit transmitted is the R/Wb bit that determines if the operation is a read (set) or write (clear). The MSB of the address is the next bit transmitted from the BBIC, followed by the remaining 14 bits in order from next MSB to LSB. If the operation is a write, the BBIC will transmit the next 8 bits MSB to LSB. If the operation is a read, the device will transmit the next 8 bits MSB to LSB.

Single-byte data transfer can occur where a single address/data transaction occurs between the $\overline{\text{SPI\_EN}}$ transition from low to high. This is called single instruction mode as shown in Figure 35:

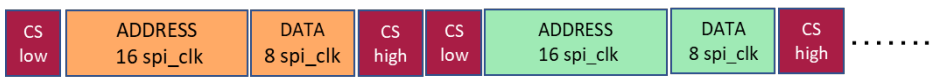


Figure 35. SPI Transactions Under Single Instruction Operation

Alternatively, it is possible to make multiple address/data transactions between the $\overline{\text{SPI\_EN}}$ transition from low to high. This is called the single instruction buffered mode. This uses the transfer format of address followed by data (Byte Order: A A D A A D …) until the $\overline{\text{SPI\_EN}}$ signal is driven high. The address must be written for each data byte transfer when using this mode as shown in Figure 36.

SERIAL PERIPHERAL INTERFACE (SPI)

Figure 36. SPI Transactions under Single Instruction Buffered Operation

Multi-Byte Data Transfer (SPI Streaming)

Multi-byte data transfer (also called SPI streaming) is not utilized in standard communication between the BBIC and the device. When enSpiStreaming = 1, data is transferred in multi-byte packets depending on the streaming mode that is enabled. This mode is used to transfer data indirectly to internal ARM memory using a direct memory access (DMA) controller.

SPI API FUNCTIONS

Table 15. List of SPI API Functions

API Method Name	Comments
adi_ADRV903x_SpiCfgSet()	Sets the configuration of the SPI interface.
adi_ADRV903x_SpiCfgGet()	Get the configuration of the SPI interface.

Configure the SPI interface by calling adi_ADRV903x_SpiCfgSet(). The adi_ADRV903x_HwOpen() command is the first command called before any other API interaction with the device, calls the adi_ADRV903x_HwReset() which issues a reset pin toggle followed by configuration of the SPI interface via adi_ADRV903x_SpiCfgSet().

Users can configure SPI settings for the device with different SPI controller configurations by configuring member values of the adi_ADRV903x_SpiSettings_t data structure. The adi_ADRV903x_SpiSettings_t data structure is defined below:

```
typedef struct adi_ADRV903x_SpiSettings
{
    uint8_t                         msbFirst;
    uint8_t                         enSpiStreaming;
    uint8_t                         autoIncAddrUp;
    uint8_t                         fourWireMode;
    adi_ADRV903x_CmosPadDrvStr_e    cmosPadDrvStrength;
} adi_ADRV903x_SpiSettings_t;
```

The parameters for this structure are listed in Table 16.

Table 16. SPI Bus Setup Parameters

Structure Member	Value	Function	Default
msbFirst	0x00	Least significant bit first	0x01
	0x01	Most significant bit first	
enSpiStreaming	0x00	Enable single-byte data transfer mode. All communication between the BBIC and the device uses this mode.	0x00
	0x01	Enable streaming to improve SPI throughput for indirect data transfer using an internal DMA controller.	
autoIncAddrUp	0x00	Auto-increment. Functionality intended to be used with SPI Streaming. Sets address auto-increment -> next addr = addr+4	0x01
	0x01	Auto-decrement. Functionality intended to be used with SPI Streaming. Sets address auto-decrement -> next addr = addr -4	
fourWireMode	0x00	SPI hardware implementation. Use 3-wire SPI (SPI_DIO pin is bidirectional). Figure 37 shows example of SPI 3-wire mode of operation.	0x01
	0x01	SPI hardware implementation. Use 4-wire SPI. Figure 38 and Figure 39 show examples of SPI 4 –wire mode of operation. NOTE: Default mode for ADI's FPGA platform is 4-wire mode.	
cmosPadDrvStrength	0x00	Applies to all output CMOS pins (GPIO, SPI_SDO, GPINT). 5 pF load @ 75 MHz	0x01

SERIAL PERIPHERAL INTERFACE (SPI)

Table 16. SPI Bus Setup Parameters (Continued)

Structure Member	Value	Function	Default
	ADI_ADRV903X_CMOS-PAD_DRV_WEAK 0x01 ADI_ADRV903X_CMOS-PAD_DRV_STRONG	Applies to all output CMOS pins (GPIO, SPI_SDO, GPINT). 100 pF load @ 100 MHz	

Table 17 lists the timing specifications for the SPI bus. The relationship between these parameters is in Figure 37. This diagram shows a 3-wire SPI bus timing diagram with the device returning a value of 0xD4 from register 0x00A and timing parameters marked. Note that this is a single read operation, so the bus-ready parameter after the data is driven from the device (t_{HZS}) is not shown in the diagram.

Table 17. SPI Bus Timing Constraint Values

Parameter	Min	Typical	Max	Description
t_{CP}	20 ns		100 ns	SPI_CLK cycle time (clock period)
t_{MP}	10 ns			SPI_CLK pulse width
t_{SC}	4 ns			$\overline{SPI\_EN}$ setup time to first SPI_CLK rising edge
t_{HC}	0 ns			Last SPI_CLK falling edge to $\overline{SPI\_EN}$ hold
t_S	4 ns			SPI_DIO data input setup time to SPI_CLK
t_H	0 ns			SPI_DIO data input hold time to SPI_CLK
t_{CO}	10 ns		16 ns	SPI_CLK falling edge to output data delay (3-wire or 4-wire mode)
t_{HZM}	t_H		$t_{CO\ (max)}$	Bus turnaround time after BBIC drives the last address bit
t_{HZS}	0 ns		$t_{CO\ (max)}$	Bus turnaround time after device drives the last data bit

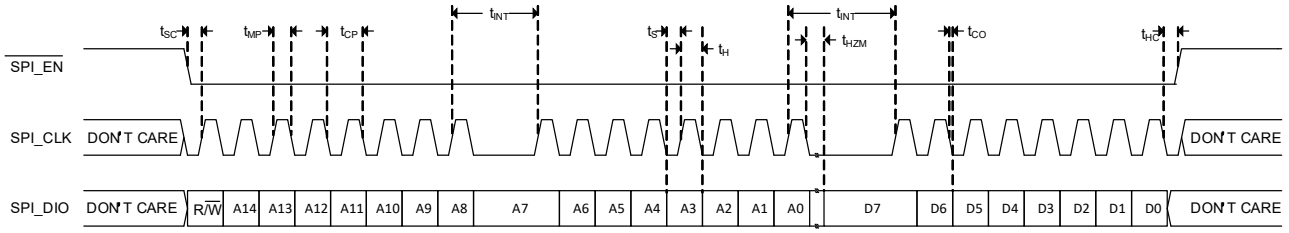


Figure 37. 3-Wire SPI Timing with Parameter Labels

TIMING DIAGRAM EXAMPLES

The diagrams in Figure 38 and Figure 39 illustrate the SPI bus waveforms for a single-register write operation and a single-register read operation, respectively. In the first figure, the value 0x55 is written to register 0x00A. In the second value, register 0x00A is read and the value returned is 0x55. If the same operations were performed with a 3-wire bus, the SPI_DO line in Figure 38 would be eliminated, and the SPI_DIO and SPI_DO signal in Figure 39 would be combined on the SPI_DIO signal. Note that both operations use MSB-first mode and all data is latched on the rising edge of the SPI_CLK signal.

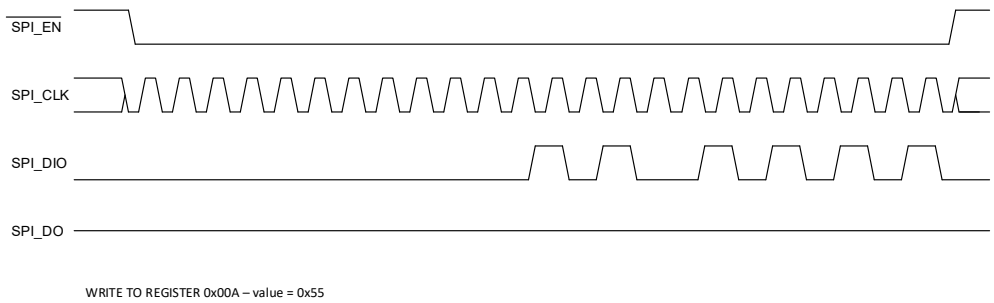


Figure 38. Nominal Timing Diagram, SPI Write Operation

SERIAL PERIPHERAL INTERFACE (SPI)

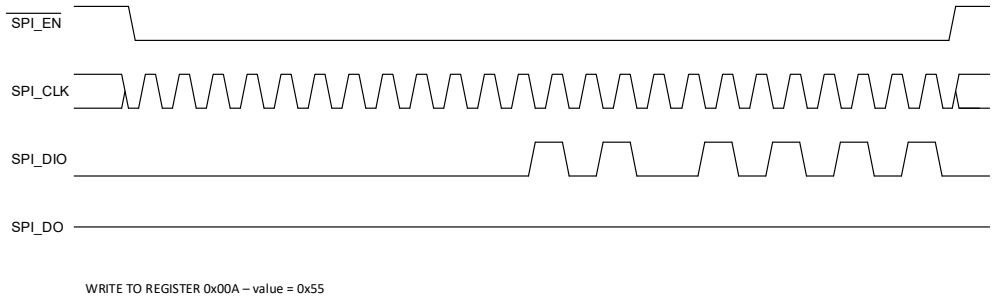


Figure 39. Nominal Timing Diagram, SPI Read Operation

AUXILIARY SPI OVERVIEW

The ADRV903x features an optional secondary SPI target instance. This is designated the Auxiliary SPI port. The Auxiliary SPI interface, if enabled, uses the same configuration as the primary SPI interface. This interface is multiplexed with the digital GPIO pins, however only specific GPIO pins must be used for the Auxiliary SPI feature. This is documented below.

Table 18. Allowed Pins for Auxiliary Interface

Aux SPI Signal	Allowed Pin Configuration 1	Allowed Pin Configuration 2
SPI_DIO	GPIO[0]	GPIO[13]
SPI_DO	GPIO[1]	GPIO[14]
SPI_CLK	GPIO[2]	GPIO[15]
$\overline{\text{SPI\_EN}}$	GPIO[3]	GPIO[16]

The Auxiliary SPI interface cannot be used to initialize the device. It only serves as an alternate SPI port for runtime interactions. When the Auxiliary SPI interface is enabled, the selected GPIO pins are reserved only for Auxiliary SPI utilization.

AUXILIARY SPI API FUNCTIONS

Table 19. List of Auxiliary SPI Related API Functions

API Method Name	Comments
adi_ADRV903x_AuxSpiCfgSet()	Sets the enable/disable and configuration of the Auxiliary SPI interface
adi_ADRV903x_AuxSpiCfgGet()	Returns the enable/disable status and current configuration of the Auxiliary SPI interface.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

The ADRV903x uses a SERDES high speed serial interface based on the JESD204B/C standards to transfer ADC and DAC samples between the device and a BBIC. The device can support lane rates up to 32,440.32 Mbps. An external clock distribution solution provides a device clock and SYSREF to both the device and the BBIC. The SYSREF signal ensures deterministic latency between the ADRV903x and the BBIC.

The ADRV903x supports the device subclass 0 and device subclass 1 operation modes. In subclass 0 deterministic latency is not supported. In subclass 1 the SYSREF signal is used to achieve deterministic latency.

The JESD204C standard is backward compatible with JESD204B. It supports both 8B/10B encoding and 64B/66B encoding. JESD204B and JESD204C with 8B/10B are considered the same when discussed here.

JESD204 STANDARD

The JESD204 specification defines four key blocks that implement the JESD protocol, as shown in Figure 40. The transport layer maps the conversion between samples and framed, unscrambled octets. The scrambler/descrambler blocks (optional in JESD204B) scramble/descramble the octets, spreading the spectral peaks to reduce EMI. The data-link layer handles link synchronization, setup, and maintenance, and encodes/decodes the scrambled octets to/from 10-bit characters in the case of 8B/10B encoding (204B and 204C) and 66-bit characters in the case of 64B/66B encoding (204C only). The physical layer is responsible for transmission and reception of characters at the lane bit rate.

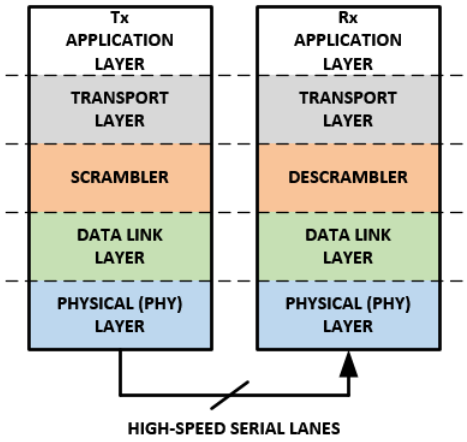


Figure 40. Key Blocks of the JESD 204B/C Standard

Figure 41 and Figure 42 illustrate how the JESD204 transmit and receive protocols are implemented.

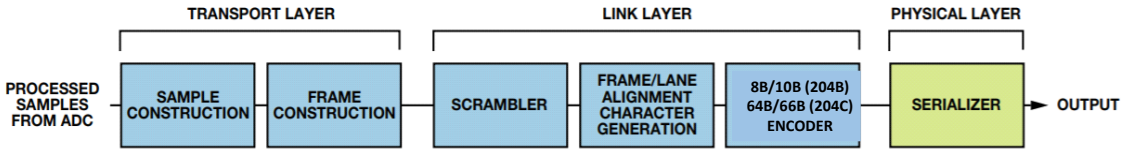


Figure 41. JESD204B/C Framer (JTX)

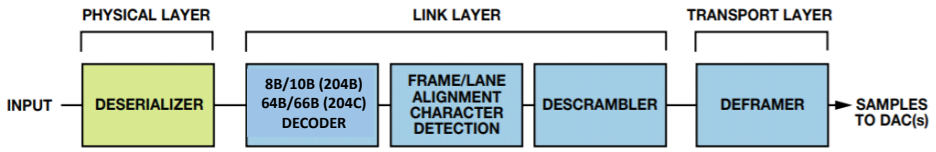


Figure 42. JESD 204B/C Deframer (JRX)

The data interface blocks in the ADRV903x can operate in either 204B or 204C modes. Fewer number of lanes may be needed when operating in 204C, which results in simpler PCB layout and lower power consumption.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

OVERVIEW OF THE DIFFERENCES BETWEEN JESD204B AND JESD204C

The initial revision of the JESD204 interface (JESD204A) provided support for both single and multiple lanes per convertor device. Revision B (JESD204B) added programmable deterministic latency, usage of device clock as main clock source and data rate up to 12.5 Gbps. In the Revision C specification (JESD204C), the data rate is increased up to 32 Gbps and three link layers are defined as 8B/10B, 64B/66B and 64B/80B where the 8B/10B link layer is backward compatible to the JESD204B link layer.

In the 8B/10B link layer, the data is organized into frames and multiframes. The 8B/10B link layer uses a SYNCB feedback signal from deframer to framer, along with the Code Group Synchronization (CGS) and Initial Lane Alignment Sequence (ILAS) steps to achieve lane and frame alignment. A Local Multiframe clock (LMFC) is used to mark the multiframe boundaries. The 8B/10B link layer uses the LMFC to achieve deterministic latency. In Subclass 1, the alignment between multiple converter devices is done through the alignment of their respective LMFCs to an external SYSREF signal.

In the 64B/66B link layer, the data is organized into blocks, multiblocks and extended multiblocks. Each block contains eight octets, each multiblock contains 32 blocks, and each extended multiblock contains one or more multiblocks. The concept of frames also exists in the 64B/66B protocol. No SYNCB feedback signal is used for the 64B/66B link layer, the transmission is entirely feed forward. The CGS and ILAS steps are not used either. Sync header bits are instead used for block, multiblock and extended multiblock alignment, and as result lane alignment. A Local Extended Multiblock clock (LEMC) is used to mark extended multiblock boundaries. The 64B/66B link layer utilizes the LEMC to achieve deterministic latency. In Subclass 1, the alignment between multiple converter devices is done through the alignment of their respective LEMCs to an external SYSREF signal.

In the 64B/66B link layer, one sync header per block is decoded to a single bit, and 32 of these bits across a multiblock makes a 32-bit sync word. The sync word can contain the following information:

- Pilot signal (marks the borders of the multiblocks and extended multiblocks)
- CRC-12 signal or optional CRC-3 signal (used for error detection, ADRV903x deframers only support CRC-3)
- FEC signal (error detection and correction, only supported on ADRV903x framers)
- Command channel (transmitting commands and status information)

Refer to the JESD204B and JESD204C specifications for further details.

JESD204B/C FRAMERS

The main function of the framers consists of mapping converter data samples into octets spread across one or more lanes and performing 8B/10B (JESD204B) or 64B/66B (JESD204C) encoding on the data transported over the lanes.

The ADRV903x features three framers to allow flexibility in configuring the output data streams. These framers are highly configurable in terms of interface rates and combinations of main RF receiver/observation receiver data streams, either separately or utilizing link sharing (Rx/ORx data time-multiplexed on the same lane according to the Rx−Tx frame timing) for up to eight lanes. To assist in debugging, they contain internal data generators allowing several test patterns and PRBS patterns to be sent across the link.

Figure 43 provides a high-level overview of the framers present in the ADRV903x. The data samples available to the framers come from the main RF receiver (Rx) datapaths and from the observation receiver (ORx) datapaths. Further details are provided in the sections below.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

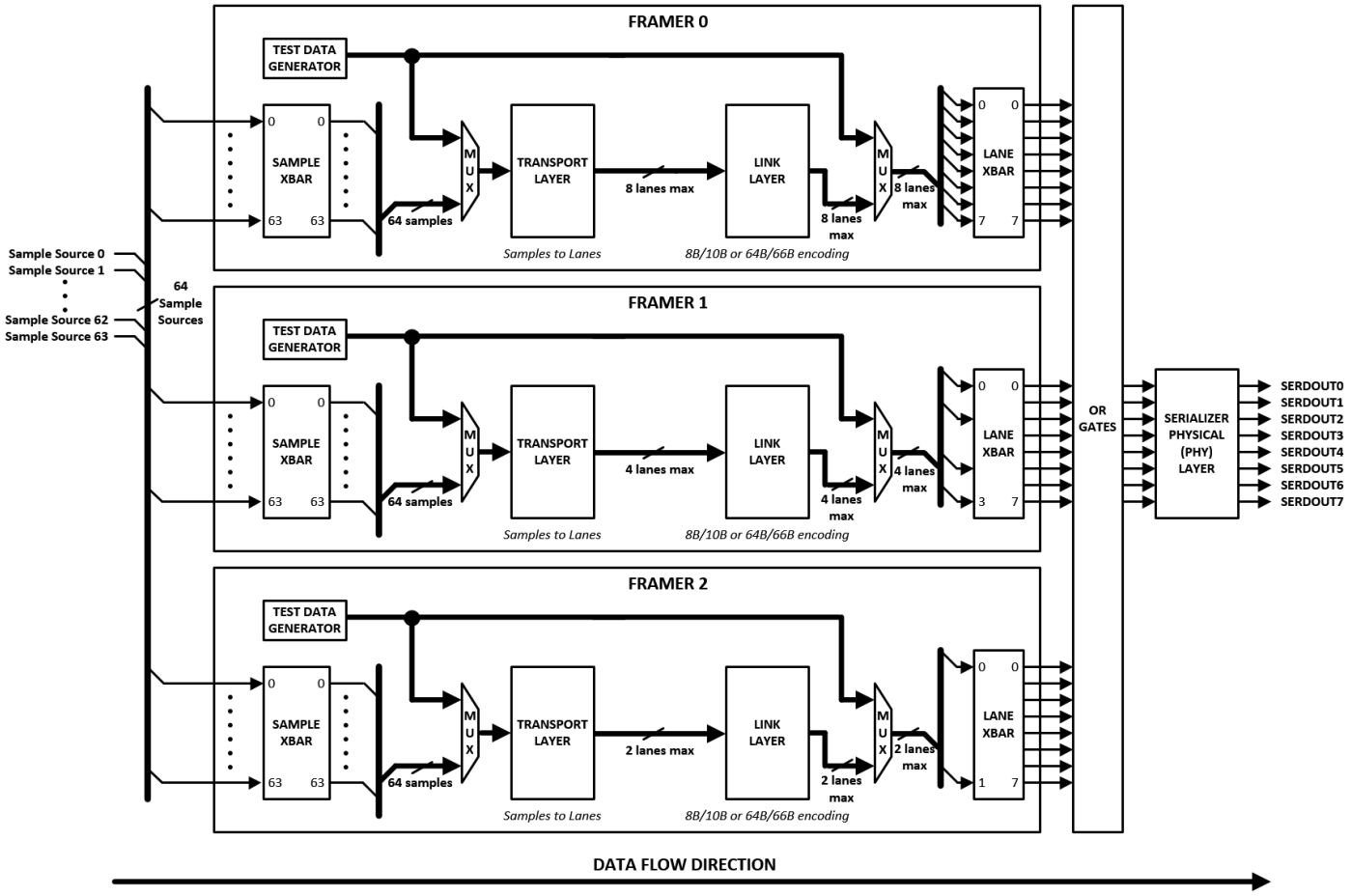


Figure 43. High Level Overview of the JESD204B/C Framers

JESD204B/C Framers Parameters

Table 20 provides a list of the supported framer parameter values. Note that not all combinations of those framer parameter values are supported.

Table 20. List of JESD204B/C Framers Parameters

Parameter	Parameter Description	Possible Parameter Values[1]
M	Number of converter devices	1 to 16
L	Number of Lanes to transmit and encode data on	1 (all framers), 2 (all framers), 4 (framer 0 and framer 1 only), 8 (framer 0 only)
F	JESD204B: Number of octets per lane in a frame cycle	2, 3, 4, 6, 8, 12, 16, 24, 32
	JESD204C: Used in conjunction with K to set the extended multiblock period. F is calculated as in JESD204B mode	2, 3, 4, 6, 8, 12, 16, 24, 32
S	Number of samples per converter per frame cycle	1, 2, 4
N'	Number of bits in a sample	12, 16, 24
K	JESD204B: Number of frames in one multiframe	1 to 32. F × K must be a multiple of four and (20 ≤ F × K ≤ 256)
	JESD204C: Used in conjunction with F to set the extended multiblock period	K × F = 256 × E, K ≤ 256
E	JESD204C only: Number of blocks in an extended multiblock	1 to 32
CS	Number of control bits per converter sample	0 to 3

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

Table 20. List of JESD204B/C Framers Parameters (Continued)

Parameter	Parameter Description	Possible Parameter Values[1]
HD	High Density mode	0, 1

[1] Not all combinations of the framer parameter values are supported.

Due to the DDC bands in the Rx datapaths and to the ORx de-interleaver block, there are more data sample sources than the number of actual converter devices (ADCs). For this reason, the M parameter can be viewed as a virtual number of converter devices, corresponding to the number of sample sources provided to the transport layer. The possible M values can therefore be higher than the total number of actual converters (ADCs).

Sample Data Sources and Sample Crossbars

A sample crossbar is present in each of the three framers in the ADRV903x to allow the selection and mapping of data samples from 64 data sample sources. Figure 44 below illustrates the possible data sample sources available to each sample crossbars.

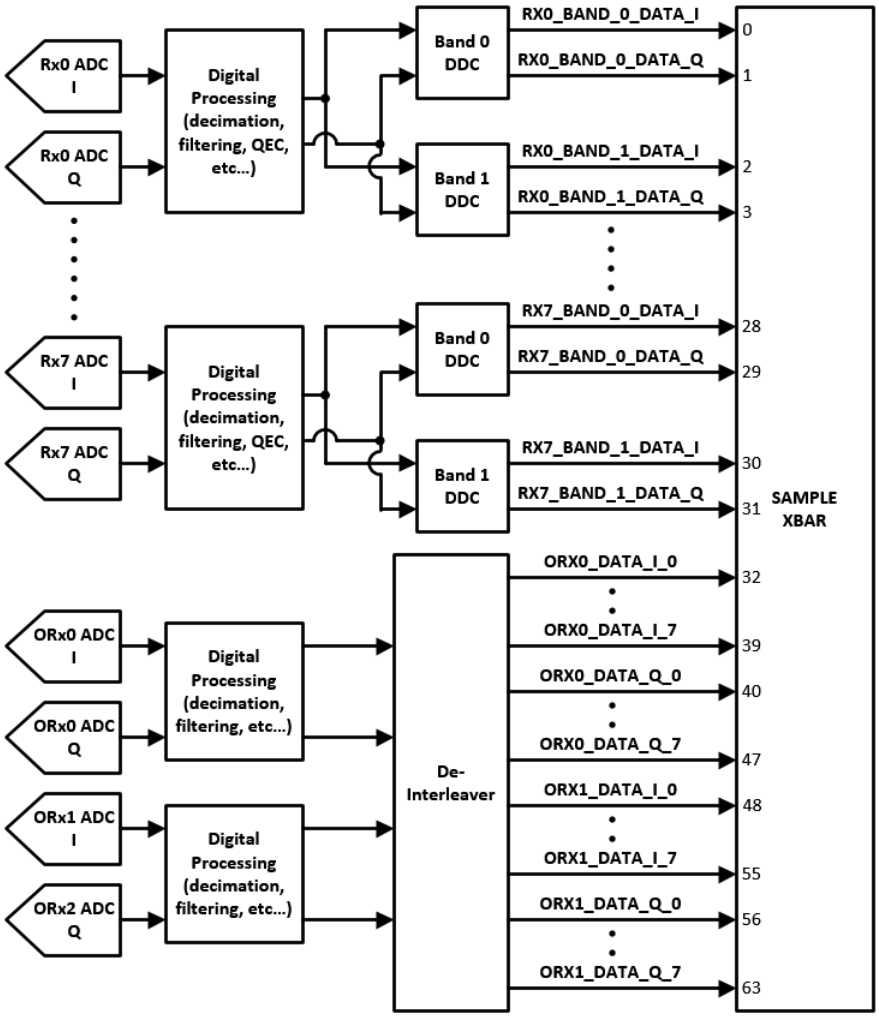


Figure 44. Sample Data Sources Available to Each Sample Crossbar

The data sample sources can be categorized as follows:

- Data samples from one of the DDC bands within an Rx datapath (eight Rx datapaths, two DDC bands per Rx datapath and two data samples going through each Rx datapath, resulting in 32 possible data sample sources)

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

- Data samples from an ORx datapath (two ORx datapaths, two data samples going through each ORx datapath at the IQ data rate and stored in the de-interleaver, and up to 16 data samples per ORx datapath output at a lower data rate by the de-interleaver resulting in 32 possible data sources)

Table 21 provides a list of all the sample crossbar input value along with the corresponding sample data sources.

Table 21. Sample Crossbar Input Selection Values

Sample Crossbar Input Value	Data Sample Source
0	RX0_BAND_0_DATA_I
1	RX0_BAND_0_DATA_Q
2	RX0_BAND_1_DATA_I
3	RX0_BAND_1_DATA_Q
4	RX1_BAND_0_DATA_I
5	RX1_BAND_0_DATA_Q
6	RX1_BAND_1_DATA_I
7	RX1_BAND_1_DATA_Q
8	RX2_BAND_0_DATA_I
9	RX2_BAND_0_DATA_Q
10	RX2_BAND_1_DATA_I
11	RX2_BAND_1_DATA_Q
12	RX3_BAND_0_DATA_I
13	RX3_BAND_0_DATA_Q
14	RX3_BAND_1_DATA_I
15	RX3_BAND_1_DATA_Q
16	RX4_BAND_0_DATA_I
17	RX4_BAND_0_DATA_Q
18	RX4_BAND_1_DATA_I
19	RX4_BAND_1_DATA_Q
20	RX5_BAND_0_DATA_I
21	RX5_BAND_0_DATA_Q
22	RX5_BAND_1_DATA_I
23	RX5_BAND_1_DATA_Q
24	RX6_BAND_0_DATA_I
25	RX6_BAND_0_DATA_Q
26	RX6_BAND_1_DATA_I
27	RX6_BAND_1_DATA_Q
28	RX7_BAND_0_DATA_I
29	RX7_BAND_0_DATA_Q
30	RX7_BAND_1_DATA_I
31	RX7_BAND_1_DATA_Q
32	ORX0_DATA_I_0
33	ORX0_DATA_I_1
34	ORX0_DATA_I_2
35	ORX0_DATA_I_3
36	ORX0_DATA_I_4
37	ORX0_DATA_I_5
38	ORX0_DATA_I_6
39	ORX0_DATA_I_7
40	ORX0_DATA_Q_0
41	ORX0_DATA_Q_1
42	ORX0_DATA_Q_2
43	ORX0_DATA_Q_3

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

Table 21. Sample Crossbar Input Selection Values (Continued)

Sample Crossbar Input Value	Data Sample Source
44	ORX0_DATA_Q_4
45	ORX0_DATA_Q_5
46	ORX0_DATA_Q_6
47	ORX0_DATA_Q_7
48	ORX1_DATA_I_0
49	ORX1_DATA_I_1
50	ORX1_DATA_I_2
51	ORX1_DATA_I_3
52	ORX1_DATA_I_4
53	ORX1_DATA_I_5
54	ORX1_DATA_I_6
55	ORX1_DATA_I_7
56	ORX1_DATA_Q_0
57	ORX1_DATA_Q_1
58	ORX1_DATA_Q_2
59	ORX1_DATA_Q_3
60	ORX1_DATA_Q_4
61	ORX1_DATA_Q_5
62	ORX1_DATA_Q_6
63	ORX1_DATA_Q_7

ORx De-Interleaving

A de-interleaver block is at the end of the ORx digital datapaths and before the framer inputs, as illustrated in Figure 44. The de-interleaver stores ORx data samples coming at the original ORx IQ data rate and can output those data samples in parallel at a lower data rate. The de-interleaver allows consecutive ORx ADC samples to be separated and placed anywhere in the frame. An overview of the de-interleaver block is in Figure 45.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

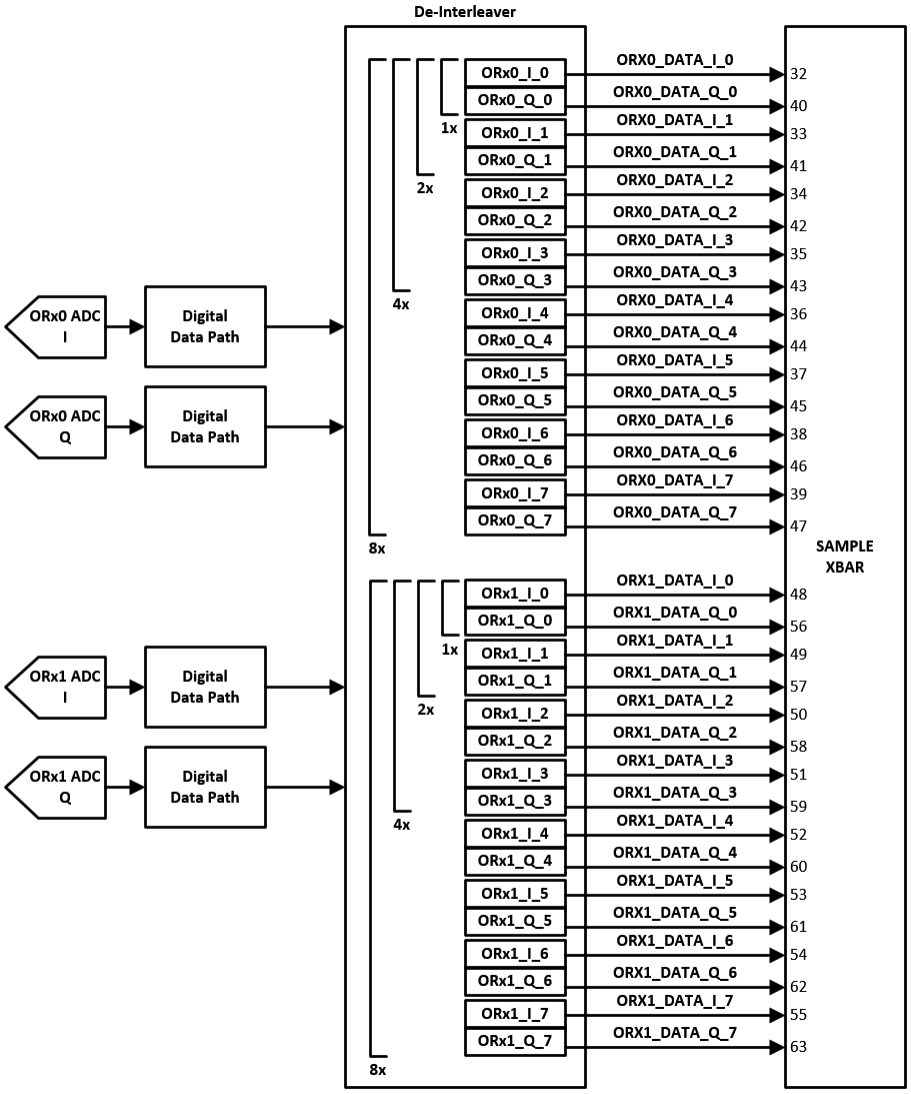


Figure 45. ORx De-Interleaver

The de-interleaver can operate in 4 modes as described in Table 22.

Table 22. De-Interleaver Operating Modes

De-Interleaver Mode	Mode Description
1x (bypass)	This corresponds to a bypass mode. In this mode, only the ORx0_I/Q_0 and ORx1_I/Q_0 samples are occupied in the de-interleaver and are output at the same sample rate as the input IQ sample rate
2x	In this mode, the ORx0_I/Q_0/1 and ORx1_I/Q_0/1 samples are occupied in the de-interleaver and are output in parallel at a sample rate equal to the input IQ sample rate divided by 2
4x	In this mode, the ORx0_I/Q_0/1/2/3 and ORx1_I/Q_0/1/2/3 samples are occupied in the de-interleaver and are output in parallel at a sample rate equal to the input IQ sample rate divided by 4
8x	In this mode, all the samples are occupied in the de-interleaver and are output in parallel at a sample rate equal to the input IQ sample rate divided by 8

The de-interleaver mode can be set independently for the ORx0 samples and for the ORx1 samples.

The de-interleaver allow the repetition of ORx ADC samples in the frame. For example, let's consider a case where the goal is to repeat every sample in order to fill the lane, e.g. AABBCCDD. Using a standard mode with S set to two will not achieve this, it results in two consecutive samples being repeated, i.e. ABABCDCD. Using the de-interleaver in 2x mode allows for the placement of consecutive samples in any pattern.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

By connecting the output of the de-interleaver to the input of the sample crossbar, any selection can be created. This example is illustrated in Figure 46.

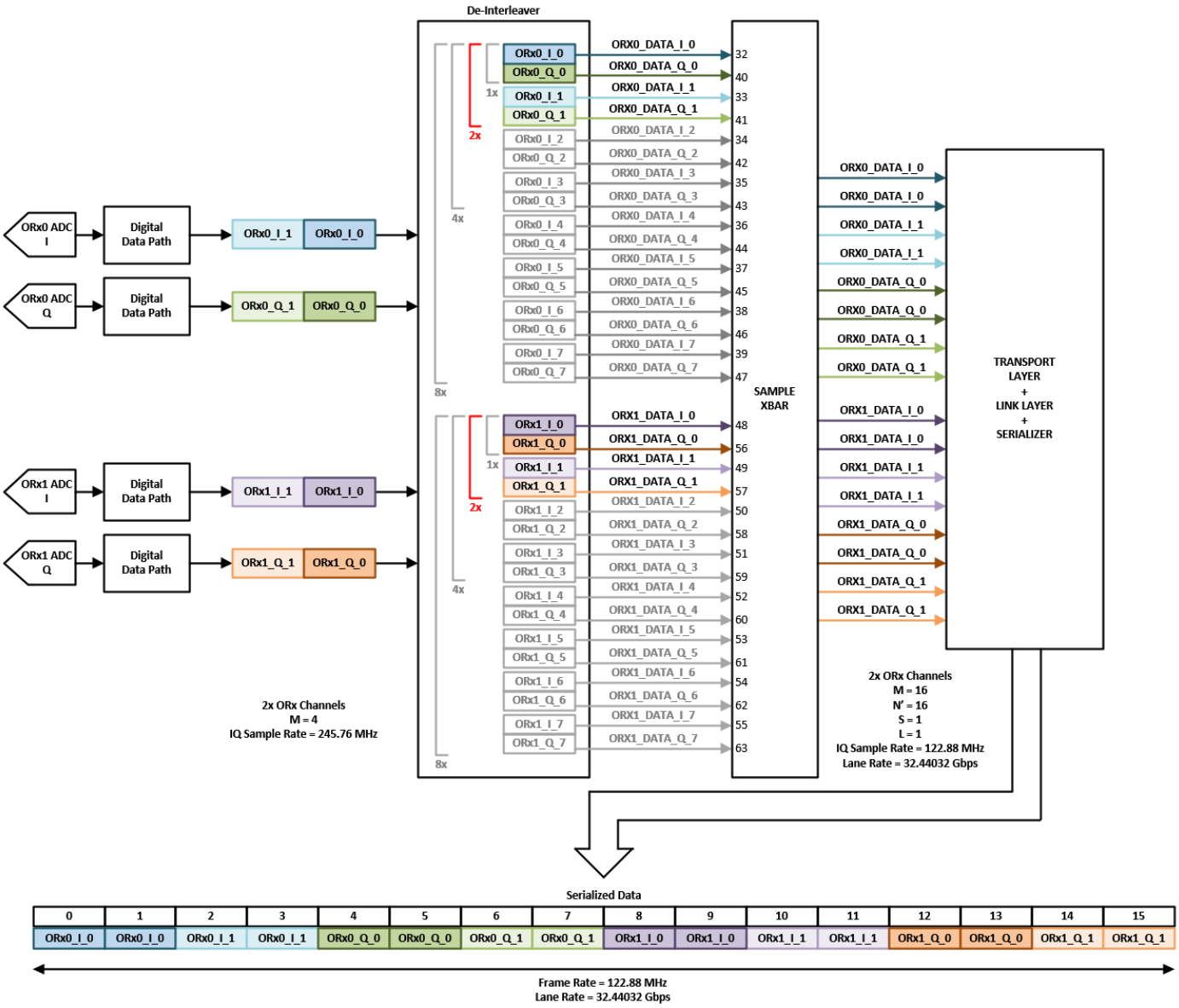


Figure 46. De-Interleaver 2x Mode Example

Transport Layer

The transport layer of a framer is responsible for mapping converter data samples, consisting each of N' bits, into octets that are spread across L lanes. Figure 47 illustrates the transport layer in a framer. The transport layer also handles the SYSREF signal that is used to achieve deterministic latency.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

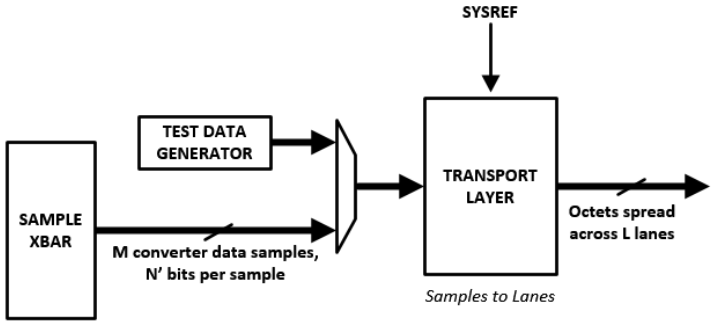


Figure 47. High-Level Overview of the Transport Layer in a Framer

The mapping changes depending on the number of lanes (L), the number of converters (M), the samples bit width (N'), and the number of samples per converter (S). Figure 48 and Figure 49 provide two examples of converter data samples mapping by the transport layer for two different sets of M, N', S and L parameters.

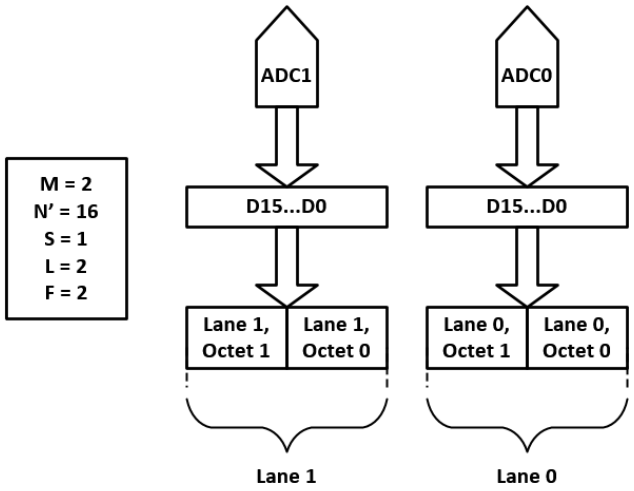


Figure 48. Converter Data Samples Mapping Example with M = 2, N' = 16, S = 1, and L = 2

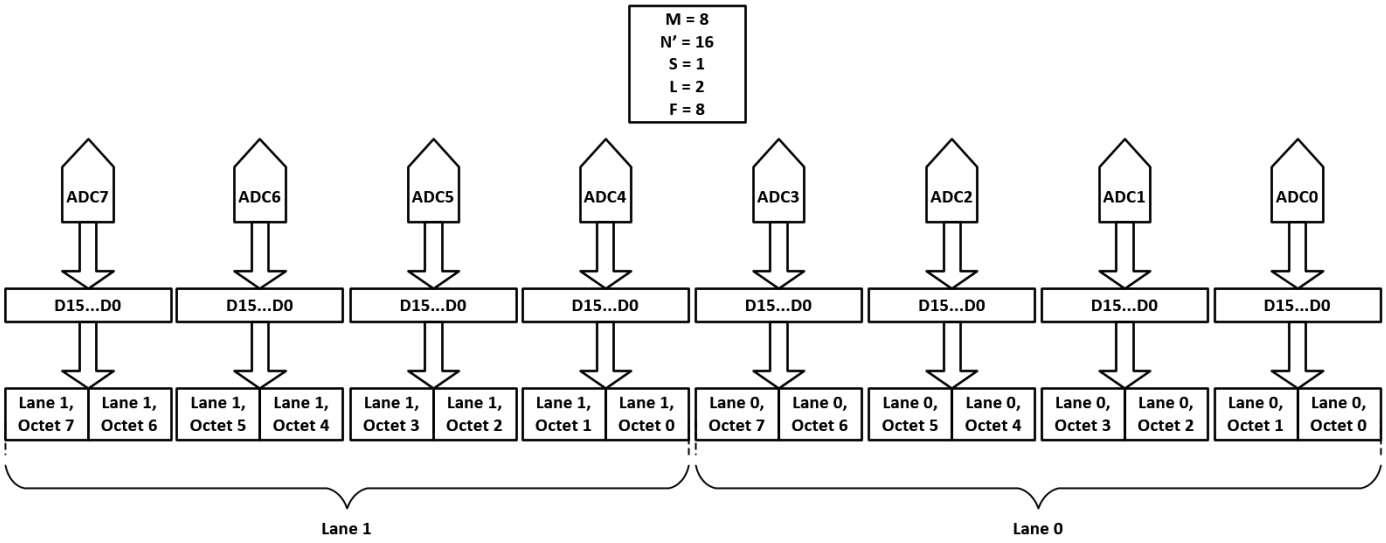


Figure 49. Converter Data Samples Mapping Example with M = 8, N' = 16, S = 1 and L = 2

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

SYSREF Signal

The SYSREF signal provides a timestamp used to achieve deterministic latency and multichip synchronization. For all three framers in the ADRV903x, the transport layer handles the SYSREF signal.

Link Layer

In JESD204B mode, the link layer of a framer is responsible for performing 8B/10B encoding on the data received from the transport layer. It handles Code Group Synchronization (CGS), generates the Initial Lane Alignment Sequence (ILAS) when required, and it performs character replacement during the transmission of user data. It can optionally perform data scrambling on the data received from the transport layer prior to 8B/10B encoding. The link layer also handles the SYNCIN~ signal.

In JESD204C mode, the link layer of a framer is responsible for scrambling the data received from the transport layer, and then performing 64B/66B encoding by inserting a 2-bit sync header before a block of 64 bits of data. The 2-bit sync headers are used to transmit information such as synchronization information, 12 CRC bits or FEC bits.

Figure 50 provides a high-level overview of the link layer in a framer.

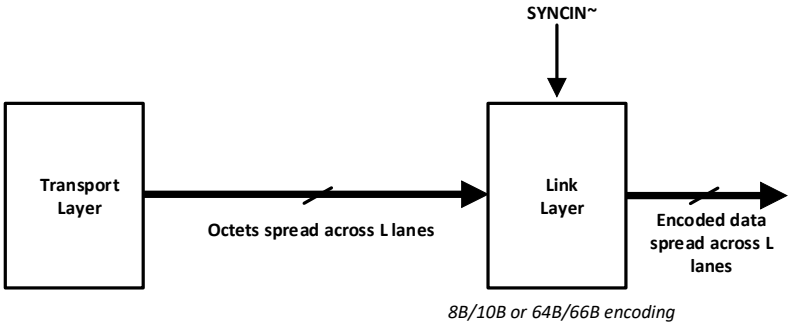


Figure 50. High-Level Overview of the Link Layer in a Framer

SYNCIN~ Signal

In JESD204B mode, the SYNCIN~ signal is sent by the receiver to the transmitter to request the start of the synchronization process, which begins with CGS. For all three framers in the ADRV903x, the SYNCIN~ signal is handled by the link layer in JESD204B mode. The ADRV903x has three SYNCIN~ inputs, which can be mapped to any of the framers through a SYNCIN~ crossbar. Each framer has its own SYNCIN~ input signal. This is illustrated in Figure 51 below.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

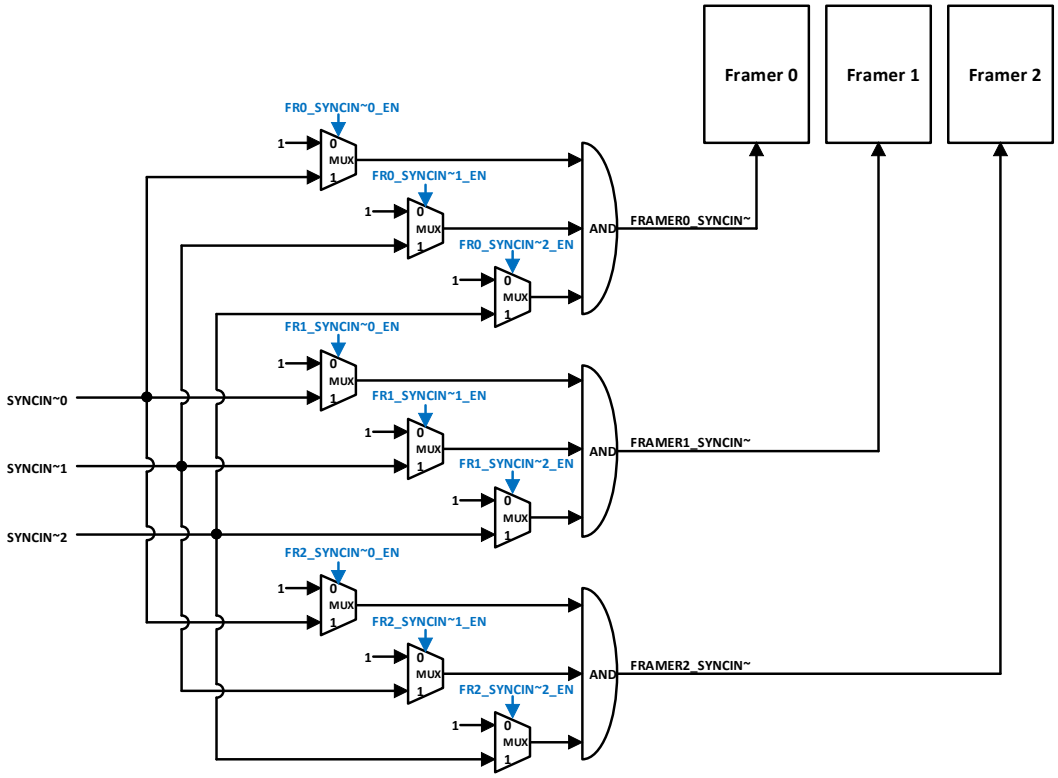


Figure 51. SYNCIN~ Crossbar

Lane Crossbars

Each of the ADRV903x framers includes a lane crossbar after its link layer to enable the routing of the used logical lanes within the framer block to the desired physical lanes. The framer 0 lane crossbar enables the mapping of any of the framer 0 link layer eight logical output lanes to any of its eight output lanes. The framer 1 lane crossbar enables the mapping of any of the framer 1 link layer four logical output lanes to any of its eight output lanes. The framer 2 lane crossbar enables the mapping of any of the framer 2 link layer 2 logical output lanes to any of its eight output lanes. This is illustrated in Figure 52 for framer 0, Figure 53 for framer 1 and Figure 54 for framer 2.

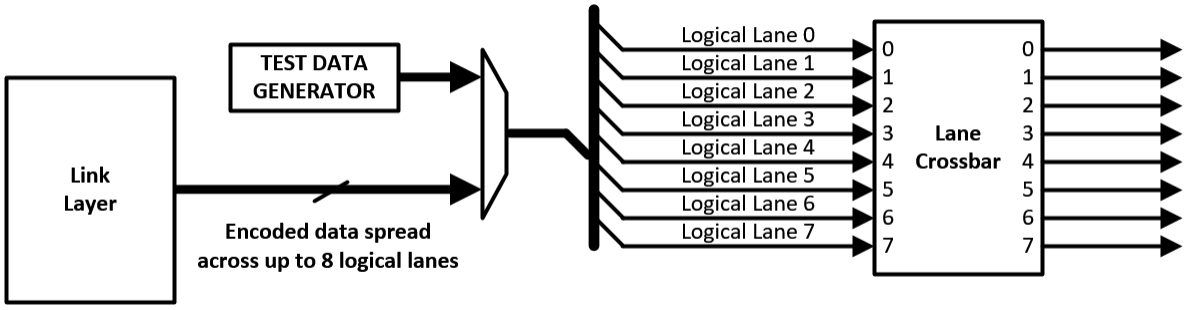


Figure 52. Framer 0 Lane Crossbar

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

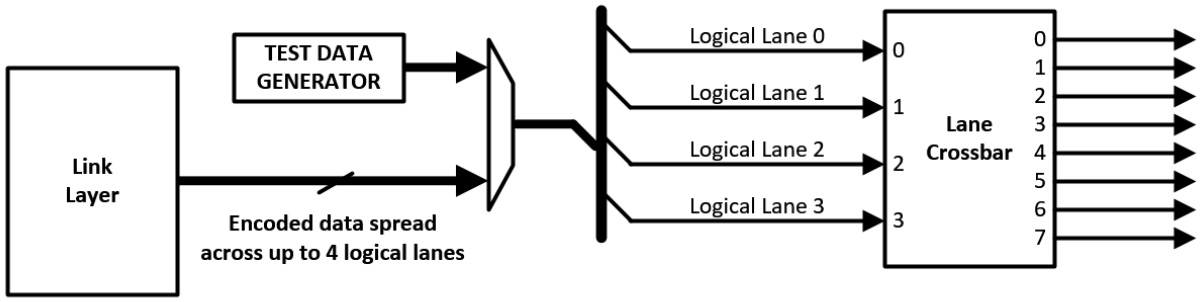


Figure 53. Framer 1 Lane Crossbar

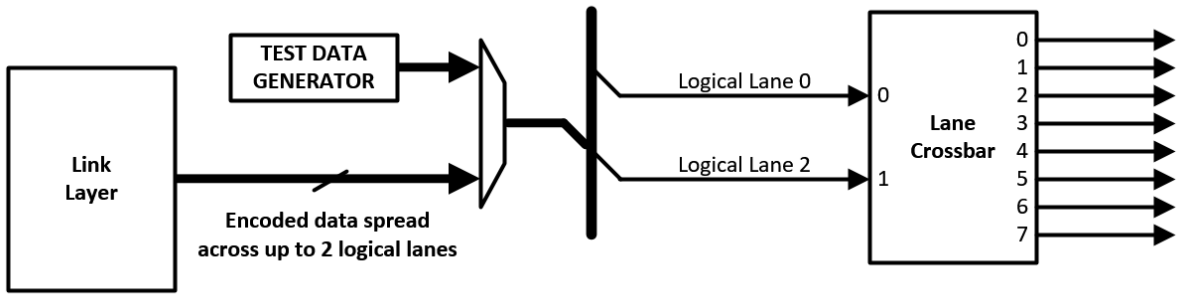


Figure 54. Framer 2 Lane Crossbar

OR Gates

The lane crossbars outputs for all the framers get OR'ed together using OR gates before being provided to the physical layer. For example, the lane 0 output of the framer 0 lane crossbar, the lane 0 output of the framer 1 lane crossbar and the lane 0 output of the framer 2 lane crossbar all go the same OR gate. The output of that OR gate goes to the serializer physical layer and ultimately goes to the SERDOUT0 output pin. Similarly, the lane 1 output of the framer 0 lane crossbar, the lane 1 output of the framer 1 lane crossbar and the lane 1 output of the framer 2 lane crossbar all go the same OR gate. The output of that OR gate goes to the serializer physical layer and ultimately goes to the SERDOUT1 output pin. The same principle applies to all the other lanes. Only one of the three lanes going to an OR gate can therefore be used at a time. Figure 55 illustrates this for lane 0.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

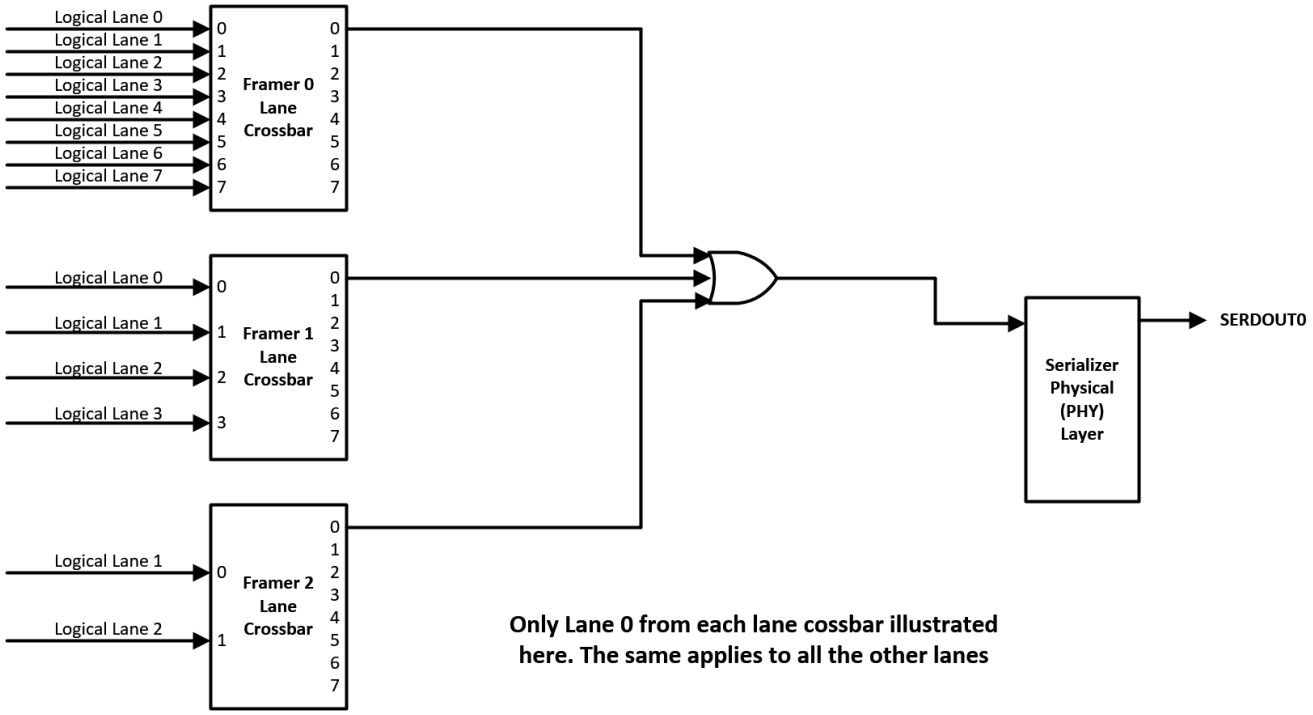


Figure 55. OR Gate Example for Lane 0

Test Data Generators

Each of the three framers feature a Test Data Generator that can be used to insert test data in either of the following two positions:

1. At the input of the Transport Layer (enables checking of transport layer, link layer and serializer PHY all together)
2. At the input of the Lane Crossbar (enables checking of serializer PHY only)

The Test Data Generator can be set in the following modes:

- Disabled: Test Data Generator not used and data from the selected data sample sources used
- Checkerboard mode: Test Data Generator enabled, and checkerboard pattern transmitted at its output
- Word toggle mode: Test Data Generator enabled, and word toggle pattern transmitted at its output
- PRBS31 mode: Test Data Generator enabled, and PRBS31 pattern transmitted at its output
- PRBS15 mode: Test Data Generator enabled, and PRBS15 pattern transmitted at its output
- PRBS7 mode: Test Data Generator enabled, and PRBS7 pattern transmitted at its output
- Ramp mode: Test Data Generator enabled, and ramp pattern transmitted at its output
- User Repeat mode: Test Data Generator enabled, and user specified pattern transmitted repeatedly at its output
- User Single mode: Test Data Generator enabled, and user specified pattern transmitted one time at its output

Clocking

The SERDES PLL provides the clock that is used to generate the clocks that are used to provide timing to the Framers Link Layer blocks and to the output of the Framer Transport Layer blocks.

IQ Sample Rate and Output Lane Data Rate Relationship Between Framers

The lane data rate at the output of the Serializer PHY depends on the on the number of lanes (L), the number of converters (M), the samples bit width (N') and the IQ sample rate.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

In JESD204B mode, the framer serialized output lane data rate is defined by the following equation:

$$Serialized\ Ouput\ Lane\ Data\ Rate = \frac{\left[M \times N' \times \left(\frac{10}{8}\right)\right]}{L} \times IQ\ Sample\ Rate \quad (6)$$

In JESD204C mode, the framer serialized output lane data rate is defined by the following equation:

$$Serialized\ Output\ Lane\ Data\ Rate = \frac{\left[M \times N' \times \left(\frac{66}{64}\right)\right]}{L} \times IQ\ Sample\ Rate \quad (7)$$

The three framers can operate at different IQ sample rates. The ratio of the IQ sample rates between two framers must be a power of two (2×, 4×, 8×, etc...).

Similarly, the three framers can operate at different serialized output lane data rates. The ratio of the serialized output lane data rates between two framers must also be a power of two (2×, 4×, 8×, etc...).

Link Sharing

Link sharing, only allowed in TDD mode, enables a reduction in the number of physical lanes used. Rx data is put on the lanes in the receive time slot and the ORx data is put on the same lanes during transmit time slots.

Figure 56 shows an example where Framer 0 would be used to handle the Rx data and the ORx data in link sharing mode. The same four physical lanes would be used to transmit the Rx data during the Receive time slots, and the ORx data during the Transmit time slots.

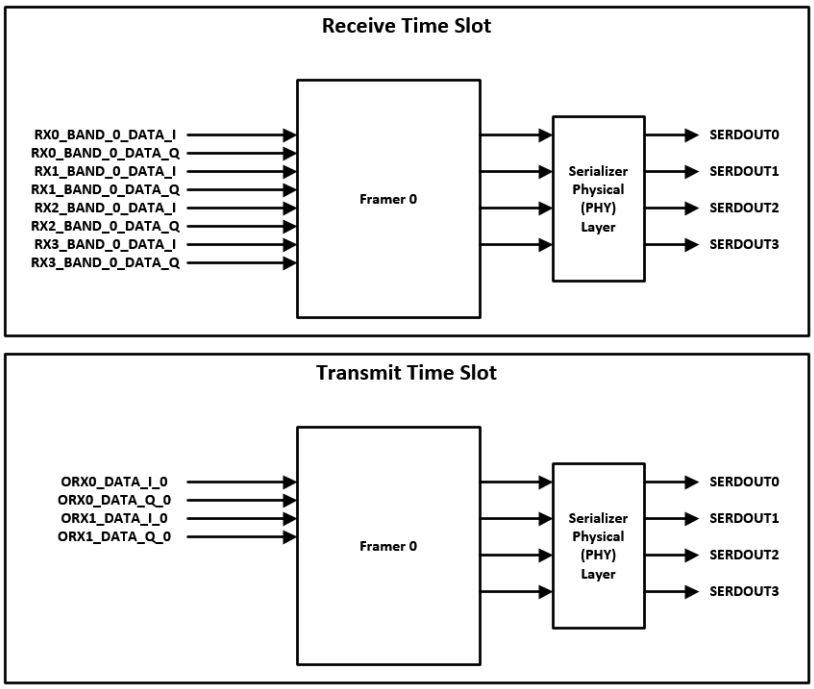


Figure 56. Link Sharing Example

JESD204B/C DEFRAMERS

Deframers receive encoded data from the SERDIN lanes, establish the JESD link, perform 8B/10B (JESD204B) or 64B/66B (JESD204C) decoding on the data and unpack the decoded octets into converter samples for the transmitter paths' DACs.

The ADRV903x has two deframers to allow flexibility in mapping SERDES inputs to transmitter DACs. A dual-deframer link configuration allows for the possibility of powering down one deframer and its associated transmitters during times of low data traffic without disturbing the other deframer and its associated transmitters. The deframers have highly configurable interface rates and supports up to eight SERDES lanes. To assist in debugging, they contain an internal PRBS receiver allowing PRBS patterns to be verified to check the link's signal integrity.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

Figure 57 provides a high-level overview of the deframers in the ADRV903x. Further details are provided in the sections that follow. Note that the flow of data in all figures in this section is from right to left.

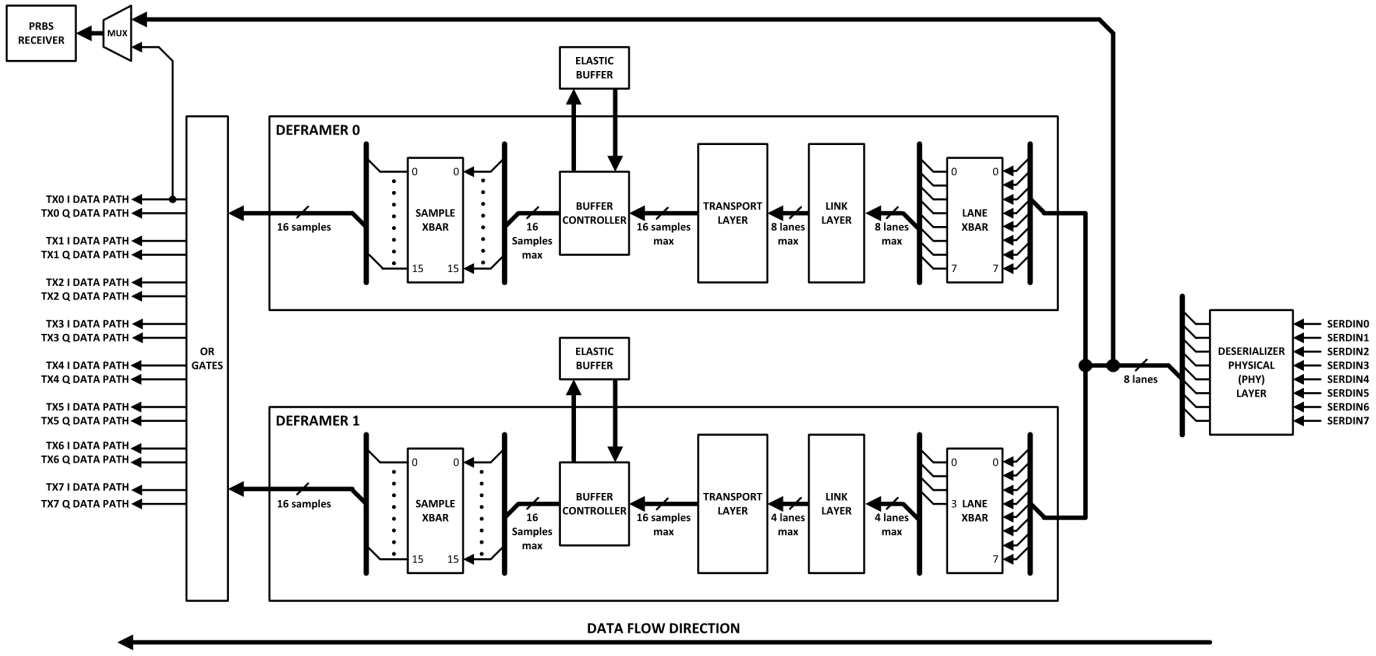


Figure 57. High Level Overview of the JESD204B/C Deframers

JESD204B/C Deframers Parameters

Table 23 provides a list of the supported deframer parameter values. Note that not all combinations of those deframer parameter values are supported.

Table 23. List of JESD204B/C Deframers Parameters

Parameter	Parameter Description	Possible Parameter Values[1]
M	Number of converter devices	1 to 16
L	Number of Lanes to receive and decode data from	1 (both deframers), 2 (both deframers), 4 (both deframers), 8 (deframer 0 only)
F	JESD204B: Number of octets per lane in a frame cycle	2, 3, 4, 6, 8, 12, 16
	JESD204C: Used in conjunction with K to set the extended multiblock period. F is calculated as in JESD204B mode.	2, 3, 4, 6, 8, 12, 16
S	Number of samples per converter per frame cycle	1, 2
N'	Number of bits in a sample	12, 16, 24
K	JESD204B: Number of frames in one multiframe.	1 to 32. F × K must be a multiple of four and (20 ≤ F × K ≤ 256)
	JESD204C: Used in conjunction with F to set the extended multiblock period	K × F = 256 × E, K ≤ 256
E	JESD204C only: Number of blocks in an extended multiblock	1 to 32
CS	Number of control bits per converter sample	0 to 3
HD	High Density mode	0, 1

[1] Not all combinations of the framer parameter values are supported.

Lane Crossbars

Each ADRV903x deframer has a lane crossbar before its link layer to enable the routing of physical lanes to the desired logical lanes within the deframer block. The deframer 0 lane crossbar enables the mapping of any of the eight physical input lanes to any of its eight logical input lanes.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

The deframer 1 lane crossbar enables the mapping of any of the eight physical input lanes to any of its four logical input lanes. Figure 58 shows the lane crossbars in detail.

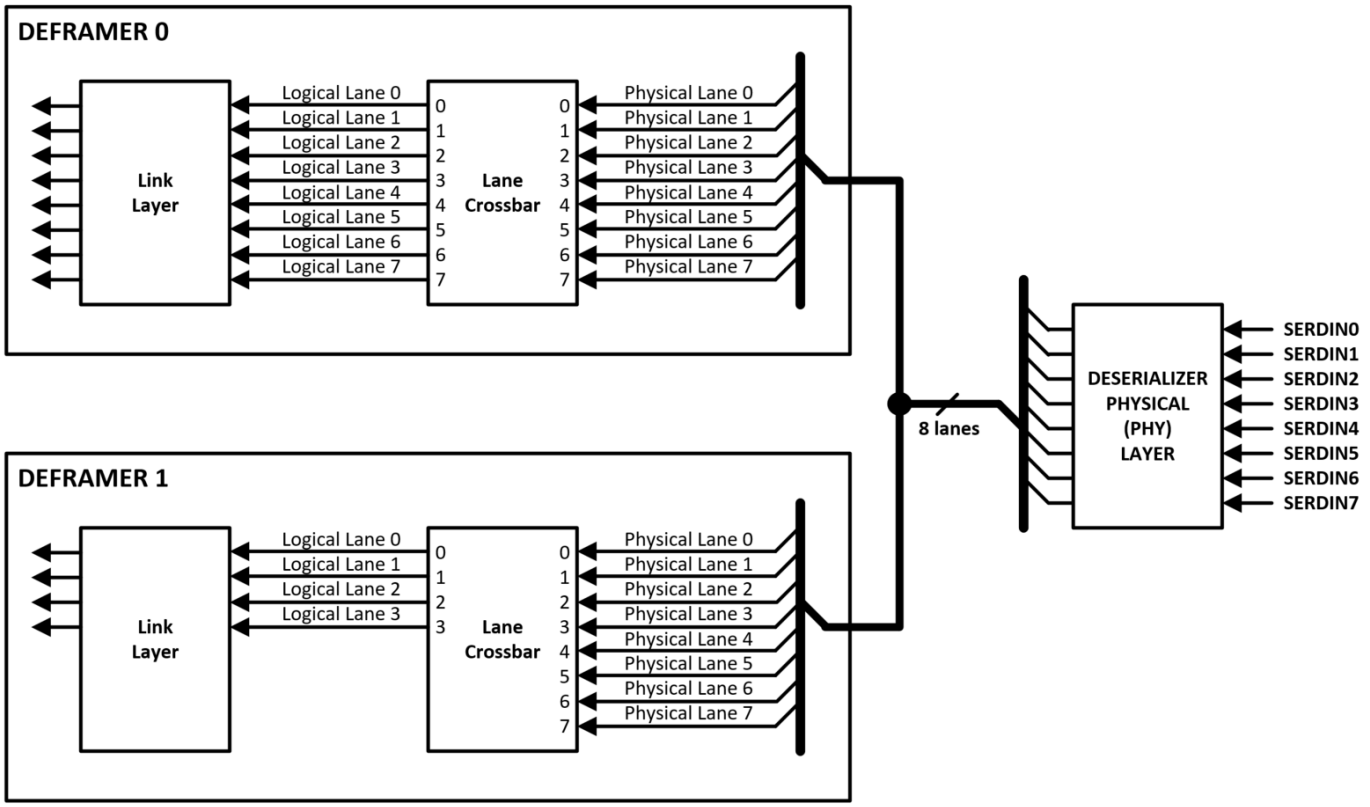


Figure 58. Deframer 0 and 1 Lane Crossbars

Link Layer

The link layer of a deframer is responsible for establishing the JESD link.

In JESD204B mode, the link layer controls the SYNCOUT~ signal, synchronizes to Code Group Synchronization (CGS), receives and verifies the Initial Lane Alignment Sequence (ILAS) and performs 8B/10B decoding on the data received from the SERDES input lanes. It also performs character re-insertion as necessary during the reception of user data and can optionally perform data descrambling on the received data after 8B/10B decoding.

In JESD204C mode, the link layer detects the location of the 2-bit sync headers, performs 64B/66B decoding by removing the 2-bit sync headers before each block of 64 bits of data and then descrambles the remaining 64 bits of received data. The 2-bit sync headers are decoded to extract information such as EoEMB synchronization information, and CRC or FEC bits. Note that FEC is not supported in the deframers and these bits, if sent, are ignored. Figure 59 provides a high-level overview of the link layer in a framer.

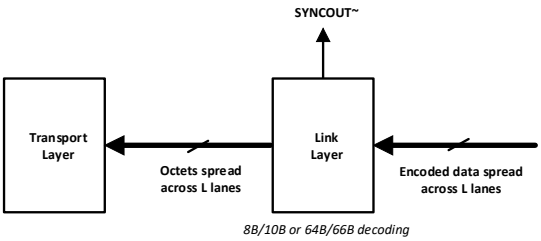


Figure 59. High-Level Overview of the Link Layer in a Deframer

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

SYNCOUT~ Signal

In JESD204B mode, the SYNCOUT~ signal is controlled by the transmitter and used to indicate to the receiver, the state of link synchronization. The SYNCOUT~ signal is not used in JESD204C mode. The ADRV903x has two SYNCOUT~ output pins and each deframer has its own SYNCOUT~ output signal. These signals can be combined and mapped to either of the SYNCOUT~ pins through muxes and AND gates. This is illustrated in Figure 60.

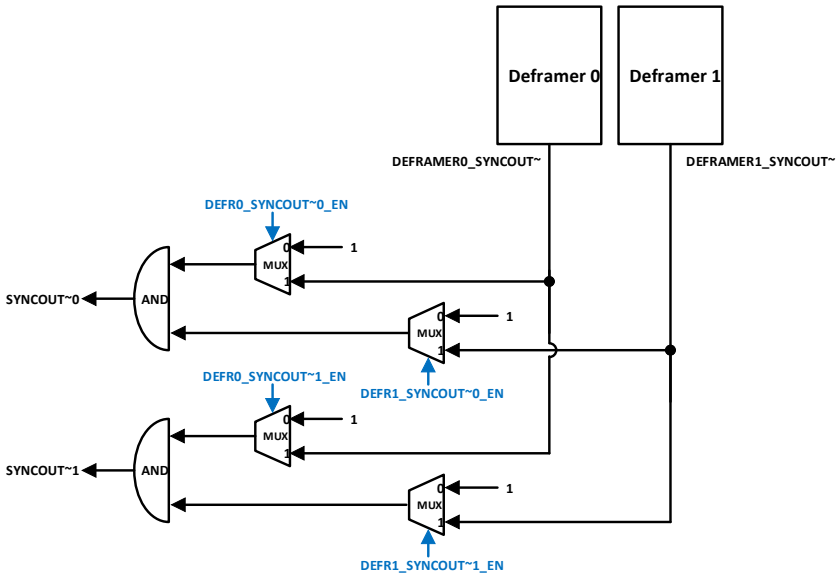


Figure 60. Deframer to SYNCOUT~ Output Options

Transport Layer

The transport layer of a deframer is responsible for de-skewing the lanes, unpacking the octets that are spread across L lanes and combining them into M converter data samples, consisting each of N' bits. The mapping changes depending on the number of lanes (L), the number of converters (M), the samples bit width (N'), and the number of samples per converter (S). The transport layer also handles the SYSREF signal that is used to achieve deterministic latency. Figure 61 shows the transport layer in a deframer.

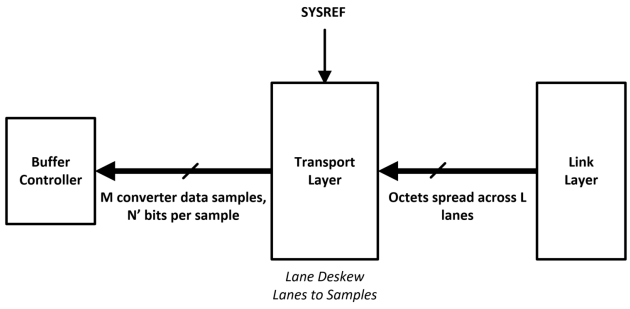


Figure 61. High-Level Overview of the Transport Layer in a Deframer

SYSREF Signal

The SYSREF signal is an externally applied global timing signal used to synchronize the part. The SYSREF signal is used initially for the multi-chip synchronization during chip initialization. The MCS during initialization ensures that all clock domains come into a deterministic phase with respect to the SYSREF signal. Subsequently, the SYSREF signal is used to reset the JESD core counters which then aligns the octet counters in all devices in the system. This is required to achieve deterministic latency as part of the subclass 1 specification. SYSREF is not needed if deterministic latency is not required.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

Elastic FIFO, Phase Adjustment and Deterministic Latency

An elastic FIFO is used to achieve deterministic latency for subclass 1 operation. The link layer in the deframer detects the LMFC/LEMC timing embedded in the incoming data and starts writing to the elastic FIFO at the start of an LMFC/LEMC period. The global LMFC/LEMC timing signal inside the ADRV903x, which has been synchronized by the SYSREF signal, triggers the reading of data from the elastic FIFO. The depth of the FIFO therefore represents the delta between the embedded LMFC/LEMC timing in the incoming data and the system wide LMFC/LEMC timing as set by SYSREF. A phase adjustment control allows adjustment of the LMFC/LEMC timing inside the ADRV903x versus the externally applied SYSREF signal, allowing tuning of the FIFO depth and overall link latency. It is essential that the FIFO depth is tuned to a suitable level as part of platform characterization, otherwise deterministic latency is not guaranteed. Details on obtaining a suitable FIFO depth are discussed in a separate section. Figure 62 shows the elastic buffer overview.

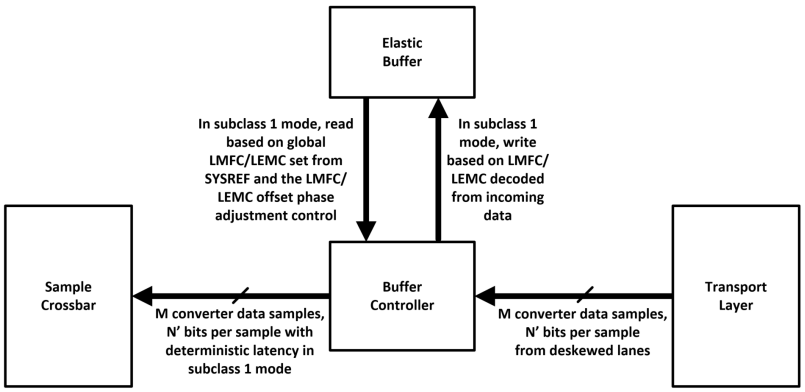


Figure 62. Elastic Buffer Overview

Sample Crossbars

A sample crossbar after the output of each deframer allows the mapping of the deframers' outputs to the transmitter DACs. Each sample crossbar contains sixteen 1:16 muxes and are used to map the deframer outputs to the desired Tx DAC. Each of the 16 muxes can select from any of the logical outputs of the deframer. Figure 63 illustrates the sample crossbar architecture for deframer 0. An identical crossbar is used for deframer 1.

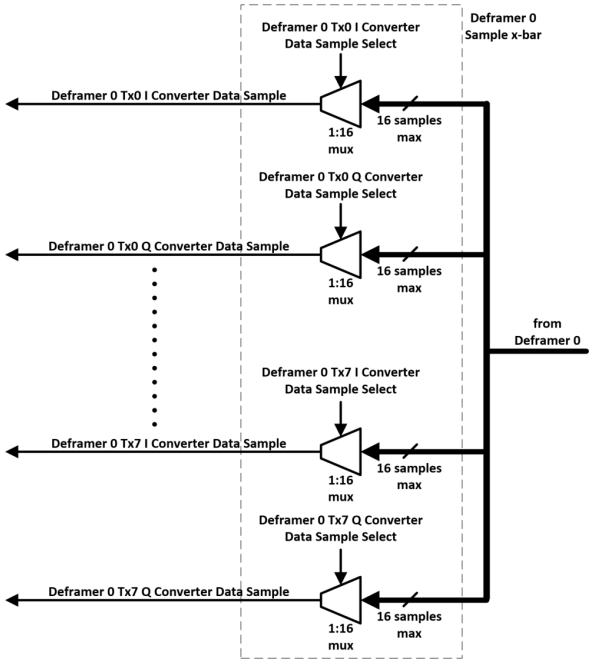


Figure 63. Deframer 0 Sample Crossbar

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

OR Gates

The 32 outputs from the two sample crossbars get OR'ed together using sixteen OR gates before being provided to the sixteen transmit DACs. For example, the sample 0 output of the deframer 0 sample crossbar and the sample 0 output of the deframer 1 sample crossbar go to the same OR gate. The output of that OR gate goes to the Tx0_I datapath. Similarly, the sample 1 output of the deframer 0 sample crossbar and the sample 1 output of the deframer 1 sample crossbar go to the same OR gate. The output of that OR gate goes to the Tx0_Q datapath. The same principle applies to all the other sample crossbar outputs. Therefore, the crossbars should be configured so that only one of the crossbars provides an active input to an OR gate. Figure 64 illustrates this for sample 0 and Figure 65 illustrates the full solution showing both sample crossbars.

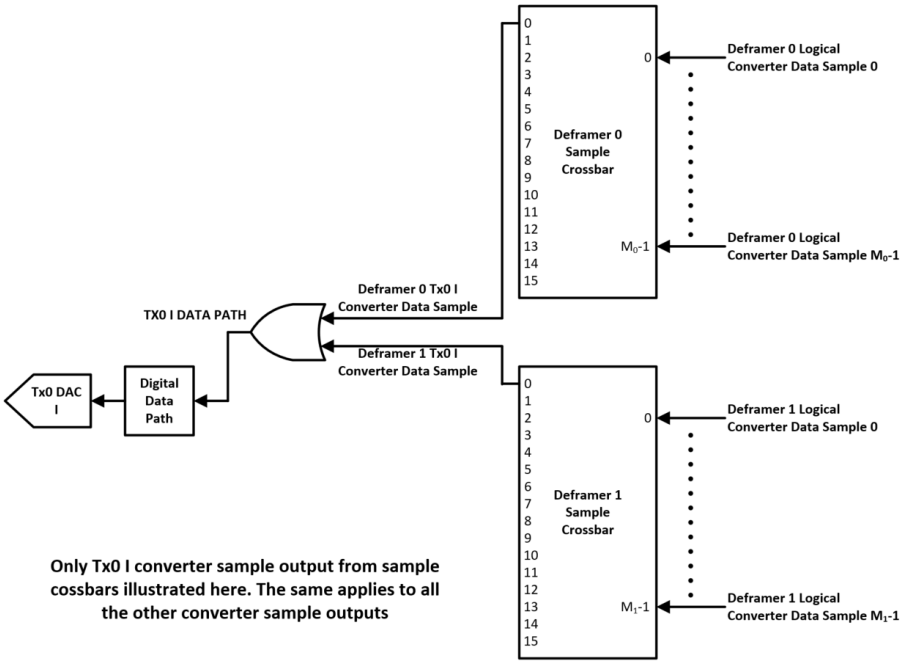


Figure 64. OR Gate Example for Sample 0

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

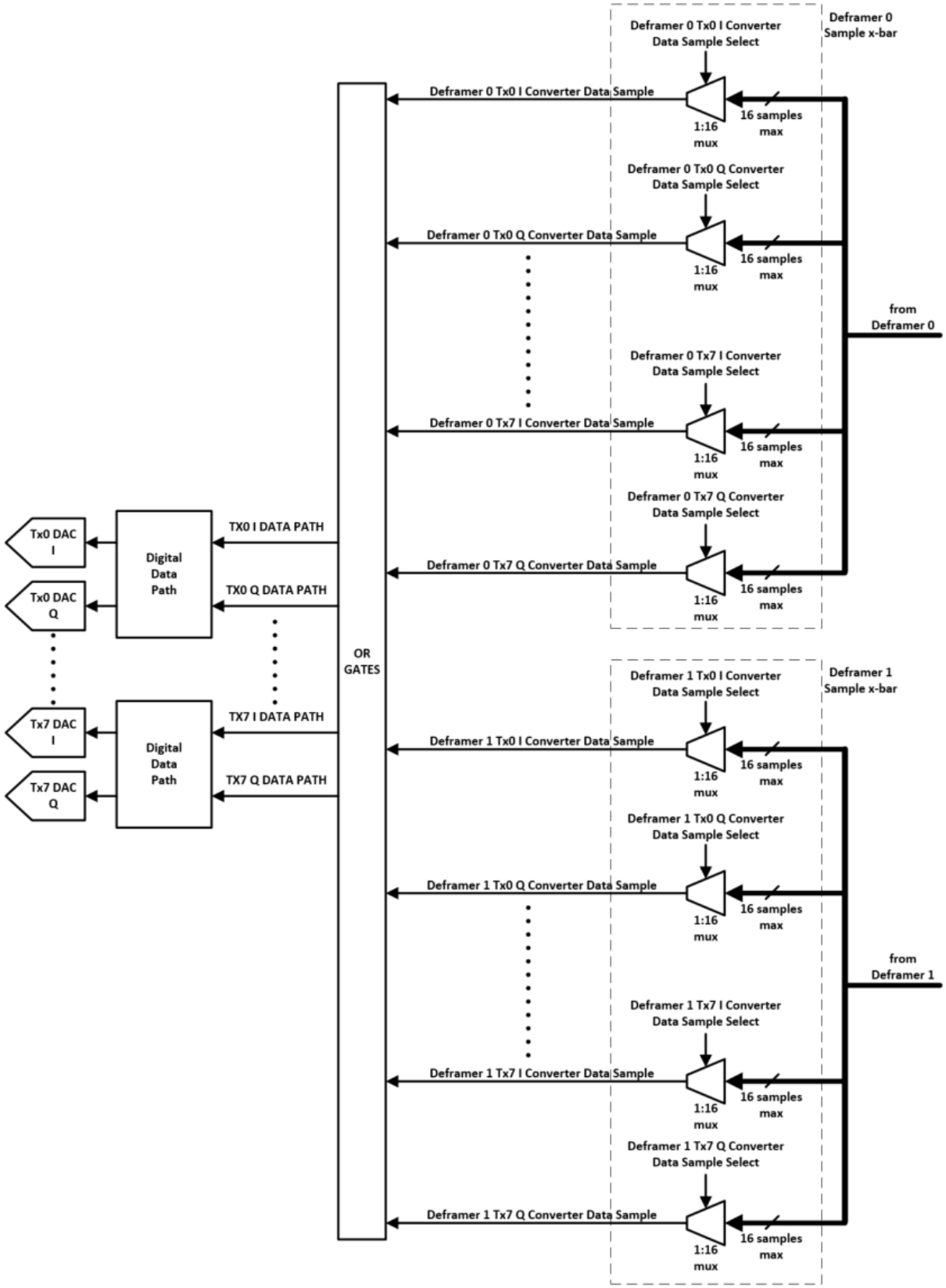


Figure 65. Sample Crossbars and OR Gates Allow Routing of Any Deframer Output to Any Tx DAC

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

PRBS Receiver

A PRBS receiver is available in the deframer block to check PRBS data on each SERDES input lane, or on the Tx0_I DAC output. When checking the PRBS on the SERDES input lanes, the raw SERDES data is checked. Therefore, the generator sending the data should apply the PRBS sequence directly to its SERDES output lanes. When checking the PRBS on the Tx0_I DAC output, the decoded and unpacked sample data is checked. Therefore, the generator sending the data should apply the PRBS sequence as data samples to the transport layer of its framer where they're packed and encoded before being sent across the SERDES link.

The PRBS receiver can be set in the following modes:

- Disabled: PRBS receiver is not used
- PRBS7 mode: PRBS receiver checks for PRBS7 pattern
- PRBS9 mode: PRBS receiver checks for PRBS9 pattern
- PRBS15 mode: PRBS receiver checks for PRBS15 pattern
- PRBS31 mode: PRBS receiver checks for PRBS31 pattern

Figure 66 shows the location of the PRBS Receiver in the deframer block. The PRBS receiver is circled in blue.

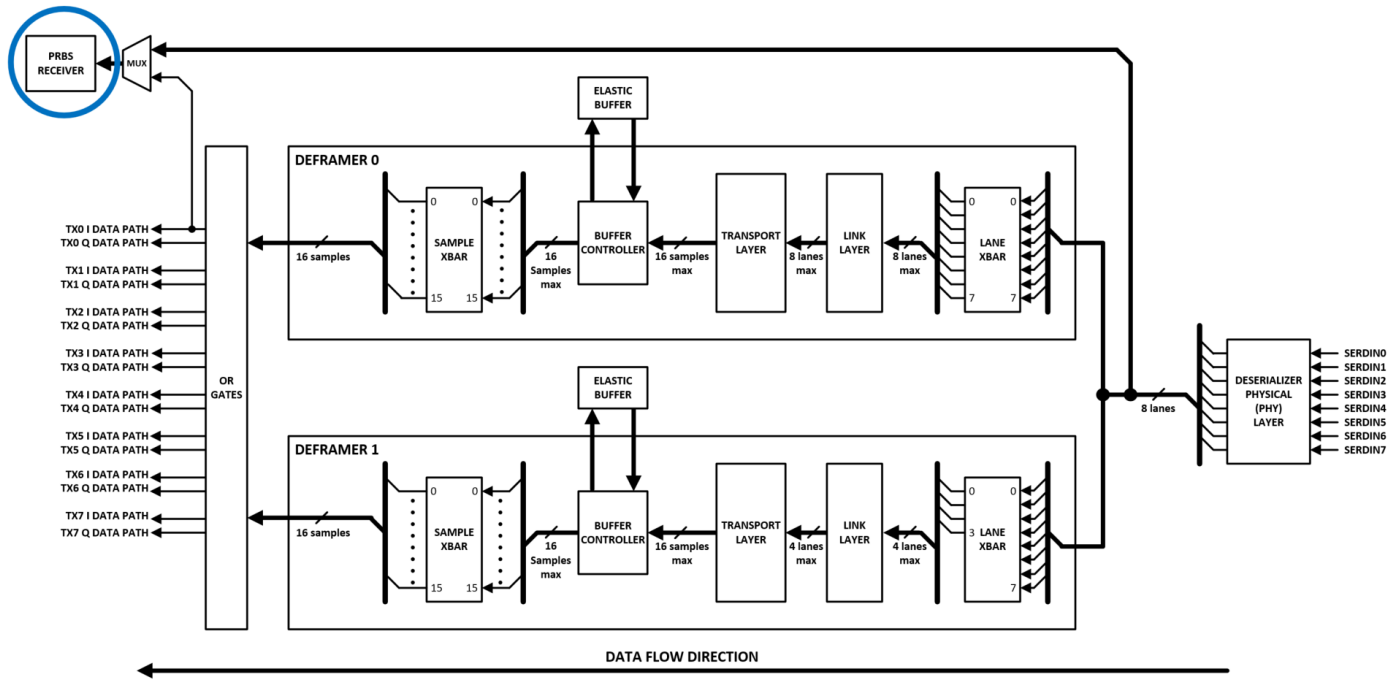


Figure 66. PRBS Receiver Location Circled in Blue

Clock Distribution

The SERDES PLL provides the clock that is used to derive the clocks that are provided to the Deframers Link Layers and to the input of the Deframer Transport Layers. Programming of the SERDES PLL and associated clock dividers is handled automatically by the embedded firmware, based on the profile information passed to the ADRV903x during initialization.

JESD PHY LAYER

The ADRV903x has eight serializer and deserializer lanes that can receive and transmit data. All these lanes use a single reference (SYSREFP/N) and device clock (DEVCLKP/N). The maximum lane rate that is supported on ADRV903x is 32,440.32 Mbps.

The JESD physical layer establishes a reliable channel between the transmitter (Tx) and the receiver (Rx). A high-level view of a typical JESD serial link is shown in Figure 67.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

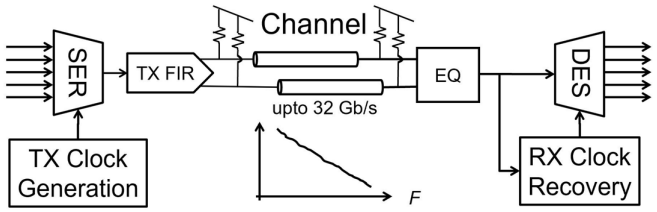


Figure 67. Overview of a Typical High-Speed Serial Link

The serializer above does a N:1 multiplexer on the transmitter. On its input there is a 40-bit or 66-bit parallel word, which is turned into a non-return-to-zero (or NRZ) waveform with lane rates up to 32,440.32 Mbps. This data is then driven by a JESD TX driver onto a terminated lane to the receiver. On the receiver side, a deserializer implements a 1-to-N demultiplexing of the data down to the original lower rate which is then sent to subsequent digital blocks to be decoded. There are no clocks transmitted on the serial link so a clock data recovery block is needed on the receiver side to find the optimum sampling phase of the clock. In Figure 67 note the frequency response of the channel. At higher data rates, Figure 67, large losses are seen across the channel causing the received signal to be attenuated resulting in large inter-symbol interference (ISI).

ISI is when the effects of a lossy transmission line cause a data symbol to be spread in time, into the symbol time slot of an adjacent symbol. Figure 68 shows an impulse response (data=..010000..) at the end of a lossy transmission line. Each color represents a different bit period. All the energy should be in the green section, which is referred to as the cursor. Energy in the blue, yellow and orange sections are caused by ISI. The time slot just prior to the impulse (in dotted red) is referred to as the pre-cursor, whereas the time slots after the impulse are referred to as first, second, and so on post cursors.

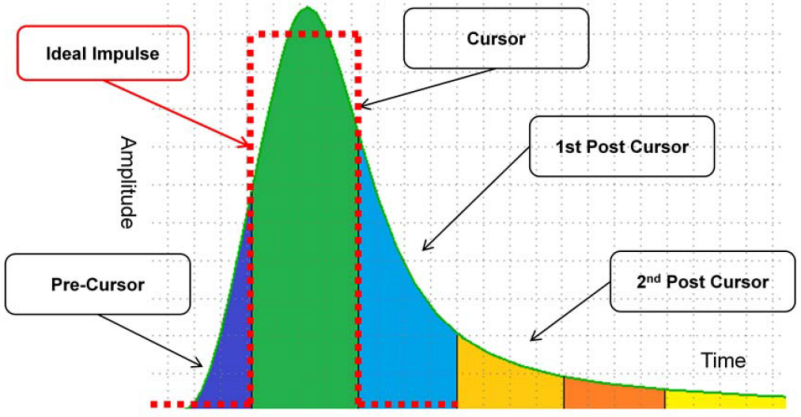


Figure 68. Inter-Symbol Interference of an Impulse Response

In order to overcome insertion loss caused by ISI, a pre-emphasis and de-emphasis (or equalization) circuits are used. An equalizer on the receiver side helps with de-emphasis, which will subtract the energy in the pre and post-cursors, so that only the energy at the cursor remains after the signal travels through the channel.

For both equalization and pre-emphasis, the idea is to apply the inverse of the channel transfer function to the signal being sent on the channel. This can be done at the input to the channel (pre-emphasis) or at the output of the channel (equalization), shown in Figure 69. The top diagram shows an impulse as it goes into the channel which acts as a low pass filter. The resulting waveform out of the channel shows the ISI effect of the impulse energy being spread into adjacent bit intervals. In the bottom diagram, the same impulse passes through a high-pass filter before being sent into the low-pass channel. The theoretical result is an impulse coming out of the channel.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

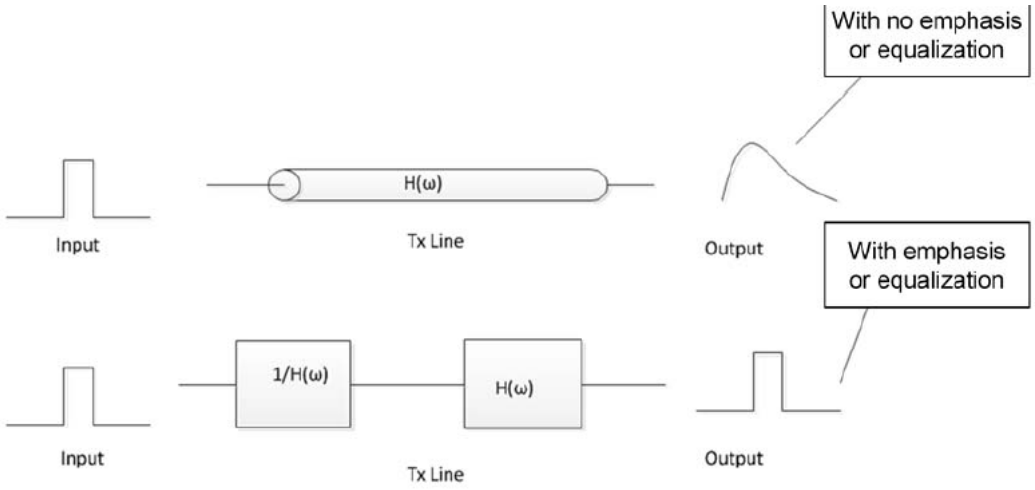


Figure 69. Theoretical Diagram of the Operation of Pre-emphasis and De-emphasis

The serializer implements a 3-tap FIR filter, whereas the deserializer implements CTLE (continuous time linear equalization), PGA (Programmable Gain Amplifier) and DFE (Decision Feedback Equalizer) to help compensate for the channel losses.

Serializer Physical (PHY) Layer

The amplitude of the serializer is represented by a 3-bit number that is not linearly weighted. The JESD204B/C transmitter mask requires a differential amplitude greater than 360 mV and less than 1V. The default amplitude is 0 (maximum amplitude setting).

Table 24. Serializer Amplitude Settings

Serializer Amplitude(decimal)	Average Differential Amplitude (V_{TT} = 1 V)
0	1 × V_{TT}
1	0.85 × V_{TT}
2	0.75 × V_{TT}
3	0.5 × V_{TT}

It is recommended to verify the eye diagram in the system after building a PCB to verify any layout related performance differences. If possible, the eye should be verified using an internal eye monitor after the equalizer circuit of the receiver as this shows the actual eye that the receiver circuit will receive.

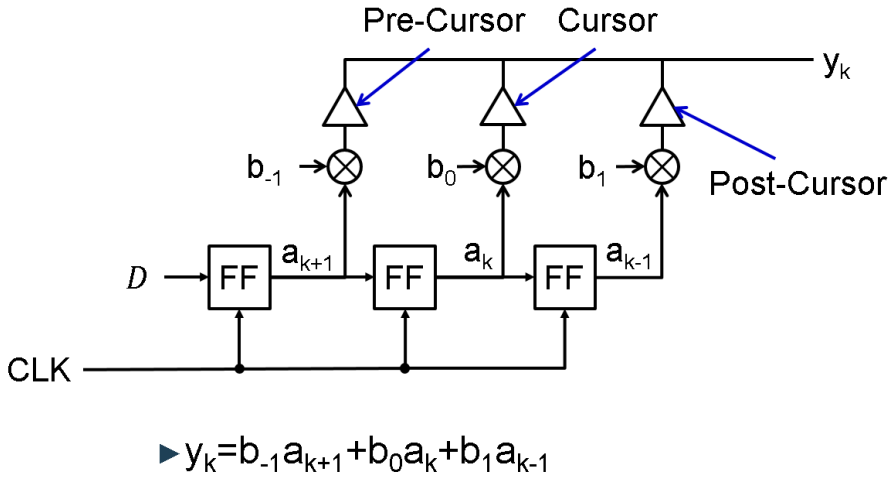


Figure 70. Serializer Emphasis Implementation

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

A three tap FIR equalizer is implemented in the serializer as shown in Figure 70. Here, the cursor, or largest tap weight multiplying a_k is in the center. There is a pre-cursor tap b_{-1} multiplying a_{k+1} and a post cursor tap b_1 multiplying a_{k-1} to realize the following difference equation for y_k. Transmit pre-emphasis is used as it's easier to realize bit delays with flip flops than trying to implement analog delays at the receiver.

The serializer pre-emphasis circuit allows boosting of the amplitude anytime the serial bit changes state. If no bit transition occurs, the amplitude is left unchanged. Pre-emphasis helps open the eye for longer PCB traces or when the parasitic loading of connectors has a noticeable effect. In most cases, to find the best setting, a simulation or measurement of the eye diagram with a high-speed scope at the receiver is recommended, or as mentioned above an internal eye monitor after the equalizer is the optimum solution. The serializer pre-emphasis is controlled by setting a pre-cursor and a post-cursor setting which are listed in Table 25 and Table 26. Use the API function adi_ADRV903x_SerLaneCfgSet() to set these parameters.

Table 25. Pre-Cursor Amplitude Settings

Emphasis (decimal)	Emphasis (dB)
0	0
1	3
2	6

Table 26. Post-Cursor Amplitude Settings

Emphasis (decimal)	Emphasis (dB)
0	0
1	3
2	6
3	9
4	12

Deserializer Physical (PHY) Layer

The deserializer connects the baseband IC to the Tx output via the JESD PHY. The ADRV903x has eight deserializer lanes that can be used to connect to the BBIC serializer for transmitting the baseband data. ADRV903x supports lane rates up to 32,440.32 Mbps. The deserializer PHY can either use the SERDES PLL or the CLK PLL to provide the reference clocks necessary to set up the JESD link. The reason to have two separate PLLs for clocking the JESD PHY is to support use cases where lane rates are not integer multiple of the sample rates and to support high data rates. Note that CLK PLL can only be used for lane rates up to 14.75 Gbps, above which only the SERDES PLL can be used.

To achieve high rates while having the VCO running between 8-16 GHz, the phase of the reference clock into PHY is divided to in-phase and quadrature phase. A high-level block diagram of the ADRV903x JESD deserializer PHY is in Figure 71.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

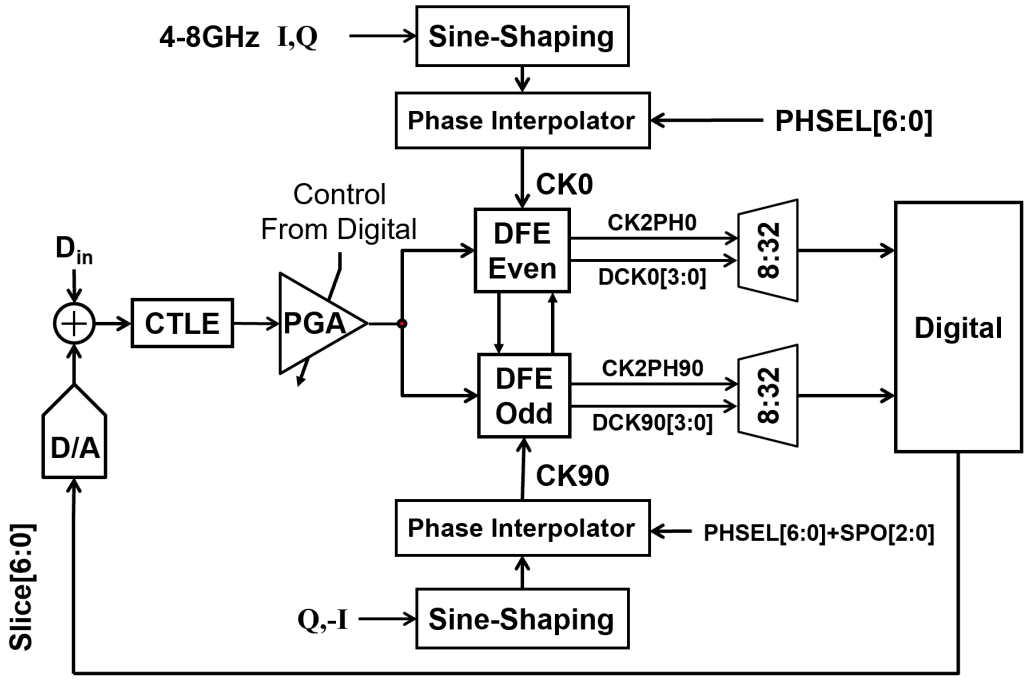


Figure 71. Block Diagram of ADRV903x Deserializer PHY

In the above block diagram, D_{in} is the data received from the BBIC over the terminated lossy channel. An offset correcting DAC adds to the input data and goes into the CTLE. The CTLE block provides an inverse transfer function of the channel so that the total composite response of the channel (at CTLE output) is flat. The CTLE attenuates the low frequency content of the incoming signal to flatten the net response cleaning up the signal and resulting in a good eye at the output of the CTLE. The amplitude of the received data is low, due to attenuation over the channel and a boost is applied via CTLE. The amplitude is boosted by a PGA block (Programmable Gain Amplifier) to rail the incoming data to $V_{ref}(1)$ or $-V_{ref}(0)$. The output of the PGA has strong 1's or 0's, which is then sampled by clocks to decode as a 1 or a 0. Note that the PGA only rails the input data in full rate and half rate modes. It linearly amplifies the incoming data for quarter rate mode (explained in subsequent sections).

A DFE samples the data using the original reference clock signal and another signal with a 90 degree phase shift relative to reference clock signal. A DFE is used to accommodate faster data rates and support higher insertion losses on ADRV903x. This eases the dependency on the CTLE block which attenuates the incoming signal and trades amplitude for a good eye causing noise/crosstalk issues. The DFE blocks look at the impulse response instead of responding to the amplitude response of the incoming signal. The DFE block subtracts the energy in the post-cursors so that zero ISI is seen for next sampling instants. This helps in achieving higher signal fidelity for lanes with relatively high insertion loss at higher data rates. The sampled data bits from the DFE block are deserialized down to a local digital block which runs a phase detection scheme to obtain a 7-bit phase code. This code is used to determine the phase selected by the phase interpolator block and adjusts the phase of CK0 and CK90 clock signals needed for sampling the incoming data.

The phase interpolator block at the top receives the incoming reference clock (as I and Q (square waves)) through a sine-shaping block. The sine-shaping block provides the sine and cosine signals which can be phase interpolated using the 7-bit phase code obtained from the digital to provide the CK0 signal. Another phase interpolator block at the bottom receives the incoming reference clock (as –I and Q (square waves)) and are passed through the same sine-shaping and phase interpolator block to obtain the CK90 (90 degrees out of phase) signal. It uses the same 7-bit phase code obtained from digital to adjust its phase. It is important to note that CK90 lags 90 degree in phase as compared to CK0. Both these signals are derived from the incoming reference clock and not the data. These 0 and 90 degree offset sampling clocks helps to get the data and edge sampling instances of the incoming data to run CDR.

With this setup, the digital can track the best sampling phase that can be used to sample the incoming data as part of clock data recovery circuit. The digital also sends the decoded data further down to the deserializer block (input into the deframer).

It is important to note that it is expected that the PLL and high-speed distribution clocks are settled before enabling the deserializer. This is done by the firmware during boot up.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

The deserializer PHY can be set up in three distinct modes based on the lane rate being used, the three modes are:

1. Full Rate Mode:

   In this mode, ADR903x has lane rates within the range of 4-8 Gbps. In this mode, only the sampling clock CK0 is used, and its frequency is matched in full to the data rate. In Figure 72 the serial input data goes through the CTLE to clean up the signal before sampling. The CTLE is also configured to provide its maximum gain to attempt to rail the signal before sampling. The timing diagram shows that samples are taken on both the rising and falling edges of CK0, and in fact CK90 is not used in this mode. The phase detector aligns the falling edge of CK0 to the transitions in the data and is a bang-bang phase detector.

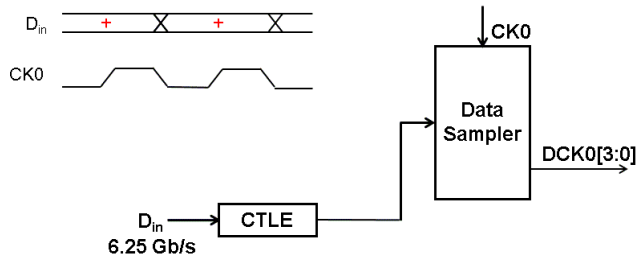


Figure 72. Simplified Block Diagram of DES in Full Rate Mode with Timing Diagram

   Note that SERDES calibrations are not needed for this mode.

2. Half Rate Mode:

   In this mode, ADR903x has lane rates within the range of 8-16 Gbps. The sampling clock is now half as fast as the data rate. From the timing diagram in Figure 73 the samples on the rising and falling edges of CK0 are all data samples, and neither takes samples on the data transitions. As these edge samples are needed, a second set of circuitry takes samples in the same fashion as before but now is driven by CK90. Each set of circuitry is referred to as a slice, thus in Full rate mode we only use slice 0, but in half rate mode we need slice 1 as well. In the same way as in Full rate mode, the phase detector does a bang-bang phase detector to align the rising and falling edges of CK90 to the transitions in the input data. The CTLE is configured in a similar fashion, to clean up the signal and max out the gain to drive the samplers. SERDES calibrations are not needed for this mode.

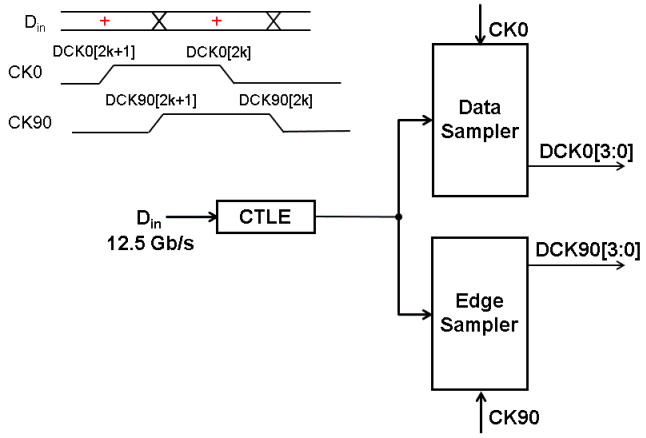


Figure 73. Simplified Block Diagram of DES in Half Rate Mode with Timing Diagram

3. Quarter Rate Mode:

   In this mode, ADR903x has lane rates within the range of 16-32 Gbps. If we look closely at the Figure 74 below, we will see that the timing diagram is almost the same as half rate mode, except that the bit-period (therefore, data rate) has been doubled. Now all rising and falling edges of both CK0 and CK90 are taking data samples, and there is nothing directly sampling the data transitions.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

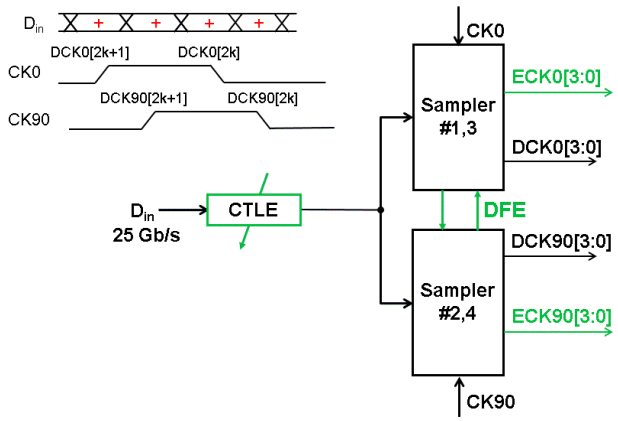


Figure 74. Simplified Block Diagram of DES in Quarter Rate Mode with Timing Diagram

In quarter rate mode the CTLE alone cannot provide enough equalization. The DFE is needed in addition to the CTLE.

For DFE to work properly, the system needs to be linearly amplified by the PGA up until the sampling point. The DFE looks at the impulse response and subtracts the energy in the post-cursors so that zero ISI is seen for next sampling instants. This helps in achieving higher signal fidelity for high lane rates (>16 Gbps).

Figure 75 shows a high-level overview of a single tap canonical DFE. The received signal *RX* is written as a linear sum of the desired cursor value plus a sum of pre- and post-cursors (in the example below, only a single post-cursor exists). This description of *RX* is a representation of the impulse response from the transmitter, through the channel, through the CTLE, up to the sampling instant.

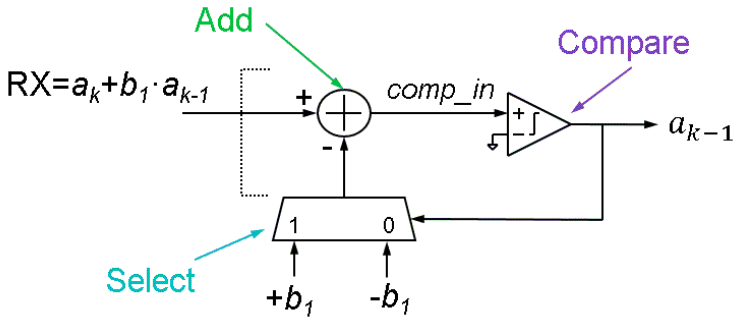


Figure 75. High Level Block Diagram of a Single Tap DFE

When a sample of the comparator's input *comp_in* is taken a comparison is made, to generate the output sample. Referring to the 'next' sample, the sample we just took is a_{k-1}. We feed this previous sample back, weight it by the value of the first post-cursor, and subtract it out (or add it negatively). Based on the comparators result, we would either select $+b_1$ or $-b_1$. This would subtract the energy in the post-cursor and reduce ISI. Note that the PGA is set up such that incoming signal is linear to accommodate this analog subtraction via DFE.

In quarter rate mode, the complete process of the DFE – Compare -> Select -> Add, is fast enough so that it is completed within a single UI. ADRV903x DFE uses three post-cursors (b_1, b_2 and b_3) to get the best equalized performance for data rates up to 32.44 Gbps

ADRV903x can run on-chip init and tracking calibrations to maintain signal integrity over the entire operating temperature. This helps to avoid JESD link dropouts during operation which cause glitches at the TX output. The ARM and API automatically enable these cals depending on the lane rates being used. Note that currently SERDES calibrations are only needed for Quarter rate mode (where lane rate >16 Gbps). The ARM will optimize the CTLE and DFE parameters in quarter rate mode to maintain signal integrity over the entire operating temperature. The calibration routines will set all the offset correction bits and DFE tap weights when operating in quarter rate mode.

Table 27 summarizes the design targets for the amount of insertion loss that ADRV903x can support for different lane rates.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

Table 27. Insertion Loss Support (over operating temperature) vs. Serdes Lane Rates

SERDES Lane Rate (Gb/s) versus Design Targets for Insertion Loss (over operating temperature) Support on ADRV903x		
Data Rate (Gb/s)	**Min Insertion Loss (dB)**	**Worst Case Max Insertion Loss (dB)**
8	3	15
12	4	17
14	4.5	17
16	4	25
25	4.5	23
32	4	10

The ADRV903x Serdes block integrates two different methods to verify link integrity:

1. The user can test the signal integrity of each of the lanes using an in-built horizontal and vertical eye monitor. This helps the user to debug link bring up issues and check if the signal integrity holds well over the entire operating temperature.
   a. The Horizontal eye monitor can be run using the RunEyeSweep_v2() API
   b. The Horizontal + Vertical eye monitor can be run using the RunVerticalEyeSweep_v2() API
2. An internal PRBS checker can be used to verify the link integrity over temperature and ensure you are meeting your Bit Error Target for long duration tests.

Deserializer Lane Configuration

The deserializer includes an adaptive CTLE and DFE equalizer per lane. This helps in compensating for signal integrity distortions for each channel due to PCB trace length and impedance.

The deser_lane_cfg structure in the json file contains the information required to properly configure the Serdes calibration defaults. This structure is repeated 8 times for each lane.

```
"deser_lane_cfg":
        {
       "desInvertLanePolarity": 0,
        "highBoost": 0,
        "configOption1": 0,
        "configOption2": 0,
        "configOption3": 0,
        "configOption4": 0,
        "configOption5": 0,
        "configOption6": 0,
        "configOption7": 0,
        "configOption8": 0,
        "configOption9": 0,
        "configOption10": 0,
        }
```

The latest configOptions recommendations are typically made available in the profiles that come with the software and would differ depending on the Serdes lane rate. For later software (greater than or equal to SW2.9), the configOptions are generally kept by default at 0 and the only parameter that is needed to tune is the Insertion Loss per lane. You can select this in the GUI by selecting the Enhanced Tuning Mode (ETM) checkbox in the Deframer section. Please contact ADI for a more detailed application note on using the ETM feature. If using older software pre-SW2.9, then please contact ADI to confirm the configOptions settings are appropriate for your channel IL and Serdes lane rate.

LINK INITIALIZATION AND DEBUGGING

Link initialization occurs during the post MCS phase of device initialization. The link bringup procedure in general follows the following steps:

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

JESD204B

For the deframer side, follow these steps:

1. Initialize and bring up the baseband processor framer side.
2. Deframer is held in reset state until INIT command, then deframer issues a synchronization request by asserting the SYNC signal.
3. Framer starts sending K28.5 characters, then deframer is brought out of reset.
4. Deframer identifies four consecutive K28.5 characters then deasserts SYNC and goes into ILAS phase.
5. If SYNC stays asserted, this indicates it is stuck in CGS phase. Check that the link parameters match. If they do, check the signal integrity (refer to the Sample Iron Python Code for PRBS Testing section).
6. After the deframer link is up, the user needs to enable the serdes tracking cal. Please note that SERDES tracking calibration needs randomized scrambled data to be sent on the deserializer lanes to be able to maintain link integrity.

For the framer side, link establishment follows the same flow. First the framer is enabled and the baseband processor deframer synchronizes to the signal.

JESD204C

For the deframer side, follow these steps:

1. Initialize and bring up the baseband processor framer side.
2. Send the JESD204C initialization calibration command. This brings the link up since it is now protocol based.
3. After the deframer link is up, the user needs to enable the serdes tracking cal. Please note that SERDES tracking calibration needs randomized scrambled data to be sent on the deserializer lanes to be able to maintain link integrity.

For the framer side, link establishment follows the same flow. First the framer is enabled and then the baseband processor deframer synchronizes to the signal.

An example API function jesdBringup can be referred to configure and establish the datalinks.

FIRST TIME SYSTEM BRING UP—CHECKING LINK INTEGRITY

1. For ease of debug during bring up, it is recommended to start with single lane on both sides and with minimum possible link speed.
2. Check that the parameters are configured the same at both ends transceiver and FPGA. The adi_ADRV903x_DeframerCfg_t data structure contains the information required to properly configure each deframer.
3. There is a PRBS checker available that can be used to check signal integrity related issues. Initialize the transceiver as outlined in the link establishment section. Enable the PRBS generator on the baseband processor with the desired PRBS sequence.
4. Confirm that the lanes baseband processor is transmitting PRBS on are the actually configured in the transceiver. Start with the PRBS errors. Ensure baseband processor and the transceiver are both using the same PRBS signal and the transceiver expects the same PRBS 7 from baseband processor.
5. Call the API adi_ADRV903x_DfrmPrbsCheckerStateSet() passing the actual device being evaluated, the PRBS sequence to check, and the location at which to check the PRBS sequence.
6. After some amount of time, call the API function to check the PRBS errors. This can be done by calling the API function adi_ADRV903x_DfrmPrbsErrCountGet() passing the actual device being evaluated, the counter selection lane to be read and the error count is returned in the third parameter passed.
7. The user can use adi_ADRV903x_DeframerSysrefCtrlSet() API so that the external SYSREF signal at the pin can be gated off internally so the deframer does not see a potential invalid SYSREF pulse before it is configured correctly.
8. After bringing up of JESD204B link or for debugging the deframer, the baseband processor can check the status of the deframer using adi_ADRV903x_DeframerStatusGet().

SAMPLE IRON PYTHON CODE FOR PRBS TESTING

The following Iron Python script can be loaded into the Iron Python tab in the GUI to run the PRBS test.

```
def generatePrbsInFpga():
        print "generatePrbsInFpga() is running"
        prbsmode = adi_fpgagen6_PrbsTestModes_e.ADI_FPGAGEN6_PRBS_31
```

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

```
        fpgaDevice_0.prbs.PrbsSerializerEnable(0xff,prbsmode)
def OnBoardPRBSChecker():
        print "OnBoardPRBSChecker() is running"
        #DfrmPrbsCheckerStateGet
        DfrmPrbsCfg_t = adi_adrvgen6_DfrmPrbsCfg_t()
        adrvgen6Device_0.dataInterface.DfrmPrbsCheckerStateGet(DfrmPrbsCfg_t)
        print
        print "DfrmPrbsCheckerState DfrmPrbsCfg_t.polyOrder: %s" % str(DfrmPrbsCfg_t.polyOrder)
        print "DfrmPrbsCheckerState DfrmPrbsCfg_t.checkerLocation: %s" % str(DfrmPrbsCfg_t.checkerLoca▶
tion)
        print
        #DfrmPrbsCheckerStateSet
        DfrmPrbsCfg_t = adi_adrvgen6_DfrmPrbsCfg_t()
        DfrmPrbsCfg_t.polyOrder = adi_adrvgen6_DeframerPrbsOrder_e.ADI_ADRVGEN6_PRBS31
        DfrmPrbsCfg_t.checkerLocation = adi_adrvgen6_DeframerPrbsCheckLoc_e.ADI_ADRVGEN6_PRBSCHECK_LA▶
NEDATA
        #OR
        #DfrmPrbsCfg_t.checkerLocation = adi_adrvgen6_DeframerPrbsCheckLoc_e.ADI_ADRV▶
GEN6_PRBSCHECK_SAMPLEDATA
        adrvgen6Device_0.dataInterface.DfrmPrbsCheckerStateSet(DfrmPrbsCfg_t)
        #DfrmPrbsCheckerStateGet
        DfrmPrbsCfg_t = adi_adrvgen6_DfrmPrbsCfg_t()
        adrvgen6Device_0.dataInterface.DfrmPrbsCheckerStateGet(DfrmPrbsCfg_t)
        print
        print "DfrmPrbsCheckerState DfrmPrbsCfg_t.polyOrder: %s" % str(DfrmPrbsCfg_t.polyOrder)
        print "DfrmPrbsCheckerState DfrmPrbsCfg_t.checkerLocation: %s" % str(DfrmPrbsCfg_t.checkerLoca▶
tion)
        print
        #adrvgen6Device_0.dataInterface.DfrmErrCounterReset(adi_adrvgen6_DeframerSel_e.ADI_ADRVGEN6_DE▶
FRAMER_0,0,7)
        #adrvgen6Device_0.dataInterface.DfrmErrCounterReset(adi_adrvgen6_DeframerSel_e.ADI_ADRVGEN6_DE▶
FRAMER_0,1,7)
        #adrvgen6Device_0.dataInterface.DfrmErrCounterReset(adi_adrvgen6_DeframerSel_e.ADI_ADRVGEN6_DE▶
FRAMER_0,2,7)
        #adrvgen6Device_0.dataInterface.DfrmErrCounterReset(adi_adrvgen6_DeframerSel_e.ADI_ADRVGEN6_DE▶
FRAMER_0,3,7)
        #adrvgen6Device_0.dataInterface.DfrmErrCounterReset(adi_adrvgen6_DeframerSel_e.ADI_ADRVGEN6_DE▶
FRAMER_0,4,7)
        #adrvgen6Device_0.dataInterface.DfrmErrCounterReset(adi_adrvgen6_DeframerSel_e.ADI_ADRVGEN6_DE▶
FRAMER_0,5,7)
        #adrvgen6Device_0.dataInterface.DfrmErrCounterReset(adi_adrvgen6_DeframerSel_e.ADI_ADRVGEN6_DE▶
FRAMER_0,6,7)
        #adrvgen6Device_0.dataInterface.DfrmErrCounterReset(adi_adrvgen6_DeframerSel_e.ADI_ADRVGEN6_DE▶
FRAMER_0,7,7)
        time.sleep(0.5)
        #DfrmPrbsErrCountGet
        DfrmPrbsErrCounters_t = adi_adrvgen6_DfrmPrbsErrCounters_t()
        adrvgen6Device_0.dataInterface.DfrmPrbsErrCountGet(DfrmPrbsErrCounters_t)
        print
        for i in range(0,8):
                print "PRBS error of lane %d" %i
                print "DfrmPrbsErrCounters_t.errorStatus",DfrmPrbsErrCounters_t.errorStatus[i]
                print "DfrmPrbsErrCounters_t.laneErrors",DfrmPrbsErrCounters_t.laneErrors[i]
        print
```

When this script is run, it transmits PRBS pattern and then checks the PRBS errors received per lane.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

PRBS ERRORS

When the baseband processor is transmitting PRBS, confirm that the active lanes are also configured properly in the transceiver. Start with the PRBS errors. Ensure the baseband processor and the transceiver are both using the same PRBS signal and the transceiver expects the same PRBS 7 from baseband processor. The following are some scenarios that might occur and how to resolve issues.

If stuck in CGS mode, or if SYNC stays at logic low level or pulses high for less than four multiframes, take the following steps:

1. Check the board, unpowered for the following:
   - SYSREF and SYNC signaling is dc-coupled.
   - Check that the pull-down or pull-up resistors are not dominating the signaling, for example if values are too small or shorted and therefore cannot be driven correctly.
   - Verify that the differential-pairs traces are length matched.
   - Verify differential impedance of the traces is 100 Ω.
2. Check the board, powered:
   - If there is a buffer/translator in the SYNC path, make sure it is functioning properly.
   - Check that SYNC source is properly configured to produce compliant logic levels.
3. Check SYNC signaling:
   - If SYNC is static and logic low, the link is not progressing beyond the CGS phase. There is either an issue with the data being sent or the JESD204 receiver is not decoding the samples properly. Verify /K/ characters are being sent, verify receive configuration settings, verify SYNC source. Consider overdriving SYNC signal and attempt to force link into ILAS mode to isolate link Rx vs. Tx issues.
   - If SYNC is static and logic high, verify the SYNC logic level is configured correctly in the source device. Check pull-up and pull-down resistors.
   - If SYNC pulses high and returns to logic-low state for less than six multiframe periods, the JESD204 Link is progressing beyond the CGS phase but not beyond ILAS phase. This suggests the /K/ characters are okay and the basic function of the CDR are working. Proceed to ILAS troubleshooting.
   - If SYNC pulses high for a duration of more than six multiframe periods, the Link is progressing beyond the ILAS phase and is malfunctioning in the data phase; see the data phase section for troubleshooting tips.
4. Checking Serial Data
   - Verify the transmitter data rate and the receiver expected rate are the same.
   - Measure lanes with high-impedance probe (differential probe, if possible); if characters appear incorrect, make sure lane differential traces are matched, the return path on the PCB is not interrupted, and devices are properly soldered on the PCB. CGS characters are easily recognizable on a high speed scope.
   - Verify /K/ characters with high impedance probe. (If /K/ characters are correct, the Tx side of the link is working properly. If /K/ characters are not correct, the Tx device or the board lanes signal have an issue.
   - Verify the transmitter CML differential voltage on the data lanes.
   - Verify the receiver CML differential voltage on the data lanes.
   - Verify that the configuration parameters M and L values match between the baseband processor and the transceiver, otherwise the data rates may not match. For example, M = 2 and L = 2 expect ½ the data rate over the serial interface as compared to the M = 2 and L = 1 case.
   - Ensure the device clock is phase locked and at the correct frequency.

If the user is stuck in ILAS mode, or if SYNC pulses high for approximately four multiframes, take the following steps:

1. Link parameter conflicts
   - Verify ILAS multiframes are transmitting properly, verify link parameters on the Tx device, the Rx device and those transmitted in ILAS second multiframe.
   - Calculate expected ILAS length (tframe, tmultiframe, 4xtmultiframe), verify ILAS is attempted for approximately four multiframes.
2. Verify all lanes are functioning properly. Ensure there are no Multilane/Multilink conflicts.

If the interface enters data phase but occasionally link resets (returns to CGS and ILAS before returning to data phase), take the following steps:

1. Invalid setup and hold time of periodic or gapped periodic SYSREF or SYNC signal.

2. Link parameter conflicts
3. Character replacement conflicts
4. Scrambling problem, if enabled
5. Lane data corruption, noisy or jitter can force the eye diagram to close
6. Spurious clocking or excessive jitter on device clock

SELECTING THE OPTIMAL LMFC/LEMC OFFSET FOR A DEFRAMER

This section describes how to set the LMFC/LEMC offset for a deframer, how to read back the corresponding elastic buffer depth, and how to select the optimal LMFC/LEMC offset value for a given system.

Deterministic Latency in JESD 204B Mode

In JESD204B mode, the transceiver digital data interface follows the JESD204B Subclass 1 standard, which has provisions to ensure repeatable latencies across the link from power-up to power-up or over link re-establishment by using the SYSREF signal.

To achieve this deterministic latency, the transceiver deframers include elastic buffers for each of their lanes. The elastic buffers are also used to de-skew each lane before aligning them with the LMFC signal. The depth of the elastic buffers can therefore be different for each lane of a given deframer.

A deframer starts outputting data out of its elastic buffers on the next LMFC (that is, multiframe) boundary following the reception of the first characters in the ILA sequence by all the active lanes. It is therefore possible to adjust when the data is output from the elastic buffers, and therefore how much data is stored in those buffers (called buffer depth), by adjusting the phase relationship between the external SYSREF signal and the internally generated LMFC signal. This phase relationship is adjustable by using the LMFC offset parameter, which is programmable for each of the deframers. This is illustrated on Figure 76 and Figure 77.

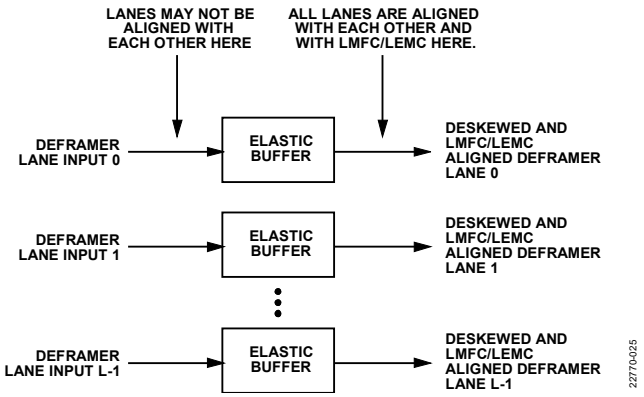


Figure 76. Elastic Buffers in the Deframers

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

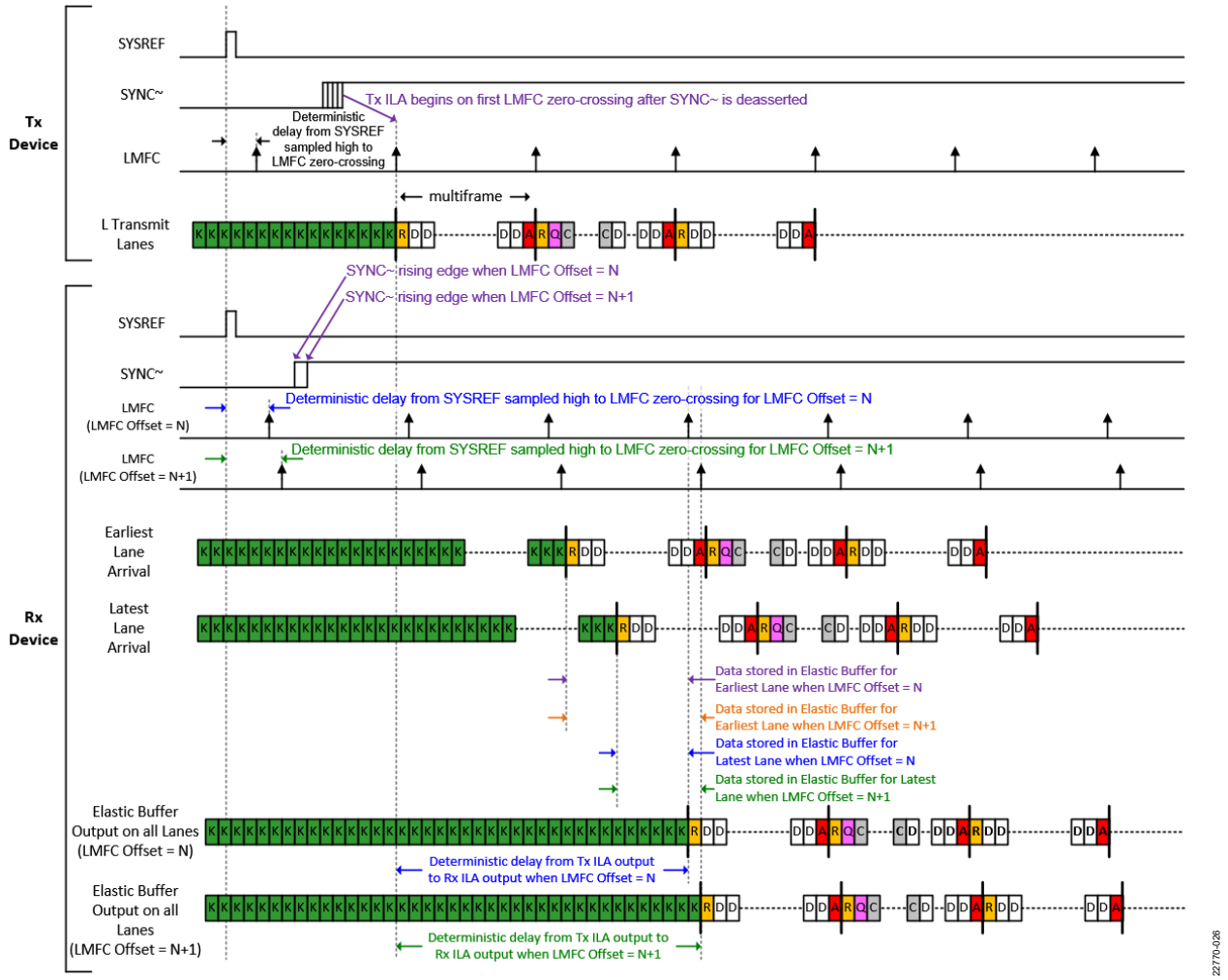


Figure 77. Impact of LMFC Offset on Elastic Buffer Depth in JESD204B Mode

Deterministic Latency in JESD204C Mode

In JESD204C mode, deterministic latency can also be achieved thanks to the elastic buffers in the deframers. The elastic buffers are still used to de-skew each lane before aligning them with the LEMC signal. The depth of the elastic buffers can, therefore, be different for each lane of a given deframer.

A deframer starts outputting data from its elastic buffers on the next LEMC (extended multiblock) boundary following the reception of the first multiblock in an extended multiblock by all the active lanes. As a result, it is possible to adjust when the data is output from the elastic buffers and, therefore, how much data is stored in those buffers (the buffer depth) by adjusting the phase relationship between the external SYSREF signal and the internally generated LEMC signal. This phase relationship is adjustable by using the LEMC offset parameter, which is programmable for each of the deframers. This is illustrated on Figure 76 and Figure 78.

It is important to note that the size of each elastic buffer is 512 octets. When the JESD204C E parameter (number of multiblocks in an extended multiblock) is bigger than 2, the elastic buffer is not able to store enough data for some LEMC offset values.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

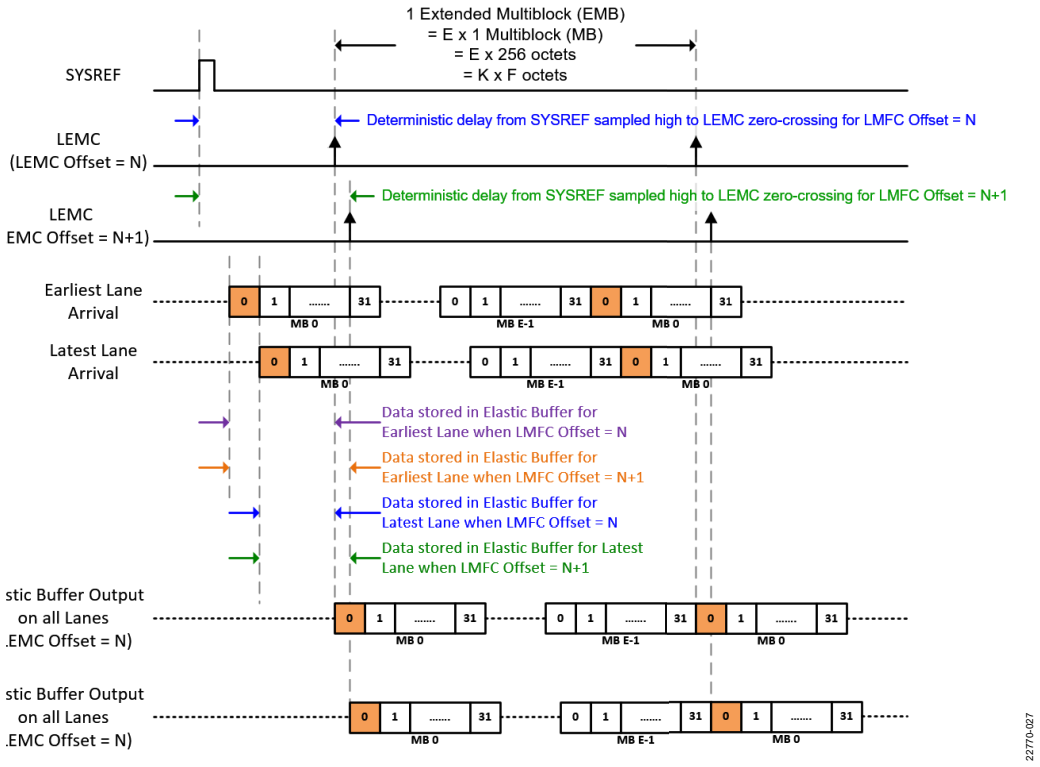


Figure 78. Impact of LEMC Offset on Elastic Buffer Depth in JESD204C Mode

Programming the LMFC Offset for a Deframer

There are three ways to program the LFMC offset for a given deframer.

1. By modifying the profile file being used
2. By using the adi_ADRV903x_DeframerCfg data structure
3. By using the DfrmLmfcOffsetSet() API method

Each method is addressed in the following sections.

Setting the LMFC/LEMC Offset in the Profile File

There is a deframer_lmfc_offset field for each of the two deframers in the profile file. This field corresponds to the LMFC offset in JESD 204B mode, and to the LEMC offset in JESD 204C mode. It can be set to a decimal value between 0 and (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle). For example, for the ADRV903X_UC48_204C_8T8R2OR_NLS.json file, the “deframer_lmfc_offset ” field is located around Line 669 for Deframer 0 and around Line 721 for Deframer 1 (see Figure 79).

```
646             255,
647             255,
648             255,
649             255,
650             255
651           ],
652           "deframer_K": 0,
653           "deframer_E": 0,
654           "deframer_S": 0,
655           "deframer_subclass": 1,
656           "deframer_bankId": 0,
657           "deframer_deviceId": 0,
658           "deframer_lane0Id": 0,
659           "deframer_syncbOutSelect": 3,
660           "deframer_syncbOutLvdsMode": 1,
661           "deframer_syncbOutLvdsPnInvert": 0
662           "deframer_syncbOutCmosSlewRate": 0
663           "deframer_syncbOutCmosDriveLevel":
664           "deframer_newSysrefOnRelink": 0,
665           "deframer_sysrefForStartup": 0,
666           "deframer_sysrefNShotEnable": 0,
667           "deframer_sysrefNShotCount": 0,
668           "deframer_sysrefIgnoreWhenLinked":
669           "deframer_lmfc_offset": 0,
670           "deframer_scramble": true
671         },
672         {
673           "deframer_M": 0,
674           "deframer_sample_xbar": [
675             128,
```

Figure 79. Deframer 0 deframer_lmfc_offset Field for the ADRV903X_UC48_204C_8T8R2OR_NLS.json File

Note that the device must be reprogrammed after changing an LMFC/LEMC offset in the profile file and loading it into Arm memory for the change to take effect. Also note that if the goal is to sweep the LMFC/LEMC offset values for test purposes without any need for RF performance (for example, to determine the optimal LMFC/LEMC value), it is not necessary to run the init cals when programming the transceiver. Not running the init cals make the programming process quicker. For this case, it is quicker to just use the API method.

Setting the LMFC/LEMC Offset in the Adi_ADRV903x_DeframerCfg_t Data Structure

An alternative way of programming the LMFC/LEMC offset consists in using the lmfcOffset field of the adi_ADRV903x_DeframerCfg data structure for the relevant deframer (see Figure 80). Note that the device must be reprogrammed after changing the LMFC/LEMC offset for a given deframer in the adi_ADRV903x_DeframerCfg data structure for the change to take effect. Also note that if the goal is to sweep the LMFC/LEMC offset values for test purposes without any need for RF performance (for example, to determine the optimal LMFC/LEMC value), it is not necessary to run the init cals when programming the transceiver. Not running the init cals make the programming process quicker.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

```
typedef struct adi_adrv903x_DeframerCfg
{
    uint8_t enableJesd204C;           /*!< 1= Enable JESD204C framer, 0 = use JESD204B framer */
    uint8_t bankId;                   /*!< Extension to Device ID. Range is 0..15 , bankId is not s
    uint8_t deviceId;                 /*!< Link identification number. Range is 0..255 */
    uint8_t laneId[ADI_ADRVGEN6_MAX_DESERIALIZER_LANES];
    uint8_t jesd204M;                 /*!< Number of DACs (0, 2, or 4) - 2 DACs per transmit chain
    uint16_t jesd204K;                /*!< Number of frames in a multiframe. Default = 32, F*K = mc
    uint8_t jesd204F;                 /*!< Number of bytes(octets) per frame . */
    uint8_t jesd204Np;                /*!< converter sample resolution (12, 16) */
    uint8_t jesd204E;                 /*!< JESD204C E parameter. This is E -1 value. */
    uint8_t decrambling;              /*!< decrambling off if decramble = 0, if decramble > 0 decra
    uint8_t deserializerLanesEnabled; /*!< Deserializer lane select bit field. Where, [0] = Lane0 e
    uint16_t lmfcOffset;              /*!< LMFC offset value to adjust deterministic latency. */
    uint8_t syncbOutSelect;           /*!< Selects deframer SYNCBOUT pin (0 = SYNCBOUT0, 1 = SYNCBC
    uint8_t syncbOutLvdsMode;         /*!< Ignored if syncbOutSelect = 3. Otherwise 1 - enable LVDS
    uint8_t syncbOutLvdsPnInvert    ; /*!< Ignored if syncbOutSelect = 3. Otherwise 0 - syncb LVDS
    uint8_t syncbOut0CmosDriveStrength; /*!< CMOS output drive strength. Max = 15 */
    uint8_t syncbOut1CmosDriveStrength; /*!< CMOS output drive strength. Max = 15 */
    adi_adrvgen6_DeserLaneXbar_t deserializerLaneCrossbar; /*!< Lane crossbar to map physical lanes
    adi_adrvgen6_DacSampleXbarCfg_t dacCrossbar;          /*!< Deframer output to DAC mapping */
    uint8_t newSysrefOnRelink;      /*!< Flag for determining if SYSREF on relink should be set. Wh
    uint8_t sysrefForStartup;       /*!< Suggested to enable for deframer so deframer will not asse
    uint8_t sysrefNShotEnable;      /*!< 1 = Enable SYSREF NShot (ability to ignore first rising ed
    uint8_t sysrefNShotCount;       /*!< Count value of which SYSREF edge to use to reset LMFC phas
    uint8_t sysrefIgnoreWhenLinked; /*!< When JESD204 link is up and valid, 1= ignore any sysref pu
    uint32_t  iqRate_kHz;           /*!< Framer I/Q rate */
    uint32_t  laneRate_kHz;         /*!< Framer Lane rate */
} adi_adrv903x_DeframerCfg_t;
```

Figure 80. LMFC Offset Field in adi_ADRV903x_DeframerCfg Data Structure

Note that a SYSREF pulse must be applied and then the link between the JESD framer and JESD deframer of the transceiver must be reestablished after changing the LMFC/LEMC offset through SPI writes for a given deframer for the change to take effect.

Setting the LMFC/LEMC Offset Through API

It is possible to set the LMFC/LEMC offset value by using the API method or function, DfrmLmfcOffsetSet(). You just need to select either Deframer0 or Deframer1 and pass an lmfcOffset parameter.

The valid range of lmfcOffset adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).

Note you should restablish the link between the JESD framer and the transceiver JESD deframer after changing the LMFC/LEMC offset. To do this, you should toggle the DeframerLinkStateSet() and also toggle the DeframerSysrefCtrlSet(). A new Sysref is not required.

Reading Back the Buffer Depths for Each Deframer

The API function adi_ADRV903x_CoreBufDepthGet can be used to readback the deframer FIFO depth for each of the deframers.

In JESD204B mode, the unit of the values read back in those registers is 4 octets. In other words, an increment of the buffer depth value read back by 1 unit corresponds to an actual increment by 4 octets. The values read back range from 0 to (K × F)/4 (where K is the number of frames per multiframe, and F is the number of octets per lane in a frame cycle).

In JESD204C mode, the unit of the values read back in those registers is 8 octets. In other words, an increment of the buffer depth value read back by 1 unit corresponds to an actual increment by 8 octets. The values read back range from 0 to E × 32 (where E is the number multiblocks in an extended multiblock). Note that the size of the elastic buffer is 512 octets. When E > 2, the maximum buffer depth values read back are therefore limited to 64, which corresponds to 512 octets.

Note that the values reported in each of those registers correspond to a value based on the positions of the elastic buffer read and write pointers. The value has a fixed offset and does not represent the exact number of octets in the elastic buffer.

Selecting the Optimal LMFC/LEMC Offset for a System

The buffer depths are expected to slightly change from power up to power up or from one JESD link establishment to another due to the variance in, for example, synchronization delays and physical lane skews. They are also expected to slightly change from system to system due to process, voltage and temperature (PVT) variations.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

It is therefore recommended to select an LMFC/LEMC offset value resulting in optimal buffer depths to account for those variations and maintain deterministic latency on all boards for a given system.

JESD API FUNCTIONS

Table 28. List of JESD Related API Functions

API Method Name	Comments
adi_ADRV903x_AdcSampleXbarSet()	Sets the ADC sample crossbar for the specified ADRV903x framer.
adi_ADRV903x_AdcSampleXbarGet()	Gets the ADC sample crossbar converter configuration map for the chosen JESD204B/C framer converter.
adi_ADRV903x_DacSampleXbarSet()	Sets the DAC sample crossbar for the specified ADRV903x deframer.
adi_ADRV903x_DacSampleXbarGet()	Gets the DAC sample crossbar for the specified ADRV903x deframer.
adi_ADRV903x_FramerCfgGet()	Gets the ADRV903x Framer's configuration.
adi_ADRV903x_DeframerCfgGet()	Gets the ADRV903x Deframer's configuration.
adi_ADRV903x_FramerLinkStateGet()	Gets the link states for all the framers.
adi_ADRV903x_FramerLinkStateSet()	Sets the link states for all the framers.
adi_ADRV903x_DfrmPrbsCountReset()	Reset Deframer PRBS Count error.
adi_ADRV903x_DeframerLinkStateGet()	Gets the link states for all the deframers.
adi_ADRV903x_DeframerLinkStateSet()	Sets the link states for all the deframers.
adi_ADRV903x_DfrmPrbsCheckerStateSet()	Sets the lane/sample PRBS checker configuration.
adi_ADRV903x_DfrmPrbsCheckerStateGet()	Gets the lane/sample PRBS checker configuration.
adi_ADRV903x_FramerSysrefCtrlSet()	Enables or disables the external SYSREF signal to the framers.
adi_ADRV903x_FramerSysrefCtrlGet()	Get the status of the external SYSREF signal to the framers.
adi_ADRV903x_DeframerSysrefCtrlSet()	Enables or disables the external SYSREF signal to the deframers.
adi_ADRV903x_DeframerSysrefCtrlGet()	Get the status of the external SYSREF signal to the deframers.
adi_ADRV903x_FramerTestDataSet()	Sets the PRBS type and enables/disables framer PRBS generation.
adi_ADRV903x_FramerTestDataGet()	Gets the PRBS framer test mode and inject points.
adi_ADRV903x_DfrmPrbsErrCountGet()	Gets the deframer sample PRBS error counts.
adi_ADRV903x_SerializerReset()	Resets the serializer.
adi_ADRV903x_FramerLmfcOffsetSet()	Sets LMFC offset for selected framer.
adi_ADRV903x_FramerLmfcOffsetGet()	Gets LMFC offset for selected framer.
adi_ADRV903x_DfrmLmfcOffsetSet()	Sets LMFC offset for selected deframer.
adi_ADRV903x_DfrmLmfcOffsetGet()	Gets LMFC offset for selected deframer.
adi_ADRV903x_DfrmPhaseDiffGet()	Gets phase diff value for selected deframer.
adi_ADRV903x_FramerStatusGet()	Get framer status for the selected framer.
adi_ADRV903x_DeframerStatusGet()	Get deframer status for the selected framer.
adi_ADRV903x_DeframerStatusGet_v2()	Get deframer status for the selected framer.
adi_ADRV903x_DfrmErrCounterStatusGet()	Get deframer status for the selected deframer. JESD204B only.
adi_ADRV903x_DfrmErrCounterReset()	Clear the error counters selected deframer. JESD204B only.
adi_ADRV903x_Dfrm204cErrCounterStatusGet()	Get deframer status for the selected deframer. JESD204C only.
adi_ADRV903x_Dfrm204cErrCounterReset()	Clear the error counters selected deframer. Support for only JESD204C.
adi_ADRV903x_DfrmLinkConditionGet()	Get deframer link condition.
adi_ADRV903x_DfrmFifoDepthGet()	Get the deframer FIFO depth.
adi_ADRV903x_DfrmCoreBufDepthGet()	Get the deframer core buffer depth.
adi_ADRV903x_DfrmIlasMismatchGet()	Compares received ILAS to programmed ILAS. JESD204B only.
adi_ADRV903x_DfrmIlasMismatchGet_v2()	Compares received ILAS to programmed ILAS. JESD204B only.
adi_ADRV903x_FramerSyncbModeSet()	Set the framer JESD204B syncb signal mode to SPI or PIN mode.
adi_ADRV903x_FramerSyncbModeGet()	Get the framer JESD204B syncb signal mode to SPI or PIN mode.
adi_ADRV903x_FramerSyncbStatusSet()	Get the framer JESD204B syncb signal status.
adi_ADRV903x_FramerSyncbStatusGet()	Set the framer JESD204B syncb signal status.
adi_ADRV903x_FramerSyncbErrCntGet()	Get the framer JESD204B syncb signal error counter.
adi_ADRV903x_FramerSyncbErrCntReset()	Reset the framer JESD204B syncb signal error counter.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

Table 28. List of JESD Related API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_DeframerSyncbErrCntGet()	Get the deframer JESD204B syncb signal error counter.
adi_ADRV903x_DeframerSyncbErrCntReset()	Reset the deframer JESD204B syncb signal error counter.
adi_ADRV903x_DfrmIrqMaskGet()	Gets the JESD204B IRQ clear register. There is one register for all deframers
adi_ADRV903x_DfrmIrqMaskSet()	Sets the JESD204B IRQ clear register. There is one register for all deframers.
adi_ADRV903x_DfrmIrqSourceReset()	Reset the JESD204B IRQ clear register. There is one register for all deframers.
adi_ADRV903x_DfrmIrqSourceGet()	Get the JESD204B IRQ source registers for the specified deframer.
adi_ADRV903x_DfrmErrCntrCntrlSet()	Configure or reset the JESD204B deframer error counters.
adi_ADRV903x_DfrmErrCntrCntrlGet()	Get the configuration for the JESD204B deframer error counters.
adi_ADRV903x_RunEyeSweep()	Run horizontal eye sweep.
adi_ADRV903x_RunEyeSweep_v2()	Run horizontal eye sweep.
adi_ADRV903x_RunVerticalEyeSweep()	Run vertical eye sweep.
adi_ADRV903x_RunVerticalEyeSweep_v2()	Run vertical eye sweep.
adi_ADRV903x_FramerTestDataInjectError()	Injects an error into the framer test data by inverting the data
adi_ADRV903x_SerLaneCfgSet()	Set the serializer lane configuration parameters for the chosen lane.
adi_ADRV903x_SerLaneCfgGet()	Get the serializer lane configuration parameters for the chosen lane.
adi_ADRV903x_RxTestDataSet()	Set up a test signal to be sent out the framer instead of the ADC output.
adi_ADRV903x_RxTestDataGet()	Get the test signal being sent out the framer instead of the ADC output.

STREAM PROCESSOR AND SYSTEM CONTROL

The stream processor is a processor within the ADRV903x which performs a series of configuration tasks based on an event. When requested the stream processor performs a series of pre-defined actions which are loaded into the stream processor during initialization. This processor takes advantage of the internal register bus speed for efficient execution of commands. The stream processor accesses and modifies registers independently, avoiding the need for ARM interaction.

The stream processor executes streams or series of tasks for:

- Tx datapath Enable/Disable
- Rx datapath Enable/Disable
- ORx datapath Enable/Disable

The stream processor image changes with different configurations. For example, the stream that enables the receivers are different depending on the JESD configuration. It is therefore necessary to save a stream image for each configuration. When the user saves the configuration files (.bin) using the configurator, a stream binary image is generated automatically (a separate .bin file). This stream image file should then be used when initializing the device with the configuration in question. It is also necessary to save a stream image file every time the firmware version is updated as stream image files can be specific to versions of ARM and API.

Stream files could differ for several reasons, some examples are:

- The framer choices for ORx and Rx
- Link sharing profiles use different stream files to non-linksharing profiles
- If floating point formatting is being used on Rx and ORx paths

Nineteen separate stream processors exist in the device, each of which is responsible for the execution of some dedicated functionality within the device. These can be divided into two broad categories: slice stream processors and the core stream processor.

SLICE AND CORE STREAM PROCESSORS

There are eighteen slice stream processors, one each for the eight Tx, Rx datapaths, and two for the ORx datapaths. These ORx datapaths are not shared with the internal Tx channel loopback paths that facilitate data collection during the various Tx calibrations however for external LOL calibrations the ORx path is used. The existence of individual slice stream processors for each datapath enables true real-time parallel operation of all individual Tx, Rx and ORx datapaths.

Each slice stream processor may only access the digital register sub maps corresponding to its specific functionality. For example, the Tx slice stream processors can only access the Tx digital sub maps.

The core stream processor has access to the entire device. The core stream processor services GPIO pin-based streams and any custom streams that are cross domain.

STREAM PROCESSOR API FUNCTIONS

Table 29. Stream Processor API Functions

API Method Name	Comments
adi_ADRV903x_StreamVersionGet()	Reads back the version of the Stream binary loaded.
adi_ADRV903x_StreamProcErrorGet()	Sweeps through all 19 stream processors and checks for failures and errors.

SYSTEM CONTROL

The signal paths within the device can be controlled by either the API or through pin control mode. API control relies on the SPI bus and its inherent unpredictable timing with respect to register access. For critical time alignment, pin control is recommended. API mode is the default on power up.

Pin Control

The individual channels can also be controlled using a series of enable pins. In pin control mode, the Rx and Tx signal chains are controlled using eight TRX[A-H]_CTRL pins. Any one of the TRX[A-H]_CTRL pins can control any of the eight Rx and/or eight Tx paths. A TRX[A-H]_CTRL pin can even control multiple paths at the same time. Using the configurator, the user will define what TRX[A-H]_CTRL pins control what Tx and Rx channels. When these TRx pins are toggled high, the pre-determined signal chain(s) are enabled. When these pins are toggled low, the pre-determined signal chain(s) are disabled.

STREAM PROCESSOR AND SYSTEM CONTROL

The ORx paths are controlled by dedicated pins, ORXA_CTRL and ORXB_ CTRL. When these pins are toggled high, the relevant signal chain is enabled. When these pins are toggled low, the relevant signal chain is disabled.

Please see for recommendations on what to do with unused TRX_CTRL pins.

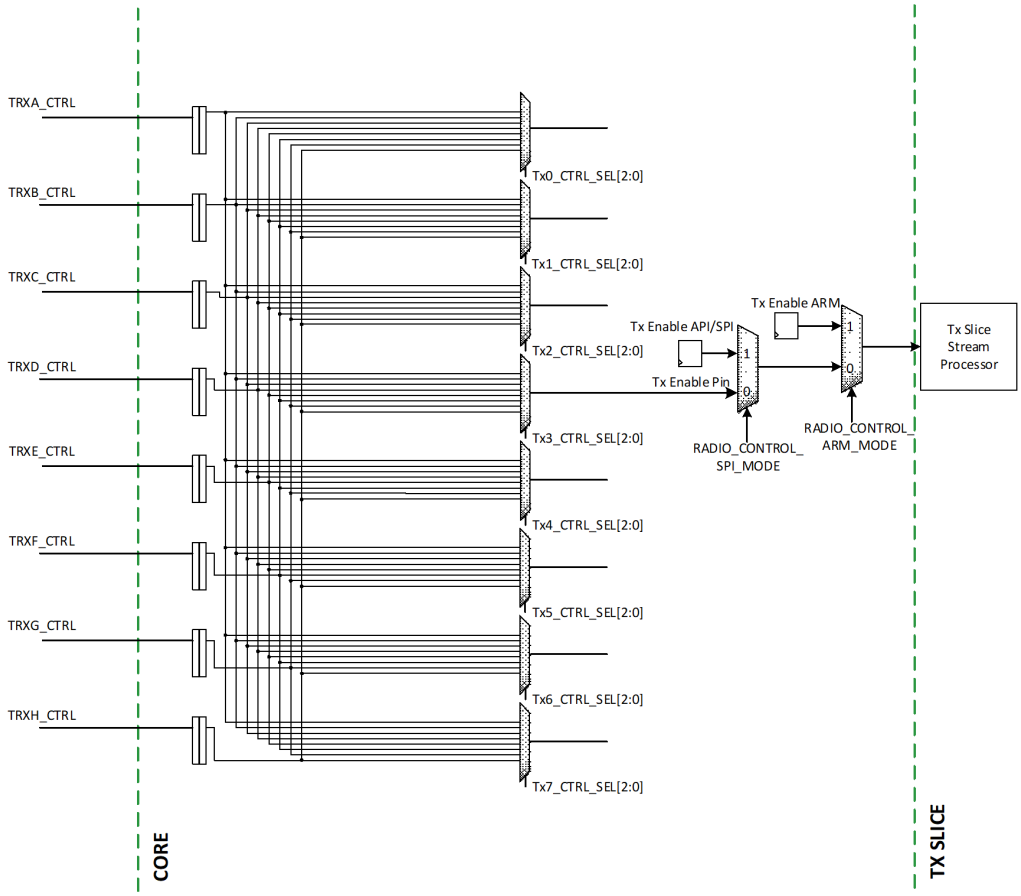


Figure 81. TRXn_CTRL Pins for Tx Slice

Figure 81 shows that each TRX[A-H]_CTRL pin can be wired to control any Tx datapath or multiple Tx datapaths at the same time. The value of Tx#_CTRL_SEL[2:0] in Table 30 dictates what TRX[A-H]_CTRL pin a given Tx slice is pointing to. The mux RADIO_CONTROL_SPI_MODE will choose whether the TRX[A-H]_CTRL pins or the API (via the SPI) controls the enable/disable of the Tx datapaths. Further on in the mux chain the RADIO_CONTROL_ARM_MODE chooses whether the ARM or the API/ TRX[A-H]_CTRL pins are able to control the Tx datapaths. This is because the ARM needs to have priority over when it can control the Tx datapaths for calibrations etc. and thus when the ARM takes control of the signal paths it cannot be stopped.

Table 30. Tx#_CTRL_SEL[2:0]

TRX PIN	Tx#_CTRL_SEL[2:0]
TRXA_CTRL	000
TRXB_CTRL	001
TRXC_CTRL	010
TRXD_CTRL	011
TRXE_CTRL	100
TRXF_CTRL	101
TRXG_CTRL	110

STREAM PROCESSOR AND SYSTEM CONTROL

Table 30. Tx#_CTRL_SEL[2:0] (Continued)

TRX PIN	Tx#_CTRL_SEL[2:0]
TRXH_CTRL	111

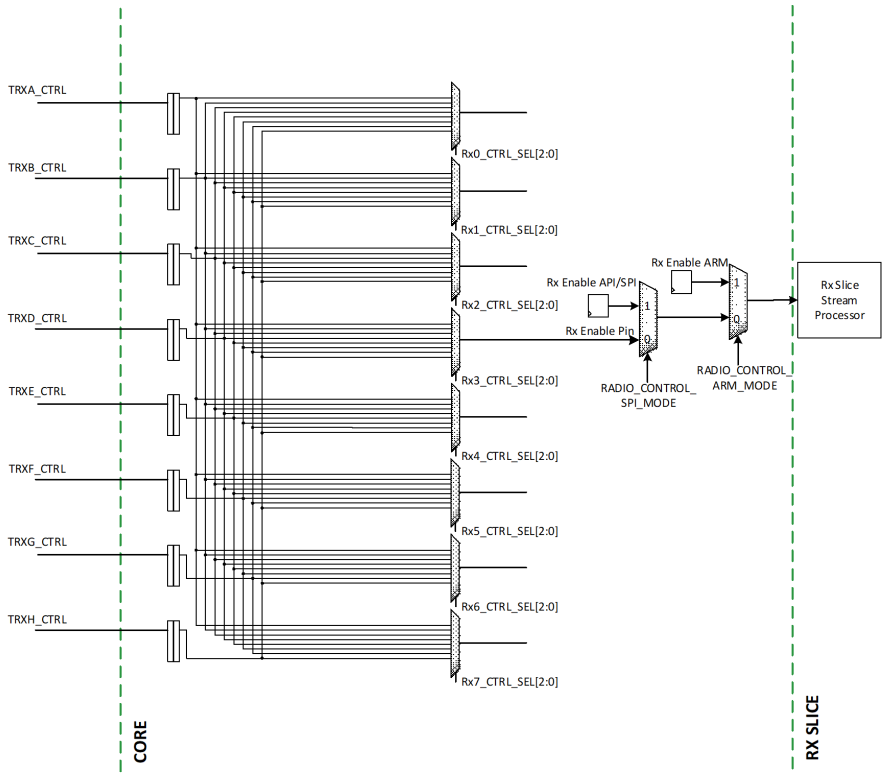


Figure 82. TRXn_CTRL Pins for Rx Slice

Figure 82 shows each TRX[A-H]_CTRL pin can be wired to control any Rx datapath or multiple Rx datapaths at the same time. The value of Rx#_CTRL_SEL[2:0] as stated in Table 31 will dictate what TRX_EN pin a given Rx slice is pointing to. The mux Rx#_CTRL_SEL[2:0] is set to point at a given TRX[A-H]_CTRL. The mux RADIO_CONTROL_SPI_MODE will choose whether the TRX_EN pins or the API (via the SPI) controls the enable/disable of the Rx datapaths. Note however that further on in the mux chain the RADIO_CONTROL_ARM_MODE chooses whether the ARM or the API/ TRX[A-H]_CTRL pins are able to control the Rx datapaths. This is because the ARM needs to have priority over when it can control the Rx datapaths for calibrations etc. and thus when the ARM takes control of the signal paths it cannot be stopped.

Table 31. Rx#_CTRL_SEL[2:0]

TRX PIN	Rx#_CTRL_SEL[2:0]
TRXA_CTRL	000
TRXB_CTRL	001
TRXC_CTRL	010
TRXD_CTRL	011
TRXE_CTRL	100
TRXF_CTRL	101
TRXG_CTRL	110
TRXH_CTRL	111

STREAM PROCESSOR AND SYSTEM CONTROL

Configuring the TRX[A-H]_CTRL Pins for Tx and Rx Pin Control

Although this can be configured in the configurator here is an example of how to manually edit the profile JSON file for pin control. On line 1767 in the profile JSON file there is an array called txEnMapping and the next eight lines are its elements. Element 0 corresponds to the pin TRXA_CTRL and this continues to element 7 which corresponds to the pin TRXH_CTRL. The decimal value placed in this element is the Tx channel or Tx channels that will be enabled in Tx pin control mode. Since there are up to 8 Tx channels that can be assigned to a TRX[A-H]_CTRL pin there is a bit wise mapping where bit[0] corresponds to Tx0 and bit[7] corresponds to Tx7. An example is shown below to control Tx0-Tx3 when TRXA_CTRL is triggered and Tx4-Tx7 when TRXB_CTRL is triggered. Tx0-Tx3 is bit[0:3] so in binary we get 00001111b and in decimal that is 15 and Tx4-Tx7 is bit[4:7] so in binary we get 11110000b and in decimal that is 240. Therefore, a decimal value of 15 is placed in element 0 and a decimal value of 240 is placed in element 1.

```
"txEnMapping": [
  15,
  120,
  0,
  0,
  0,
  0,
  0,
  0
],
```

On line 1787 in the profile JSON file there is an array called rxEnMapping and the next eight lines are its elements. The same concept as above applies. An example is shown below to control Rx0-Rx2 when TRXE_CTRL is triggered and Rx3-Rx7 when TRXG_CTRL is triggered. Rx0-Rx2 is bit[0:2] so in binary we get 00000111b and in decimal that is 7 and Rx3-Rx7 is bit[3:7] so in binary we get 11111000b and in decimal that is 248. Therefore, a decimal value of 7 is placed in element 4 and a decimal value of 248 is placed in element 6.

```
"rxEnMapping": [
  0,
  0,
  0,
  0,
  7,
  0,
  248,
  0
],
```

SYSTEM CONTROL API FUNCTIONS

Table 32. List of System Control Related API Functions

API Method Name	Comments
adi_ADRV903x_RadioCtrlCfgSet()	Sets the enable/disable of SPI or PIN mode for radio control. The device defaults to SPI mode on power up so this API will need to be called to enable pin mode if necessary.
adi_ADRV903x_RadioCtrlCfgGet()	Gets the radio control mode. SPI or PIN.
adi_ADRV903x_RadioCtrlTxRxEnCfgSet()	Sets the Tx/Rx channels to be turned on and off by the TRX[A-H]_CTRL pins. Will also need to be considered for antenna cal.
adi_ADRV903x_RadioCtrlTxRxEnCfgGet()	Gets the Tx/Rx channels that are controlled by the TRX[A-H]_CTRL pins.

STREAM PROCESSOR AND SYSTEM CONTROL

Table 32. List of System Control Related API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_RxTxEnableSet()	Enables or disables all Rx, ORx and Tx channels that are in SPI control mode (see adi_ADRV903x_RadioCtrlCfgSet). Has no effect on channels that are not set to SPI control mode. Furthermore, channels that are not initialized will not be enabled. For use cases where pin mode is not required, this function can be used to enable/disable the Rx/ORx/Tx signal paths. This function should be called after initialization and loading the stream processor.
adi_ADRV903x_RxTxEnableGet()	Get SPI mode enable/disable status of Rx/Tx/ORx.
adi_ADRV903x_ChannelEnableGet()	Get the enable/disable status of each channel. Works in Pin and SPI control.

TX TO ORX MAPPING

The Tx to ORx mapping concept is vital to obtain the best performance from any calibration that uses an external loopback path for data acquisition. For calibrations that utilize an external loopback path, such as Tx LOL tracking, the ARM must be informed which Tx channel it is currently observing at each ORx channel in order to apply corrections to the appropriate channel. The means to provide this information to the ARM is called Tx to ORx mapping.

There are two interfaces that can be used to indicate the Tx to ORx mapping condition.

1. API Command Interface: Uses an API command adi_ADRV903x_TxToOrxMappingSet(…) to indicate which Tx channel is currently input to either ORx channels.
   a. Advantage: No GPIO pins are required.
   b. Disadvantage: To synchronize that the external path connection matches the Tx to ORx mapping state indicated to the ARM is more challenging than the pin control method. This leads to a possible scenario that the external path connection is in one state while the Tx to ORx mapping state internal to the transceiver is in a different state. This could lead to issues with algorithm performance. May not be suitable for TDD applications due to higher overhead to indicate Tx to ORx mapping to ARM.
2. Pin Control Interface: Uses from 2 to 8 GPIO pins to indicate the Tx to ORx mapping state.
   a. Advantage: Much easier to achieve synchronization between the external path connection and Tx to ORx mapping state as the signaling lines that control an external switch can be used to also indicate the Tx to ORx mapping state provided that logic levels for the transceiver input pins are met.
   b. Disadvantage: GPIO pins are required. Some modes do not have a way to indicate that there is no Tx channel observable at the ORx input.

The following section goes into detail regarding the pin interface for Tx to ORx mapping.

TX TO ORX MAPPING: PIN INTERFACE

There are three aspects to stream processor configuration that needs to be input to generate the proper stream binary for a desired Tx to ORx mapping. In these modes, the GPIO input pins send a signal to the stream processor which indicates the mapping to the ARM processor so it may properly run the calibrations. These are:

1. Tx Observability: The user must configure a Tx Observability attribute for each Tx channel. This defines which ORx channel a specific Tx channel can be observed by. A single Tx channel can only be observable to one ORx channel. It is invalid for any Tx to be observed at both ORx0 and ORx1.
2. GPIO Mapping Mode: The available modes are listed below. For modes featuring fewer pins, there are additional constraints upon the user in terms of configuration of the feedback path.
   - 2-pin Mode
   - 3-pin Mode
   - 4-pin Mode
   - 6-pin Mode
   - 8-pin Mode
3. GPIOs for Tx to ORx Mapping: The user must define which GPIO pins will be used to indicate the Tx to ORx mapping. Any of the 24 digital GPIO pins can be used to indicate the mapping.

STREAM PROCESSOR AND SYSTEM CONTROL

The following section provides specific details for the GPIO mapping modes available to describe what each pin does in the application. For all the tables in the following section, the term TxToORxMap[dx:d0] is the control word for the Tx to ORx mapping state – these can be assigned to any GPIO pin. Changing any of the GPIOs in the TxToOrxMap word will latch the specified Tx to ORx mapping.

Pin Interface: 2-Pin Mode

The 2-pin mode represents the minimum GPIO requirement for Tx to ORx mapping. A constraint is that one mapping must always be active at any time. TxN in the table below can be any Tx with the constraint that no Tx is mapped to ORx0 and ORx1 at the same time. The user can define which Tx is mapped in the ACE GUI, the profile JSON file or via an API.

Table 33. Tx to ORx Mapping 2-Pin Mode Bit Description

TxToOrxMap[d1:d0]	Description
0	TxN → ORX0 and TxN → ORX1
1	TxN → ORX0 and TxN → ORX1
2	TxN → ORX0 and TxN → ORX1
3	TxN → ORX0 and TxN → ORX1

Pin Interface: 3-Pin Mode

The 3-pin mode extends the 2-pin mode by adding a third GPIO pin to indicate that there is no observable Tx channel at both ORx ports.

Table 34. Tx to ORx Mapping 3-Pin Mode Bit Description

TxToOrxMap[d2:d0]	Description
0	TxN → ORX0 and TxN → ORX1
1	TxN → ORX0 and TxN → ORX1
2	TxN → ORX0 and TxN → ORX1
3	TxN → ORX0 and TxN → ORX1
>=4	No observable Tx channel at either ORx channel

Pin Interface: 4-Pin Mode

The 4-pin mode is unique in the sense that the real time status of the pins do not indicate the full mapping for both ORx0 and ORx1. The advantage of this mode is that every Tx can be routed into a single ORx – provided the Tx observability allows it. The disadvantages are that ORx0 and ORx1 mappings cannot be changed simultaneously and the user cannot specify that no Tx is connected to an ORx.

At any given time, the pins indicate the status of one ORx depending on the MSB of the TxToORxMap word. In this mode, the MSB of the TxToOrxMap selects which ORx is to receive the Tx mapping indicated by the 3 LSBs. The following tables describe the bit mapping in this mode.

Table 35. Tx to ORx Mapping 4-Pin Mode TxToOrxMap[d3] Bit Description

TxToOrxMap[d3]	Description
0	ORx0 Tx Mapping Select
1	ORx1 Tx Mapping Select

Table 36. Tx to ORx Mapping 4-Pin Mode TxToOrxMap[d2:d0] Bit Description

TxToOrxMap[d2:d0]	Description
0	TxN Observable at ORx indicated by TxToORxMap[d3]
1	TxN Observable at ORx indicated by TxToORxMap[d3]
2	TxN Observable at ORx indicated by TxToORxMap[d3]
3	TxN Observable at ORx indicated by TxToORxMap[d3]
4	TxN Observable at ORx indicated by TxToORxMap[d3]
5	TxN Observable at ORx indicated by TxToORxMap[d3]
6	TxN Observable at ORx indicated by TxToORxMap[d3]
7	TxN Observable at ORx indicated by TxToORxMap[d3]

STREAM PROCESSOR AND SYSTEM CONTROL

Pin Interface: 6-Pin Mode

The 6-pin mode is like the 3-pin mode except the interface is extended such that independent control of Tx to ORx mapping on either side of the chip is enabled. The bit mapping is described below.

Table 37. Tx to ORx Mapping 6-Pin Mode TxToOrxMap[d2:d0] Bit Description

TxToOrxMap[d2:d0]	Description
0	TxN → ORX0
1	TxN → ORX0
2	TxN → ORX0
3	TxN → ORX0
>=4	No observable Tx channel at ORx0

Table 38. Tx to ORx Mapping 6-Pin Mode TxToOrxMap[d5:d3] Bit Description

TxToOrxMap[d5:d3]	Description
0	TxN → ORX1
1	TxN → ORX1
2	TxN → ORX1
3	TxN → ORX1
>=4	No observable Tx channel at ORx1

Pin Interface: 8-Pin Mode

The 8-pin mode provides an extension of the 4-bit mode which allows simultaneous control of the mapping to ORx0 and ORx1 in contrast to the 4-pin mode which only allows individual control of the mapping to ORx0 and ORx1. The bit mapping is described below:

Table 39. Tx to ORx Mapping 8-Pin Mode TxToOrxMap[d3] Bit Description

TxToOrxMap[d3]	Description
0	ORx0 observes some Tx indicated by TxToOrxMap[d2:d0]
1	ORx0 does not observe a Tx channel

Table 40. Tx to ORx Mapping 8-Pin Mode TxToOrxMap[d6:d4], TxToOrxMap[d2:d0] Bit Description

TxToOrxMap[d6:d4] OR TxToOrxMap[d2:d0]	Description
0	TxN Select
1	TxN Select
2	TxN Select
3	TxN Select
4	TxN Select
5	TxN Select
6	TxN Select
7	TxN Select

Table 41. Tx to ORx Mapping 8-Pin Mode TxToOrxMap[d7] Bit Description

TxToOrxMap[d7]	Description
0	ORx1 observes some Tx indicated by TxToOrxMap[d2:d0]
1	ORx1 does not observe a Tx channel

TX TO ORX MAPPING API FUNCTIONS

Table 42. Tx to ORx Mapping API Functions

API Method Name	Comments
adi_ADRV903x_TxToOrxMappingConfigGet()	Gets the Tx to ORx Mapping Configuration setup.
adi_ADRV903x_TxToOrxMappingSet()	Sets the Tx to ORx external signal routing for Tx calibrations that use the ORx for observation.
adi_ADRV903x_TxToOrxMappingGet()	Gets the Tx to ORx external signal routing for Tx calibrations that use the ORx for observation.

STREAM PROCESSOR AND SYSTEM CONTROL

Table 42. Tx to ORx Mapping API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_TxToOrxMappingPresetAttenSet()	Sets ORx preset Atten value for selected Tx Channel(s) to be used when mapped to an ORx Channel.
adi_ADRV903x_TxToOrxMappingPresetAttenGet()	Retrieves ORx preset Atten value for selected Tx Channel to be used when mapped to an ORx Channel.
adi_ADRV903x_TxToOrxMappingPresetAttenGet_v2()	Retrieves ORx preset Atten value for selected Tx Channel to be used when mapped to an ORx Channel.
adi_ADRV903x_TxToOrxMappingPresetNcoSet()	Sets ORx preset NCO values for selected Tx Channel(s) to be used when mapped to an ORx Channel.
adi_ADRV903x_TxToOrxMappingPresetNcoGet()	Gets ORx preset NCO values for selected Tx Channel to be used when mapped to an ORx Channel.
adi_ADRV903x_TxToOrxMappingPresetNcoGet_v2()	Retrieves ORx preset NCO values for selected Tx Channel to be used when mapped to an ORx Channel.

FRONT END ANALOG SIGNAL PATH

TRANSMIT PATH

Figure 86 below illustrates the transmit analog front end used in the ADRV903x family of devices. It is duplicated eight times for each transmitter. Each transmitter consists of a tuning baseband low pass filter, up-convert quadrature mixers and a RF variable gain amplifier. The baseband filter has a tunable bandwidth of 300MHz to 840MHz. The bandwidth is tuned at initialization via the loopback path and implemented in the ARM firmware. The up converter has fixed gain which reduces quadrature errors that would be associated with attenuation changes in the mixer stages. A unique architecture of up converters was chosen to reduce 3rd and 5th harmonics. By lowering these harmonics reduces the linearity requirements of the RF VGA as well as reducing the risk of aliasing these harmonics in the TX loopback path.

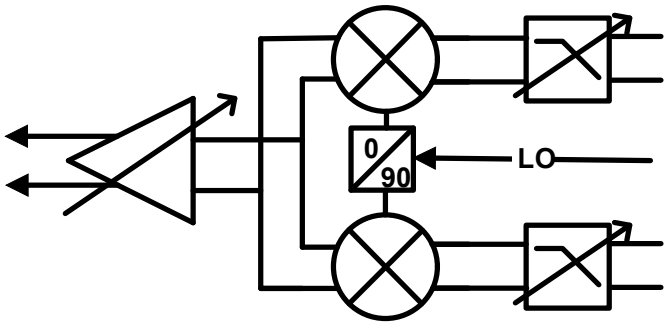


Figure 83. Transmit Analog Front End Block Diagram

The RF VGA has 32dB of attenuation range and higher gain resolution is achieved by use of digital gain adjustments. Transmit power control is implemented to minimize the interaction with the baseband processor.

TX ATTENUATION CONTROL

The ADRV903x uses an accurate and efficient method of transmit power control (Tx attenuation control) that involves a minimum of interaction with the baseband processor. The power control in the transmit chain is implemented with two variable attenuations, one in the digital domain and one in the analog domain. Furthermore, the maximum output level of the transmitter can be adjusted between two levels, allowing a tradeoff between linearity and LOL performance. There are two different modes available to control the attenuation setting of the transmitter. The attenuation can be set immediately via the API, incremented or decremented using GPIO pins to trigger the increment or decrement. See the Digital GPIO Input Modes section for more details on GPIO attenuation controls.

The attenuation is controlled via a lookup table, which is programmed into the product during initialization. The lookup table maps a desired value in dB to the appropriate analog and digital attenuation settings to be applied in the data path. The default table provides a range of 0 dB to 41.95 dB of attenuation, with a step size of 0.05 dB, resulting in 840 available attenuation settings. The step size resolution of 0.05 dB cannot be modified.

The Tx datapath can be configured to automatically ramp the attenuation to the maximum level under certain conditions, such as the JESD link dropping or the Tx PLL unlocking, to prevent spurious transmission in the event of these types of system errors. See the PA Protection section for more details.

TX ATTENUATION API FUNCTIONS

The following commands can be used after device initialization to re-configure and control the Tx attenuators settings. A short description of each API and any usage limitations are listed in Table 43. Details of the parameters, members, enums, etc. for each command are presented in the doxygen help files included with each software build. Please refer to those files when developing your software code. The structures and enumerators for these API functions are detailed in the doxygen documentation.

Table 43. List of Tx Attenuation Related API Functions

API Method Name	Comments
adi_ADRV903x_TxAttenTableRead()	Get Tx RF output attenuation table data.
adi_ADRV903x_TxAttenSet()	Set the desired attenuation in units of mdB.
adi_ADRV903x_TxAttenGet()	Get the desired attenuation in units of mdB.
adi_ADRV903x_TxAttenCfgSet()	Sets the configuration of the attenuation mechanism for one or more channel.

FRONT END ANALOG SIGNAL PATH

Table 43. List of Tx Attenuation Related API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_TxAttenCfgGet()	Gets the configuration of the attenuation mechanism for one or more channel.
adi_ADRV903x_TxAttenPhaseSet()	Sets the Tx Attenuation phase. A bug in this API requires that you invert the sign in front of the phase.
adi_ADRV903x_TxAttenPhaseGet()	Gets the Tx Attenuation phase.
adi_ADRV903x_TxAttenS0S1Set()	A second mode of Tx attenuation control is the S0/S1 attenuation feature. In this mode, the Tx attenuation is set as either S0 (State 0) or S1 (State 1) and the level of a GPIO pin determines whether the Tx channel is set into either S0 or S1. Multiple Tx channels can use the same pin for S0/S1 state control.
adi_ADRV903x_TxAttenS0S1Get()	A second mode of Tx attenuation control is the S0/S1 attenuation feature. In this mode, the Tx attenuation is set as either S0 (State 0) or S1 (State 1) and the level of a GPIO pin determines whether the Tx channel is set into either S0 or S1. Multiple Tx channels can use the same pin for S0/S1 state control.
adi_ADRV903x_TxAttenUpdateCfgSet()	Sets the Tx attenuation update configuration for several channels: attenuation level source (S0, S1), update trigger (GPIO, SPI, None).
adi_ADRV903x_TxAttenUpdateCfgGet()	Gets the Tx attenuation update configuration for several channels: attenuation level source (S0, S1), update trigger (GPIO, SPI, None).
adi_ADRV903x_TxAttenUpdate()	Simultaneously update the Tx attenuation level for several channels. This function only has an effect for channels that have already had their attenuation update trigger set to SPI using TxAttenUpdateCfgSet().

RECEIVER PATH

The ADRV903x RX path is instantiated eight times and shown in Figure 87. It consists of an RF attenuator followed by a current mode passive mixer. The output current of the mixer is passed through a transimpedance amplifier (TIA) filter then digitized with continuous time pipeline ADC. The digital baseband provides most of the filtering and decimation required. Analog power detectors are not in the signal chain but are built into the ADC of each RX channel.

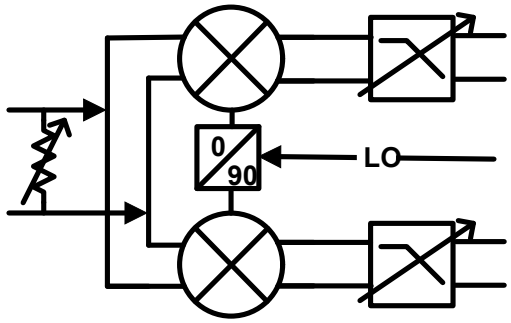


Figure 84. Receive Analog Front End Block Diagram

RF input is 100 ohms differential. Single ended 50-ohm sources would drive into a 1:2 balun. The RF attenuator is pi resistive network with 256 gain settings but only 0-32dB range is used. The RX AGC indexes the digital gain look up table to control the attenuation of the Rx front-end. The methods of gain control is explained in the Gain Control Modes section.

RX MANUAL GAIN API FUNCTIONS

Table 44. Rx Manual Gain API Functions

API Method Name	Comments
adi_ADRV903x_RxGainSet()	Sets the Rx Channel Manual Gain Index. If the value passed in the gainIndex parameter is within range of the gain table minimum and maximum indices, the Rx channel gain index will be written to the transceiver
adi_ADRV903x_RxGainGet()	Reads the Rx AGC Gain Index for the requested Rx channel.
adi_ADRV903x_RxMgcGainGet()	Reads the Rx MGC Gain Index for the requested Rx channel.
adi_ADRV903x_RxTempGainCompSet()	Sets the temperature gain compensation parameter for Rx channel only.
adi_ADRV903x_RxTempGainCompGet()	Gets the temperature gain compensation parameter for Rx channel only. Only one channel can be retrieved per call.
adi_ADRV903x_RxGainTableLoad()	Loads the Rx Gain Table csv file.

FRONT END ANALOG SIGNAL PATH

Table 44. Rx Manual Gain API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_RxGainTableChecksumRead()	Reads Rx Gain Table file checksum value.
adi_ADRV903x_RxGainTableChecksumCalculate()	Calculates Rx Gain Table file checksum value.

OBSERVATION PATH

The ADRV903x Observation RX path is shown in Figure 85 and is instantiated two times for ORx1 and ORx2 paths. As shown the observation path implements direct RF sampling. An RF ADC eliminates the need for a LO which will eliminate spurious often seen with LO coupling. The attenuator is a resistor ladder to provide 16 dB attenuation in analog domain with roughly 1 dB step size. The attenuation is for differential signals, but also works for common mode signals at least starting from 400MHz.The attenuator also provides 50 ohms on-chip RF matching for ORx input. The attenuator is designed to provide 0dB, 1dB, 6dB, and 12dB steps. The user must check the maximum operation input voltage level to the attenuator. The API functions to adjust the attenuation of the ORx are listed below.

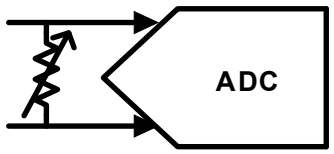


Figure 85. Observation Receive Analog Front End and ADC Block Diagram

ORX ATTENUATION API FUNCTIONS

Table 45. ORx Attenuation API Functions

API Method Name	Comments
adi_ADRV903x_OrxAttenSet()	Set the desired attenuation in units of dB.
adi_ADRV903x_OrxAttenGet()	Get the attenuation in units of dB.

SYNTHESIZER CONFIGURATION

OVERVIEW

The ADRV903x has four phase locked loop (PLL) synthesizers: Clock, RF (x2), and SERDES. Each PLL is based on a fractional-N architecture and consists of a reference clock divider, phase frequency detector, charge pump, loop filter, feedback divider and high-performance voltage-controlled oscillator cores (VCO). The SERDES PLL VCO's tuning range is 8.125 GHz to 16.25 GHz. The CLK PLL, RF PLL0 and RF PLL1 VCOs have a tuning range of 7.1 GHz to 14.2 GHz. Each RF PLL drives its own local oscillator (LO) generator block. The output of the LOGEN block is a divided version of the VCO frequency. The LO divide range goes in binary steps from 2 to 512. No external components are required to cover the entire frequency range. The reference frequency for the PLL is a scaled version of the input device clock (DEVCLK). Figure 86 below illustrates the PLL & associated distribution blocks used in the ADRV903x family of devices.

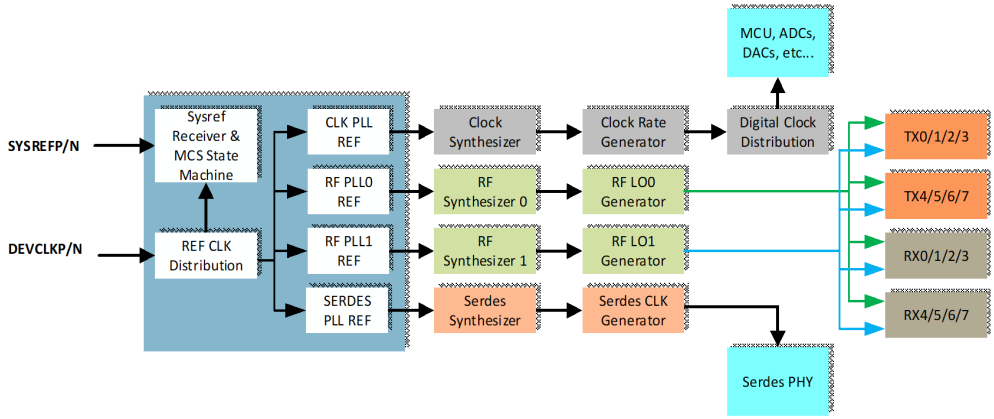


Figure 86. Synthesizer Interconnection and Clock/LO Distribution Block Diagram

DEVCLK

The external DEVCLK is used as a reference clock for the four synthesizers on chip and for optimum performance it needs to be a low noise, high quality clock source. Connect the external clock source to the DEVCLKP (E11) and DEVCLKN (E12) pins via AC coupling capacitors and terminate with 100 Ω close to the device as shown in Figure 87. The device clock receiver is a noise sensitive differential RF receiver. The frequency range and amplitude specs are in the data sheet. The equivalent AC Circuit is given in Figure 88.

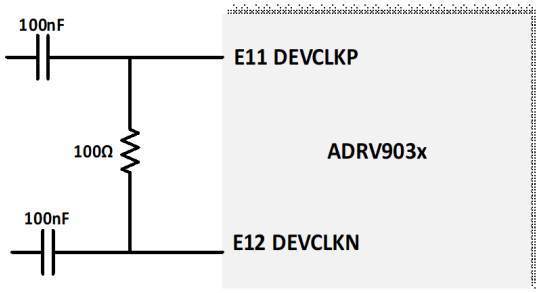


Figure 87. Device Clock Input Connections

SYNTHESIZER CONFIGURATION

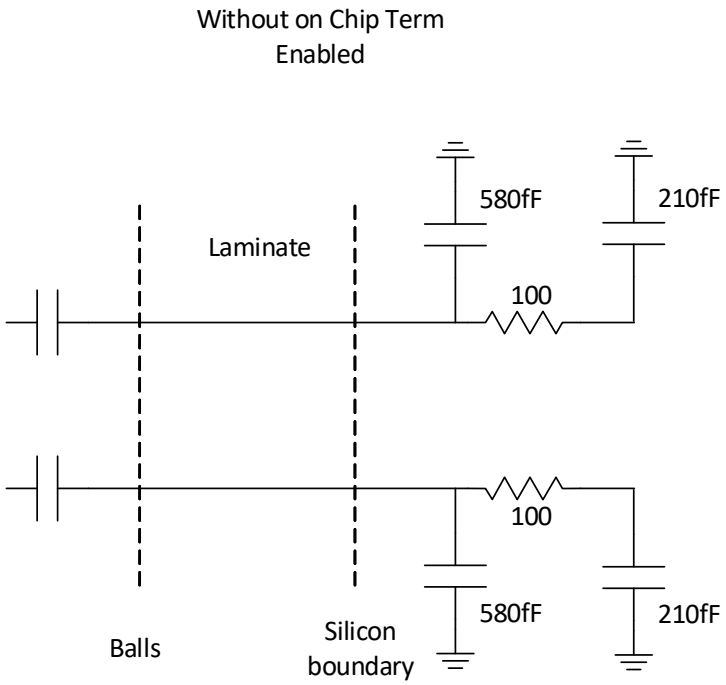


Figure 88. Equivalent AC Circuit

External Reference Clock (DEVCLK) Requirements

Each RF synthesizer takes the DEVCLK reference and multiplies it up to the required LO frequency. The phase noise performance at the final frequency has a dependency on the phase noise of the input reference clock (DEVCLK).

The LO frequency is related to the reference clock by the following equation:

$$F_{lo} = N \times DEVCLK \quad (8)$$

$$DEVCLK\ Noise\ Gain = 20 \times LOG_{10}(N) x\ H(s) \quad (9)$$

Where N is the multiplier applied to the device clock frequency to generate the desired LO frequency and H(s) is the PLL loop transfer function. Inside the loop bandwidth, the noise power from the reference sees a multiplication factor equal to the 20logN term. Outside the loop bandwidth, the multiplied reference noise is attenuated by the loop filter. This means the reference phase noise is typically only a contributor for close-in offsets less than the loop bandwidth. The loop bandwidth and phase margin used for a given phase noise measurement in the datasheet are typically provided in the caption of the figures.

Figure 89 illustrates three DEVCLK noise gain responses with different loop bandwidths and phase margins. Each response is normalized to 0 dB by subtracting out the 20logN term. For example, for a F_{lo} of 2600 MHz and a F_{ref} of 245.76 MHz the gain would be 20.5dB. With the reference clock noise, the RF LO phase noise as specified in the data sheet, and this transfer function, the total noise can be calculated.

SYNTHESIZER CONFIGURATION

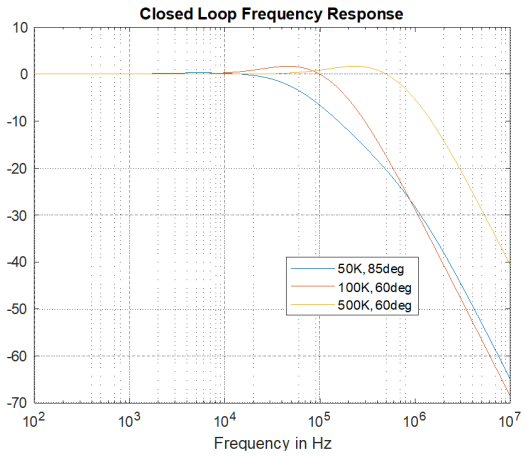


Figure 89. Normalized Devclk Noise Gain/PLL Closed Loop Response

The plots provided in the ADRV903x data sheet are generated with a high-quality, low noise reference clock. To meet the datasheet specs for integrated phase noise it is recommended to use a clock source with a phase noise at 1 kHz offset as per Table 46 below. This scales 6 dB per doubling of DEVCLK frequency. A clock source with a higher phase noise results is some degradation to close in phase noise and integrated phase noise. The table is only a guideline, system engineers should evaluate the performance of the PLL with the clock source for their system. The HMC7044 Clock Generation IC meets the requirements in the first column, while the AD9528 meets the requirements in the second column.

Table 46. DEVCLK Phase Noise Recommendations

DEVCLK Frequency	CLK Phase Noise at 1 kHz offset to meet data sheet integrated phase noise specifications	CLK Phase Noise at 1 kHz offset to keep integrated phase noise specifications within ~10%
122.88 MHz	≤ −133 dBc/Hz	≤ −125 dBc/Hz
245.76 MHz	≤ −127 dBc/Hz	≤ −119 dBc/Hz
491.52 MHz	≤ −121 dBc/Hz	≤ −113 dBc/Hz

Figure 90 shows a simulated phase noise result showing the impact of reference clock phase noise on the RFLO phase noise at 3.6 GHz. Since the reference noise is only impacting the phase noise up to ~20 kHz, the integrated phase noise only changes from 0.16 deg to 0.17 deg when the DEVCLK source is changed from the HMC7044 to the AD9528.

SYNTHESIZER CONFIGURATION

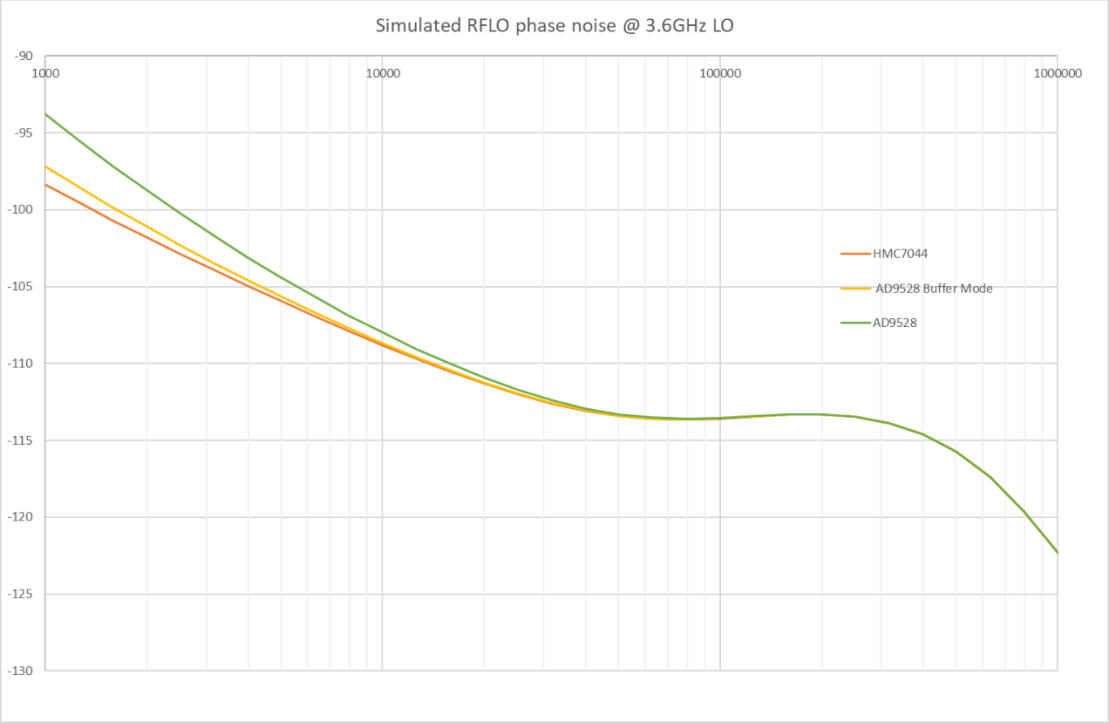


Figure 90. Simulated RFLO Phase Noise at LO = 3.6 GHz Comparing Impact of Different DEVCLK Noise Performance

SYSREF

The SYSREF receiver is a differential receiver and is compatible with LVDS/LVPECL logic levels. The recommendation is to DC-couple this signal with a 100 Ω differential termination resistor placed on the PCB, near pins G11 and G12 as shown in Figure 91 below. The SYSREF traces should be impedance controlled for 50 Ω. DC-coupling is important when using the single-shot SYSREF mode, otherwise the single pulse is distorted as it passes through the capacitor.

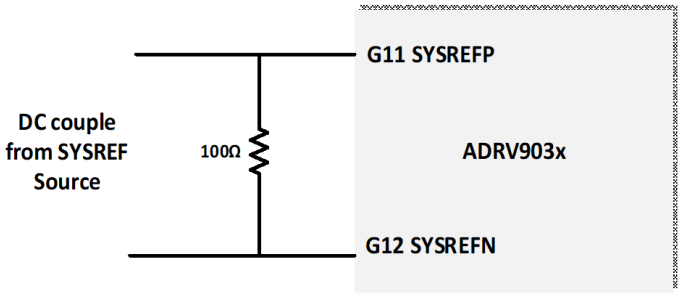


Figure 91. SYSREF Input Connections

SYSREF Setup and Hold Time Requirements

The ADRV903x SYSREF setup and hold time requirements with respect to DEVCLK are in the datasheet. To achieve maximum margin on setup/hold time, it is recommended to align SYSREF with the falling edge of DEVCLK. This would give ~1 ns setup/hold time for the case of a 491.52 MHz DEVCLK.

If the SYSREF and device clock arrive at the same time, i.e. not meeting the setup/hold time requirements, metastability could occur in the system synchronization. In this case you cannot guarantee on which edge of DEVCLK a given ADRV903x will clock in the SYSREF signal, resulting in synchronization variation or latency differences between devices or from power-up to power-up. This is shown in Figure 92. Meeting the setup/hold times ensures SYSREF has no transitions within the keep out window.

SYNTHESIZER CONFIGURATION

Ideally setup/hold times should be confirmed by oscilloscope measurements for each SYSREF/DEVCLK signal pair, as close as possible to the destination pins as possible, e.g. at the termination resistor.

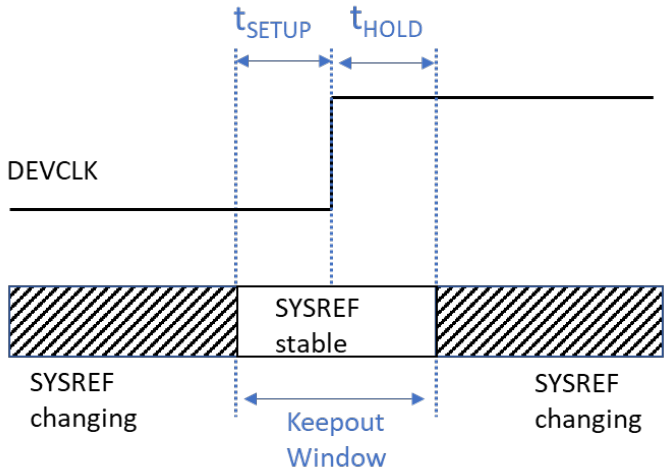


Figure 92. Sysref Timing Keep out Window

CLOCK SYNTHESIZER

The clock synthesizer is used to generate all the clocking signals necessary to run the device and therefore is the first PLL to be brought up during initialization. Although the clock PLL is a fractional-N architecture, the signal sampling relationships to the DEVCLK rates typically require that the synthesizer operates in integer mode. Profiles that are included in the ADRV903x ACE software configure the clock synthesizer appropriately based on the desired IQ and Converter clock rates. Reconfiguration of the clock synthesizer is not necessary after initialization. The clock generation block of the clock synthesizer provides clock signals for the high-speed digital clock, Rx ADC sample and interface clocks, ORx and loopback ADC sample and interface clocks, and Tx DAC sample and interface clocks.

The Rx ADC and Tx DAC operate at the same rate and can be configured to run at different rates, for example, 2949.12 MHz, 3686.4 MHz or 3932.16 MHz. The ORx ADC will either run at the same rate or twice this. To provide the converter clock rates listed, the CLK PLL would operate at either 7372.8 MHz, 7864.2 MHz, or 11796.48 MHz.

For GSM applications we recommend changing the CLKPLL filter Loop bandwidth from its default value to a narrow-band 60 kHz value. This will optimize the phase noise contribution of the CLKPLL at 800 kHz offsets at the expense of slightly higher integrated phase noise. To do this set the clkpll_loop_filter_bw_hz in the profile JSON file = 60000 and clkpll_loop_filter_pm_deg = 60. For non-GSM applications you should use the default setting = 0. This will set the LBW to ~ 500 kHz and phase margin = 60 deg.

RF SYNTHESIZER

The device contains two RF PLLs. Each RF PLL uses the PLL block common to all synthesizers in the device and employs a high performance VCO for best phase noise performance. The reference for RF PLL 0 and 1 are sourced from the reference generation block of the device. The RF PLLs are fractional-N architectures. A default modulus value is programmed automatically by firmware to provide an exact frequency on at least a 1 kHz raster using reference clocks that are integer multiples of 122.88 MHz. More details of the divider options and actual raster achieved are in Table 47. The ORx NCO will also operate on an exact 1 kHz raster providing 0 Hz error between the Tx LO and the ORx NCO.

The RF LO frequency is derived by dividing down the VCO output in the LOGEN block. The tunable range of the RF LO is 450−7100 MHz. The LO divider boundaries are given in Table 47. It is recommended to re-run the init cals when crossing a divide-by-2 boundary or when changing the LO Freq by ±100 MHz or more from the frequency at which the init cals were performed.

Table 47. RF Synthesizer Divider Ranges

	LO Frequency Limits (MHz)									
	Lower Limit	Upper Limit	Lower Limit	Upper Limit	Lower Limit	Upper Limit	Lower Limit	Upper Limit	Lower Limit	Upper Limit
RFPLL0/1	221.875	443.75	443.75	887.75	887.5	1775	1775	3550	3550	7100
Div by	32		16		8		4		2	

SYNTHESIZER CONFIGURATION

LO

Configurable LO Options

A highly configurable LO generation network is implemented in the device to provide flexibility in LO assignment for the two RF LO sources. The LOGEN network is shown in Figure 93 and consists of a root divider located close to the RFVCO and separate leaf dividers located at each Rx and Tx slice. The root divider ratio is from 1/2/4… 64 in binary steps. The leaf divider range is 2/4/8. This setup allows the generation of power of two LO's in each Tx and Rx path and also non-power of two LOs in either 4T4R section. For example, in a multi-band setup, where you want to generate a 3.5 GHz LO for a 4T4R section, 1.8 GHz for 2T2R and 900 MHz for the remaining 2T2R, the RF LOs could be configured as follows:

1. To generate the 3.5 GHz for Tx/Rx 0-3, RFLO1 VCO is programmed to 14GHz followed by divide-by-4, where root divider = 2 and leaf divider = 2.
2. To generate the 1.8 GHz for Tx/Rx 4-6, RFLO2 VCO is programmed to 7.2 GHz followed by divide-by-4, where root divider = 2 and leaf divider = 2.
3. To generate the 900 MHz for Tx/Rx 6-7, RFLO2 VCO is programmed to 7.2 GHz followed by divide-by-8, where root divider = 2 and leaf divider = 4.

In many cases, given the wide bandwidth of the ADRV903x, a single RFLO in combination with the on-chip NCOs can be sufficient to support several multi-band configurations. In this case the unused RFLO can be powered down to save power.

Note that it is not recommended to set RFLO1 = RFLO2; this could cause unwanted coupling between the two PLLs. If a common RFLO is desired, then either of RFPLL1 or RFPLL2 should be set to the desired frequency and muxed to both TxLO and RxLO. That is, the configuration should be set to either TXLO = RXLO = RFLO1 or TXLO = RXLO = RFLO2 with the unused RFLO powered down.

SYNTHESIZER CONFIGURATION

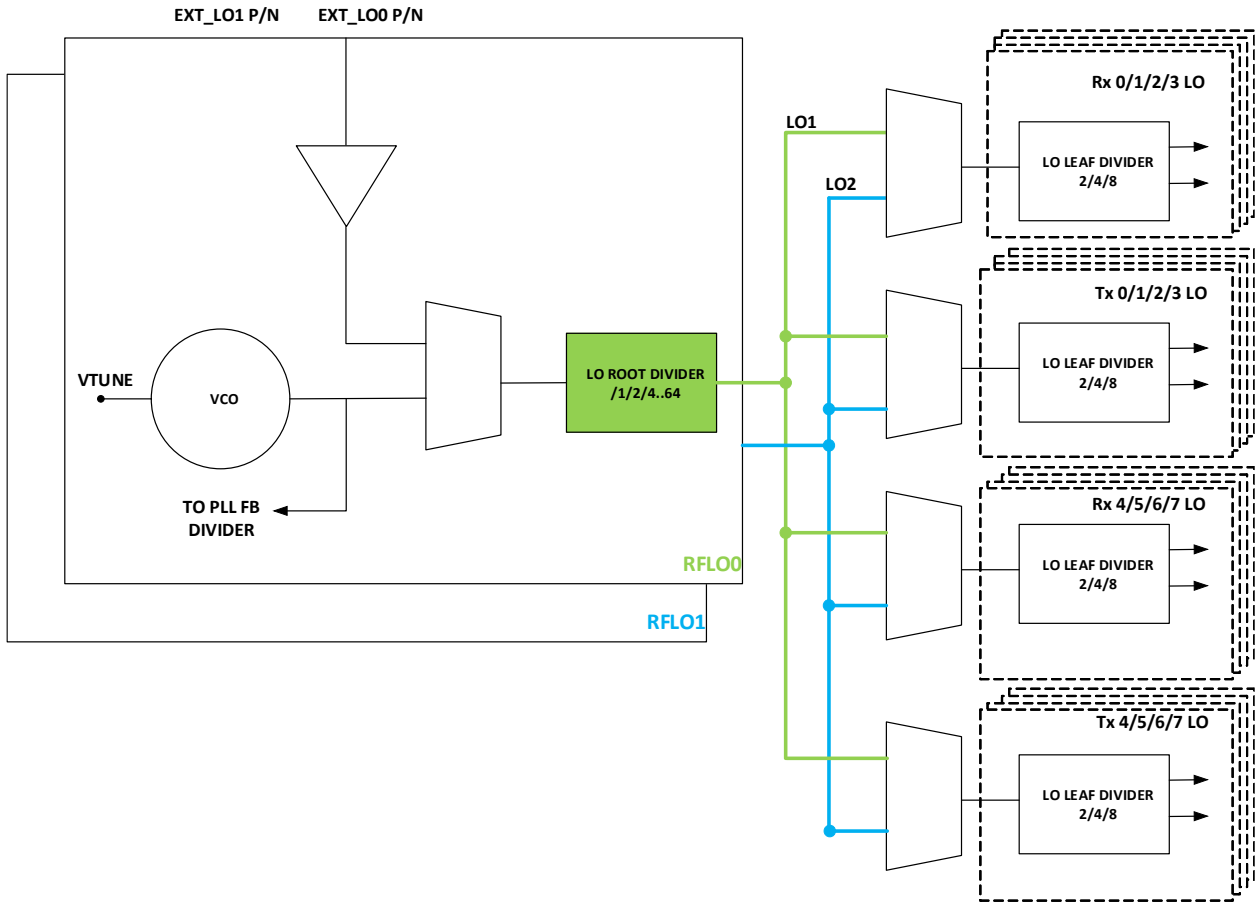


Figure 93. LO Switching Network

External LO

Two external LO ports provide the option to drive the LO with an external LO source to improve the phase noise performance relative to the internal PLL. Refer to Figure 93 for illustration of the external LO connection in reference to the RF LO sources.

The external LO can receive a signal between 3.55 GHz and 12 GHz through a matched differential impedance of 100Ω. The inputs are internally ac-coupled. The external LO frequency should be at least 2x, 4x, or 8x the desired internal LO, depending on the required LO frequency. As an example, Table 48 shows different combinations for selecting the external LO frequency based on the desired internal LO Output.

Table 48.

Desired LO Output Frequency (MHz)	Divider	Required External LO Input Frequency (MHz)
450	8	3600
887.5	8	7100
887.5	4	3550
3000	4	12000
3000	2	6000
6000	2	12000

The on-chip LO dividers provide a programmable LO signal between 450 MHz and 6 GHz for the transmitters and receivers. Refer to the data sheet for the input amplitude requirements on the external LO pin and for the minimum LO divide ratio versus LO frequency.

SYNTHESIZER CONFIGURATION

External LO Setup

The external LO is configured in the ADRV903x JSON file.

To enable external LO 0 set line 227 to be true.

```
"use_extLO0": true,
```

To set the frequency, the external LO pin will be accepting modify line 228. This will automatically set the internal dividers.

```
"extLO0_frequency_in_kHz": 0,
```

The same applies to external LO1, except it is lines 229 and 230.

Another two variables have to be set:

- rx_lo_frequency_in_kHz See lines 108, 119, 130, 141.
- tx_lo_frequency_in_kHz See lines 157, 171, 185, 199.

The following mathematical relationship must also be maintained:

rx_lo_frequency_in_kHz/tx_lo_frequency_in_kHz multiplied by 2^N must equal extLO0_frequency_in_kHz and extLO1_frequency_in_kHz, where N is from 1 to 7.

LO CONFIGURATION USING API FUNCTIONS

The basic LO configuration, i.e. whether you are using the internal or external LO, using a single LO or dual LO and the required Rx and Tx LO frequencies are specified in the profile binary, as described in the Software Architecture section, which is then used to configure the device during device initialization.

The following commands can be used after device initialization to re-configure and control the internal and external LO settings. A short description of each API and any usage limitations are listed in Table 49. Details of the parameters, members, enums, etc. for each command are presented in the doxygen help files included with each software build. Please refer to those files when developing your software code.

You can set the desired LO Frequency using the adi_ADRV903x_LoFrequencySet() command as listed in the table. Note, the PLL unlock bits in the GP Interrupt Mask are masked off temporarily when setting a new LO Frequency, so this should avoid the GP Interrupt unnecessarily triggering while the PLL is temporarily unlocked as you change frequency. The PLL Loop Bandwidth and Phase Margin are automatically set by the firmware and should normally be left at their defaults. You can also manually Set and Get these values. If you do need to update the PLL loop bandwidth, you will need to first call the LoopFilterSet() command followed by an LoFrequencySet() command. The PLL loop bandwidth will be updated on the call to LoFrequencySet. The structures and enumerators for these API functions are detailed in the doxygen documentation. You can check the lock status of the various on-chip PLLs via the API also.

Table 49. List of LO Configuration Related API Functions

API Method Name	Comments
adi_ADRV903x_LoFrequencySet()	You need to pass the loName of the PLL you want to program, along with the LO frequency in Hz and a configuration option to select the NCO to auto update or not with the LO frequency change. Note, you should not change the frequency for either the CLKPLL or SerdesPLL from their configured defaults.
adi_ADRV903x_LoFrequencyGet()	You need to pass the loName of one of the four PLLs you want to readback the LO frequency in Hz from. Note, if you are using different TxLO/RxLO leaf divider values across the slices, then this method will just return the highest frequency. E.g. if LO0 = RxLO is set to 1.8GHz for Rx0/1/2/3 and a further divide-by-2 or 0.9GHz for Rx4/5/6/7, then this method will return a frequency of 1.8GHz. In this case you should use the adi_ADRV903x_RxTxLOFreqGet(…) method listed below.
adi_ADRV903x_RxTxLOFreqGet()	Similar to LOFrequencyGet, but this method allows you to readback the LO frequency individually for each Rx or Tx slice.
adi_ADRV903x_RxLoSourceGet()	Gets the LO source for the selected Rx Channel.
adi_ADRV903x_TxLoSourceGet()	Gets the LO source for the selected Tx Channel.
adi_ADRV903x_LoopFilterSet()	This function allows the user to set the PLL loop filter bandwidth & phase margin.
adi_ADRV903x_LoopFilterGet()	This function allows the user to get the PLL loop filter bandwidth & phase margin.

SYNTHESIZER CONFIGURATION

Table 49. List of LO Configuration Related API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_PllStatusGet()	This function allows the user to get the PLL lock status for each of the four on-chip PLLs.

MULTICHIP SYNCHRONIZATION (MCS)

Multi-chip synchronization (MCS) is a mechanism on-chip to allow deterministic timing to be established and thus enable data alignment down to the sample level across multiple serial lane links. MCS will therefore align multiple ADRV903x in the system.

As shown in Figure 94, the multi-chip synchronization (MCS) state machine takes in both SYSREF and DEVCLK as its inputs. As part of device initialization, the multi-chip synchronization (MCS) state machine re-synchronizes the system reference signal (SYSREF) to the local device clock (DEVCLK) domain and uses this SYSREF to reset critical timing blocks in the device in a deterministic fashion.

The MCS sequence is performed in four stages as shown in Figure 94, each one initiated with a SYSREF rising edge.

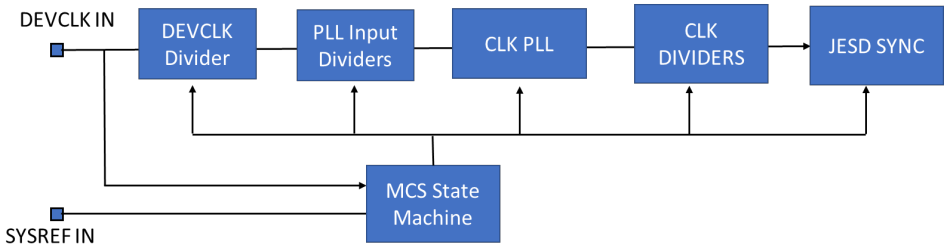


Figure 94. MCS State Machine

RF PLL PHASE SYNCHRONIZATION

The RFPLL Phase Synchronization description is included at this time for prototyping and evaluation purposes only. Consult ADI for function availability.

This function has been added to allow the internally generated LO to be phase synchronized and aligned across multiple devices. This function allows all devices to align the RFPLL to the same phase, and therefore the phase between each device is aligned at startup so that the phase between devices is repeatable and fixed. As part of device initialization, the multi-chip synchronization (MCS) state machine re-synchronizes the system reference signal (SYSREF) to the device clock (DEVCLK) domain and uses this SYSREF to reset the data converter clocks and all other clocks at the baseband rate. These same signals are also used to initialize an on-chip counter which is later used during PLL programming to synchronize the LO phase. No additional signals are required to take advantage of the LO phase synchronization mechanism. From the on-chip counter and the PLL fractional word programming, a digital representation of the desired LO phase can be computed at each PLL reference clock edge and is remembered in the digital phase accumulator (DPA).

The LO phase sync hardware operates by directly sampling the LO signal (in quadrature) using the PLL reference clock signal (DEVCLK). Averaging is required to increase the accuracy of the LO phase measurement, so at every sample, the observed LO phase is de-rotated by the digitally desired phase. This is done by performing a vector multiplication of the complex conjugate of the digital phase. The result is a vector representing the phase difference between the LO and the digitally desired phase, and these vectors can be averaged over many DEVCLK cycles to obtain an accurate measurement of the phase adjustment required.

After the phase difference has been measured, the adjustment can be applied into the PLL's first stage sigma delta modulator (SDM) by adding it to the first stage modulator input. The total adjustment amount is added over many reference clock cycles in order to stay within the PLL loop bandwidth and not cause the PLL to come unlocked. To counteract temperature effects after calibration, a PLL phase tracking mode can be activated. Figure 95 is a block diagram of the phase synchronization system.

SYNTHESIZER CONFIGURATION

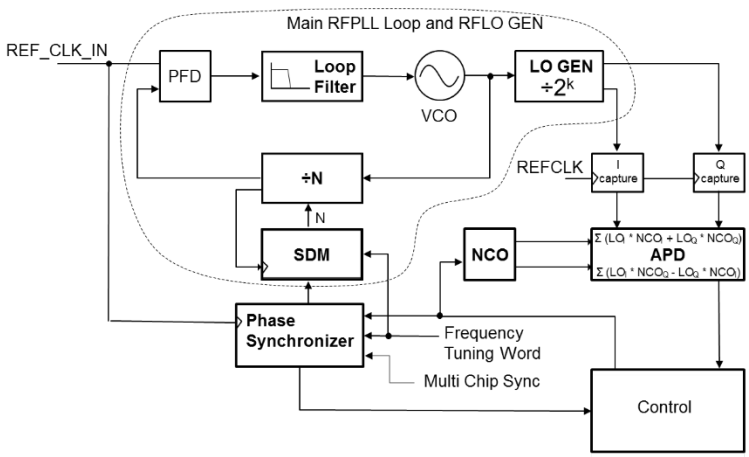


Figure 95. LO Phase Sync Functional Diagram

System Level Considerations

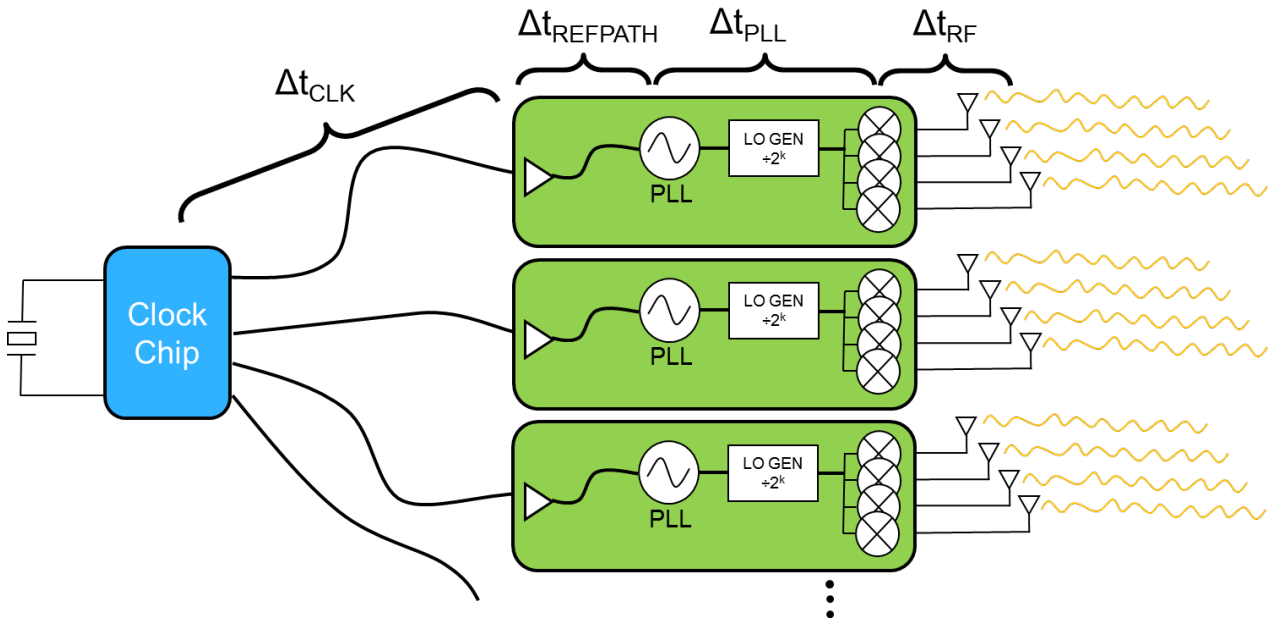


Figure 96. High Level Contributions to System Phase Per Antenna

Overall phase synchronization is determined by a number of factors, including the board level clock routing (t_{CLK}), the on-chip reference path routing ($t_{REFPATH}$), the PLL and LO divider path (t_{PLL}), and the RF & antenna paths (t_{RF}). In a beamforming/MIMO system a system level antenna calibration is performed to equalize the sum of these paths between all channels. The goals of this are:

- Reduce the complexity of the antenna calibration by initializing to a more consistent startup condition with deterministic PLL phase and LO divider state,
- Reduce the temperature dependence of the system phase synchronization to allow the antenna calibration to run less frequently during operation,
- Allow transceivers to be stopped and started in an operational system and “hot synchronize” with the other transceiver elements.

The LO phase synchronization method addresses the initial PLL phase and LO divider state and reduces their temperature dependence to a negligible amount compared to other sources of phase drift in the system.

ARM PROCESSOR AND DEVICE CALIBRATIONS

ARM PROCESSOR

The transceiver is equipped with two Arm M4 processors, CPU0 and CPU1. The firmware for these Arm processors is loaded during the initialization process. CPU0 is loaded first, followed by CPU1 then CPU0 is started which will then start CPU1. The firmware memory size is 641 kB. The Arms are tasked with configuring the transceiver for the selected use case, performing initial calibrations of the signal paths and maintaining device performance over time through tracking calibrations.

Arm State Machine Overview

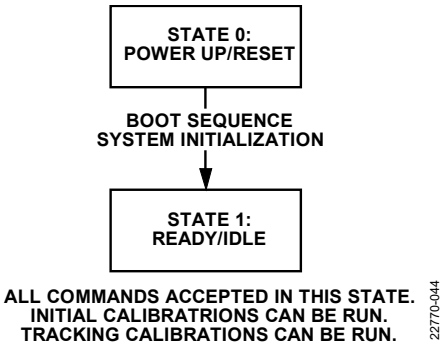


Figure 97. Arm State Machine

State 0: When the Arm core is powered up, the Arm moves into its power-up/reset state. The Arm firmware image is loaded at this point. Once the Arm image has been loaded, the Arm is enabled and begins its boot sequence.

State 1: After the Arm has been booted, it enters its ready/idle state. In this state, it can receive configuration settings or commands (instructions), such as performing the initial calibrations or enabling tracking calibrations.

System Initialization

This section provides a detailed description of the initialization procedure. There are three main sections to the initialization procedure.

1. Pre-MCS Init
2. MCS
3. Post-MCS Init

Pre-MCS Init initializes the device up to the multi-chip synchronization (MCS) procedure. The pre-MCS init sequence is split into two commands that the application layer function calls. These are adi_ADRV903x_PreMcsInit() and adi_ADRV903x_PreMcsInit_NonBroadcast(). adi_ADRV903x_PreMcsInit() is a broadcastable command that can simultaneously issue commands to multiple transceivers to save time during system initialization for systems with multiple transceivers. Arm and stream binaries are downloaded to the chip during this step. The broadcast functionality is realized by issuing SPI write commands only. The adi_ADRV903x_PreMcsInit_NonBroadcast() verifies that the Arm is programmed properly by verifying the Arm checksum and that the Arm is in the Ready/Idle state.

The multichip sync (MCS) step uses SYSREF pulses to synchronize internal clocks within the transceiver. Required for deterministic latency of JESD links so is needed even if only one ADRV903x in the system. The function adi_ADRV903x_MultichipSyncSet() sets up the transceiver to listen for incoming SYSREF pulses and initiate the MCS procedure.

Post-MCS Init continues initialization following MCS. The application layer command that performs the post-MCS initialization is adi_ADRV903x_PostMcsInit(). This command programs the PLLs, configures the radio control initialization structure and instructs the Arms to perform initialization calibrations.

Pre-MCS Initialization

The adi_ADRV903x_PreMcsInit() function is in the adi_ADRV903x_utilities.c file. It performs a sizeable part of the full chip initialization. The first step is to load the Arm image using adi_ADRV903x_CpuImageLoad(). The Arm image, ADRV903x_FW.bin, is provided in the 'firmware' folder of the GUI installation folder. Following the Arm firmware image being loaded, the next step is to load the device configuration into

ARM PROCESSOR AND DEVICE CALIBRATIONS

data memory using adi_ADRV903x_CpuProfileWrite(). The Arm is then started and begins its boot sequence. This process is initiated by adi_ADRV903x_CpuStart().

As part of the boot sequence, the Arm configures the device for the required profile (Tx/Rx/ORx path configuration as determined by the use case), configures and enables the clock PLL and configures the JESD framers and deframers. The Arm also computes a checksum for the Arm firmware image loaded, for each of the streams loaded and the profiles loaded (determining if they are valid profiles). adi_ADRV903x_CpuStartStatusCheck() is called after the arm boots and compares the computed checksums during boot to the precomputed checksums. If a checksum is found not to be valid, this function returns an error.

MCS

The MCS procedure resets critical timing blocks and each one is initiated with a sysref rising edge. The sysref signal needs to be source synchronous with the ADRV903x device clock. The function adi_ADRV903x_MultichipSyncSet() sets up the transceiver to listen for incoming SYSREF pulses and initiate the MCS procedure.

Post-MCS Initialization

After the MCS sequence has been completed, the Arm is ready to configure the radio, perform its initialization calibrations and bring up the JESD link. This is achieved using the adi_ADRV903x_PostMcsInit() function. Once complete, the tracking calibrations can be enabled. The RF data paths can then be enabled using either SPI or pin modes.

Note that there is no absolute requirement to follow this sequence. The initialization calibrations and tracking calibrations do not need to be run in order for the paths to be enabled in the device. It is ultimately up to the user to ensure that the paths have been correctly configured prior to operation.

ARM Memory Dump

A useful debug tool is the ability to see what is in the Arm's memory when an issue occurs. This can be achieved by using the adi_ADRV903x_CpuMemDump() function. This function dumps the ADRV903x Arm program and data memory. The binary file that is generated needs to be sent to Analog Devices for analysis.

ARM API FUNCTIONS

Table 50. List of ARM API Functions

API Method Name	Comments
adi_ADRV903x_CpuImageLoad()	Loads the ADRV903x CPU Binary Image.
adi_ADRV903x_StreamImageLoad()	Loads the ADRV903x Stream Binary Image.
adi_ADRV903x_DeviceInfoExtract()	Extract the Init info from the ADRV903x CPU Profile Binary Image.
adi_ADRV903x_CpuProfileImageLoad()	Loads ADRV903x CPU Profile Binary Image.
adi_ADRV903x_PreMcsInit()	Executes ADRV903x Pre-MCS Initialization Sequence
adi_ADRV903x_PreMcsInit_NonBroadcast()	Executes the non-broadcastable part of Pre-MCS Initialization Sequence.
adi_ADRV903x_MultichipSyncSet()	Prepare for incoming SYSREF pulses to synchronize the internal clock tree.
adi_ADRV903x_MultichipSyncStatusGet()	Read the multi-chip sync status.
adi_ADRV903x_PostMcsInit()	Executes ADRV903x Post-MCS Initialization Sequence.
adi_ADRV903x_CpuMemDump()	Captures the ADRV903x CPU program and data memory.

DEVICE CALIBRATIONS

The Arm is tasked with performing calibrations for the transceiver to achieve its performance specifications. These are split into two categories: initial calibrations which are run either before the transceiver is operational or after LO frequency change; and tracking calibrations which are used to maintain performance during runtime.

INITIAL CALIBRATIONS

The Arm processor in the transceiver is tasked with scheduling/performing initial calibrations to optimize the performance of the signal paths prior to device operation. These calibrations are run as part of the utility API function adi_ADRV903x_PostMcsInit( ).

ARM PROCESSOR AND DEVICE CALIBRATIONS

In some cases, it is required to run an initial calibration outside of adi_ADRV903x_PostMcsInit(). This adi_ADRV903x_InitCalsRun() command instructs the Arm to perform the requested calibrations.

Table 51 shows the bit assignments of the calibration mask. Note that Table 51 provides a full list of initialization calibrations for the device. Some initial calibrations are not available for certain transceivers and applications.

The Arm sequences the initial calibrations as required, not necessarily in the bit order presented below. It is mandatory that the user wait for calibrations to complete before continuing with the initialization of the device.

Table 51. adi_ADRV903x_InitCalibrations Bit Assignments

Bit	Enum	Calibration	Description
D0	(Reserved)		
D1	ADI_ADRV903X_IC_RX_DC_OFFSET	Rx DC Offset	Corrects for DC Offset within the receiver chain.
D2	ADI_ADRV903X_IC_ADC_RX	ADC Rx	Calibrates the receiver ADC
D3	ADI_ADRV903X_IC_ADC_ORX	ADC ORx	Calibrates the observation receiver ADC
D4	ADI_ADRV903X_IC_ADC_TXLB	ADC Tx Loopback	Calibrates the Tx loopback receiver ADC
D5	ADI_ADRV903X_IC_TXDAC	Tx DAC	Calibrates the transmitter DAC
D6	ADI_ADRV903X_IC_TXBBF	Tx BB filter	Tunes the corner frequency of the transmitter filter.
D7	ADI_ADRV903X_IC_TXLB_PATH_DLY	Tx LB path delay	Computes the transmitter to internal loopback path delay, which is required for the TxQEC initial calibration and tracking.
D8	(Reserved)		
D9	ADI_ADRV903X_IC_HRM	HRM	Performs HRM (harmonic reject mixer) harmonics rejection corrections.
D10	ADI_ADRV903X_IC_TXQEC	TxQEC	Performs an initial QEC calibration for the transmitter path. It utilizes the transmitter path and an internal loopback path
D11	ADI_ADRV903X_IC_TXLOL	TxLOL	Performs an initial LO leakage calibration for the transmitter path. Utilizes the transmitter path and the internal loopback path
D12	ADI_ADRV903X_IC_SERDES	SerDes	Performs an initialization calibration for the JESD 204C data interface.
D13	(Reserved)		
D14	(Reserved)		
D15	(Reserved)		
D16	(Reserved)		
D17	ADI_ADRV903X_IC_TXRX_PHASE	Ext_LO phase	Calibrates Ext_LO phase for phase consistency across channels

Initialization Calibrations Durations

To achieve best performance, the transceiver features autonomous internal calibrations that are performed during device initialization. The calibrations are run in the Post-MCS section of device initialization. The majority of the calibrations are run with a single API call once the calibration structure is set. These are the internal calibrations that utilize internal loopback paths.

All of the calibrations are overseen and scheduled by the Arm processor so the user does not need to be concerned about what order the calibrations are run. The sequence is defined such that those calibrations that depend on others are scheduled appropriately. The amount of time it takes for the calibrations to complete are related to the internal high speed clock and the resulting IQ rates of the Rx, Tx and ORx paths. The Arm clock is derived from the clock PLL.

In the following diagram the slices show the relative timing of each common initialization calibration relative to the total time. Some of the calibrations are very short and mostly involve for example loading coefficients and initializing for operation or measuring the delay of the calibration path. Some others require observation of either internally generated calibration tones or pseudo-random noise to calculate the required coefficients that are used to define the characteristics of the channel. Still others for example the Tx QEC calibration use an algorithm to determine the correction factors which can be influenced by the actual load conditions the transmitter is connected to. For these reasons, the amount of time each of the calibrations needs to complete may vary slightly. The calibration times will vary with UseCase and can also vary with the software version, for example where improvements were made to reduce Serdes Initcal times in later software. The InitCal times are measured and stored by the internal ARM processer and can be queried using the InitCalsWait_v2() or InitCalsDetailedStatusGet_v2() APIs.

ARM PROCESSOR AND DEVICE CALIBRATIONS

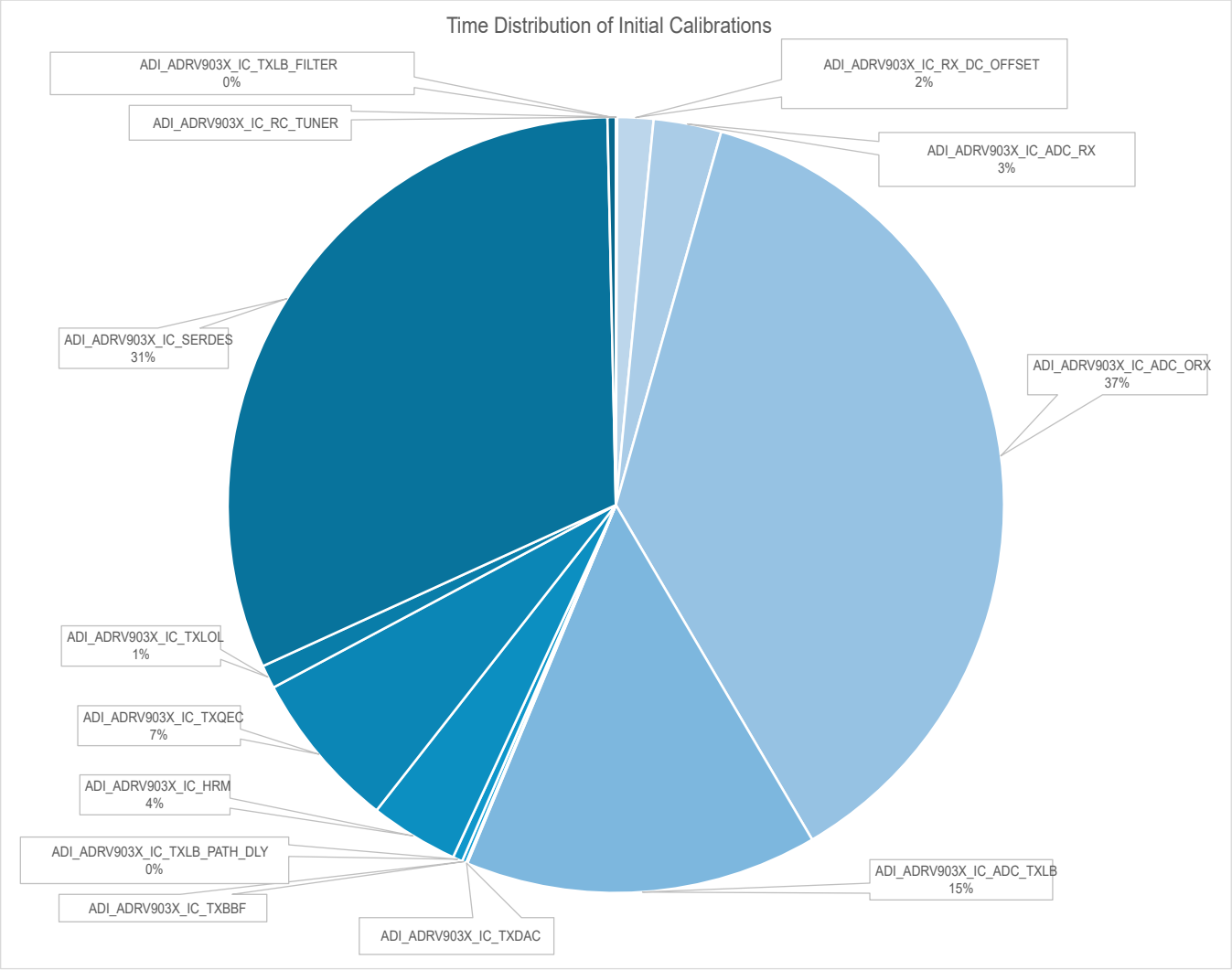


Figure 98. Initial Calibration Timings

Table 52. Inital Calibration Timings

Cal	Time (s)
ADI_ADRV903X_IC_RC_TUNER	0.01
ADI_ADRV903X_IC_RX_DC_OFFSET	0.26
ADI_ADRV903X_IC_ADC_RX	0.49
ADI_ADRV903X_IC_ADC_ORX	6.45
ADI_ADRV903X_IC_ADC_TXLB	2.55
ADI_ADRV903X_IC_TXDAC	0.01
ADI_ADRV903X_IC_TXBBF	0.03
ADI_ADRV903X_IC_TXLB_PATH_DLY	0.07
ADI_ADRV903X_IC_HRM	0.64
ADI_ADRV903X_IC_TXQEC	1.15
ADI_ADRV903X_IC_TXLOL	0.17
ADI_ADRV903X_IC_SERDES	5.46
ADI_ADRV903X_IC_TXLB_FILTER	0.06
Total	17.34

ARM PROCESSOR AND DEVICE CALIBRATIONS

TRACKING CALIBRATIONS

The Arm processor is tasked with ensuring that QEC and LOL corrections are optimal throughout device operation over time, attenuation and temperature. It achieves this by performing calibrations at regular intervals. These calibrations are termed tracking calibrations and utilize normal traffic data to update the path correction coefficients. The adi_ADRV903x_TrackingCalsEnableSet() API function enables the tracking calibrations in the Arm. Table 53 shows the bit assignments of the enable mask.

Table 53. adi_ADRV903x_TrackingCalibrationMask Bit Assignments

Bit	Enum
D0	ADI_ADRV903X_TC_RX_QEC
D1	ADI_ADRV903X_TC_TX_LOL
D2	ADI_ADRV903X_TC_TX_QEC
D3	ADI_ADRV903X_TC_TX_SERDES
D4	ADI_ADRV903X_TC_RX_ADC
D5	ADI_ADRV903X_TC_TX_LB_ADC
D6	ADI_ADRV903X_TC_ORX_ADC

SYSTEM CONSIDERATIONS FOR CALIBRATIONS

The following diagrams are used to show how the transceiver is configured for notable calibrations with external system requirements, such as the QEC and LOL calibrations. In all diagrams, gray lines and blocks are not active in the calibration. Lines showing the path of the LOs are shown in color to distinguish them from the signal paths. A brief explanation of the calibration is provided. Note that as the Arm performs each of the initial calibrations, it is tasked with configuring the device as per Figure 100, with respect to enabling/disabling paths, for example. No user input is required in this regard.

TX LO LEAKAGE CALIBRATION

The ADRV903x uses a zero-IF architecture which has energy transmitted at the Local Oscillator (LO) frequency. To reduce this undesired emission, the transceiver has a Tx LO Leakage (LOL) correction algorithm. For optimal performance, the Tx LOL calibration requires an initial calibration followed by a tracking calibration used during runtime operation. The Tx LOL algorithm applies complex valued correction to both I and Q data paths at the Tx QEC block in order to reduce the Tx LOL level observed at the output of the transmitter.

TX LOL INITIAL CALIBRATION

The Tx LOL initial calibration is used to make an initial correction to the LO Leakage level. The initial calibration uses the internal loopback path only, so it is not necessary to provide an external loopback connection. The signal path used during the initial calibration is in the figure below where unused components are greyed out.

ARM PROCESSOR AND DEVICE CALIBRATIONS

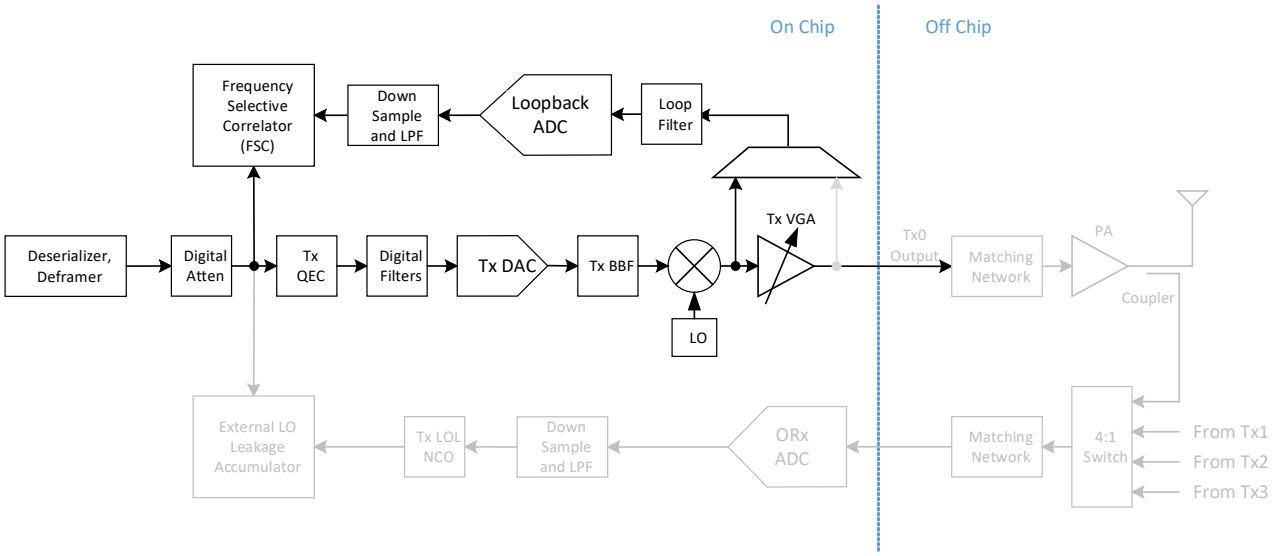


Figure 99. Tx LO Leakage Initial Calibration Correction Loop

Each Tx channel has a corresponding internal loopback channel that is used for calibration purposes. The loopback channel has a correlator block (FSC) which allows for correlation of the transmitted signal to the internally observed loopback signal. The correlation generates LO leakage correction coefficients. Because the Tx LOL initial calibration uses the internal loopback path, the Tx to ORx mapping state (i.e. which Tx channel is connected to a specific ORx channel) does not matter for the operation of this calibration.

During the Tx LOL initial calibration, the user does not need to be transmitting traffic data. Instead, the device will generate perturbations to estimate the initial correction for the channel. Additionally, the Tx VGA will be powered down during this calibration to prevent large level signals entering the power amplifier stage. Note, it is also ok to send data or an idle pattern over JESD when the TxLOL InitCal is run, as the firmware will temporarily disconnect the input data during the cal.

TX LOL TRACKING CALIBRATION

The Tx LOL initial calibration is a prerequisite before enabling the Tx LOL tracking calibration. The Tx LOL tracking calibration uses an external loopback loop to be able to track and correct changes over time. The external loop is shown in orange in the next figure.

ARM PROCESSOR AND DEVICE CALIBRATIONS

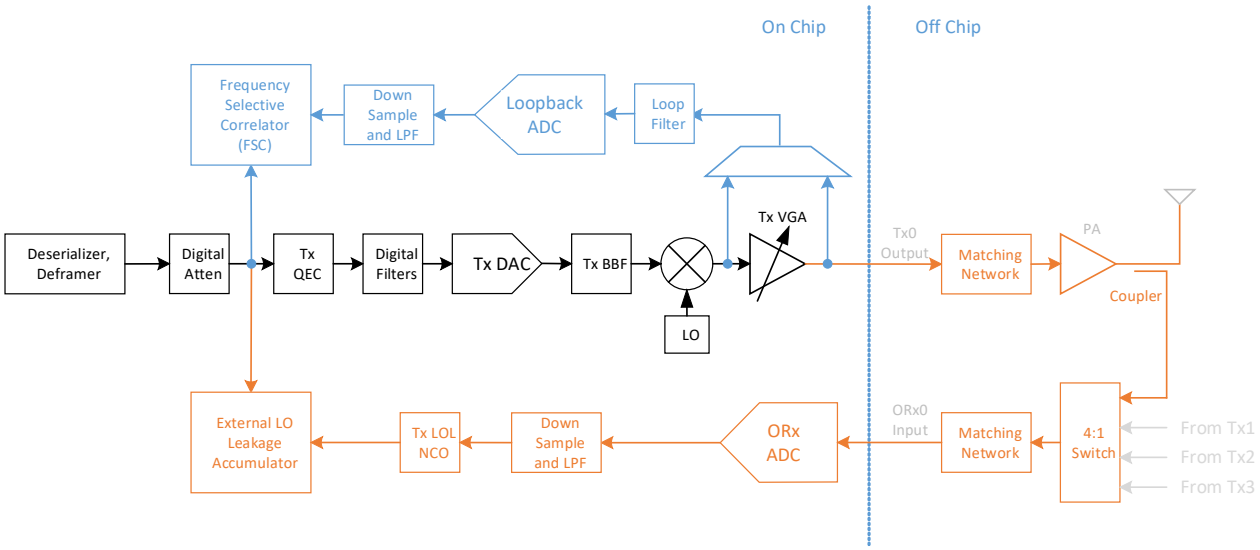


Figure 100. Tx LO External Tracking Calibration Loop (Orange Loop). This shows a sample configuration that uses a 4:1 mapping of Tx0/1/2/3 into ORx0 channel and currently Tx0 is available for external path observation.

The necessity of the tracking calibration is a result of the variable levels of LOL emissions over time and operating conditions (temperature and voltage) of the device. At this time, it is recommended that users that require data sheet performance levels for Tx LOL include support for Tx-ORx mapping in their software to enable the utilize Tx LOL tracking loop. When Tx LOL tracking calibration is running, the user can transmit regular traffic data and the algorithm will be able to discern the emitted LOL level and apply improvements in the correction, if possible.

The tracking calibration is responsible for building up a correction table of LOL corrections versus Tx attenuation.

Tx LOL Tracking Calibration

The Tx LOL tracking loop allows the observation of contributors to LO leakage at the output of the PA. This allows for observation of LO Leakage contributors external to the device and generates corrections accordingly at the Tx QEC block.

The Tx LOL tracking loop requires knowledge of the Tx-ORx mapping state in order to determine which Tx is currently connected to an ORx and can apply the correction to the correct transmitter channel. If a Tx channel is never mapped to an ORx channel, the Tx LOL tracking calibration will not run for that Tx channel.

Note that the Tx LOL tracking calibration can operate while the ORx channel is enabled and sending data to the BBP.

External Path System Requirements for LOL Tracking

- Each Tx must be observable at the ORx at least 100ms every 6 seconds.
- Tx observation duration must be at least 1 symbol (17µs). A 1-symbol every 1-ms configuration is supported.
- The gain from Tx input to ORx output must be at least −10 dB. That is the digital input level to the Tx DAC to the digital output level from the ORx ADC. For example, if you are transmitting a signal with average power of −13dbFS at the TxDAC output, the ORx signal at the ORX ADC output should be > −23dbFS.
- A Tx channel must only be observed on the same ORx channel. For example, this means that Tx0 should not be looped into ORx0 and ORx1 at different times. This affects the ability of the calibration to estimate characteristics of the Tx0 to ORx0 or Tx0 to ORx1 external loopback path which can lead to issues. An acceptable loopback scenario in this example is that Tx0 would only be observed at the ORx0 port.
- If the gain of the path between Tx output and ORx input changes suddenly the user must issue a reset channel command using the TxLolReset() API. You can use the SOFT_RESET to reset the learned external channel while keeping the LOL corrections words relatively constant. You can use the HARD_RESET to reset both the channel and the stored correction from the TxLOL InitCal. Because the LOL corrections words are reset in this case, the LO leakage could temporarily degrade to uncorrected levels, but the convergence time should be quicker in this case as the calibration will make more aggressive updates. You do not need to use the TxLolReset for gradual gain

ARM PROCESSOR AND DEVICE CALIBRATIONS

changes due to temperature dependent gain variation but specifically to larger step gain changes due to, for example, an external gain change through a Digitally Stepped Attenuator (DSA). Sudden gain or phase variation on the external path may cause instability algorithm performance.

TX QEC CALIBRATION

The ADRV903x uses a zero-IF Tx architecture. Quadrature errors within the transmit path shows up as unwanted energy on image frequency. To reduce this undesired emission, the transceiver has a Tx Quadrature Error Correction (QEC) algorithm. For optimal performance, the Tx QEC calibration requires an initial calibration followed by a tracking calibration used during runtime operation. The Tx QEC algorithm applies complex valued correction to I and Q data paths at the Tx QEC block in order to reduce the unwanted emissions at the image frequency observed at the output of the transmitter.

TX QEC INITIAL CALIBRATION

The Tx QEC initial calibration is used to make an initial correction to the quadrature error. The initial calibration uses the internal loopback path that samples the Tx signal before Tx VGA. The signal path used during the initial calibration is shown in the figure below where unused components are greyed out.

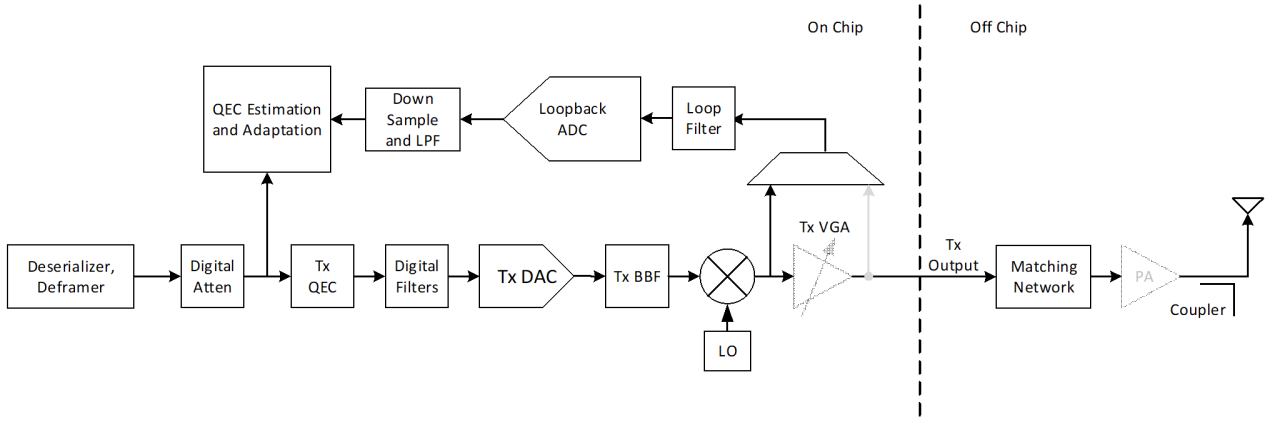


Figure 101. Tx QEC Initial Calibration Loop

Each Tx channel has a corresponding internal loopback channel that is used for calibration purposes. The loopback channel has a QEC Estimation and Adaptation engine which allows for correlation of the transmitted signal to the internally observed loopback signal and generates QEC coefficients. Because the Tx QEC initial calibration uses the internal loopback path, the Tx to ORx mapping state (i.e. which Tx channel is connected to a specific ORx channel) does not matter for the operation of this calibration.

During the Tx QEC initial calibration, the user will not be transmitting traffic data. Instead, the device will transmit tones at various offsets from LO in order to estimate the initial correction for the channel. Additionally, the Tx VGA will be powered down during this calibration to prevent large level signals entering the power amplifier stage. Powering down the Tx VGA offers additional isolation for conditions where turning power amplifier off might cause an impedance mismatch.

TX QEC TRACKING CALIBRATION

The Tx QEC initial calibration is required before enabling the Tx QEC tracking calibration. There are two observation tap-off points in the Tx path, one before the VGA and one after, as shown in figure below. Both tap-off points are connected to internal loopback path. This allows the tracking calibration to observe and track impairments that couple to the Tx output, including but not limited to the Tx VGA. The QEC Estimation and Adaptation engine is still responsible for evaluating cross-correlation between the transmitted signal and the observed signal and generating updated coefficients. Tx QEC tracking calibration does not require an external loopback path which utilizes ORx path.

ARM PROCESSOR AND DEVICE CALIBRATIONS

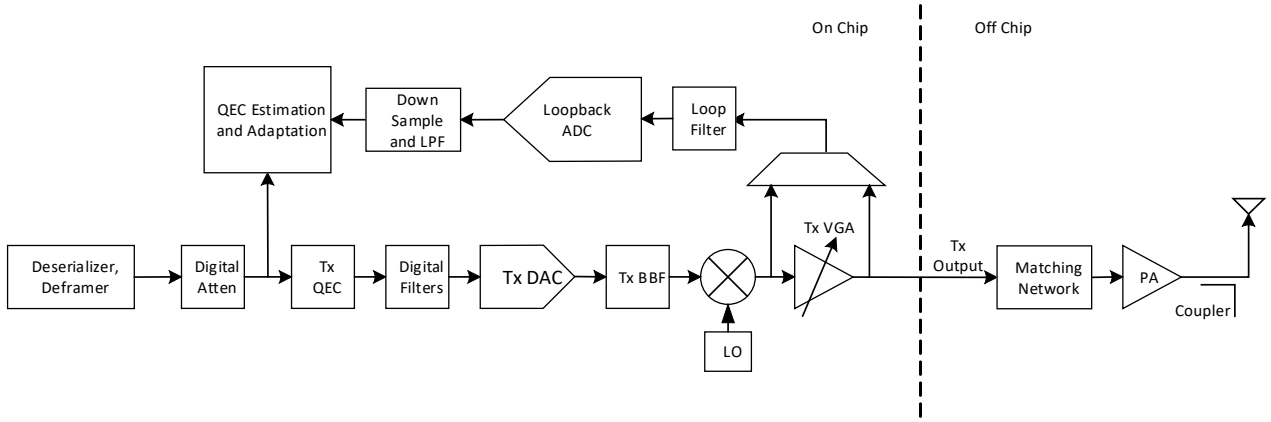


Figure 102. Tx QEC Tracking Calibration Loop with Two Tap-Off Points

As the device operating conditions change, the updates to the QEC is achieved through tracking calibration. Tx QEC tracking calibration makes use of actual transmit data and determines optimal coefficients. The tracking calibration is responsible for building up a correction table of QEC versus Tx attenuation.

Tx QEC Tracking Calibration Frequency Planning

For Tx QEC Tracking Calibration to operate as intended, the user has to consider the frequency planning below.

Let f_S be the Tx DAC sampling rate. The Tx LO frequency f_{LO} must satisfy

$$\left| f_{LO} - i\frac{f_S}{2m} \right| > 1 \text{ MHz} \quad (10)$$

for all integers i and for m = 2, 4, 6, 8, 10, 12. The table gives a calculated example for LO = 2.6GHz.

Table 54. Tx QEC Tracking Calibrations Frequency Planning

m	Value of i that minimizes $\left\| f_{LO} - i\frac{f_S}{2m} \right\|$ $i = \text{round}\left(2m\frac{f_{LO}}{f_S}\right)$	Nearest "blackout" frequency (in MHz) $i\frac{f_S}{2m}$	Distance from desired f_{LO} to nearest "blackout" frequency (in MHz). $\left\| f_{LO} - i\frac{f_S}{2m} \right\|$
2	4	2949.12	349.12
4	7	2580.48	19.52
6	11	2703.36	103.36
8	14	2580.48	19.52
10	18	2654.208	54.208
12	21	2580.48	19.52

For this example, because f_{LO} is always at least 1 MHz away from the "blackout" frequencies, the Tx LO can be placed at exactly 2.6 GHz. If f_{LO} is within 1MHz of the blackout frequencies, then the LO must be moved by at most +/− 1 MHz to avoid the nearest "blackout" frequency. The Palau configurator has this LO frequency checking built in and will output an error if the LO frequency does not meet the requirements.

RX QEC CALIBRATION

The ADRV903x uses a zero-IF architecture and quadrature errors in the received path show up as unwanted energy in the received spectrum. In order to prevent this unwanted energy appearing as an image in the received spectrum the ADRV903x implements a Quadrature Error Correction (QEC) algorithm in FW which is designed to perform blind, digital-only, frequency-dependent, and independent QEC.

The location of Rx QEC in the Rx path is shown in Figure 103. The goal of the algorithm is to estimate and track the quadrature error and apply QEC using a filter. The algorithm is designed to keep tracking the Rx input dynamics in both the frequency and time domain in the bandwidth of interest. The tracking is accomplished by continuously monitoring the input spectrum in search of adverse blockers and desired signals.

ARM PROCESSOR AND DEVICE CALIBRATIONS

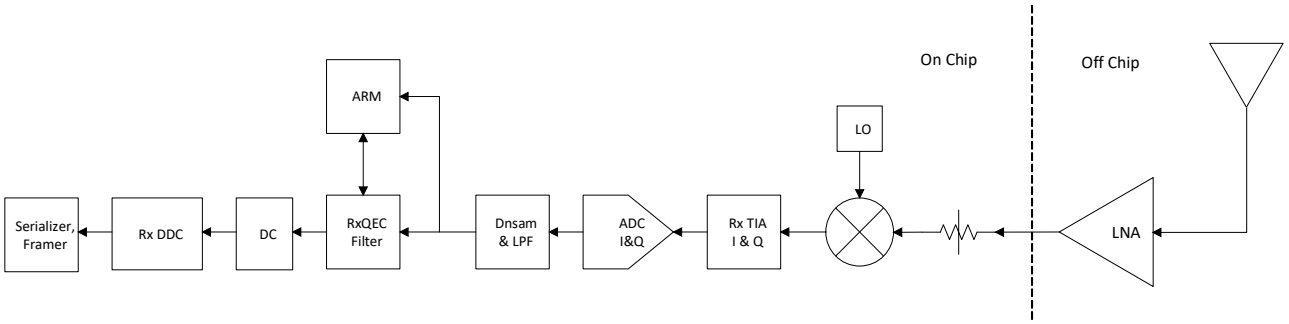


Figure 103. Rx QEC in Rx Path

QEC AND LOL CALIBRATION API FUNCTIONS

Table 55. List of QEC and LOL Calibration API Functions

API Method Name	Comments
adi_ADRV903x_InitCalsRun()	Runs the QEC and LOL init cals amongst others init cals.
adi_ADRV903x_InitCalsDetailedStatusGet()	Provides updates to represent the status of the QEC and LOL init cals amongst other init cals.
adi_ADRV903x_InitCalsDetailedStatusGet_v2()	Provides updates to represent the status of the QEC and LOL init cals amongst other init cals.
adi_ADRV903x_InitCalsWait()	Called after adi_ADRV903x_InitCalsRun to wait for init cals to finish.
adi_ADRV903x_InitCalsWait_v2()	Called after adi_ADRV903x_InitCalsRun to wait for init cals to finish.
adi_ADRV903x_InitCalsCheckCompleteGet()	Similar to adi_ADRV903x_InitCalsWait except it does not block the thread waiting for the enabled init cals to complete. This function returns the init cal status immediately allowing the application layer to do other work while the cals are running
adi_ADRV903x_InitCalsCheckCompleteGet_v2()	Similar to adi_ADRV903x_InitCalsWait except it does not block the thread waiting for the enabled init cals to complete. This function returns the init cal status immediately allowing the application layer to do other work while the cals are running
adi_ADRV903x_InitCalsAbort()	The init cals can take several seconds. This function is called when the BBIC needs to intercede and stop the current init cal sequence.
adi_ADRV903x_InitCalStatusGet()	Read back the status of the cal including metrics like error codes, percentage of data collected for current cal, the performance of the cal and the number of times the cal has run and updated the hardware.
adi_ADRV903x_TrackingCalsEnableSet()	Enables or disables QEC or LOL tracking cals amongst other tracking cals.
adi_ADRV903x_TrackingCalsEnableSet_v2()	Enables or disables QEC or LOL tracking cals amongst other tracking cals.
adi_ADRV903x_TrackingCalsEnableGet()	Gets the set of tracking cals that are enabled.
adi_ADRV903x_TrackingCalAllStateGet()	Gets the current state of all tracking cals
adi_ADRV903x_TrackingCalStatusGet()	Returns the status of the tracking cal.
adi_ADRV903x_TxLolReset()	Resets the Tx channel LOL tracking cal.
adi_ADRV903x_CalPvtStatusGet()	Returns detailed status information specific to the private cal.
adi_ADRV903x_CalSpecificStatusGet()	Returns status specific information calibration.

TX ANALOG LPF CALIBRATION

The baseband low pass filter is designed as a second order Butterworth filter between DAC's output and the input of mixer of I and Q path of TX channels as shown in Figure 104 below. Its bandwidth (3dB corner) is adjustable for different user cases. And the initial calibration is implemented to,

1. Set the 3dB corner of this filter precisely at the band edge associated with the user case.
2. Achieve the frequency response as a Butterworth filter with best in-band flatness.

ARM PROCESSOR AND DEVICE CALIBRATIONS

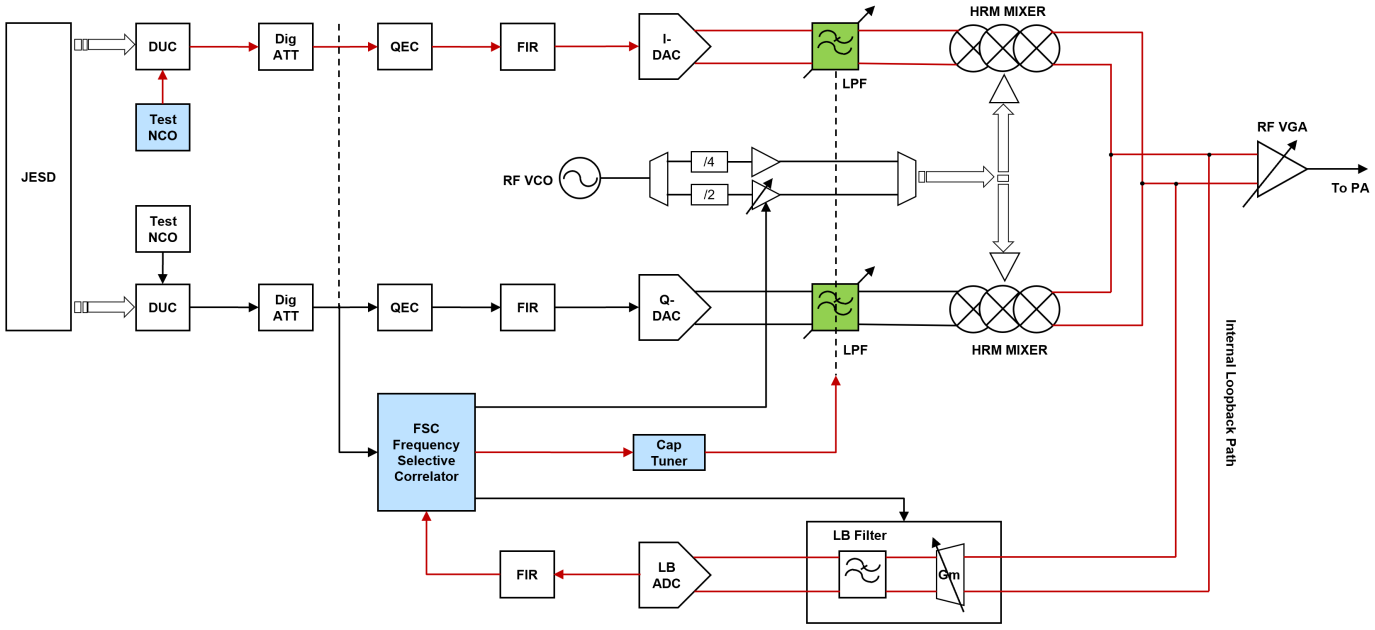


Figure 104. Tx Analog LPF Initial Calibration Loop (Red)

To calibrate the bandwidth of LPF, the calibration tones are generated separately and sequentially from the Test NCO that feeds into the TX signal path through DUC block, and looped back internally at the output of the Mixer to the loopback path. The power at each tone is measured by the frequency selective correctors (FSC). And the capacitors at the analog LPF are tuned simultaneously based on the ratio of the power measurements done by FSC to achieve the accurate 3dB corner frequency and in-band flatness.

LOOPBACK PATH DELAY INITIAL CALIBRATION

The loopback path delay calibration is used to measure the delay between the TX QEC actuator and the loopback RX input to the frequency selective correlator (FSC). And the result is used to configure the FSC correlator for accurate estimation of TX QEC and TX LOL corrections. So this calibration needs to run before TX QEC initial calibration and TX LOL initial calibration. Note if multiple calibrations are enabled in the CalMask, the ARM Firmware will correctly sequence the cals to make sure the Tx loopback cal is run before TxLOL Init or TxQEC Init.

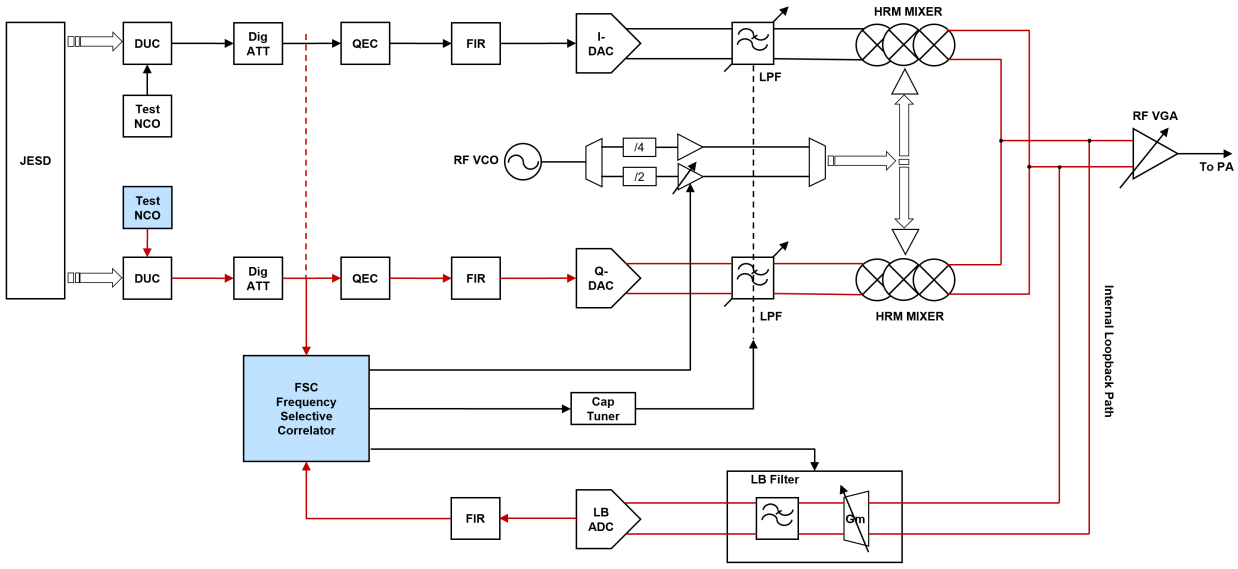


Figure 105. Loopback Path Delay Initial Calibration Loop (Red)

ARM PROCESSOR AND DEVICE CALIBRATIONS

As shown in the block diagram in Figure 105, the calibration tones are generated by the Tx Test NCO, and feed through the whole TX channel and looped back internally at the output of TX mixer to the loopback path. The FSC hardware measures the cross-correlation between the TX data and the Loopback RX ADC data to estimate the path delay.

Requirement:

- Loopback path delay initial calibration must run with all tracking calibration disabled.
- Loopback path delay initial calibration must run before TX QEC, TX LOL initial calibrations.
- Loopback path delay initial calibration must run after TX DAC, TX LPF, loopback filter and ADC initial calibrations.

RX DC OFFSET CALIBRATION

Overview

The Rx DC calibration comprises of two different corrections, the Rx RFDC calibration which attempts to correct the DC offset introduced by the analog front end and a Rx BBDC correction which attempts to null any remaining DC power by using a narrowband filter to heavily reject DC in the digital datapath. All Rx DC Offset calibrations are hardware based and have very little interaction with the embedded ARM processors.

The sense point for the Rx RFDC calibration is after the PFIR. The RFDC calibration attempts to correct all DC introduced by the analog front end by converting the offset observed just after the PFIR using a DAC and applying the inverse to the input of the TIA filter. The Rx BBDC offset block is directly after the sense point for the Rx RFDC (after the PFIR) and this will attempt to remove any remaining DC offset using a notch filter. It is recommended that both the RFDC and BBDC components of the calibration are enabled during operation for maximum performance.

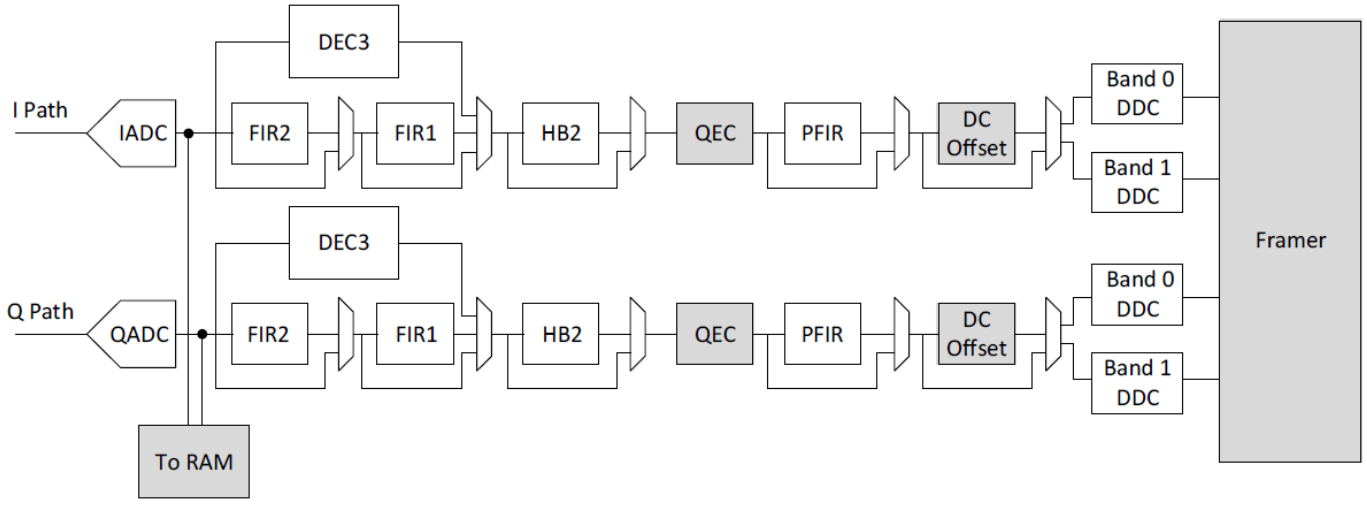


Figure 106. Rx Channel Datapath

Rx RFDC Offset Initialization Calibration

The Rx DC initialization calibration must be run for both the RFDC calibration and the BBDC calibration to be enabled. During the RFDC initialization calibration, the device sweeps the front-end gain from maximum to minimum and for each I/Q path, accumulators calculate and store the Rx DC correction in a table for each value of front-end gain. This initial calibration is the first step towards correcting the DC offset and gives the device a good starting point for applying subsequent corrections during the tracking calibration. The initialization calibration is not aimed at meeting data sheet performance however, both tracking calibrations must be enabled to guarantee maximum DC offset correction.

Maximum Time for Initial Calibration using an IQ rate of 245.76 MHz is approximately 1.5 milliseconds per channel.

Maximum Time for Initial Calibration using an IQ rate of 307.2 MHz is approximately 1.2 milliseconds per channel.

ARM PROCESSOR AND DEVICE CALIBRATIONS

Rx RFDC Offset Tracking Calibration

The Rx RFDC offset tracking calibration is responsible for updating the RFDC correction while in normal user data. The RFDC tracking algorithm uses normal Rx data to determine the DC offset during runtime and update the RFDC table with the latest correction for the current front-end gain which means that the DC offset correction will maintain performance across temperature and aging of the device among other factors. The correction required is applied to the input of the Rx TIA using a DAC to remove the DC offset calculated. This calibration can be enabled or disabled during runtime, but it must be enabled to obtain maximum DC rejection.

Rx BBDC Tracking Calibration

The Rx BBDC tracking calibration uses normal Rx data to collect information regarding the Rx DC offset and apply a correction. The Rx BBDC uses an accumulator in the baseband path to determine the amount of DC offset remaining following the correction made by the RFDC algorithm. Whatever DC value accumulates in this block will be subtracted out and will reduce the DC offset even further to meet the datasheet spec. This implementation results in a very steep notch filter around DC which removes heavily any DC but does not reject/rejects very little data not on DC. This calibration can be enabled or disabled during runtime, but it must be enabled to obtain maximum DC rejection.

As well as disabling the calibration, the user can modify the M-Shift value using the API in order to change the notch filer BW. Below is a table which shows the filter BW and convergence time in number of samples and for a sample rate of 307.2 MHz:

Table 56. M-Shift Configuration vs BW for 307.2 MHz IQ Rate

M-Shift	3 dB Filter BW (Hz)	Number of Samples for Convergence	Convergence Time for Rx IQ Rate of 307.2 MHz (ms)
9	1220	1625	0.01
10	610	3250	0.01
11	340	6500	0.02
12	160	13000	0.04
13	110	25000	0.08
14	40	50000	0.17
15	20	100000	0.33
16	10	200000	0.65
17	5	400000	1.3
18	3	600000	1.96
19	3	800000	2.61
20	2	1000000	3.26

The DC Offset Initialization/Tracking Calibrations are responsible for keeping the DC offset in the Rx data path in line with datasheet specifications. Initialization calibrations can be run during initialization of the device and can also be run during runtime. This also applies to tracking calibrations such as the Rx DC Tracking Calibration. During runtime, you may need to adjust the Rx BBDC Tracking Calibrations and the function calls associated with these are described in Table 57.

RX DC OFFSET CONFIGURATION API FUNCTIONS

Table 57. List of Rx DC Offset Configuration Related API Functions

API Method Name	Comments
adi_ADRV903x_DigDcOffsetEnableSet()	Sets the enable/disable of the BBDC Offset tracking calibration.
adi_ADRV903x_DigDcOffsetEnableGet()	Returns the enable/disable status of the Rx BBDC Offset mask representing the enable/disable status of each Rx BBDC Offset channel where a 0 is disabled and a 1 is enabled using the adi_ADRV903x_RxChannels_e mapping.
adi_ADRV903x_DigDcOffsetCfgSet()	Sets the digital DC offset configuration for the selected channel/s. The user can select the 1dB corner for the DC offset filter through adi_adrv903x_dcOffsetCfg_t structure. FW calculates the required mshift and multiplier values based on this corner frequency and configures the transceiver device.
adi_ADRV903x_DigDcOffsetCfgGet()	This function reads back the BBDC Offset configuration for the selected channel.

ANTENNA CALIBRATION

The ADRV903x supports a Tx Antenna Calibration Mode and a Rx Antenna Calibration Mode. An antenna calibration (AC) mode allows channels to be activated with special programming calibrations disabled. ACs can be triggered by pin control by using the TRX[A-H]_CTRL pins

or user selected GPIO pins. The pin mode to use is selected before runtime in the streamSettings structure of the profile JSON file. Setting EnableAntCal to true will enable the TRX[A-H]_CTRL pins to control the antenna cal. Setting EnableGpioAntCal to true will enable the GPIO pins to control the antenna cal. EnableAntCal and EnableGpioAntCal cannot both be set to true.

Configuring the TRX[A-H]_CTRL Pins for Tx and Rx Antenna Calibration

First, setup the AC to use TRX[A-H]_CTRL pins. In lines 88-89 of the profile JSON file set:

"EnableGpioAntCal": false,

"EnableAntCal": true,

Second, assign the TRX[A-H]_CTRL pins to the ACs. This can be done using the API function adi_ADRV903x_RadioCtrlTxRxEnCfgSet() or it can be set up in the profile JSON file which is described below.

This is how to assign the Rx channel or channels for Tx AC in the profile JSON file. On line 1797 in the profile JSON file there is an array called rxAltMapping and the next eight lines are its elements. Element 0 corresponds to the pin TRXA_CTRL and this continues to element 7 which corresponds to the pin TRXH_CTRL. The decimal value placed in this element is the Rx channel or Rx channels that will be enabled in Tx AC. Since there are up to 8 Rx channels that can be assigned to a TRX[A-H]_CTRL pin there is a bit wise mapping where bit[0] corresponds to Rx0 and bit[7] corresponds to Rx7. An example is shown below to setup Rx4 for Tx AC when TRXD_CTRL is triggered. Rx4 is bit[4] so in binary we get 00010000b and in decimal that is 16. Therefore, a value of 16 is placed in element 3.

```
"txEnMapping": [
  15,
  120,
  0,
  0,
  0,
  0,
  0,
  0
],
```

This is how to assign the Tx channel or channels for Rx AC On line 1777 in the profile JSON file there is an array called txAltMapping and the next eight lines are its elements. Element 0 corresponds to the pin TRXA_CTRL and this continues to element 7 which corresponds to the pin TRXH_CTRL. The decimal value placed in this element is the Tx channel or Tx channels that will be enabled in Rx AC. Since there are up to 8 Tx channels that can be assigned to a TRX[A-H]_CTRL pin there is a bit wise mapping where bit[0] corresponds to Tx0 and bit[7] corresponds to Tx7. An example is shown below to setup Tx3 for Rx AC when TRXB_CTRL is triggered. Tx3 is bit[3] so in binary we get 00001000b and in decimal that is 8. Therefore, a value of 8 is placed in element 1.

```
"rxEnMapping": [
  0,
  0,
  0,
  0,
  7,
  0,
  248,
  0
],
```

Configuring the GPIO Pins for Tx and Rx Antenna Calibration

First setup the AC to use GPIO pins. In lines 88-89 of the profile JSON file set:

"EnableGpioAntCal": true,

"EnableAntCal": false,

Second, assign the GPIO pins to the ACs using the profile JSON file. Lines 29-52 of the profile JSON file allow the definition of GPIO_0 to GPIO_23. A decimal 8 sets the GPIO to Rx AC and a decimal 9 sets the GPIO to Tx AC. An example where GPIO_0 and GPIO_1 are used for the Rx and Tx ACs respectively is shown below.

"Gpio00Selection": 8,

"Gpio01Selection": 9,

Finally, determine which Tx/s turn on during Rx AC or which Rx/s turn on during Tx AC use the API function adi_ADRV903x_RadioCtrlAntCalGpioChannelSet().

Tx Antenna Calibration Mode

In the simplest terms it works by taking a Rx channel or Rx channels, applying a fixed front end RF gain, disabling the Rx AGC, disabling internal calibrations and if required adjusting the Rx NCO. Once the Tx AC is complete then all the changed Rx settings are returned to their values before the calibration started.

The Tx AC can be setup to operate in two different ways.

- Single Pin Control. A single pin will both modify and enable/disable the Rx channel/s that is monitoring the Txs.
- Dual Pin Control. One pin, the pin assigned for the AC, will modify the Rx channel/s that is monitoring the Txs. Another pin will enable/disable the Rx channel that is monitoring the Txs. This pin is assigned in the same way pins are assigned Rx enable/disable for TDD pin control. See System Control section for further details.

Before enabling a Tx AC there are predefined parameters that need to be setup:

- The pin control type. For single pin control set DisableTxRx to false in the streamSettings structure of the profile JSON file. For dual pin control set DisableTxRx to true.
- The Rx NCO enable which is determined by EnableNco parameter in the streamSettings structure of the profile JSON file.
- The frequency the Rx NCO must change to, determined by the adi_ADRV903x_RadioCtrlAntCalConfigSet or the adi_ADRV903x_RadioCtrlAntCalConfigSet_v2 API function.
- The Rx gain which is determined by the adi_ADRV903x_RadioCtrlAntCalConfigSet or the adi_ADRV903x_RadioCtrlAntCalConfigSet_v2 API function.

An example of single pin and dual pin control Tx AC are given below

1. Single Pin Control. The pin assigned to the Tx AC goes high and this starts streams that make the required changes. Streams apply the fixed gain to the Rx, disable the AGC and cals, adjust the Rx NCO (if EnableNCO is set to True) and enable the Rx channel. Once the BBIC has completed the Tx AC, the same pin goes low to disable the Rx and restore all the Rx parameters back to their previous state.
2. Dual Pin Control. The pin assigned to the Tx AC goes high and this starts streams that make the required changes. Streams apply the fixed gain to the Rx, disable the AGC and cals, adjust the Rx NCO (if EnableNCO is set to True). A second pin is responsible for enabling the Rx channel. Once the BBIC has completed the Tx AC the second pin will trigger low to disable the Rx and the first pin will also trigger low and restore all the Rx parameters back to their previous state.

All gain/NCO settings restored at the falling edge of RX_ANT_CAL, see Figure 107 for more details. If AGC was enabled prior to AC mode entry, it is disabled on the rising edge of AC mode and re-enabled upon exit.

Rx Antenna Calibration Mode

In the simplest terms it works by taking a single Tx channel or Tx channels, applying a fixed Tx attenuation, disabling internal calibrations and if required adjusting the Tx NCO. There is also an option to change the Rx gain control from AGC to MGC. Once the Rx AC is complete the all the changed settings are returned to their values before the calibration started.

The Rx AC can be setup to operate in two different ways.

- Single Pin Control. A single pin will both modify and enable/disable the Tx channel that is monitoring the Rxs.

ARM PROCESSOR AND DEVICE CALIBRATIONS

- Dual Pin Control. One pin, the pin assigned for the AC, will modify the Tx channel that is monitoring the Rxs. Another pin will enable/disable the Tx channel that is monitoring the Rxs. This pin is assigned in the same way pins are assigned Tx enable/disable for TDD pin control. See System Control section for further details.

Before enabling a Rx AC there are predefined parameters that need to be setup:

- The pin control type. For single pin control set DisableTxRx to false in the streamSettings structure of the profile JSON file. For dual pin control set DisableTxRx to true.
- The Tx NCO enable which is determined by EnableNco parameter in the streamSettings structure of the profile JSON file.
- The frequency the Tx NCO must change to, determined by the adi_ADRV903x_RadioCtrlAntCalConfigSet or the adi_ADRV903x_RadioCtrlAntCalConfigSet_v2 API function.
- The Tx attenuation which is determined by the adi_ADRV903x_RadioCtrlAntCalConfigSet or the adi_ADRV903x_RadioCtrlAntCalConfigSet_v2 API function.
- The gain (rxGain) to set the Rx MGC to if the AGC freeze option is used (enableFreeze = 1). These are set in the adi_ADRV903x_RadioCtrlAntCalConfigSet_v2 API function. Note, enableFreeze only has an impact if the AGC is used. If using MGC mode then enableFreeze should be set to 0 to avoid any confusion.

An example of single pin and dual pin control Tx AC are given below

- Single Pin Control. The pin assigned to the Rx AC goes high and this starts streams that make the required changes. Streams apply the fixed Tx attenuation, disable the cals, adjust the Tx NCO (if EnableNCO is set to True) and cause the Tx channel to enable. The Rx gain control, if enableFreeze = 1, changes from AGC to MGC using the value of rxGain. Once the BBIC has completed the Rx AC, the same pin will trigger low to disable the Tx and restore all the Tx parameters back to their previous state and if enableFreeze = 1 when the Rx AC ran, the AGC is restored on the Rxs.
- Dual Pin Control The pin assigned to the Rx AC goes high and this starts streams that make the required changes. Streams apply the fixed Tx attenuation, disable the cals, adjust the Tx NCO (if EnableNCO is set to True). The Rx gain control, if enableFreeze = 1, changes from AGC to MGC using the value of rxGain. A second pin is responsible for enabling the Tx channel. Once the BBIC has completed the Rx AC the second pin will trigger low to disable the Tx and the first pin will also trigger low and restore all the Tx parameters back to their previous state and restore the Rx back to AGC if it was selected before the Rx AC ran.

Antenna Calibration Example

Consider a TRX[A-H]_CTRL pin example where:

- TRXA_CTRL enables all TX channels
- TRXB_CTRL enables TX3 for AC
- TRXC_CTRL enables all RX channels
- TRXD_CTRL enables RX4 for AC
- Single pin mode is used so DisableTxRx is set to false
- Rx MGC is not enabled in Rx AC so enableFreeze = 0

This figure below shows example timing of TRX[A-D]_CTRL pins for TDD radio control and AC modes.

- The term “Tx3_ANT_CAL” means Tx3 enabled for the purposes of Rx AC.
- The term “Rx4_ANT_CAL” means Rx4 enabled for the purposes of Tx AC.

ARM PROCESSOR AND DEVICE CALIBRATIONS

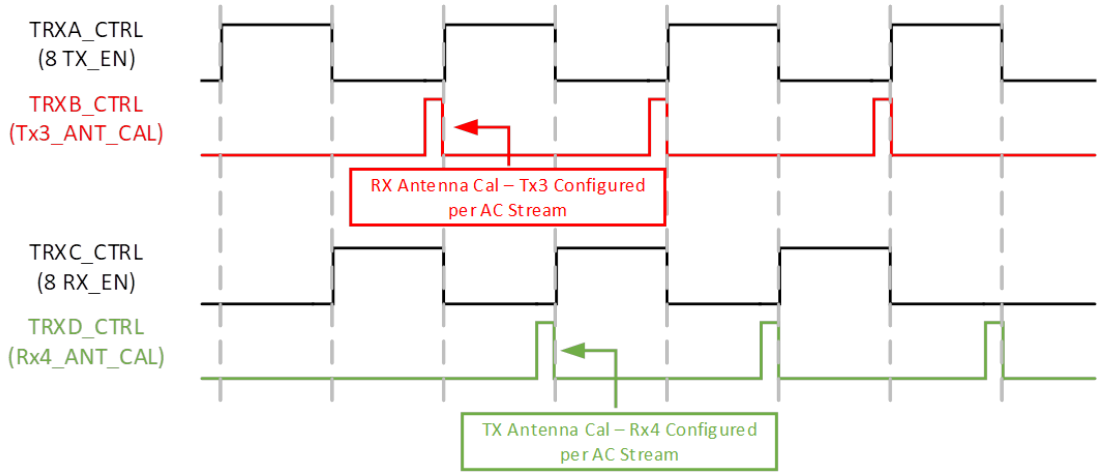


Figure 107. TDD Radio Control and Antenna Calibration

ANTENNA CALIBRATION API FUNCTIONS

Table 58. List of Antenna Calibration Related API Functions

API Method Name	Comments
adi_ADRV903x_RadioCtrlTxRxEnCfgSet()	Selects the Tx/Rx channels to be turned on during TRX[A-H]_CTRL antenna cal.
adi_ADRV903x_RadioCtrlTxRxEnCfgGet()	Reads Tx/Rx channel selection for TRX[A-H]_CTRL antenna cal.
adi_ADRV903x_RadioCtrlAntCalGpioChannelSet()	Selects the Tx/Rx channels to be turned on during GPIO antenna cal. GPIO pin can be selected via 'GpioXXSelection' fields in streamSettings structure in the profile JSON file as explained in the Configuring the GPIO Pins for Tx and Rx Antenna Calibration section.
adi_ADRV903x_RadioCtrlAntCalGpioChannelGet()	Reads Tx/Rx channel selection for GPIO antenna cal.
adi_ADRV903x_RadioCtrlAntCalConfigSet()	Sets the gain and NCO for the Tx/Rx channels in antenna cal.
adi_ADRV903x_RadioCtrlAntCalConfigGet()	Reads the gain and NCO for the Tx/Rx channels in antenna cal.
adi_ADRV903x_RadioCtrlAntCalConfigSet_v2()	Sets the gain, NCO and AGC for the Tx/Rx channels in antenna cal.
adi_ADRV903x_RadioCtrlAntCalErrorClear()	Clear error bits for selected channels.
adi_ADRV903x_RadioCtrlAntCalErrorGet()	Reads back the AC error status.

CALIBRATION GUIDELINES AFTER RF LO FREQUENCY CHANGES

The RF LO frequency changes are one of the two types:

Type 1: LO frequency change this is described by all three of the following criteria:

- The LO frequency change is less than 100 MHz.
- The LO frequency change does NOT step over the LO divider boundary or VCO transition boundary that is shown in the table below.
- The LO frequency change does NOT step over the TX upconverter bias settings switching boundary (at 1600 MHz and 3600 MHz).

Type 2: LO frequency change that is described by either of the following criteria:

- The LO frequency change is greater than 100 MHz.
- The LO frequency change does step over the LO divider boundary or VCO transition boundary.
- The LO frequency change does step over the TX upconverter bias settings switching boundary (at 1200 MHz, 1600 MHz, and 3600 MHz).

Table 59. VCO Transition Boundary and LO Divider Boundary

Frequency MHz		VCO Boundary MHz				Leaf DIV	Root DIV
650	887.5			10400	14200	4	4
887.5	1256.25	7100	10049			4	2
1256.25	1775			10050	14200	4	2
1775	2512.5	7100	10049			4	1

ARM PROCESSOR AND DEVICE CALIBRATIONS

Table 59. VCO Transition Boundary and LO Divider Boundary (Continued)

Frequency MHz		VCO Boundary MHz				Leaf DIV	Root DIV
2512.5	2800			10050	11200	4	1
2800	3550			11200	14200	2	2
3550	5025	7100	10049			2	1
5025	7100			10050	14200	2	1

The LO frequency change procedure for Type 1:

1. Disable all tracking calibrations
2. Disable all RF channels. If TRX_CTRL/ORX_CTRL pins cannot stop toggling, put the device into API command control mode via adi_ADRV903x_RadioCtrlCfgSet(), then call adi_ADRV903x_RxTxEnableSet() to disable all channels.
3. Call adi_ADRV903x_LoFrequencySet() to set the LO frequency.
4. Call adi_ADRV903x_TxLolReset with resetType set as ADI_ADRV903X_TX_LOL_SOFT_RESET to reset the channel estimation for Tx LOL calibration.
5. Call adi_ADRV903x_TxQecReset with resetType set as ADI_ADRV903X_TX_QEC_TRACKING_CHANNEL_RESET to reset the channel estimation for Tx QEC calibration
6. Enable relevant tracking calibrations.
7. Transition back to pin control mode, if necessary.

The LO frequency change procedure for Type 2:

1. Disable all tracking calibrations
2. Disable all RF channels. If TRX_CTRL/ORX_CTRL pins cannot stop toggling, put the device into API command control mode via adi_ADRV903x_RadioCtrlCfgSet(), then call adi_ADRV903x_RxTxEnableSet() to disable all channels.
3. Call adi_ADRV903x_LoFrequencySet() to set the LO frequency
4. Rerun the following initial calibrations with the order shown below separately or with the calMask set as 0x40E80.
    a. HRM Init Cal (ADI_ADRV903X_IC_HRM)
    b. Loopback filter Calibration (ADI_ADRV903X_IC_TXLB FILTER)
    c. Loopback path delay Cal (ADI_ADRV903X_IC_TXLB_PATH_DLY)
    d. Tx QEC Init Cal (ADI_ADRV903X_IC_TXQEC)
    e. Tx LOL Init Cal (ADI_ADRV903X_IC_TXLOL)
5. Enable relevant tracking calibrations.
6. Transition back to pin control mode, if necessary.

PA PROTECTION

The ADRV903x has two Power Amplifier (PA) protection blocks, a peak power block and an average power block. They can be used individually or in parallel. These blocks can monitor the signal at the output of the QEC correction block or at the input to the digital attenuation block. The alarms raised by these blocks can be made sticky, requiring user intervention to clear them once they trigger.

PA PROTECTION – PEAK POWER

The PA protection peak power block looks at individual I and Q samples, squares them and adds them together ($I^2 + Q^2$) before comparing them to a programmable threshold. If the ($I^2 + Q^2$) result is greater than the threshold, a peak is detected and the peak counter incremented. If enough peaks are found within a programmable, non-overlapping window of time, the peak power alarm is raised. The peak counter is reset at the end of each measurement period. If the peak power alarm has been set to be sticky, the user must take action to reset the alarm. If the alarm is non-sticky it will automatically be cleared at the end of the subsequent measurement period if that measurement period does not contain enough peaks to also trigger it.

Note that if the signal received by the ADRV903x is interpolated prior to the PA protection monitoring circuits, a single peak in the original signal may become multiple peaks in the interpolated signal. This is illustrated in the figures below.

Figure 108 shows the digital samples of a 50 MHz CW sampled at 491.52 Msps. The red line shows an arbitrary threshold set such that only one sample per positive cycle of the CW will exceed it.

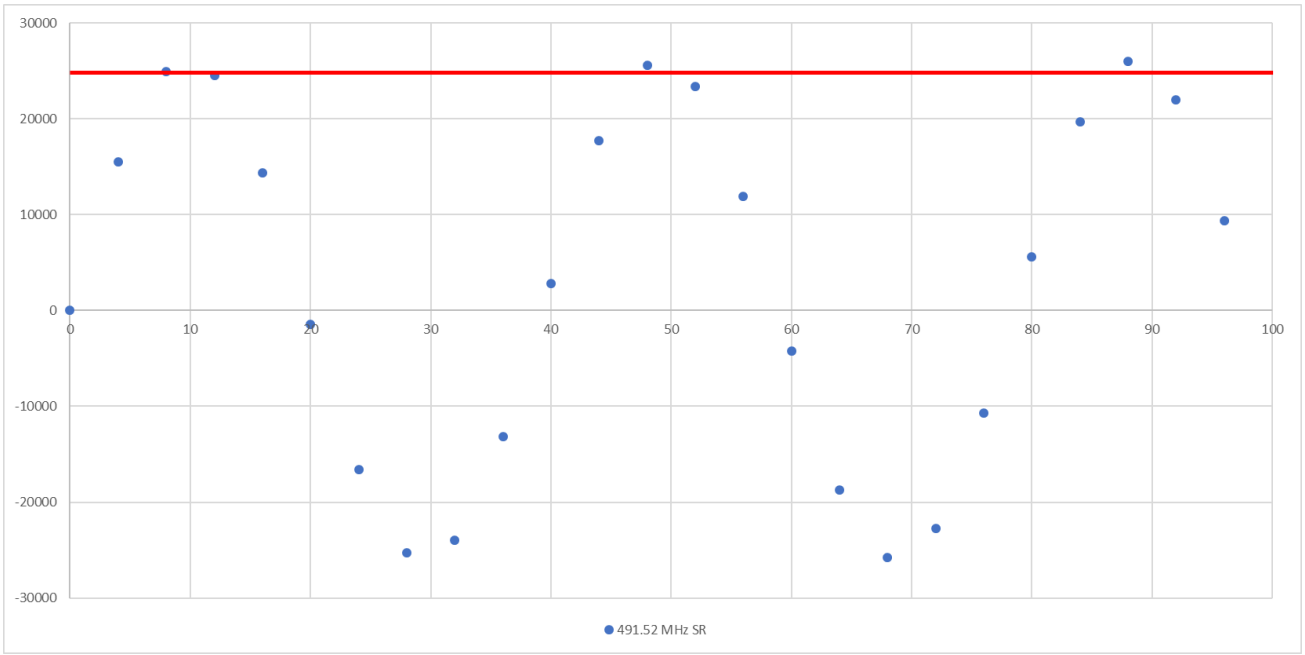


Figure 108. 50 MHz CW Sampled at 491.52 Msps

Figure 109 shows the same 50 MHz CW interpolated by 2, thus resampled at 983.04 Msps. Two samples per positive cycle of the CW now exceed the threshold.

PA PROTECTION

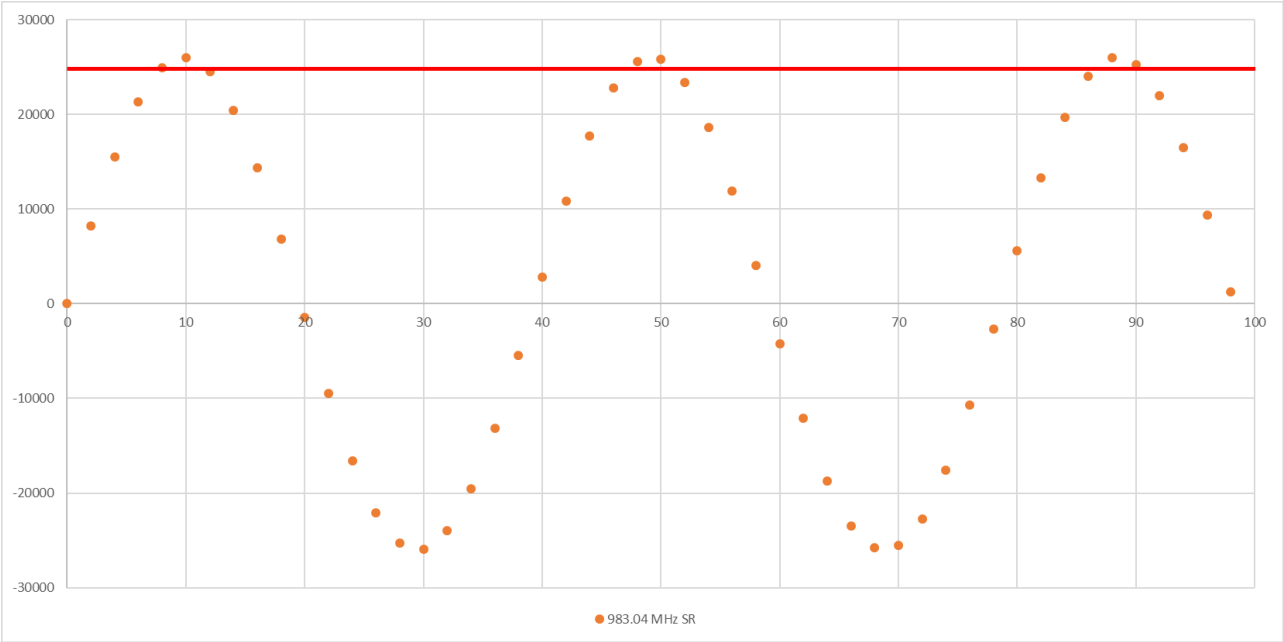


Figure 109. 50 MHz CW Sampled at 983.04 Msps

Figure 110 shows a further interpolation by two so the 50 MHz CW is now resampled to 1966.08 Msps. Four samples per positive cycle of the CW now exceed the threshold

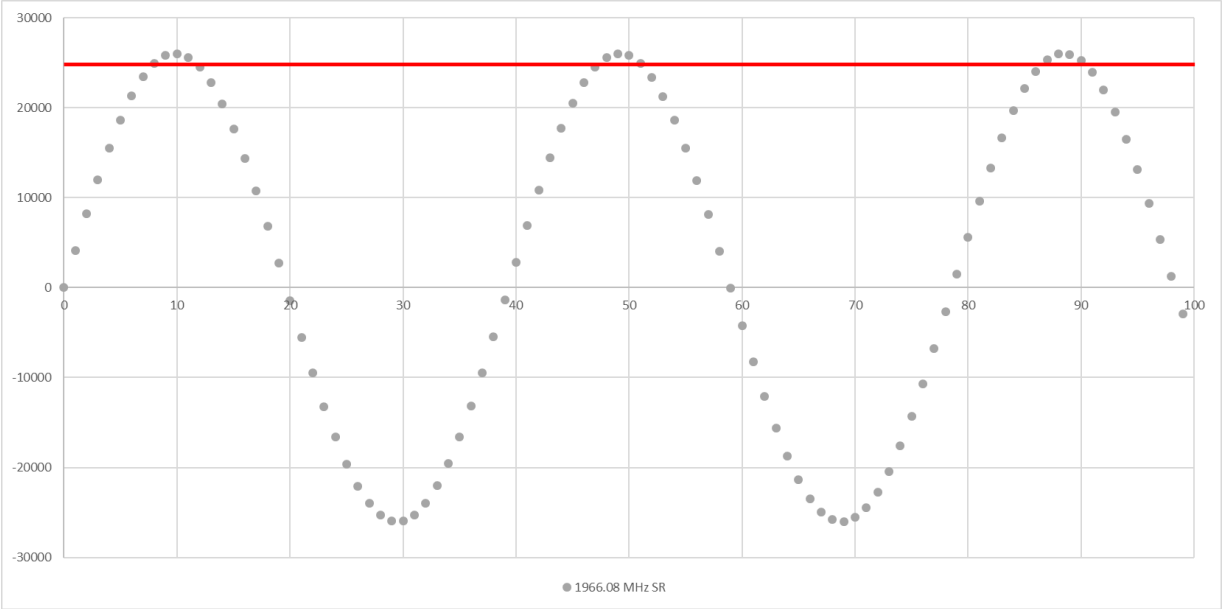


Figure 110. 50 MHz CW Sampled at 1966.08 Msps

To prevent a single sample in the original waveform from triggering the peak counter two or four times (depending on the interpolation factor) the PA peak detection circuit can be configured to only consider a peak 'detected' if it sees multiple samples exceeding the threshold in close proximity. Figure 111 describes the configuration options, and descriptions of each parameter are provided after the figure.

PA PROTECTION

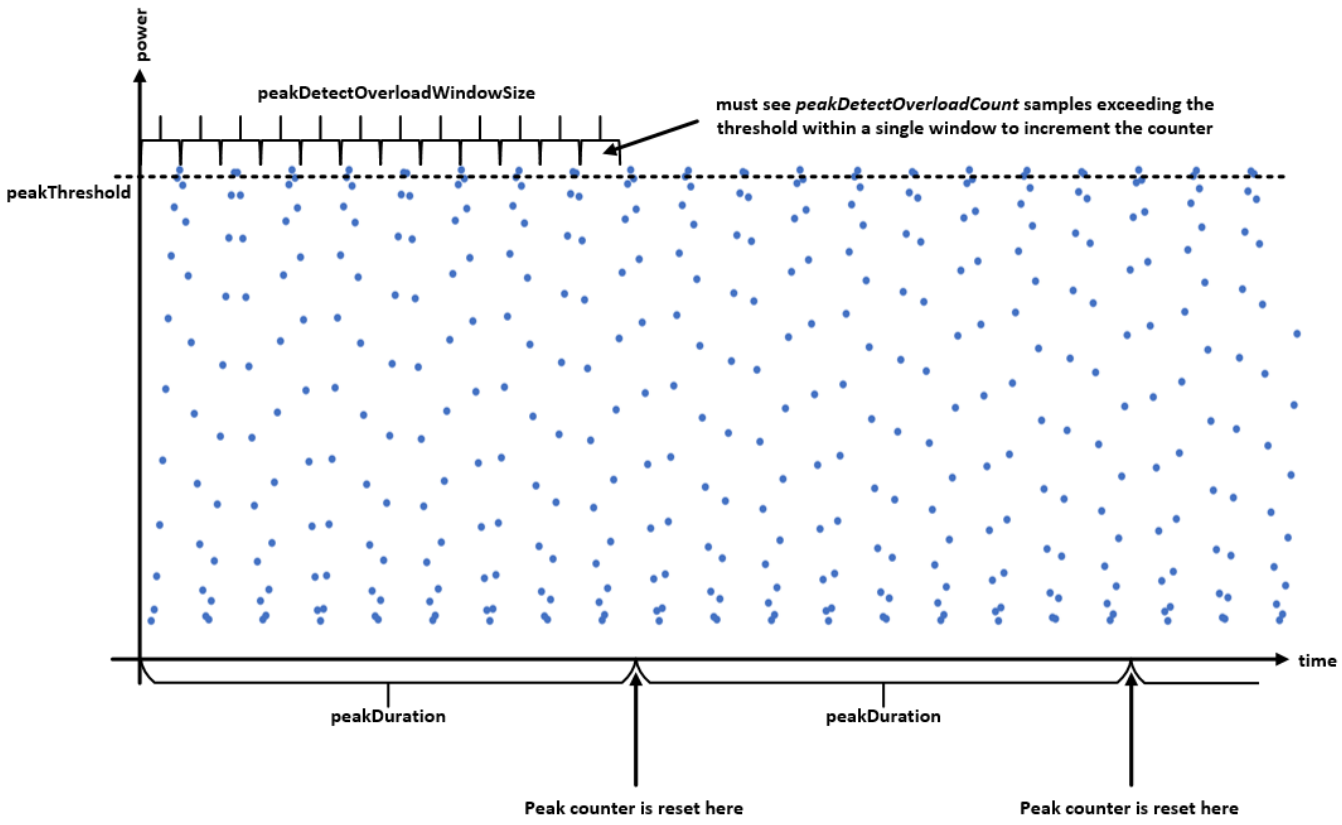


Figure 111. PA Peak Power Configuration Options

measDuration sets the width of the non-overlapping window which triggers the reset of the peak counter. Values range from 0 to 15. The value is translated into an IQ sample count using the formula 2^(measDuration+ 7). If this parameter is set to three the peak counter will be reset every 1024 IQ samples. If interpolation has been enabled prior to the monitoring circuit, this number must be divided by the interpolation factor to refer to the number of IQ samples sent across the JESD interface.

peakDetectOverloadWindowSize sets the width of the non-overlapping window which looks for multiple samples exceeding the threshold to trigger the peak counter. Values correspond directly to the number of IQ samples observed at the IQ rate of the protection circuit. Note, this value is automatically set to total *Input Interpolation Rate* * 2 to account for the interpolation effect (the user does not have to program this in the API).

peakDetectOverloadCount sets the number of samples that must exceed the threshold within a widow of peakDetectOverloadWindowSize before the peak counter is incremented. Values correspond directly to the number of peaks which must be found. This value is automatically set equal to total *Input Interpolation Rate* to account for the interpolation effect described above (the user does not have to program this in the API).

peakThreshold sets the threshold which must be exceeded for a peak to be found. Values range from 0 to 65535. When setting this threshold, the I and Q samples are considered to have a max value of 1.0. It is assumed that each (I,Q) sample should be within the unit circle, so an example of a max sample would be 1,0. Taking a signal with peaks at -3 dBFS, a peak (I,Q) amplitude value example would be (~0.707,0) and the power ($I^2 + Q^2$) of this sample would be ~0.5. Therefore, the corresponding threshold setting for this -3 dBFS peak sample would be ~0.5 * 2^16 = 32,846.

PA PROTECTION – AVERAGE POWER

The PA protection average power block looks at the average power of a programmable number of ($I^2 + Q^2$) results, where I and Q are samples, and compares it to a programmable threshold. If the average ($I^2 + Q^2$) result is greater than the threshold, the average power alarm is raised. The parameters used to configure the average power measurement are detailed below. If the average power alarm has been set to be sticky, the user must take action to reset the alarm after it is raised. If the alarm is non-sticky it will automatically be cleared at the end of the subsequent measurement period if that measurement period does not contain sufficient average power to also trigger it.

PA PROTECTION

measDuration sets the width of the non-overlapping window during which the average power is calculated. Values range from 0 to 15. The value is translated into an IQ sample count using the formula 2^(measDuration + 7). If this parameter is set to three the power will be averaged over 1024 IQ samples. If interpolation has been enabled prior to the monitoring circuit, this number must be divided by the interpolation factor to refer to the number of IQ samples sent across the JESD interface.

avgThreshold sets the threshold which the average power result must exceed to set the alarm. Values range from 0 to 65535. When setting this threshold, the I and Q samples are considered to have a max value of 1.0. It is assumed that the (I,Q) sample pairs should be within the unit circle, so a max sample pair would be 1,0. Taking a signal with an average power of −3 dBFS, an average (I,Q) amplitude value example would be (~0.707,0) and the power (I^2 + Q^2) of this sample would be ~0.5. Therefore, the corresponding threshold setting for this −3 dBFS average power would be ~0.5 * 2^16 = 32,846.

SLEW RATE DETECTION AND LIMITING

The slew rate detection block can monitor the signal at the output of the QEC correction block, or at the input to the digital attenuation block. The slew between two adjacent samples is calculated as $\frac{2^{16}}{2^{2Np}}\left((I_n - I_{n-1})^2 + (Q_n - Q_{n-1})^2\right)$, where Np is the converter resolution sent over the JESD. If the result of this calculation is above a programmable threshold, called srdOffset, a slew rate violation is detected. The srdOffset threshold can be converted to dbFS by applying $10 \cdot \log_{10}(\mathrm{srdOffset}/65535)$. The alarm raised by the slew rate detection block can be made sticky, thus requiring user intervention to clear it once it triggers, or it can be set to auto-clear after a programmable length of time during which no further slew violations are detected.

Upon a slew violation event, the last good digital sample at the output of the PFIR filter may be latched and held, thus blocking the offending samples from propagating through the datapath. This is enabled using the API function adi_ADRV903x_TxProtectionRampSampleHoldEnable-Set(). This will result in a DC level persisting in the datapath. Therefore, the slew alarm would usually be used to also trigger a ramp of the analog attenuator at the Tx output to ramp the output to max attenuation.

In automatic recovery mode a programmable timer is started when a slew event has been detected. Every time a new slew event is detected this timer is reset. If the timer expires, meaning that no new slew events have occurred for the preset time, the circuit automatically releases the latch at the output of the PFIR and triggers a ramp up of the Tx attenuator back to the level it was set to prior to the original slew event.

Slew statistics can be recorded. Either the maximum slew delta, or the number of slew events may be recorded. Both statistics may not be recorded simultaneously.

The value srdOffset sets the threshold for slew detection. The range of values which can be programmed is from 0 to 65535 and directly corresponds to the result of the $\frac{2^{16}}{2^{2Np}}\left((I_n - I_{n-1})^2 + (Q_n - Q_{n-1})^2\right)$ calculation. When comparing the value with the input signal for debug or to estimate the srdOffset value, the enabled interpolation blocks in the chain have to be taken into account as they will reduce the slew rate of the original signal as can be seen in Figure 112.

PA PROTECTION

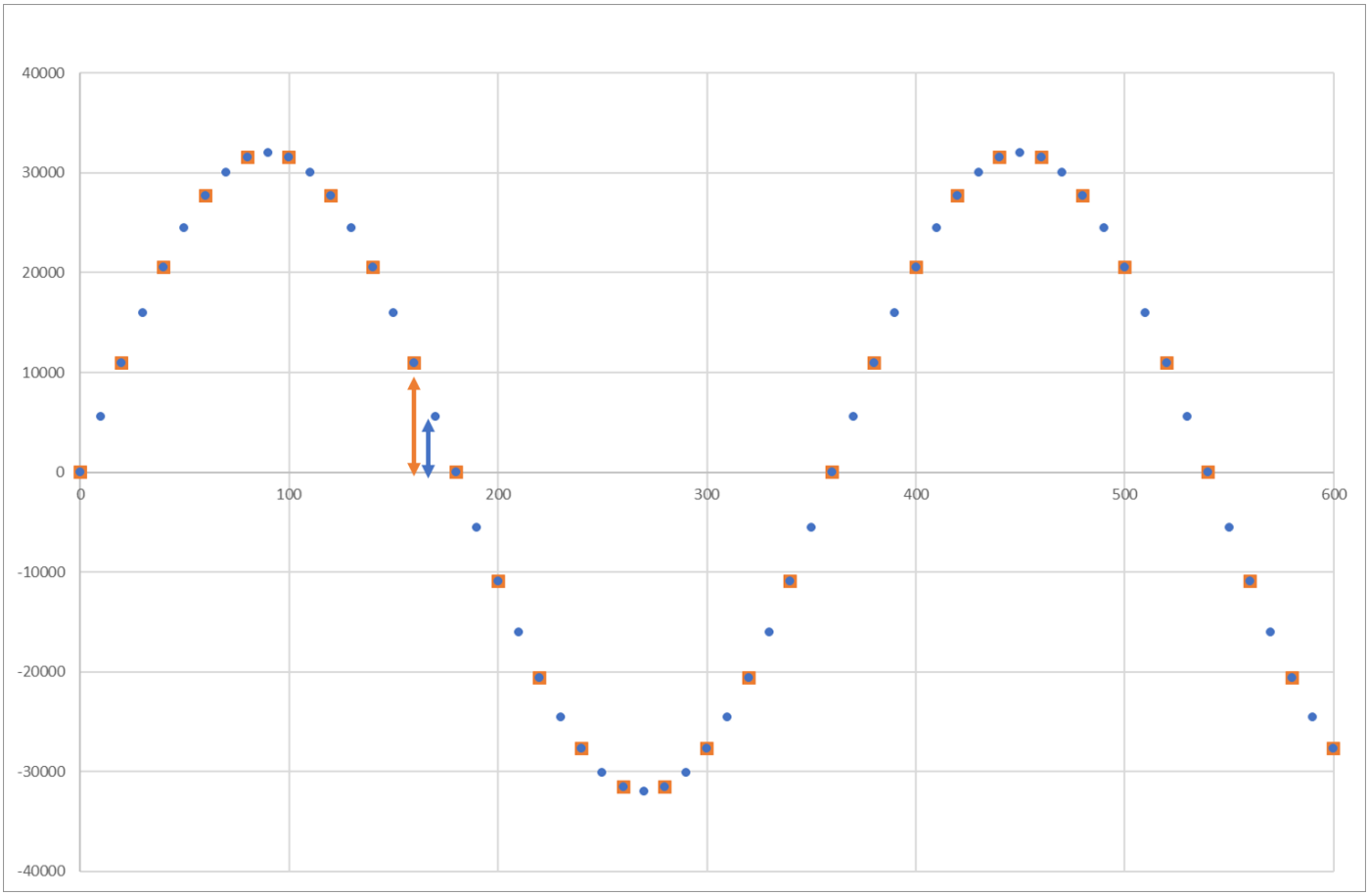


Figure 112. Slew Rate Comparison Between a Signal (Orange) and the Same Signal Interpolated by 2 (Blue)

The value autoRecoveryWaitTime sets the counter time for auto recovery mode. Values range from 0 to 15. The value is translated into an IQ sample count using the formula $2^{(autoRecoveryWaitTime + 6)}$. If this parameter is set to four the counter will be reset every 1024 IQ samples (at the IQ rate as seen by the SRD block).

It is possible to enable the Slew Rate Limiter interrupt so that it can be read from the main GP_INT Status Register. This can then be unmasked at the GPINT hardware pin to trigger a hardware interrupt as described in the General Purpose Interrupt section. If the analog rampdown has been enabled on occurrence of a slew rate detection, the GPINT pin will assert from Low to High and the rampdown will start typically within 100ns. When the slew rate detector is cleared, the GPINT pin will de-assert from High to Low and the Tx analog front-end will ramp back up to its previous value. If the user wants the Tx front-end to fully ramp up before the GPINT pin de-asserts it is possible to enable this by additionally unmasking the ARM0 Force GP Interrupt bit, or D9 in the GP Interrupt Word and ARM1 Force GP Interrupt bit, or D4 in the GP Interrupt Word. Note if you are splitting the Interrupt sources separately to GPINT0 and GPINT1, you should make sure you map ARM0 Force GP Interrupt to GPINT0 and ARM1 Force GP Interrupt to GPINT1.

PA PROTECTION API FUNCTIONS

Table 60. List of PA Protection API Functions

API Method Name	Comments
adi_ADRV903x_TxProtectionErrorGet()	Reads the status of events causing Tx power ramp down.
adi_ADRV903x_TxProtectionErrorClear()	Clears the error flags causing Tx power ramp down.
adi_ADRV903x_TxProtectionRampSampleHoldEnableSet()	Enables/disables the sample hold for Tx PA Protection ramp.
adi_ADRV903x_TxProtectionRampSampleHoldEnableGet()	Reads back sample hold ramp down configuration for selected Tx channel.
adi_ADRV903x_TxProtectionRampCfgSet()	Set the Tx Protection Ramp configuration. This function configures Tx ramp-down in case of Average Power Error/Peak Power Error/SRD Error/Pll Unlock/Dfrm Irq.
adi_ADRV903x_TxProtectionRampCfgGet()	Read the Tx Protection Ramp configuration.

PA PROTECTION

Table 60. List of PA Protection API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_TxPowerMonitorCfgSet()	Set the Tx power monitor configuration.
adi_ADRV903x_TxPowerMonitorCfgGet()	Get the Tx power monitor configuration.
adi_ADRV903x_TxPowerMonitorStatusGet()	Reads average and peak power, average to peak ratio (if enabled), average power at the time when last average power error occurred, peak power at the time when last peak power error occurred
adi_ADRV903x_TxSlewRateDetectorCfgSet()	Set the Tx Slew Rate Detector configuration.
adi_ADRV903x_TxSlewRateDetectorCfgGet()	Get the Tx Slew Rate Detector configuration.
adi_ADRV903x_TxSlewRateStatisticsRead()	Read the Slew Rate Detector statistics.

RX GAIN CONTROL AND GAIN COMPENSATION

The ADRV903x main receivers (Rx0-Rx7) have a front end with a variable gain setting. Each receiver has an independent gain setting that is controlled externally in Manual Gain Control (MGC) or by an internal state machine that makes gain change decisions based on input signal levels in Automatic Gain Control (AGC). AGC enables the fastest reduction of gain in response to the sudden onset of a strong interferer that may overload the receiver datapath. The AGC state machine controls the gain of the device based on inputs from a number of signal peak and power detectors and responds by stepping through gain indices from the programmed gain table. The resolution of gain changes possible depends on the gain table used. In MGC mode changes in gain are initiated by the Baseband Processor (BBIC) typically over the SPI interface. The gain control blocks are configured by the API data structures and several API functions allow for user interaction with the gain control mechanisms.

The AGC is highly flexible and has two configurations. During 3G/4G/5G operation Peak Detect Mode operation should be sufficient since the received signal is typically a multi-carrier signal. In this case a gain change should be performed only under large over range or under range conditions. If a blocker does appear, a 'fast attack' mode exists that should be able to reduce the gain at a fast rate.

To manage GSM blockers and radar pulses that have fast rise and fall times, a 'fast attack, fast recovery, peak detect only' mode is provided. This mode can recover receiver gain quickly in addition to the fast attack capability mentioned earlier.

This section contains the following functional descriptions:

- Receiver Datapath: Outlines the gain control and signal observation elements of the receiver chain. It also describes the concept of the receiver gain table.
- ORx Gain Control: Outlines the differences between Rx and ORx Gain Control.
- Gain Control Modes: Explains how to select between the gain control modes.
- Manual Gain Control (MGC): Describes how to operate the device in manual gain control mode.
- Automatic Gain Control (AGC): Describes the two principal modes of AGC operation, peak detect mode and peak/power detect mode.
- AGC Clock and Gain Block Timing: Describes the speed of the AGC clock and the various gain event and delay timers.
- Peak and Power Detectors: Outlines the operation and configuration of the gain control detectors in the device.
- API Programming: List out the API functions used for AGC and MGC.
- AGC Sample Script: Example python code.
- Gain Compensation, Floating Point Formatter, and Slicer: In AGC modes, it is recommended to implement gain compensation. Gain Compensation allows changes in Rx analog gain to appear transparent to the baseband processor by applying a compensating digital gain.

GLOSSARY OF IMPORTANT TERMS

Automatic Gain Control (AGC): The gain control mode where each receiver's internal AGC state machine controls the receiver gain settings.

Manual Gain Control (MGC): The gain control mode where the receiver gain is controlled by the BBIC.

Gain Attack: A reduction in receiver gain index due to an over range condition.

Gain Recovery: An increase in receiver gain index due to an under range condition.

Gain Compensation: The process of compensating for the analog attenuation in the device (prior to the ADC) with a corresponding amount of digital gain before the digital signal is sent to the user.

High Threshold: Each peak detector has multiple threshold levels. The highest level is referred to as the high threshold. High thresholds set an upper bound to the signal input level above which the gain can be decreased.

Low Threshold: A level in a peak detector which is lower than the high threshold. Some detectors have multiple low thresholds. Low thresholds set a lower bound to the signal input level below which the gain can be increased.

Threshold Overload: When a threshold is exceeded in a peak detector, this is referred to as an overload. An overload can occur for both high and low threshold.

Over range Condition: An over range condition exists when the AGC is required to reduce the gain. This can either be a peak condition, where a programmable number of individual overloads of a high threshold have occurred within a defined period of time, or a power condition, where the measured power exceeds a high-power threshold.

RX GAIN CONTROL AND GAIN COMPENSATION

Under range Condition: An under range condition exists when the AGC is required to increase the gain. This can either be a peak condition, where a lower threshold is not exceeded a programmable number of times within a defined period, or a power condition, where the measured power does not exceed a low power threshold.

RECEIVER DATAPATH

Figure 113 is a block diagram of the Rx datapath, highlighting the variable gain elements and the signal detection blocks. The amount of gain provided by the variable gain elements is determined by the gain index. The gain index is a row within a table called the gain table where each row within the table defines settings for all variable gain element blocks. If desired, the user can modify the gain table to suit their application. Manual control over each individual gain block is not supported.

The user can manually set the gain index (MGC) or allow control of the gain index to be handled by the receiver based on inputs from the signal detection blocks (AGC).

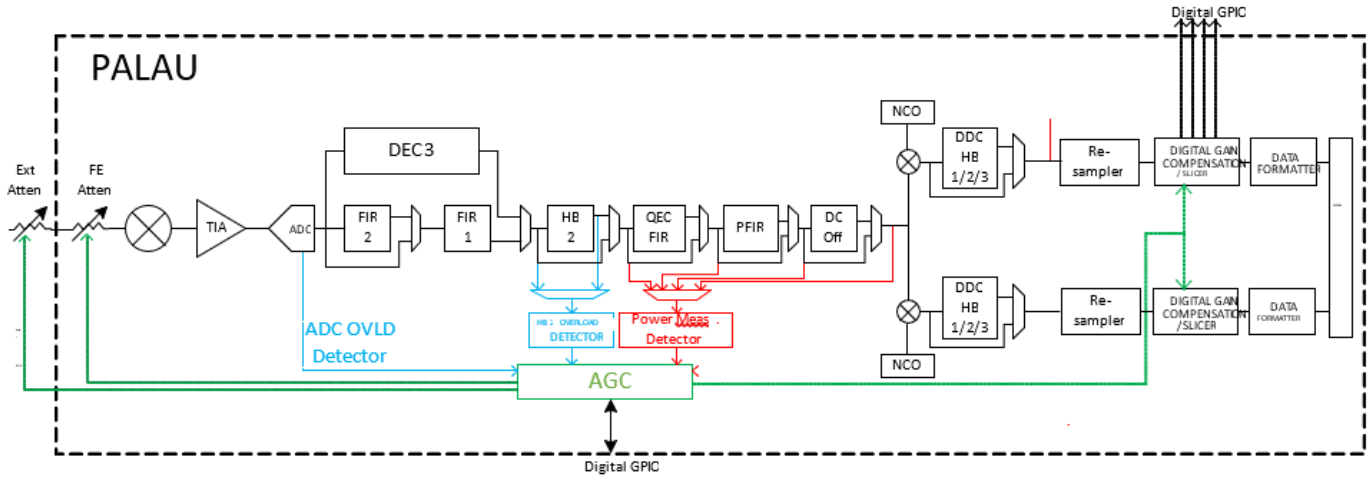


Figure 113. Rx Datapath and Gain Control Blocks

Variable Gain Elements

The “FE Atten” block is the front-end attenuator stage, a variable gain analog input stage that is used to attenuate the input signal in the presence of a strong interferer that may overload the ADC. The front-end attenuator uses an 8-bit control word. The amount of attenuation applied depends on the value set in the front-end attenuator column of the selected gain table index. The following equation provides an approximate relationship between the internal attenuator and the front-end attenuation value programmed in the gain table by the 8-bit value N:

$$FE\ Atten\ Attenuation\ (dB) = 20\log_{10}\left(\frac{256-N}{256}\right) \quad (11)$$

This formula implies that as the value of N increases so does the front-end attenuation step size. Changing N from 1 to 2 results in an attenuation step size of approximately 0.03 dBs but changing N from 254 to 255 results in an attenuation step size of approximately 6.02 dB which is orders of magnitude larger. Essentially at large values of N the front-end attenuation steps have poor resolution. Given that the front-end attenuation is coarse at times it needs to work with a digital attenuation to give consistent attenuation resolution or steps across all attenuation levels.

In the digital datapath, there is a “Digital Gain Compensation/Slicer” block. This block can be used to 1) Provide small amounts of gain or attenuation to ensure the most consistent gain differences between different gain indices in the gain table, 2) To provide digital gain compensation which compensates for a change in the FE Atten gain with an approximately equal and opposite digital gain or attenuation. The second use is highly recommended in AGC scenarios because an amplitude transient would be observed in the desired baseband signal if a strong interferer required a decrease in receiver gain. In the gain table, the digital control word is an 11-bit value where the MSB determines the sign and the remaining 10-bits indicate the magnitude of gain or attenuation. The bit mapping is shown in Table 61.

RX GAIN CONTROL AND GAIN COMPENSATION

Table 61. Signed Digital Gain/Attenuation Word Definition

Bit Position	Interpretation
D10	1 = Attenuation 0 = Gain
D9:D0	Magnitude of gain/attenuation word.

The range of the digital gain is 0 to 50 dB. The range of the digital attenuation is 0 to 18 dB. The resolution of the steps is 0.05 dB. As an example, a value of 14 would result in 0.7 dB gain, and a value of −14 would result in 0.7 dB of attenuation.

The “Ext Atten” block is an optional feature that uses the GPIO_ANA pins to control an external gain element. For single band receiver modes, up to two GPIO_ANA pins can be used per Rx to control an external element. Examples of the external element include LNA disable pins or input pins to a Digital Step Attenuator (DSA) block. The GPIO_ANA pins available for this feature are documented in the General Purpose Input/Output Configuration section. For a given gain index, if a bit is set in the Ext Atten field, the corresponding GPIO is asserted.

Gain Table Format

The gain table is user programmable, however, ADI provides two standard gain tables as a starting point in the SW deliverable package. The gain table named RxGainTable.csv is the default table for all LowBand and MidBand configurations where LO < 4.5 GHz. For HighBand applications with LO > 4.5 GHz, you should use the RxGainTable_HB.csv. Each row of the table provides a combination of front-end attenuator, external gain element (if used) and digital gain settings. Based on which row of this table selected, either by the user in MGC mode, or automatically by the device in AGC mode, the gain control block updates the variable gain elements depicted by the green arrows in Figure 113.

Table 62 shows a few entries sample gain table. This happens to also be an example of a gain compensated table where the front-end attenuator applies attenuation and the signed digital gain/attenuation column applies a complementary amount of gain. Please refer to the documentation earlier in this section to translate values in the table to gain values.

Table 62. Sample Rows from the Default Rx Gain Table

Gain Table Index	Front-End Attenuator[7:0]	External Gain Control[1:0]	Signed Digital Gain/Attenuation[10:0]	Phase Offset[15:0]
255	0	0	0	0
254	14	0	9	0
253	28	0	18	0
252	41	0	27	0
251	53	0	35	0
250	64	0	44	0

The gain table index is the reference for each combination of gain settings in the programmable gain table. It is possible to have different gain tables for each receiver, though typically the same one is used. The possible range of the gain table is 255 to 0, however typically only a subset of this range is used. The gain table must be assigned in order of decreasing gain, starting with the highest gain in the maximum gain index, such as 255, and the lowest gain in the minimum gain index.

Note that the phase offset has a bug where you need to sign flip to get the value you want.

The gain index is programmed during initialization in the command adi_ADRV903x_RxGainTableLoad(). This command is called during the adi_ADRV903x_PreMcsInit() command after the ARM profile binary has been loaded.

Peak Detectors and Power Measurement Detectors

The receiver datapath has multiple observation elements that can monitor the incoming signal level. These can be used in either MGC or AGC modes. Firstly, an ADC Overload Detector exists within ADC. This peak detector has the widest bandwidth of detection and can be used for monitoring out of band blockers. The detection bandwidth is the ADC sample rate divided by two. Note that this detector is located after the TIA filter so any attenuation in the TIA affects the observed signal.

The second peak detector is called the HB2 Overload Detector, so called because it monitors the data at the HB2 filter in the receiver chain. The detection bandwidth at the HB2 overload detector is always narrower than the ADC Overload Detector. The detection bandwidth depends on whether the input or output of the HB2 filter is used as the input to the AGC. The detection bandwidth is the sample rate at the detection point multiplied by the normalized bandwidth of the filter preceding the detection point.

RX GAIN CONTROL AND GAIN COMPENSATION

A power measurement detection block is also provided in the receiver chain, which takes the RMS power of the received signal over a configurable period. The power measurement location in the datapath is user configurable.

ORX GAIN CONTROL

The ORx0 and ORx1 channels are distinct from the Rx channels in the ADRV903x. The architecture of the ORx is based on an RF ADC design as opposed to the Zero-IF architecture used in the main receivers.

The attenuation control on the ORx channels are also distinct from that of the Rx. The ORx may only be controlled via MGC. The ORx signal level is generally well known and, therefore, an AGC control is not implemented for the ORx.

The attenuation range on the ORx channels are 16 dB in 1 dB steps. There is no gain table available for the ORx signal paths. Gain adjustments are made in the analog stage of the ORx.

Table 63. List of Rx Gain API Functions

API Method Name	Comments
adi_ADRV903x_RxGainTableWrite()	Programs the gain table settings for Rx channels.
adi_ADRV903x_RxGainTableRead()	Reads the gain table entries for Rx channels requested.
adi_ADRV903x_RxMinMaxGainIndexSet()	Updates the minimum and maximum gain indices for a requested Rx Channel
adi_ADRV903x_RxGainTableExtCtrlPinsSet()	Enable or disable the routing of the external control word from an Rx channel's gain table to its external analog GPIO pins.
adi_ADRV903x_OrxAttenGet()	Get the attenuation of an ORx channel.
adi_ADRV903x_RxGainCtrlModeSet()	Set the Rx gain control mode to MGC or AGC.
adi_ADRV903x_RxGainCtrlModeGet()	Get the Rx gain control mode with Rx channel index
adi_ADRV903x_RxTempGainCompSet()	This function sets the temperature gain compensation parameter for Rx channel only.
adi_ADRV903x_RxTempGainCompGet()	This function gets the temperature gain compensation parameter for Rx channel only. Only one channel can be retrieved per call.

GAIN CONTROL MODES

There are two gain control modes for main receivers. These are designated MGC and AGC. The user can select the mode that best suits their application. If MGC is selected, the gain index control is handled primarily over SPI command and there is some latency incurred due to this control scheme. If AGC is selected, the user must ensure that the AGC data structure is configured appropriately to avoid gain oscillation scenarios that could occur if high thresholds are not separated enough from low thresholds.

MANUAL GAIN CONTROL (MGC)

The gain control block applies the settings from the selected gain index in the gain table. In MGC mode, the BBIC is in control of the selecting the gain index. There are two options: 1) API functions; and 2) pin control. By default, if MGC is chosen the part is configured for API functions.

Table 64. List of MGC API Functions

API Method Name	Comments
adi_ADRV903x_RxGainSet()	Sets the Rx Channel Manual Gain Index. If the value passed in the gainIndex parameter is within range of the gain table minimum and maximum indices, the Rx channel gain index will be written to the transceiver
adi_ADRV903x_RxMgcGainGet()	Reads the Rx MGC Gain Index for the requested Rx channel.

AUTOMATIC GAIN CONTROL (AGC)

In AGC mode, a built-in state machine automatically controls the gain based on a user defined configuration. The AGC can be configured in one of two modes:

- Peak Detect mode, where only the peak detectors are used to make gain changes.
- Peak/Power Detect mode, where information from the peak detectors and power detector is used to make gain changes.

The agcPeakThreshGainControlMode parameter of the AGC configuration structure adi_ADRV903x_AgcCfg_t is used to select the AGC mode of operation as shown in Table 65.

Table 65. agcPeakThreshGainControlMode Settings

Bit Position	Interpretation
agcPeakThreshGainControlMode[d0]	1 = AGC in peak detect mode. Power Based AGC changes cannot occur in this mode. 0 = AGC in peak/power mode

Peak Detect Mode

In this mode, the peak detectors alone are used to inform the AGC to make gain changes. The ADC peak detector and HB2 detector both have a high threshold and a low threshold. The ADC peak detector's high threshold is fixed by hardware and is not user programmable. The other thresholds are user programmable and can be configured with API parameters adcOvldLowThresh, hb2HighTresh and hb2UnderRange-HighThresh. The limit for the number of times a threshold needs to be crossed for an over range or under range condition to be flagged is also user programmable.

The high thresholds are used as limits on the incoming signal level and are typically set based on the maximum input of the ADC. When an over range condition occurs, the AGC reduces the gain (gain attack). The low thresholds are used as lower limits on signal level. When the signal peaks are not exceeding the lower threshold, then this is indicative of a low power signal, and the AGC increases gain (gain recovery). This signal condition is termed an under range. The AGC stable state (where it does not adjust gain) occurs when neither an under range nor an over range condition is occurring (the signal peaks are less than or equal to the high thresholds and greater than the lower threshold levels).

Each over range/under range condition has its own attack and recovery gain step as shown in Table 66.

Table 66. Peak Detector Gain Steps

Overload/Under Range Condition	Gain Step
adcOvldHighThresh over range	Reduce gain by adcOvldGainStepAttack
adcOvldLowThresh under range	Increase gain by adcOvldGainStepRecovery
hb2HighThresh over range	Reduce gain by hb2GainStepAttack
hb2UnderRangeHighThresh under range	Increase gain by hb2GainStepHighRecovery

An over range condition occurs when a high threshold has been exceeded a configurable number of times within a configurable period. An under range condition occurs when a low threshold has not been exceeded a configurable number of times within the same configurable period. These counts make the AGC more or less sensitive to peaks in the input signal, ensuring that a single peak exceeding a threshold does not necessarily cause the AGC to react, allowing the user to trade off bit-error rate with signal to noise ratio. Table 67 outlines these configurable counts for each threshold's over range/under range conditions.

Table 67. Peak Detector Threshold Exceeded Counts

Overload/Under Range Condition	Threshold Exceeded Count
adcOvldHighTresh over range	adcOvldUpperThreshPeakExceededCnt
adcOvldLowThresh under range	adcOvldLowerThreshPeakExceededCnt
hb2HighThresh over range	hb2UpperThreshPeakExceededCnt
hb2UnderRangeHighThresh under range	hb2UnderRangeHighThreshExceededCnt

As an example of how the threshold exceeded counts work, consider the two thresholds associated with the HB2 detector in default operation. If hb2UpperThreshPeakExceededCnt was set to 1, then hb2HighThresh must be crossed one or more times within the configurable period for the signal to be considered over ranging the hb2HighThresh. Similarly, if hb2UnderRangeHighThreshExceededCnt was set to 1, then not crossing the hb2UnderRangeHighThresh even once during the configurable period would indicate an under ranging condition. However, if hb2UnderRangeHighThresh was crossed one or more times, then the signal will be not be under ranging (no gain recovery will be required in this case).

The configurable period for determining under and over range conditions can be set through the AGC's Gain Update Counter (GUC). This counter serves as a timing reference for making gain changes. The GUC and all AGC state machine logic, excluding the current gain index, are reset when the Rx is powered on. The GUC value, and therefore the time spacing between possible gain changes, is user programmable through the agcGainUpdateCounter parameter. The GUC value thus sets a periodic interval, in AGC clock cycles, with which gain changes can be made. Typically, this might be set to frame or sub-frame boundary periods. The agcSlowLoopSettlingDelay (configured in AGC clock cycles) contributes to this periodic interval in addition to the agcGainUpdateCounter, as shown in Figure 121.

RX GAIN CONTROL AND GAIN COMPENSATION

Once the GUC expires, all the peak threshold counters are reset through the agcSlowLoopSettlingDelay. The GUC is therefore a decision period, and peak detection is suspended during agcSlowLoopSettlingDelay. The overload thresholds and counters are set based on the number of overloads considered acceptable for the application within the duration of the GUC.

Figure 114 shows an example of the AGC response to a signal versus the ADC overload threshold levels. For ease of explanation, the ADC overload peak detector is considered in isolation. The green line is representative of the peaks of the signal. Key events in this example are numbered in the figure and described below:

1. During the first GUC period shown, the peaks of the signal are within the adcOvldHighThresh and adcOvldLowThresh, hence no gain changes are made at the first GUC boundary.
2. An interferer suddenly appears whose peaks now exceed adcOvldHighThresh.
3. Assuming a sufficient number of peaks have been detected to exceed the adcOvldUpperThreshPeakExceededCnt, the AGC decrements the gain index (reduces the gain) by adcOvldGainStepAttack.
4. Since one gain attack isn't sufficient to get the signal peaks below adcOvldHighThresh, the gain is decremented again.
5. With the peaks now between the two thresholds, the gain is stable and no gain changes are made at this GUC boundary.
6. The interferer is removed, and the peaks drop below the adcOvldLowThresh, resulting in an under range condition.
7. Although the signal is under-ranging, no gain changes are made at this GUC boundary assuming that enough peaks were detected to exceed the adcOvldLowerThreshPeakExceededCnt while the interferer was present earlier in the same period.
8. As the signal continues to under range, no peaks are detected above adcOvldLowThresh. Thus the number of peaks detected does not cross adcOvldLowerThreshPeakExceededCnt and the AGC steps by adcOvldGainStepRecovery.
9. The AGC recovers gain once more to bring the signal peaks back within adcOvldLowThresh and adcOvldHighThresh.

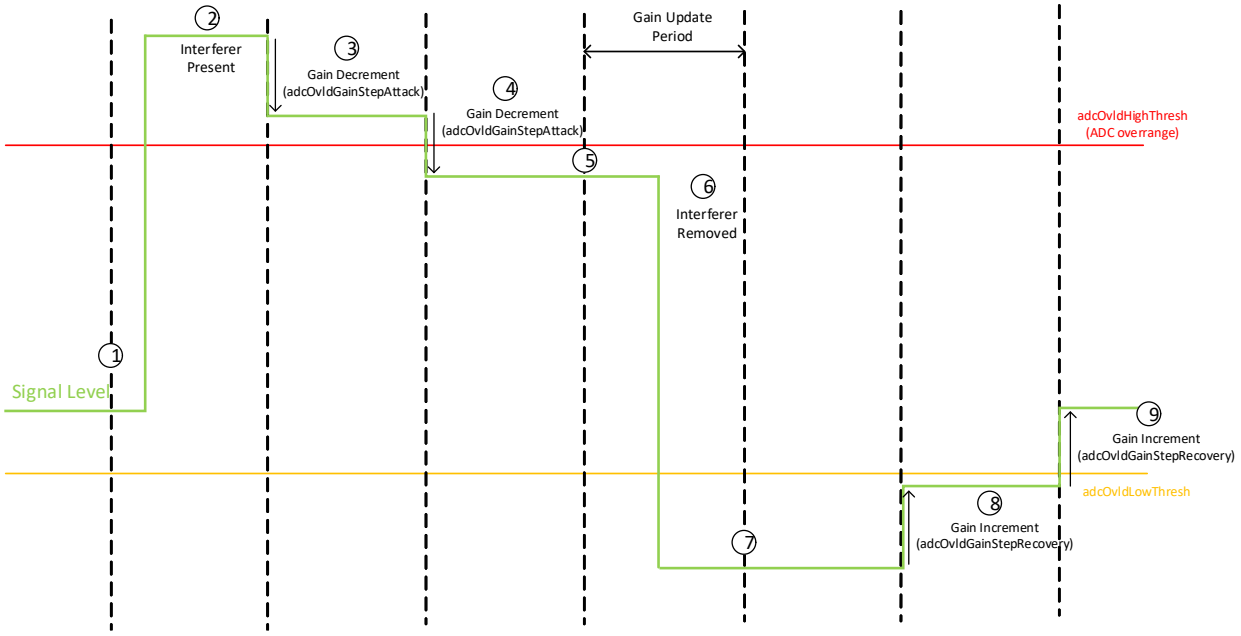


Figure 114. ADC Overload Thresholds and Gain Changes Associated with Underrange and Over range Conditions

Figure 115 shows the same scenario but from the viewpoint of the HB2 detector considered in isolation.

RX GAIN CONTROL AND GAIN COMPENSATION

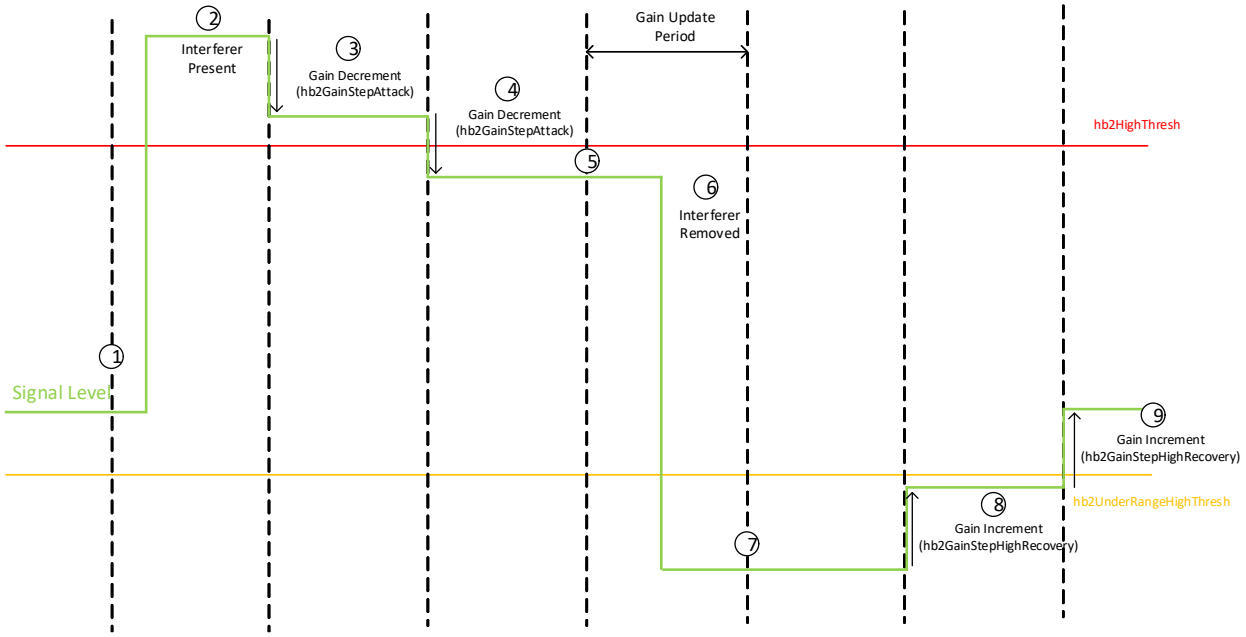


Figure 115. HB2 Thresholds and Gain Changes Associated with Under Range and Over Range Conditions

In both these cases the AGC increases or decreases gain at the expiry of the GUC period (also referred to as the GUC boundary). Alternatively, it is possible to enable a fast attack option whereby the AGC reduces gain immediately when an over range condition occurs, instead of waiting until the next expiry of the GUC. This fast attack mode can be configured independently for the ADC and HB2 overload detectors using the API parameter agcGainChangeIfThreshHigh. Values from 0−3 are valid as shown in Table 68.

Table 68. agcGainChangeIfThreshHigh Settings

agcChangeGainIfThreshHigh[1:0]	Gain Change following ADC Overload Over range	Gain Change following HB2 Over range
00	After expiry of agcGainUpdateCounter	After expiry of agcGainUpdateCounter
01	Immediately	After expiry of agcGainUpdateCounter
10	After expiry of agcGainUpdateCounter	Immediately
11	Immediately	Immediately

Figure 116 shows how the AGC reacts when the agcChangeGainIfThreshHigh[1] is set to 1 to enable fast attack for the ADC overload detector. In this case, once the interferer appears, the gain is updated immediately after the number of detected peaks above adcOvldHighThresh exceeds the adcOvldUpperThreshPeakExceededCnt. The AGC does not wait for the next expiry of the GUC to perform the gain attack. In this way, a number of gain changes can be made in quick succession, providing a much faster attack than the default operation where gain changes are made with the GUC's rhythm. Fast attack mode could be used in applications where it may be best to decrease gain immediately if the ADC is overloaded rather than wait for a suitable moment in the received signal's frame or sub-frame boundaries to change the gain.

Note that gain attack and recovery steps can be taken until the AGC reaches the minimum or maximum gain index programmed during the AGC's configuration. If a gain step has the potential of setting a gain index outside of the configured range, then the gain step is limited to keep the gain within this configured gain index range.

RX GAIN CONTROL AND GAIN COMPENSATION

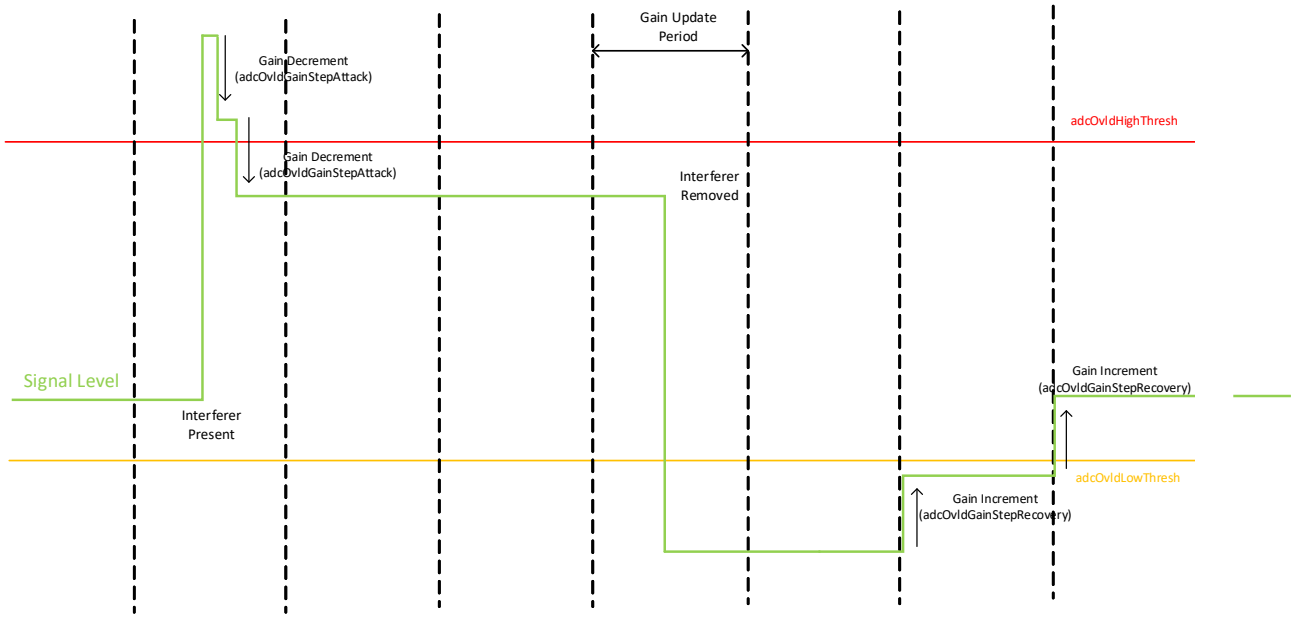


Figure 116. ADC Overload Gain Changes with Fast Attack Enabled

Figure 117 shows the same scenario but from the viewpoint of agcChangeGainIfThreshHigh being set for HB2.

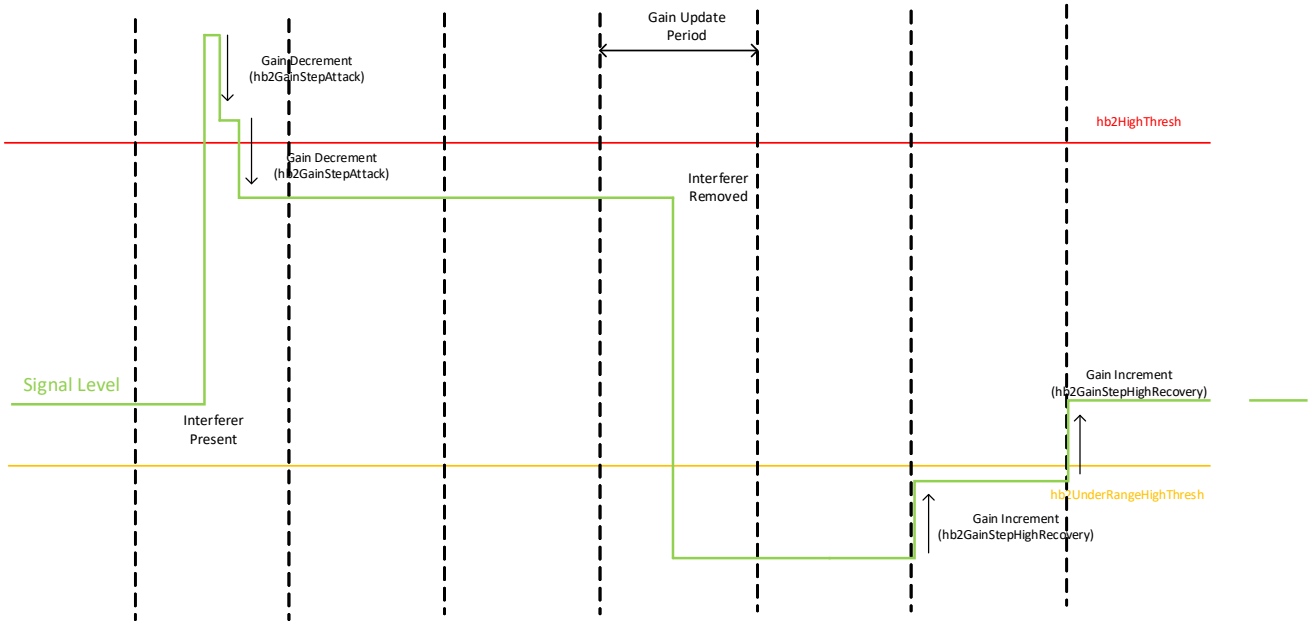


Figure 117. HB2 Gain Changes with Fast Attack Enabled

It is also possible to enable a fast recovery mode for the HB2 detector whereby gain recovery may occur with the expiry of multiple (shorter) gain update intervals. Fast recovery is enabled by setting agcEnableFastRecoveryLoop to 1, and this mode enables additional thresholds and corresponding gain update intervals for the HB2 detector. The peak exceeded counts and gain steps for each HB2 detector threshold are described later in the Half Band 2 Peak Detector section. Gain update intervals for the under range thresholds are tabulated below.

RX GAIN CONTROL AND GAIN COMPENSATION

Table 69. HB2 Peak Detector Recovery Steps and Intervals in Fast Recovery

Under Ranging Threshold	Gain Step	Gain Recovery following HB2 Under Range
hb2UnderRangeLowThresh	hb2GainStepLowRecovery	After expiry of hb2UnderRangeLowInterval (replaces GUC)
hb2UnderRangeMidThresh	hb2GainStepMidRecovery	After expiry of hb2UnderRangeMidInterval
hb2UnderRangeHighThresh	hb2GainStepHighRecovery	After expiry of hb2UnderRangeHighInterval

The multiple threshold and interval parameters allow for gain recovery such that as the input signal approaches the desired level, the size of the gain steps is reduced and the time interval between gain changes is increased. It should be noted that the time intervals associated with the three under range thresholds run in parallel, such that hb2UnderRangeMidInterval is a multiple of hb2UnderRangeLowInterval, and hb2UnderRangeHighInterval is a multiple of hb2UnderRangeMidInterval.

In fast recovery, the hb2UnderRangeLowInterval is used instead of agcGainUpdateCounter to set the gain update period. This interval also serves as the GUC for gain changes triggered by the ADC peak detector in default operation (excluding fast attack) when fast recovery is enabled for the HB2 detector. The recovery interval boundary chosen by the AGC to make its corresponding recovery gain step is prioritized between the thresholds as explained in the next section. Figure 118 illustrates fast recovery with an example scenario described below:

1. Once the signal level falls below hb2UnderRangeLowThresh, gain is incremented by hb2GainStepLowRecovery following the expiry of the gain update period hb2UnderRangeLowInterval.
2. After sufficient gain increases are implemented to bring the signal level above hb2UnderRangeLowThresh, the gain is incremented by hb2GainStepMidRecovery with the expiry of gain update periods now set by hb2GainStepMidRecovery. This applies while hb2UnderRangeMidThresh continues to under range.
3. Finally, when the signal level increases above hb2UnderRangeMidThresh, gain is incremented by hb2GainStepHighRecovery following the expiry of gain update periods as set by hb2UnderRangeHighInterval. This continues until the signal finally stabilizes in between hb2HighThresh and hb2UnderRangeHighThresh.

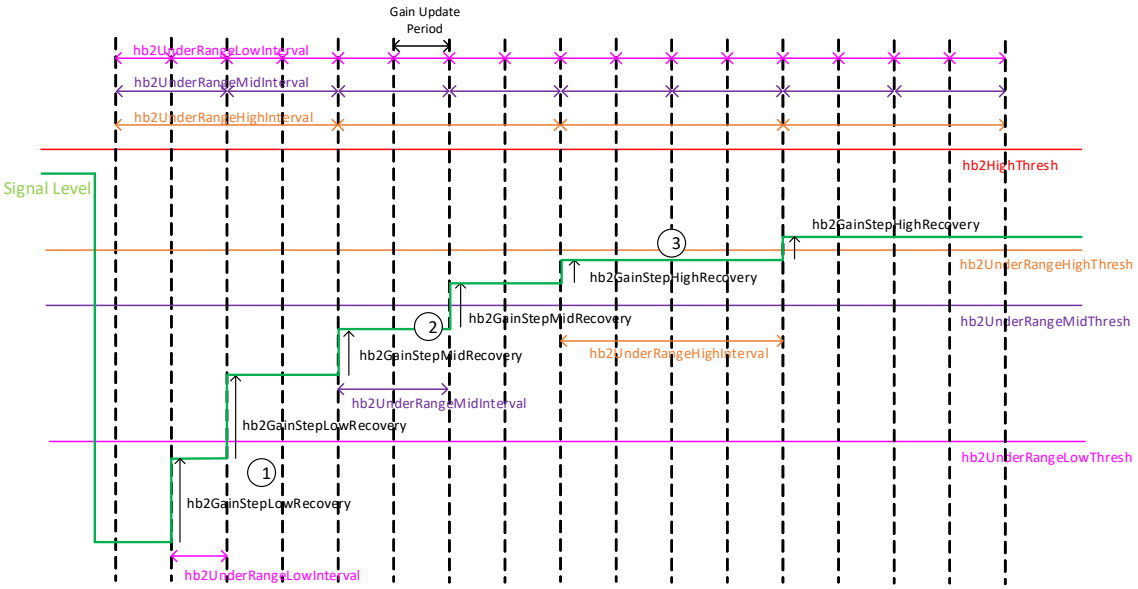


Figure 118. AGC Operation with HB2 Detector in Fast Recovery Mode

RX GAIN CONTROL AND GAIN COMPENSATION

Priorities and Overall Operation

The hb2HighThresh is typically set at approximately −3 dBFS to provide some overhead to change the gain before an interferer causes saturation of the ADC. It is highly recommended that the adcOvldLowThresh and the hb2UnderRangeHighThresh are set to equivalent values. Typical values are approximately −6 to −8 dBFS. This equivalence will be approximate, as these thresholds have unique threshold settings and will not be exactly equal. This section discusses the relevant priorities between the detectors and how the AGC will react when multiple threshold detectors have been exceeded. Table 70 shows the priorities between the detectors in default (slow attack) mode, when multiple over-ranges occur during the same GUC period.

Table 70. Priorities of Attack Gain Steps

adcOvldHighThresh Over Range	hb2HighThresh Over Range	Gain Change
No	No	No Gain Change
No	Yes	Gain Change by hb2GainStepAttack
Yes	No	Gain Change by adcOvldGainStepAttack
Yes	Yes	Gain Change by adcOvldGainStepAttack

If fast attack is enabled for either the ADC or the HB2 detectors or both, the peak exceeded counts corresponding to the high thresholds for both detectors (adcOvldUpperThreshPeakExceededCnt and hb2UpperThreshPeakExceededCnt) are reset when a fast attack is triggered by either detector, preventing any extraneous gain reduction from having both detectors triggering gain attacks in rapid succession. The peak exceeded counts corresponding to the low thresholds, however, are unaffected by the gain attacks.

For recovery, the number of thresholds is dependent on whether fast recovery is enabled or not. Considering the fast recovery scenario, the priority of the thresholds is:

1. hb2UnderRangeLowThresh Under Range Condition
2. hb2UnderRangeMidThresh Under Range Condition
3. hb2UnderRangeHighThresh Under Range Condition
4. adcOvldLowThresh Under Range Condition

Upon one under range condition, the AGC will change the gain by the corresponding gain step size of this condition. However, if multiple conditions occur simultaneously, then the AGC will prioritize based on the priorities indicated; that is, if hb2UnderRangeLowThresh is reporting an under range condition then the AGC will adjust the gain by hb2GainStepLowRecovery with two exceptions.

The adcOvldLowThresh has priority in terms of preventing recovery. If adcOvldLowThresh reports an over range condition (at least adcOvldLowerThreshPeakExceededCnt number of signal peaks have exceeded adcOvldLowThresh within a GUC period), then no further recovery is allowed. adcOvldLowThresh and hb2UnderRangeHighThresh should be configured to be as close to the same value of dBFS, but assuming some small difference between the thresholds, then as soon as adcOvldLowThresh is exceeded recovery will no longer occur. The reverse is not true, hb2UnderRangeHighThresh will not prevent the gain recovery towards the adcOvldLowThresh. Given the strong recommendation that adcOvldLowThresh and hb2UnderRangeHighThresh being set equally, then a condition whereby adcOvldLowThresh was at a lower dBFS level to hb2UnderRangeLowThresh or hb2UnderRangeMidThresh should not occur.

Another exception is if the recovery step size for a detector is set to zero. If so, the AGC makes the gain change of the highest priority detector with a non-zero recovery step. Figure 119 provides a flow diagram of the decisions of the AGC when recovering the gain in peak detect mode.

Power Detect Mode

In this mode, the power detector measurement is also used to control the gain of the Rx chain. It is possible to combine the peak detectors and power measurement detectors to create scenarios where peak detectors (and/or power measurement detectors) can inform gain reductions in over range scenarios and power measurement blocks inform gain increases in under range scenarios. Power measurement detectors by themselves cannot allow for “fast attack” style operation. The power detector will change gain solely at the expiry of the gain update counter whereas the peak detectors can be configured to make a gain change immediately when an over range is detected.

The power measurement block measures the RMS power of the receiver data at the measurement location. It can be configured to monitor the signal in locations shown in Figure 113. In power detect mode, the AGC compares the measured signal level to programmable thresholds which provide a second-order control loop, whereby gain can be changed by larger amounts when the signal level is further from the target level, and make smaller gain changes when the signal is closer to the target level.

RX GAIN CONTROL AND GAIN COMPENSATION

Figure 120 shows the operation of the AGC when using the power measurement detector. Considering the power measurement detector in isolation from the peak detectors, the AGC will not modify the gain when the signal level is between overRangeLowPowerThresh and underRangeHighPowerThresh. This range is the target range for the power measurement.

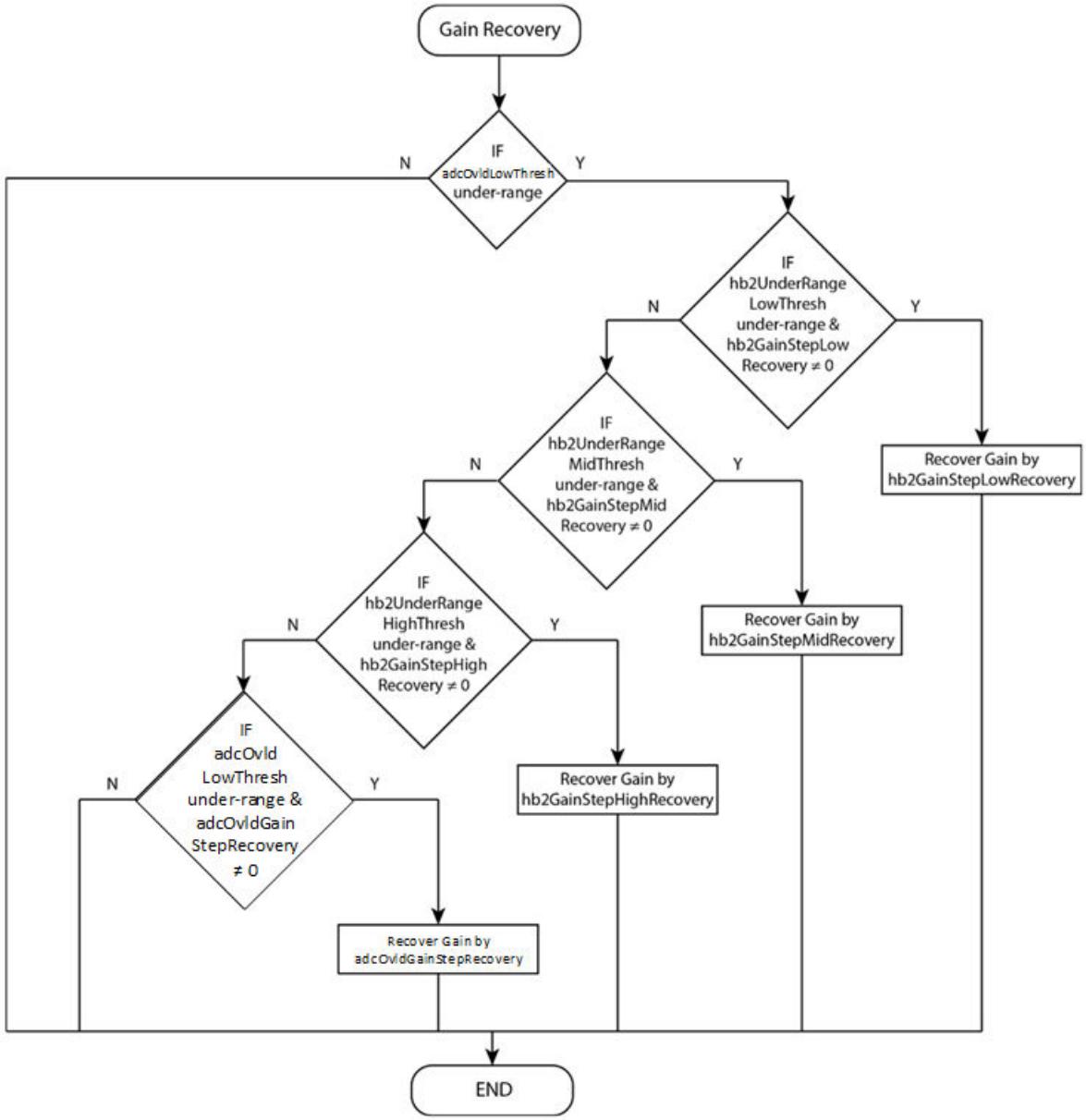


Figure 119. Flow Diagram for AGC Recovery in Peak Detect AGC Mode

When the signal level goes below underRangeLowPowerThresh, the AGC waits for the next gain update counter expiry and then increments the gain by underRangeLowPowerGainStepRecovery. When the signal level is greater than underRangeLowPowerThresh but below underRangeHighPowerThresh, the AGC will increment the gain by underRangeHighPowerGainStepRecovery. Likewise, when the signal level goes above overRangeHighPowerThresh, the AGC decreases the gain by overRangeHighPowerGainStepAttack, and when the signal level is between overRangeHighPowerThresh and overRangeLowPowerThresh, the AGC will decrease the gain by overRangeLowPowerGainStepAttack.

It is possible for the AGC to get conflicting requests from the power and peak detectors. An example would be an overloading out of band blocker visible to the ADC overload detector but significantly attenuated at the power measurement block. In this case the ADC overload detector requests a gain decrement, while the power measurement block requests a gain increment. The AGC has the following priority scheme in power detect mode:

RX GAIN CONTROL AND GAIN COMPENSATION

1. ADC Overload Detector High Threshold Over range
2. HB2 High Threshold Over range
3. ADC Overload Detector Low Threshold Over range
4. HB2 Low Threshold Over range
5. Power Measurement

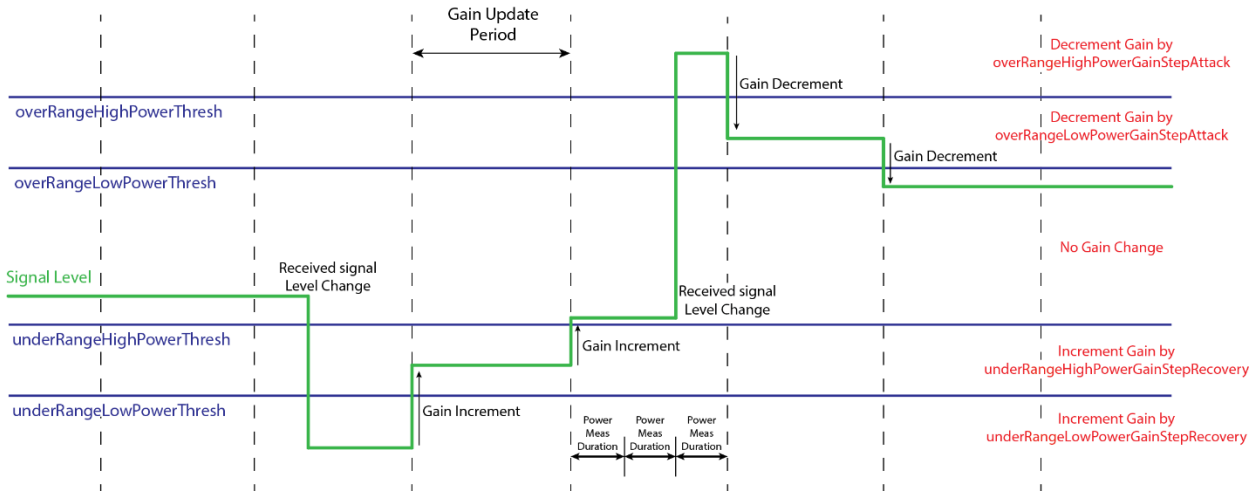


Figure 120. PMD Thresholds and Gain Changes for Under-range and Over-range Conditions

In this example, the gain would be decremented because the ADC overload high threshold over-range has a higher priority than the power measurement. Of note are the ADC overload and HB2 lower level overloads. In peak detect mode, the lower level thresholds for these detectors were used to indicate an under-range condition which caused the AGC to increase the gain. In power detect mode, these detectors are not used for gain recovery, but can be used to control gain recovery by setting the API parameter, agcLowThreshPreventGain. If this parameter is set, and if the signal level is exceeding a lower level threshold, the AGC is prevented from increasing the gain regardless of the power measurement.

This prevents an oscillation condition that could otherwise occur to a blocker visible to a peak detector but filtered before the power measurement block. In such a case, the peak detector could cause the AGC to decrease gain. It would do this until the blocker is no longer exceeding the defined threshold. At this point, the power measurement block could request an increase in gain and would do so until the detector's peak threshold was exceeded. This would decrease gain and so on. By using these lower level thresholds, the AGC is prevented from increasing gain as the signal level approaches an overload condition, providing a stable gain level for the Rx chain under such a condition.

AGC CLOCK AND GAIN BLOCK TIMING

The AGC clock is the clock which drives the AGC state machine. A number of the programmable counters used by the AGC are clocked at this rate. Its maximum frequency is 500 MHz. The clock is the greatest 2^N multiple of the IQ rate less than 500 MHz. For example, for an Rx profile with an IQ output rate of 245.76 MSPS, the AGC clock will be 491.52 MHz.

The AGC state machine's gain update period contains three factors: Gain Update Counter, followed by the Slow Loop Settling (SLS) delay, and a constant 5 AGC clock cycles delay. The total time between gain updates (gain update period) is thus a combination of the GUC (agcGainUpdateCounter or hb2UnderRangeLowInterval), the agcSlowLoopSettlingDelay, and an additional 5 AGC clock cycles. Note that the agcSlowLoopSettlingDelay set by the user through the API is doubled by hardware.

The following is an example of how to configure the agcGainUpdateCounter parameter. Consider a 100 µs gain update period and a 491.52 MHz AGC clock, a total of 49,152 AGC clocks cycles are required in the gain update period:

$$Gain\ Update\ Period\ (AGC\ Clocks) = 491.52\ MHz{*}100us = 49{,}152 \quad (12)$$

As noted, the full gain update period is the sum of the agcGainUpdateCounter, twice the agcSlowLoopSettlingDelay, and 5 additional AGC clock cycles. If the agcSlowLoopSettlingDelay is set to 4, the gain update counter must be set to 49,139.

$$Gain\ Update\ Period\ (AGC\ Clocks) = agcGainUpdateCounter + 2(agcSlowLoopSettingDelay) + 5 \quad (13)$$

RX GAIN CONTROL AND GAIN COMPENSATION

$$Gain\ Update\ Period\ (AGC\ Clocks) = 49{,}139 + 2(4) + 5 = 49{,}152$$

Figure 121 outlines the operation of the AGC state machine. The diagram outlines possible gain change scenarios rather than a practical example of AGC operation. The possible gain change scenarios are described below:

1. Signal under ranging / over ranging within GUC duration, before the SLS delay begins – Because slow loop settling (SLS) is typically several orders of magnitude smaller than gain update counter, this is the most common gain decrement scenario. The AGC state machine will proceed with any gain changes as deemed necessary at the GUC boundary (excluding fast attack and fast recovery modes).
2. Signal under ranging / over ranging detected during the SLS delay – This is a special case, but will rarely occur in applications per the reasoning in 1). As AGC's peak counters are reset during the SLS delay, any under ranging or over ranging will not be flagged over the SLS duration. The AGC is essentially blind to input signal conditions during the SLS delay. However, any signal under ranging or over ranging that persists onto the next GUC period will be addressed as in 1).

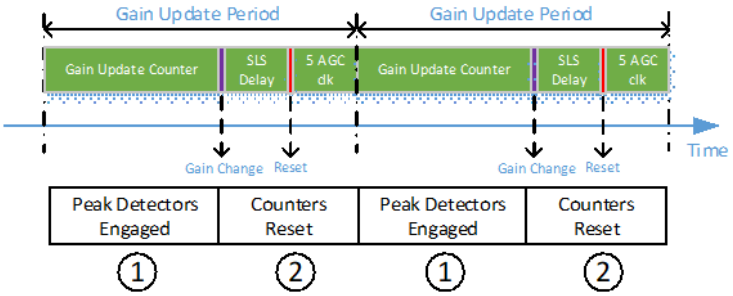


Figure 121. Gain Update Period

It should be noted that a gain attack may occur within the gain update counter period when fast attack is enabled. However, a gain recovery event may only occur at the expiry of the gain update counter. After a gain attack, the SLS delay is started, and no further gain attacks are possible while this counter is running. The SLS sets the minimum time between gain changes in fast attack mode.

When Rx is enabled, the AGC can be kept inactive for a programmable number of AGC clock cycles set by agcRxAttackDelay. This means the user can specify one delay for AGC reaction when entering Rx mode, and another for after a gain change occurs (agcSlowLoopSettlingDelay). Additionally, the API parameter agcResetOnRxon can be set to 1 to make the GUC restart whenever the Rx is re-enabled. Note that when agcResetOnRxon is 0 (default), the GUC pauses when the Rx is disabled, and resumes from its last value when Rx is re-enabled, effectively giving a time shift in the GUC's rhythm.

PEAK AND POWER DETECTORS

ADC Overload Detector

ADC overload detection is performed by an analog peak detector located at the ADC input after the TIA. This peak detector compares the incoming signal's peak level to the high threshold adcOvldHighThresh (fixed in hardware at −0.5 dBFS), and a programmable low threshold adcOvldLowThresh. In case of the high threshold, an over ranging (threshold exceeded) condition is flagged if the input signal exceeds adcOvldHighThresh at least a programmable number of adcOvldUpperThreshPeakExceededCnt times within the gain update counter (GUC) period. The GUC period is set through the agcGainUpdateCounter in AGC clock cycles. For the low threshold, under ranging is flagged if the input signal does not cross adcOvldLowThresh at least a programmable number of adcOvldLowerThreshPeakExceededCnt times within the GUC period. The two thresholds are visualized below:

RX GAIN CONTROL AND GAIN COMPENSATION

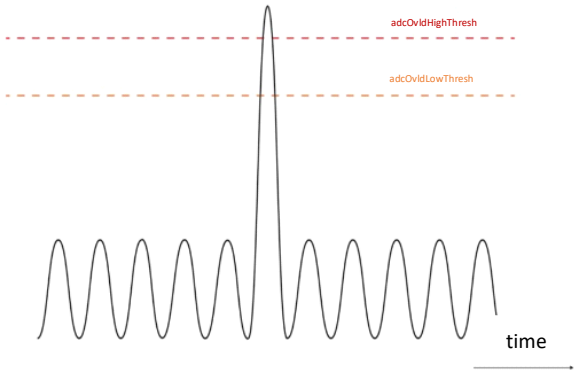


Figure 122. Analog Overload Detector Thresholds

The low threshold adcOvldLowThresh can be set to any value in the range of 0 to 8, where 8 represents the ADC's fullscale. The API value can be scaled in terms of the ADC fullscale (ADC dBFs) by using the following equation:

$$ADC\ Overload\ Low\ Threshold\ \left(in\ dBFS\right) = 20\ log\left(\frac{\text{adcOvldLowThresh}}{8}\right) \quad (14)$$

Since the analog peak detector is located after the TIA, the TIA can also attenuate the signal to a small degree, typically having a low attenuation over the passband and a steeper roll-off past the 3dB TIA corner frequency. This effectively determines the peak detector's operational bandwidth. The analog peak detector can also detect out-of-band (OOB) blockers although the blocker's level at the Rx input will get attenuated by the TIA before its level at the ADC input is compared against adcOvldHighThresh and adcOvldHighThresh.

The gain steps and peak threshold counts corresponding to each threshold were tabulated in Table 66 and Table 67.

Table 71. AgcPeak struct parameters for the Analog Peak Detector

AgcPeak Parameter	Description
adcOvldHighThresh	Fixed by hardware (−0.5 dBFS).
adcOvldUpperThreshPeakExceededCnt	Number of peak events needed where input signal exceeds adcOvldHighThresh to trigger an over ranging condition. Valid range is 2 to 255.
adcOvldLowThresh	Analog peak detector's programmable low threshold. Valid range is 0 to 8, where 8 means full scale of ADC.
adcOvldLowerThreshPeakExceededCnt	Number of peak events needed where input signal exceeds adcOvldLowThresh in order to prevent an under ranging condition. Valid range is 2 to 255.

Half-Band 2 Peak Detector

The HB2 peak detector samples data at the input or output (hb2OverloadSignalSelection) of Half-Band filter 2. The detector compares the signal level to programmable thresholds. It monitors the signal level by observing individual $I^2 + Q^2$ samples (or peak I and peak Q if hb2OverloadPowerMode = 0) over a period and compares these samples to the thresholds. If enough samples exceed a threshold in the period, then the threshold is noted as exceeded by the detector.

The HB2 detector does not operate on individual samples but on a programmable batch size of samples. The hb2OverloadDurationCnt defines the size of the small batch of samples to analyze. The hb2OverloadThreshCnt sets the necessary number of samples exceeding an HB2 threshold level within the batch to increment the peak count. This means that every peak that exceeds the threshold does not necessarily increment the peak detection count – this decreases sensitivity to a single errant peak which could be caused by the statistics of the signal itself. hb2OverloadThreshCnt and hb2OverloadDurationCnt apply for all HB2 thresholds enabled for the selected recovery mode (default mode and fast recovery). If a hb2OverloadDurationCnt is interrupted by the expiry of the GUC, the incomplete hb2OverloadDurationCnt batch does not factor into the gain change decision – it is effectively truncated from the decision criteria.

In the case of the HB2 high threshold hb2HighThresh, once enough batches containing a sufficient count (hb2UpperThreshPeakExceededCnt) of overloading samples are detected an over range condition is observed. For the HB2 low thresholds (hb2UnderRangeHighThresh in default mode and additionally hb2UnderRangeMidThresh and hb2UnderRangeLowThresh in fast recovery), if there are not a greater number of batches containing a sufficient count (hb2UnderRangeHighThreshExceededCnt, hb2UnderRangeHighThreshExceededCnt, and hb2UnderRangeHighThreshExceededCnt respectively) of overloading samples, then an under range condition is observed.

RX GAIN CONTROL AND GAIN COMPENSATION

Figure 123 shows the two-level approach. It shows the gain update counter period, with the time being broken into subsets of time based on the setting of hb2OverloadDurationCnt. Each of these periods of time is considered separately, and hb2OverladThreshCnt individual samples must exceed the threshold within hb2OverloadDurationCnt for an overload to be declared. These individual samples greater than the threshold are shown in red, while those less than the threshold are shown in green. Two examples are shown, one where the number of samples exceeding the threshold was sufficient for the HB2 peak detector to declare an overload (this time period is shown as red in the gain update counter timeline), and a second example where the number of samples exceeding the threshold was not sufficient to declare an overload (this time period is shown as green in the gain update counter timeline).

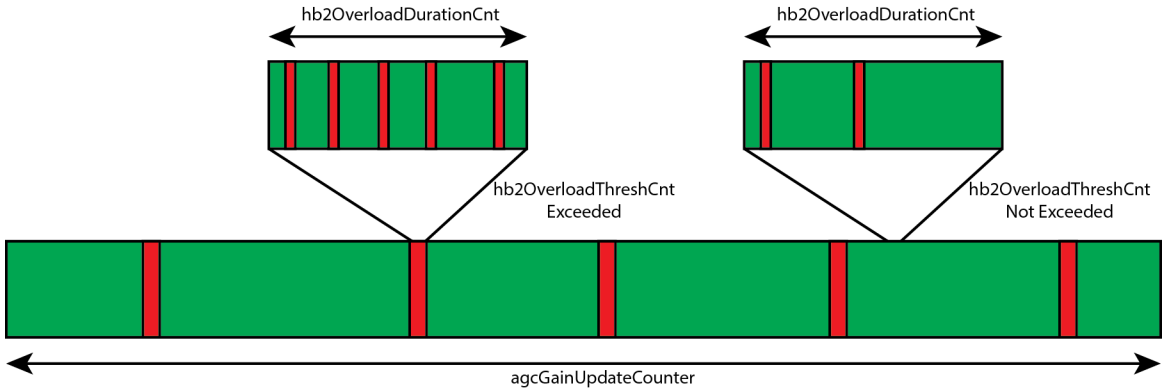


Figure 123. HB2 Detector Two-Level Approach for an Overload Condition

The HB2 detector has a number of programmable thresholds. Some of these thresholds are only used in the fast recovery mode of the peak detect AGC configuration, as summarized in Table 72.

Table 72. HB2 Overload Thresholds

HB2 Threshold	Usage
hb2HighThresh	Used for gain attack in both peak and power detect AGC modes.
hb2UnderRangeHighThresh	Used for gain recovery in peak detect AGC mode. In power detect AGC mode it is used to prevent overloads during gain recovery.
hb2UnderRangeMidThresh	Used only when the fast recovery option of the peak detect AGC mode is being utilized.
hb2UnderRangeLowThresh	Used only when the fast recovery option of the peak detect AGC mode is being utilized.

For more details of how these thresholds are used by the AGC, refer to the relevant sections of the AGC overview in this document (specifically Figure 115, Figure 117 and Figure 118).

The thresholds are related to an ADC dBFS value using the following equations:

$$hb2HighThresh = 16384 \times 10^{\left(\frac{hb2High\ dBFS}{20}\right)} \quad (15)$$

$$hb2UnderRangeHighThresh = 16384 \times 10^{\left(\frac{hb2UnderRangeHigh\ dBFS}{20}\right)} \quad (16)$$

$$hb2UnderRangeMidThresh = 16384 \times 10^{\left(\frac{hb2UnderRangeMid\ dBFS}{20}\right)} \quad (17)$$

$$hb2UnderRangeLowThresh = 16384 \times 10^{\left(\frac{hb2UnderRangeLow\ dBFS}{20}\right)} \quad (18)$$

Please note that these equations only apply if the hb2OverloadPowerMode = 0. If this parameter is set to 1, then the denominator in the exponent changes from 20 to 10.

Each threshold has an associated counter such that an over-range condition is not flagged until the threshold has been exceeded this amount of times in a gain update period.

Table 73. HB2 Overload Thresholds and Counters

HB2 Threshold	Counter
hb2HighThresh	hb2UpperThreshPeakExceededCnt

RX GAIN CONTROL AND GAIN COMPENSATION

Table 73. HB2 Overload Thresholds and Counters (Continued)

HB2 Threshold	Counter
hb2UnderRangeHighThresh	hb2UnderRangeHighThreshExceededCnt
hb2UnderRangeMidThresh	hb2UnderRangeMidThreshExceededCnt
hb2UnderRangeLowThresh	hb2UnderRangeLowThreshExceededCnt

In AGC mode, the HB2 has programmable gain attack and gain recovery step sizes.

Table 74. HB2 Attack and Recovery Step Sizes

Gain Change	Step Size
Gain Attack	hb2GainStepAttack
Gain Recovery (hb2UnderRangeHighThresh)	hb2GainStepHighRecovery
Gain Recovery (hb2UnderRangeMidThresh)	hb2GainStepMidRecovery
Gain Recovery (hb2UnderRangeLowThresh)	hb2GainStepLowRecovery

Power Detector

The power measurement block measures the RMS power of the incoming signal. It can monitor the signal level at different locations, namely the HB2 output, the RFIR output and the output of the DC correction block. To choose a location, the powerInputSelect API parameter is utilized as described in Table 75.

Table 75. Location of the Decimated Power Measurement

powerInputSelect	Value
DC Offset Output	0
RFIR Output	1
QFIR Output (QEC Filter)	2
HB2 Output	3

The number of samples that are used in the power measurement calculation is configurable using the powerMeasurementDuration API parameter:

$$Power\ Meas\ Duration\ (Rx\ Sample\ Clocks) = 8{*}2^{powerMeasurementDuration} \quad (19)$$

where Rx Sample Clocks is the number of clocks at the power measurement location. It is important that this duration not exceed the gain update counter. The gain update counter resets the power measurement block and therefore a valid power measurement must be available before this event. In the case of multiple power measurements occurring in a gain update period, the AGC will use the last fully completed power measurement, any partial measurements being discarded.

The power measurement block has a dynamic range of 60 dB by default.

AGC API FUNCTIONS

Table 76. List of AGC Configuration API Functions

API Method Name	Comments
adi_ADRV903x_AgcCfgSet()	Configures all the AGC settings as described above.
adi_ADRV903x_AgcCfgGet()	Readback the AGC settings.
adi_ADRV903x_AgcGainIndexRangeSet()	Configure min/max gain indices allowed for AGC operation.
adi_ADRV903x_AgcGainIndexRangeGet()	Function to read AGC Gain range for selected channel.
adi_ADRV903x_AgcReset()	Function to reset all AGC state machines and peak detector counters for selected channels.
adi_ADRV903x_RxGainGet()	Reads the Rx AGC Gain Index for the requested Rx channel.
adi_ADRV903x_RxTempGainCompSet()	Sets the Rx gain compensation.
adi_ADRV903x_RxTempGainCompGet()	Gets the Rx gain compensation.

RX GAIN CONTROL AND GAIN COMPENSATION

RX AND ORX POWER MEASUREMENT API FUNCTIONS

Table 77. List of Rx and ORx Power Measurement API Functions

API Method Name	Comments
adi_ADRV903x_RxDecimatedPowerCfgSet()	Set the Rx decimated power measurement block configuration.
adi_ADRV903x_RxDecimatedPowerCfgGet()	Get the Rx decimated power measurement block configuration.
adi_ADRV903x_ORxDecimatedPowerCfgSet()	Set the ORx decimated power measurement block configuration.
adi_ADRV903x_ORxDecimatedPowerCfgGet()	Get the ORx decimated power measurement block configuration.
adi_ADRV903x_RxDecimatedPowerGet()	Get the decimated power measurement for selected Rx/ORx channel.

AGC SAMPLE SCRIPT

The sample python script below is a function definition for settings the AGC values. They values below are ADI recommended values.

```
def setAGC(RxMask):
        AgcCfg1 = adi_adrvgen6_AgcCfg_t()
        AgcCfg1.rxChannelMask = RxMask
        AgcCfg1.agcAdcResetGainStep = 5
        AgcCfg1.agcChangeGainIfThreshHigh = 2
        AgcCfg1.agcEnableFastRecoveryLoop = 1
        AgcCfg1.agcLowThreshPreventGainInc = 1
        AgcCfg1.agcPeakThreshGainControlMode = 1
        AgcCfg1.agcPeakWaitTime = 2
        AgcCfg1.agcResetOnRxon = 1
        AgcCfg1.agcRxAttackDelay = 15
        AgcCfg1.agcRxMaxGainIndex = 255
        AgcCfg1.agcRxMinGainIndex = 185
        AgcCfg1.agcSlowLoopSettlingDelay = 32
        GUC = int(Update_Ts*getAgcClk_MHz(RxChannel) - 2*AgcCfg1.agcSlowLoopSettlingDelay -5)
        #Calculates Gain Update Counter based of desired Update Period (Update_Ts)
        AgcCfg1.agcGainUpdateCounter = GUC
        AgcCfg = adi_adrvgen6_AgcPeak_t()
        #Intervals
        AgcCfg.hb2UnderRangeLowInterval = 12288
        AgcCfg.hb2UnderRangeMidInterval = 1
        AgcCfg.hb2UnderRangeHighInterval = 3
        #Thresholds AND Steps
        AgcCfg.adcOvldGainStepAttack = 5
        AgcCfg.adcOvldLowThresh = 4
        AgcCfg.adcOvldGainStepRecovery= 0
        AgcCfg.hb2HighThresh = int(16384*(10**(HB2_HT/20.0)))
        AgcCfg.hb2GainStepAttack = 4
        AgcCfg.hb2UnderRangeHighThresh = int(16384*float(10**(HB2_LT1/20.0)))
        AgcCfg.hb2GainStepHighRecovery = 2
        AgcCfg.hb2UnderRangeMidThresh = int(16384*float(10**(HB2_LT2/20.0)))
        AgcCfg.hb2GainStepMidRecovery = 4
        AgcCfg.hb2UnderRangeLowThresh = int(16384*float(10**(HB2_LT3/20.0)))
        AgcCfg.hb2GainStepLowRecovery = 8
        #Thresh Counts
        AgcCfg.adcOvldUpperThreshPeakExceededCnt = 2
        AgcCfg.adcOvldLowerThreshPeakExceededCnt = 2
        AgcCfg.hb2OverloadThreshCnt = 6
        AgcCfg.hb2UpperThreshPeakExceededCnt = 3
        AgcCfg.hb2UnderRangeHighThreshExceededCnt = 4
        AgcCfg.hb2UnderRangeMidThreshExceededCnt = 4
        AgcCfg.hb2UnderRangeLowThreshExceededCnt = 4
```

RX GAIN CONTROL AND GAIN COMPENSATION

```
        #Misc
        AgcCfg.hb2OverloadDurationCnt = 32
        AgcCfg.hb2OverloadPowerMode = 0
        AgcCfg.hb2OverloadSignalSelection = 0
        AgcCfg.enableHb2Overload = 1
        AgcCfg1.agcPeak = AgcCfg
        AgcCfg1.agcPower.powerEnableMeasurement = 0
```

GAIN COMPENSATION, FLOATING POINT FORMATTER AND SLICER

The user has the option of enabling gain compensation in the device. In gain compensation mode, the digital gain block is utilized to compensate for the analog front-end attenuation. The cumulative gain across the device will be 0 dB; for example, if 5 dB analog attenuation is applied at the front end of the device then 5 dB of digital gain will be applied. This ensures that the digital data is representative of the RMS power of the signal at the Rx input port; any internal front-end attenuation changes in device in order to prevent ADC overloading are transparent to the baseband processor. In this way with gain compensation, the BBIC does not need to precisely track the current gain index to recover the received signal because the compensation is performed on the transceiver.

The digital gain block is controlled by the gain table, and a compensated gain table is required to operate in this mode. Such a gain table defines in each row a front-end attenuator setting with a corresponding amount of digital gain programmed at each index of the table. The user may create custom compensated gain tables as needed.

Gain compensation can be used in either AGC or MGC modes although has most utility in AGC mode. The maximum amount of gain compensation is 50 dB and the minimum is −18 dB. This allows for compensation of both the internal analog attenuator and an external gain component (such as a DSA or LNA).

Large amounts of digital gain will increase the bit-width of the datapath. There are several ways in which this expanded bit-width data can be sent to the BBIC, which are detailed below. Figure 124 is a block diagram of the gain compensation portion of the Rx chain, showing the locations of the various blocks.

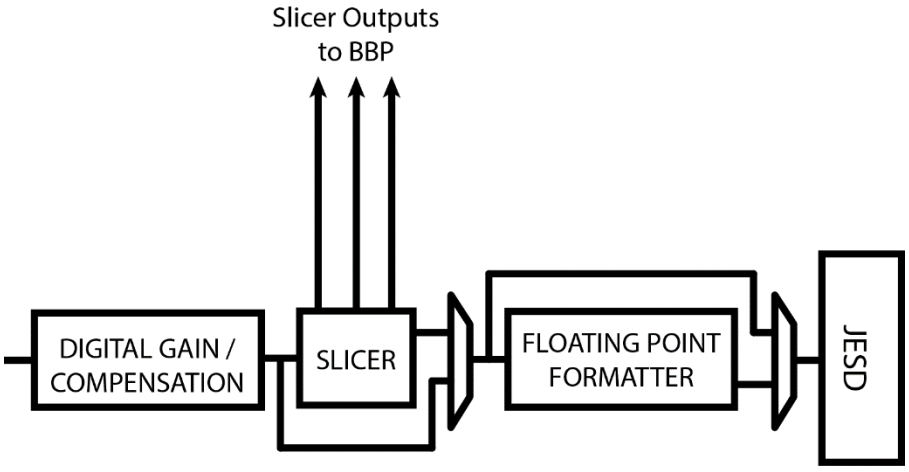


Figure 124. Gain Compensation, Floating Point Formatter, and Slicer Section of the Receiver Datapath

Please note that in addition to the standard gain compensation as defined in the gain table, there is also a digital temperature gain compensation feature in the datapath. This uses a SPI command to add or subtract digital gain. The purpose is to compensate for changes in external gain over temperature shifts. The user may have a temperature look up table defined within their system with different digital temperature gain compensation settings that are enabled when entering certain zones of operating temperature. The digital temperature gain compensation is exclusively managed by the user and is not intrinsically controlled.

Mode 1: No Digital Gain Compensation (Default)

This is the mode that the chip is configured to by default. In this mode the digital gain block is not used for gain compensation. Instead the digital gain block may be utilized to apply small amounts of digital gain/attenuation to provide consistent gain steps in a gain table. The premise

RX GAIN CONTROL AND GAIN COMPENSATION

is that because the analog attenuator does not have consistent stops in dB across its range then the digital gain block can be utilized to even out the steps for consistency (the default table utilizes the digital gain block to provide consistent 0.5 dB steps).

Neither the slicer nor floating-point formatter block is utilized. As no gain compensation is applied, there is no bit-width expansion of the digital signal. The signal is provided to the JESD port which sends it to the BBIC in either 12-bit, 16-bit, or 24-bit format depending on the use case.

Mode 2: Digital Gain Compensation With Slicer GPIO Outputs

In this mode gain compensation is used. The device should be loaded with a gain table that compensates for the analog front-end attenuation applied. Thus, as the analog front-end attenuation is increased, and equal amount of digital gain is applied. Considering 16-bit data at the input to the digital compensation block, then as more digital gain is applied the bit-width of the signal is increased. With every 6 dB of gain, the bit-width increases by 1. Figure 125 outlines this effect, with yellow boxes indicating the valid (used) bits in each case.

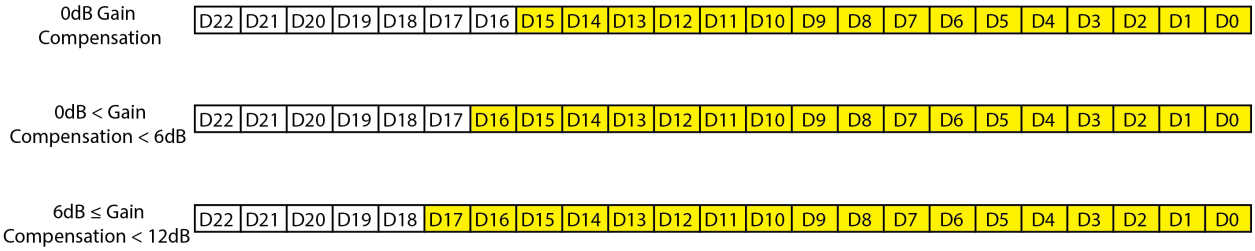


Figure 125. Bit Width of Received Signal for Increasing Gain Compensation

The slicer is used to attenuate the data after the digital gain block such that it can fit into the resolution of the JESD datapath. It then advises the user how much attenuation is being applied in real time, so that the user can compensate on the BBIC side. In this mode, the current slicer setting (amount of attenuation) is provided real time over GPIO pins.

Note that this slicer setting information is not necessarily time aligned to the data at the BBIC side. As soon as the slicer value changes, this information is provided on the GPIO pins. However, there will be some latency between this and when the corresponding data arrives across the JESD link. It is up to the user to determine an appropriate way of accounting for this latency.

This slicer can be configured for a number of attenuation resolutions, namely 1 dB, 2 dB, 3 dB, 4 dB, 6 dB, or 8 dB steps. Higher resolution (smaller steps) allows the user to follow the actual signal amplitude with finer resolution, while lower resolution (larger steps) allows for more compensation range.

The slicer can use up to three GPIOs per receiver. These require these pins to be enabled as outputs and configured for slicer output mode, see the General Purpose Input/Output Configuration section.

The following example explains the operation of the slicer in detail. In this use case, the JESD is configured for 16-bit data resolution. The slicer is configured to 6 dB resolution. Note that 6 dB is the most common setting as the baseband processor can easily approximate adding in the gain by a simple bit shift.

Figure 126 explains the operation. Initially the analog attenuator is applying no attenuation (0 dB) and hence there is 0 dB digital gain applied to the signal. The slicer is in its default (0000) position. As the attenuation increases (0 to 6 dB), a corresponding amount of digital gain is applied to the signal. With any digital gain applied to the signal, the bit-width of the signal has increased (the ADC can output 16-bits, further gain will allow a maximum input to go beyond 16-bits). In this case the signal has now a bit-width of 17. The slicer therefore applies 6 dB of attenuation, and the slicer position information across the GPIOs is updated to advise the user of this change (in this case 0001). This 6 dB attenuation ensures that the bit-width of the signal is 16 once more; that is, the 16 MSBs have been selected (sliced) with the LSB dropped. When the compensation increases beyond 6 dB, it is now possible that the signal resolution in the digital path can be 18-bit. The slicer then attenuates by 12 dB (or slices the 16 MSBs dropping the 2 LSBs).

RX GAIN CONTROL AND GAIN COMPENSATION

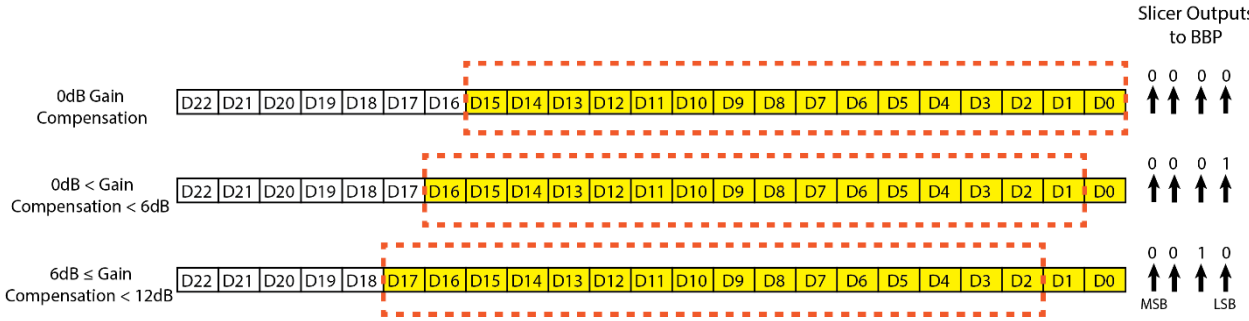


Figure 126. Slicer Bit Selection with Digital Gain

The BBIC receives these 16-bits and uses the slicer output to scale the power of the received signal to determine the power at the input to the device (or at the input to an external gain element if considered part of the digital gain compensation).

The slicer position versus digital gain for this 6 dB example is described in Table 78. Equivalent tables can be inferred for the other attenuation options.

Table 78. Slicer GPIO Output versus Digital Gain Compensation

Digital Gain Compensation (dB)	Slicer Position (Value output on GPIOs)
0	0
0 < Dig_Gain < 6	1
6 ≤ Dig_Gain < 12	2
12 ≤ Dig_Gain < 18	3
18 ≤ Dig_Gain < 24	4
24 ≤ Dig_Gain < 30	5
30 ≤ Dig_Gain < 36	6
36 ≤ Dig_Gain < 42	7
42 ≤ Dig_Gain < 48	8
48 ≤ Dig_Gain ≤ 50	9

Mode 3: Digital Gain Compensation With Embedded Slicer Position

This mode is like mode 2 but notably eliminates the GPIO pin requirement by replacing sample data bits with slicer information bits. The slicer is used to select the 16 MSBs based on the amount of digital gain used by the currently selected gain index in the gain table. However, in this mode the GPIO slicer outputs are not used. Instead the slicer position (or attenuation applied) is embedded into the data.

There are several permissible ways in which this can be configured which are shown in the figures in this section. The options are to place the slicer setting as 1 bit on both I and Q, or 2 bits on both I and Q. These can be placed at the MSBs or LSBs. For the case where 2 bits are embedded onto both I and Q data, there are further options of using three slicer bits or four. If three are used, there is a further option of inserting a zero to fill the fourth bit, or to insert a parity bit.

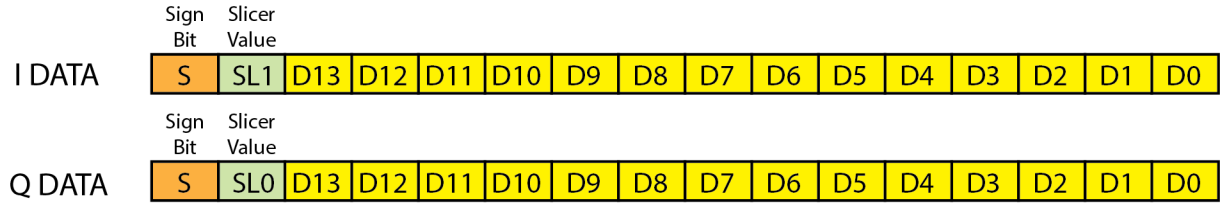


Figure 127. Encoding of Slicer Information as Control Bits (adi_ADRV903x_RxSlicerEmbeddedBits = ADI_ADRV903X_EMBED_1_SLICERBIT_AT_MSB)

RX GAIN CONTROL AND GAIN COMPENSATION

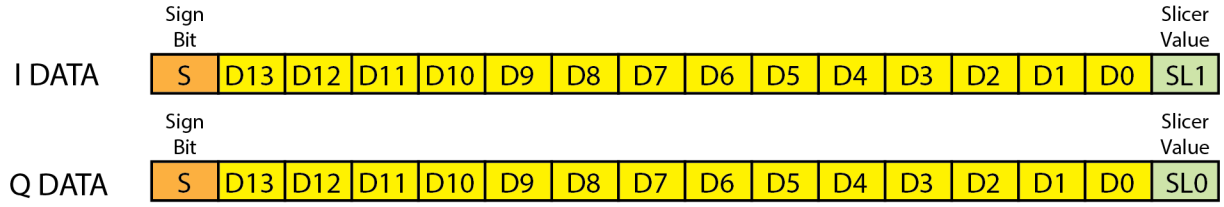


Figure 128. Encoding of Slicer Information as Control Bits (adi_ADRV903x_RxSlicerEmbeddedBits = ADI_ ADRV903X_EMBED_1_SLICERBIT_AT_LSB)

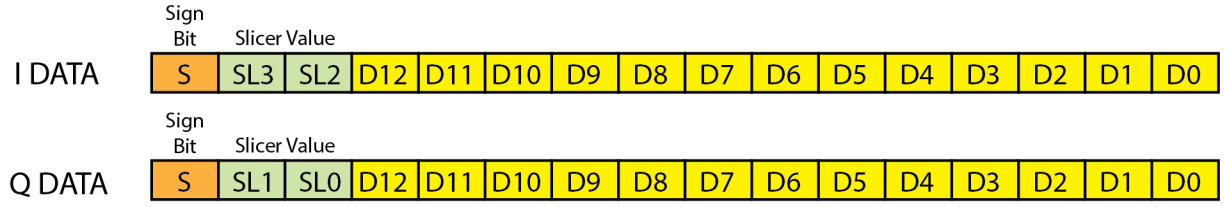


Figure 129. Encoding of Slicer Information as Control Bits (adi_ADRV903x_RxSlicerEmbeddedBits = ADI_ ADRV903X_EMBED_2_SLICERBITS_AT_MSB _3_BIT_ SLICER)

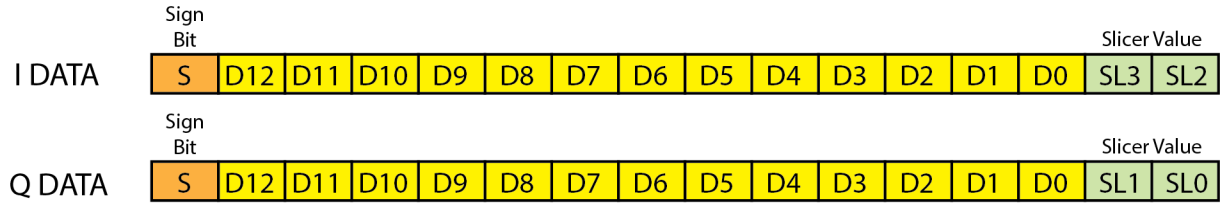


Figure 130. Encoding of Slicer Information as Control Bits (adi_ADRV903x_RxSlicerEmbeddedBits = ADI_ ADRV903X_EMBED_2_SLICERBITS_AT_LSB _3_BIT_ SLICER)

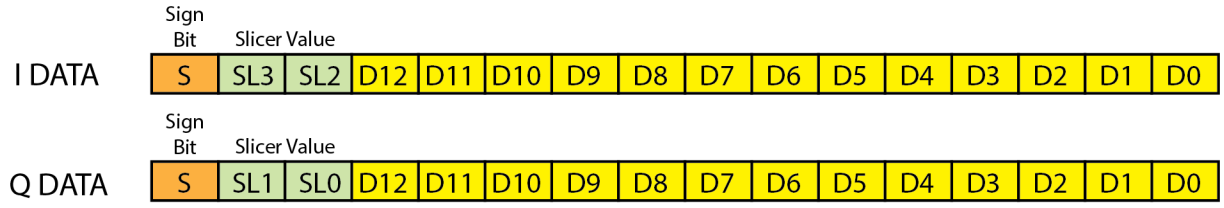


Figure 131. Encoding of Slicer Information as Control Bits (adi_ADRV903x_RxSlicerEmbeddedBits = ADI_ ADRV903X_EMBED_2_SLICERBITS_AT_MSB _4_BIT_ SLICER)

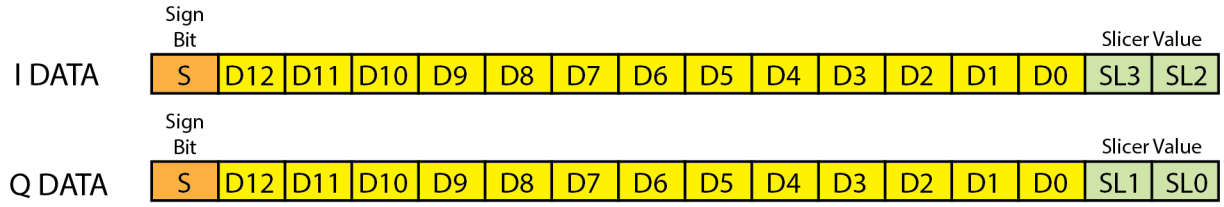


Figure 132. Encoding of Slicer Information as Control Bits (adi_ADRV903x_RxSlicerEmbeddedBits = ADI_ ADRV903X_EMBED_2_SLICERBITS_AT_LSB _4_BIT_ SLICER)

Mode 4: Digital Gain Compensation and Floating-Point Formatting

The floating-point formatter offers an alternative way of encoding the digitally compensated data onto the JESD204B link. In this mode, the data is converted to IEEE754 half precision floating point format (binary 16). There is a slight loss in resolution when using the floating-point formatter, though resolution is distributed such that smaller numbers have higher resolution.

RX GAIN CONTROL AND GAIN COMPENSATION

In binary 16 floating point format the number is composed on a sign-bit (S), an exponent (E) and a significand (T). There are a number of options in terms of the number of bits that can be assigned to the exponent. More bits in the exponent result in higher range, and thus can allow for more digital compensation to the represented, whereas more bits in the significand provides higher resolution. The available options for device's floating-point formatter are:

1. 5-bit exponent, 10-bit significand
2. 4-bit exponent, 11-bit significand
3. 3-bit exponent, 12-bit significand
4. 2-bit exponent, 13-bit significand

It is also possible to pack the data in different formats (as shown in Figure 133):

1. Sign, Exponent, Significand
2. Sign, Significand, Exponent

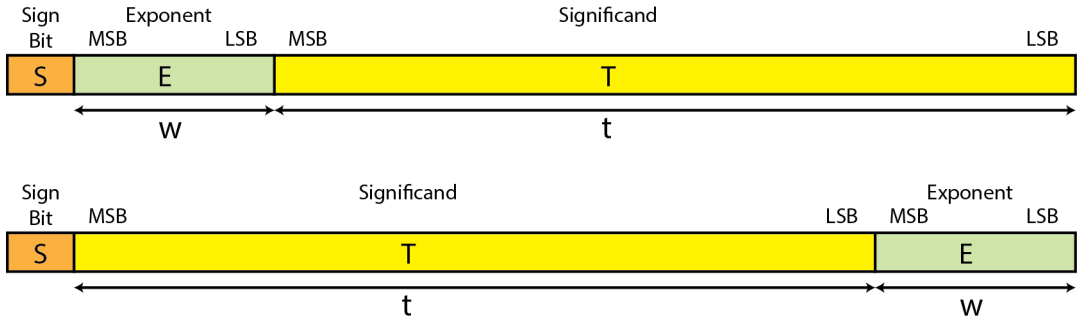


Figure 133. Floating Point Number Representation

In Figure 133, S is the sign bit, E is the value of the exponent, T is the value of the significand, w is the bit width of the exponent, and t is the bit width of the significand.

Upon receipt of an encoded floating-point formatter, the user breaks up the binary 16 number into its constituent parts. For the purposes of this explanation, let's consider a 3-bit exponent. In IEEE754, the maximum exponent (0'b111 in this case) is reserved for NaN. The minimum exponent (0'b000) is used for a signed zero (E=0, T=0) and subnormal numbers (E=0, T≠0). To decode a received floating-point sample, the following equations are used:

If E=0 and T=0:

$$Value = 0 \quad (20)$$

If E=0 and T≠0:

$$Value = (-1)^{S} \times 2^{E-bias+1} \times (0 + 2^{1-p} \times T) \quad (21)$$

If E≠0:

$$Value = (-1)^{S} \times 2^{E-bias} \times (1 + 2^{1-p} \times T) \quad (22)$$

where bias is used to convert the positive binary values to exponents which allow for values both less than and greater than the full-scale of the ADC and p is the precision of the mode (p=t+1, because you have t significand bits coupled with a sign bit). Table 79 provides the values to use in these equations for the various IEEE754 supported modes.

Table 79. Floating Point Formatter – Supported IEEE 754 Modes

Exponent Bit Width (w)	Significand Bit Width (t)	Precision (p)	Bias
5	10	11	15
4	11	12	7
3	12	13	3
2	13	14	1

RX GAIN CONTROL AND GAIN COMPENSATION

Figure 134 provides a visual representation of how the values of a waveform would be encoded in floating point format. In this case the maximum exponent (E-bias) is +3, meaning that data up to +24 dBFS of the ADC can be represented. As the signal reduces, the exponent required to represent each value differs. This is a different concept to the slicer that instead bit-shifted the data solely based on the applied digital attenuation and had a constant value for a constant digital gain. Instead the floating-point formatter interprets each data value after the digital gain compensation separately. Given the fixed precision of the significand and the sign bit, it can also be interpreted from this plot that there is higher resolution at lower signal levels then there is at higher signal levels, preserving SNR when the received signal strength is low.

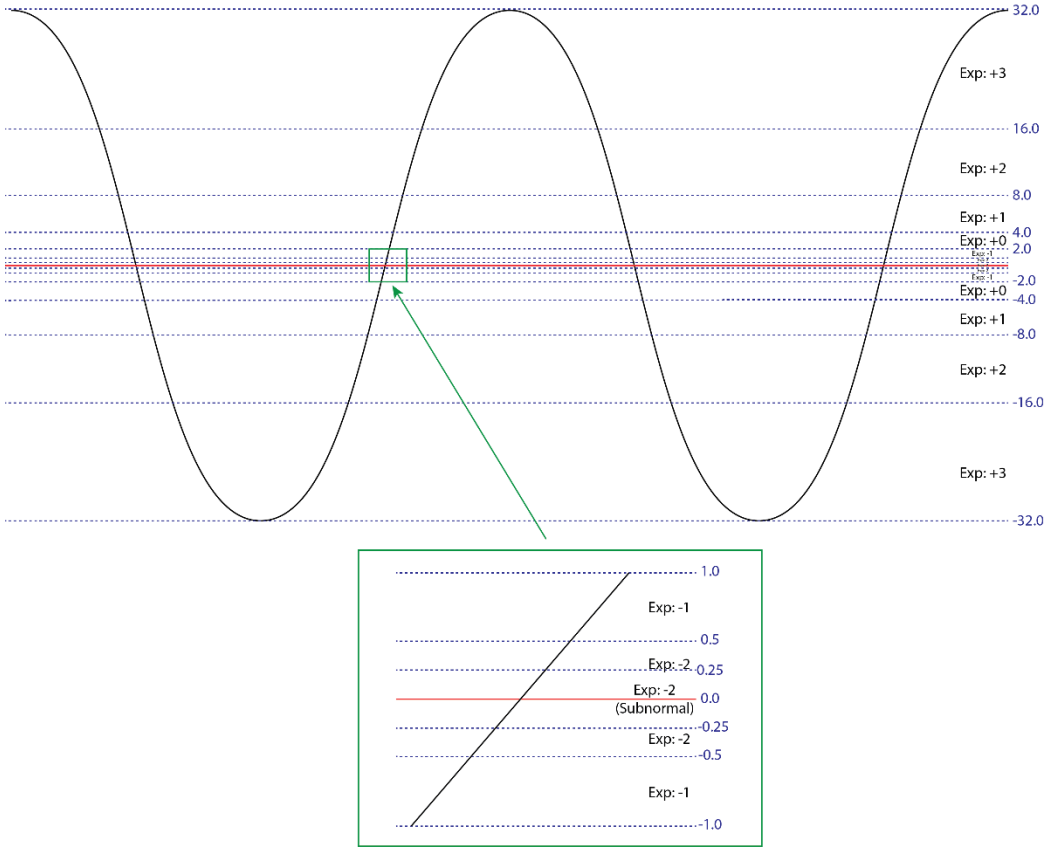


Figure 134. Visualization of the Floating-Point Formatter Values

The floating-point formatter also supports non-IEEE754 modes, referred to as Analog Devices modes, where the largest exponent is not used to express NaN in accordance with IEEE754. It is unnecessary for the device to encode NaN as none of the data values can be NaN, and therefore using this extra exponent value increases the largest value representable for a given exponent bit-width.

Table 80. Exponent Bit Widths of IEEE-754 and ADI Modes

Exponent Bit Width (w)	IEEE-754 Mode Exponent Range (after un-biasing)	ADI Mode Exponent Range (after un-biasing)
5	+15 to −14	+16 to −14
4	+7 to −6	+8 to −6
3	+3 to −2	+4 to −2
2	+1 to −1	+2 to −1

In the default floating point format, the leading one is inferred and not encoded (for normal numbers). It is possible to enable a mode where the leading one is encoded and stored in the MSB of the significand. This would reduce the precision of the values however.

If the user knows that the range of attenuation required for the worst case blocker (and therefore the digital gain required to compensate for it) will exceed the correction range allowed by the exponent width chosen, then it is also possible to enable a fixed digital attenuation (from 6 to 42 dB) prior to the floating point formatter to ensure that the signal never exceeds the maximum range encodable over the JESD link.

RX GAIN CONTROL AND GAIN COMPENSATION

Rx Data Format Data Structure

The Rx Data Formatter is not runtime configurable and therefore cannot be setup or modified by the API. In order to setup the Rx Data Formatter the Configurator must be used or by manually changing the JSON file.

RX DATA FORMATTER API FUNCTIONS

Table 81. List of Rx Formatter API Functions

API Method Name	Comments
adi_ADRV903x_RxDataFormatGet()	Get the Rx data path format configuration.
adi_ADRV903x_RxSlicerPositionGet()	Get the Rx gain slicer position.

DIGITAL FILTER CONFIGURATION

OVERVIEW

This section describes the digital filters within the ADRV903x. It provides a description of each of the filters in terms of their filter coefficients and position within the signal chain.

RECEIVER SIGNAL PATH

Each receiver input has an independent signal path including separate I/Q mixers. The signals are converted by the pipeline ADCs and filtered in half band decimation stages and the programmable finite impulse response filter (PFIR). The fixed coefficient halfband filters (FIR1, FIR2, HB2, DEC3) and the PFIR are designed to prevent data wrapping and over-range conditions.

Each receiver ADC is a high efficiency and wide bandwidth two-stage continuous time pipelined ADC, it successively converts the analog input into digital data, and processing the data in a pipelined manner. It operates at clock frequency of 2.9 GHz ~ 3.9 GHz with low OSR (oversampling rate) to realize the maximum signal bandwidth. The power consumption scales with the clock rates. The open-loop architecture makes it stable and the inherent low pass filter feature relaxes costly analog anti-alias filter requirements.

Each receiver channel can convert signals down to zero-IF real data using the standard I/Q configuration or a low-IF complex data configuration. The digital filtering stage allows the configuration flexibility and decimation options to operate in either mode.

If the carrier center frequency is same as LO frequency, you get the signal at DC and that will be Zero-IF. If the Carrier frequency configured is different then you will receive the signal as low IF, offset from DC.

The GUI allows configuration of a different LO frequency and carrier center frequency. Configuring LO frequency and Center frequency to be the same results in Zero-IF and keeping Tx/Rx center frequency different results in low IF.

The band NCO is used to shift the carriers in case of Low IF configuration as shown in band DUC block diagram in Figure 136. API for configuring the NCOs are available in Table 82.

Figure 135 shows the signal path for the Rx0, Rx1, Rx2, Rx3, Rx4, Rx5, Rx6, and Rx7 signal chain. Greyed out blocks are not described in this section but in their relevant sections of the user guide.

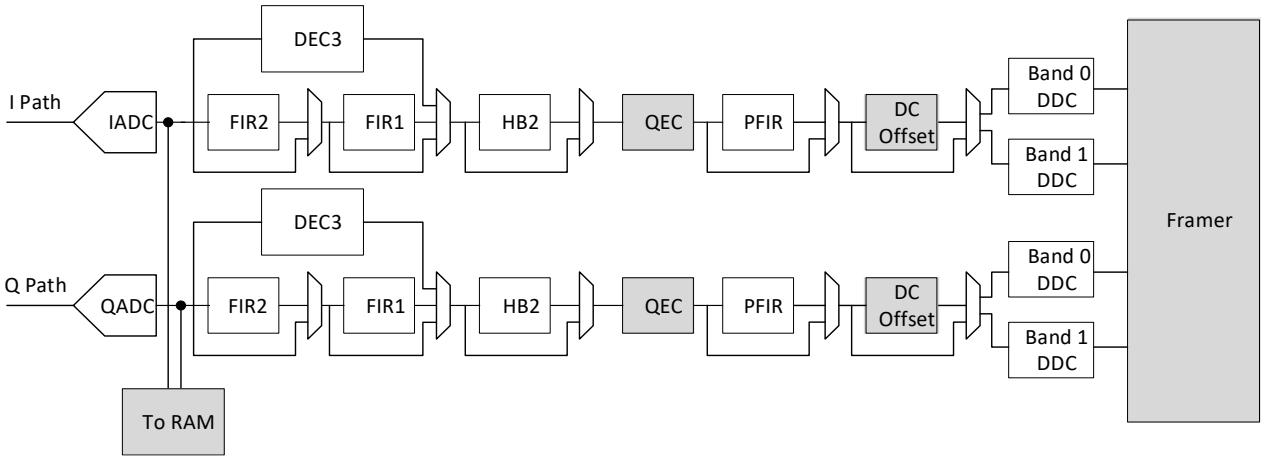


Figure 135. Rx Signal Path

Decimation Stages

The signal path can be configured so that either the decimate-by-3 filter (DEC3) or the combination of FIR2 and FIR1 along with HB2 is used in the Rx digital path. The DEC3 decimates by a factor of three while the other filter combination can be configured to decimate by factors of 2, 4, or 8. Each decimation stage can be bypassed as required.

DEC3

The DEC3 filter is a fixed coefficient decimating filter used when sample rate reduction by a factor of three is necessary. The filter coefficients are designed to account for proper passband shaping in this sampling scenario. The DEC3 filter coefficients are: [0.001220852, 0.001098767,

DIGITAL FILTER CONFIGURATION

-0.001953363, -0.012208522, -0.017336101, -0.006104261, 0.03418386, 0.065926016, 0.046026126, -0.063484312, -0.18117446, -0.177023562, 0.089244293, 0.560493224, 1.038701013, 1.230741057, 1.038701013, 0.560493224, 0.089244293, -0.177023562, -0.18117446, -0.063484312, 0.046026126, 0.065926016, 0.03418386, -0.006104261, -0.017336101, -0.012208522, -0.001953363, 0.001098767, 0.001220852]

Finite Impulse Response 2 (FIR2)

The FIR2 filter is a fixed coefficient decimating filter. The FIR2 decimates by a factor of two or it may be bypassed.

FIR2 filter coefficients: [0.01369863, 0, -0.1037182, 0, 0.59295499, 1.005870841, 0.59295499, 0, -0.1037182, 0, 0.01369863]

Finite Impulse Response 1 (FIR1)

The FIR1 filter is a fixed coefficient decimating filter. The FIR1 decimates by a factor of two or it may be bypassed.

FIR1 filter coefficients are: [-0.00097704, 0, 0.007327797, 0, -0.033707865, 0, 0.111382511, 0, -0.31704934, 0, 1.231069858, 1.996091842, 1.231069858, 0, -0.31704934, 0, 0.111382511, 0, -0.033707865, 0, 0.007327797, 0, -0.00097704,]

Receive Half Band 2 (HB2)

The HB2 filter is a fixed coefficient decimating filter. The HB2 decimates by a factor of two.

HB2 filter coefficients are: [0.000244156, 0, -0.00054935, 0, 0.000976622, 0, -0.001709089, 0, 0.002807789, 0, -0.004272722, 0, 0.006348044, 0, -0.009155832, 0, 0.012757126, 0, -0.01745712, 0, 0.023438931, 0, -0.031068791, 0, 0.040712934, 0, -0.052981749, 0, 0.068851859, 0, -0.089971312, 0, 0.119514131, 0, -0.164072514, 0, 0.241225661, 0, -0.415308552, 0, 1.266190563, 1.992980529, 1.266190563, 0, -0.415308552, 0, 0.241225661, 0, -0.164072514, 0, 0.119514131, 0, -0.089971312, 0, 0.068851859, 0, -0.052981749, 0, 0.040712934, 0, -0.031068791, 0, 0.023438931, 0, -0.01745712, 0, 0.012757126, 0, -0.009155832, 0, 0.006348044, 0, -0.004272722, 0, 0.002807789, 0, -0.001709089, 0, 0.000976622, 0, -0.00054935, 0, 0.000244156]

Rx Programmable Finite Impulse Response (PFIR)

The Rx PFIR is used to compensate for the roll-off of the analog TIA LPF. The PFIR has 24 filter taps. The PFIR also has programmable gain settings of +6 dB, 0 dB or −6 dB. The Rx PFIR filter has no decimating ability. You can bypass the Rx PFIR completely by setting the pfir_mode = 1 in the profile JSON file. Note in this case the Rx analog response will not be compensated for digitally in Palau, so channel equalisation will need to be done externally. Alternatively if you set pfir_mode = 2, this will add 0.5 dB additional attenuation into the PFIR without changing the flatness response. For all other applications you should keep pfir_mode = 0, which is the default setting.

DIGITAL FILTER CONFIGURATION

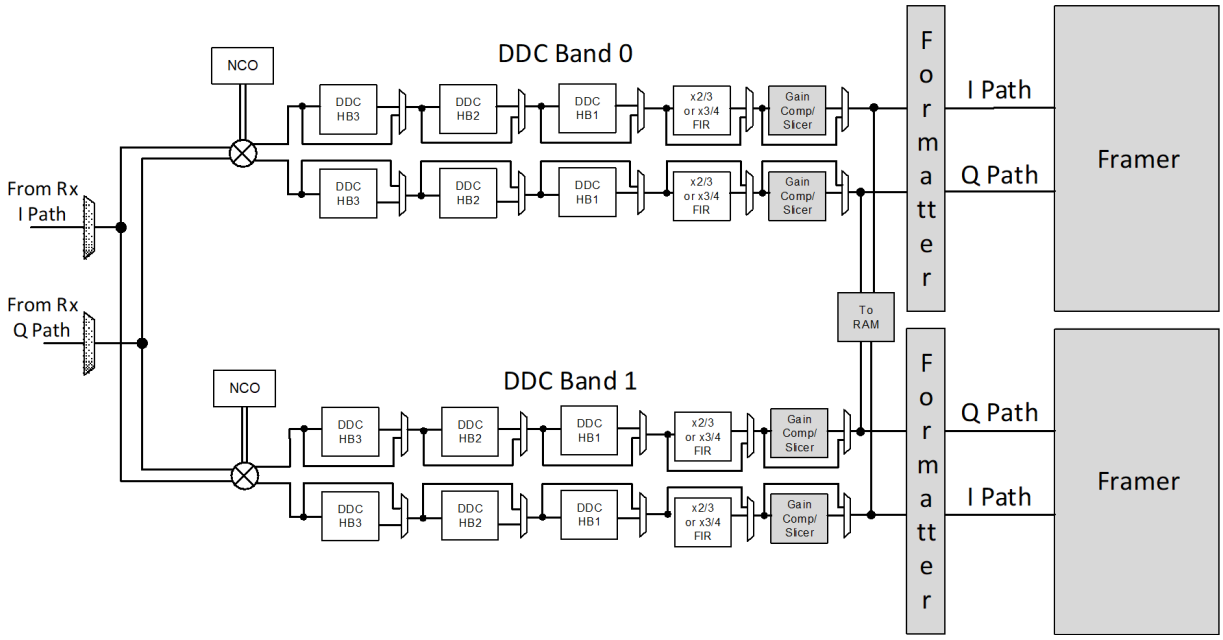


Figure 136. DDC Stage Band 0 / Band 1

DDC NCO

The DDC NCO blocks have a frequency tuning word (FTW) with 48-bit resolution and are programmable over SPI via the API. This NCO is used to place bands at the appropriate frequency using a complex multiplier. The maximum operating frequency of this NCO is 1 GHz. The NCO frequency range is also limited by the use case, In Figure 137 the JESD rate is 122.88 MSPS and we have decimation of 2x, 2x, 1x. So the rate at which NCO can operate is +/- 245.76 MSPS. If HB1 interpolation is also 2x, then the NCO can operate with the max range of +/- 491.52 MSPS.

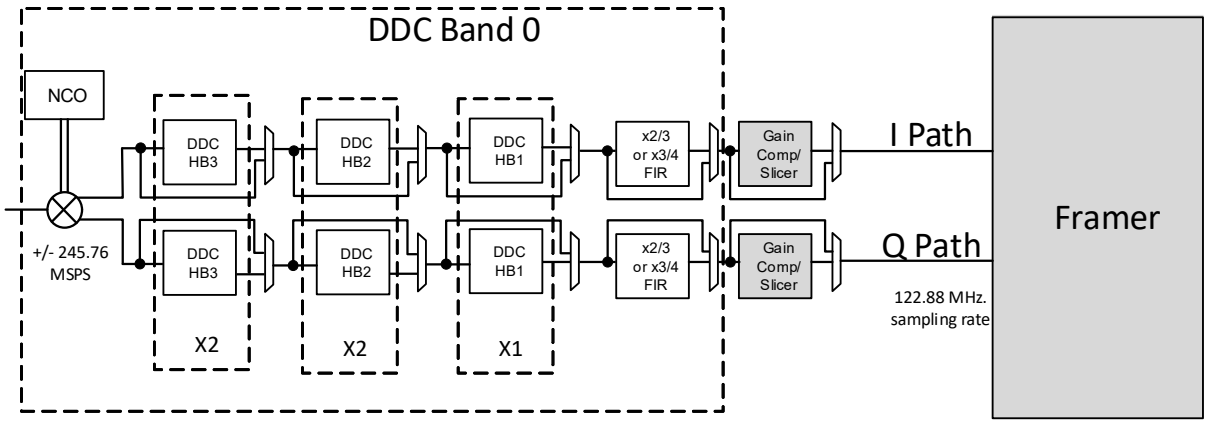


Figure 137. NCO Frequency Range and decimation in DDC

Digital Down-Conversion Half Band 3 (DDC HB3)

The DDC HB3 filter is a fixed coefficient decimating filter. The DDC HB3 decimates by a factor of two or it can be bypassed.

DDC HB3 filter coefficients: [0.01369863, 0, -0.1037182, 0, 0.59295499, 1.005870841, 0.59295499, 0, -0.1037182, 0, 0.01369863]

DIGITAL FILTER CONFIGURATION

Digital Down-Conversion Half Band 2 (DDC HB2)

The DDC HB2 filter is a fixed coefficient decimating filter. The DDC HB2 decimates by a factor of two or it can be bypassed.

DDC HB2 filter coefficients: [0.000854597, 0, -0.005982176, 0, 0.023928702, 0, -0.075204493, 0, 0.304968868, 0.497130997, 0.304968868, 0, -0.075204493, 0, 0.023928702, 0, -0.005982176, 0, 0.000854597]

Digital Down-Conversion Half Band 1 (DDC HB1)

The DDC HB1 filter is a fixed coefficient decimating filter. The DDC HB1 decimates by a factor of two or it can be bypassed.

DDC HB1 filter coefficients: [-0.000244148, 0, 0.000640889, 0, -0.001464888, 0, 0.002899258, 0, -0.005249184, 0, 0.008850368, 0, -0.014160588, 0, 0.021759697, 0, -0.032654805, 0, 0.048402356, 0, -0.072389904, 0, 0.113284707, 0, -0.203192236, 0, 0.632251961, 0.997650075, 0.632251961, 0, -0.203192236, 0, 0.113284707, 0, -0.072389904, 0, 0.048402356, 0, -0.032654805, 0, 0.021759697, 0, -0.014160588, 0, 0.008850368, 0, -0.005249184, 0, 0.002899258, 0, -0.001464888, 0, 0.000640889, 0, -0.000244148]

Resampling Finite Impulse Response (Resamp FIR)

The Resampling FIR filter can be set to 2/3 or 3/4 to get 2/3 or 3/4 of the current input rate at the output. This is especially useful to allow Tx rates such as 737.28 MHz to be combined with Rx rates of 491.52 MHz / 245.76 MHz for more flexibility when designing profiles.

2/3 Resampling FIR filter coefficients are: [0.000396741, 0.000793481, 0.000488296, -0.001312296, -0.003997925, -0.005218665, -0.002563555, 0.00347911, 0.008117924, 0.005645924, -0.004364147, -0.013946959, -0.012176885, 0.00369274, 0.021668142, 0.022644734, -0.000885037, -0.032166509, -0.039551988, -0.005981628, 0.04709006, 0.068605609, 0.021485031, -0.07269509, -0.131199072, -0.064821314, 0.146855068, 0.420911283, 0.613238929, 0.613238929, 0.420911283, 0.146855068, -0.064821314, -0.131199072, -0.07269509, 0.021485031, 0.068605609, 0.04709006, -0.005981628, -0.039551988, -0.032166509, -0.000885037, 0.022644734, 0.021668142, 0.00369274, -0.012176885, -0.013946959, -0.004364147, 0.005645924, 0.008117924, 0.00347911, -0.002563555, -0.005218665, -0.003997925, -0.001312296, 0.000488296, 0.000793481, 0.000396741]

3/4 Resampling FIR filter coefficients: [-6.1037E-05, 0.000244148, 0.000915555, 0.001922666, 0.002899258, 0.003173925, 0.002075259, -0.000396741, -0.003418073, -0.005432295, -0.004821924, -0.000976592, 0.004791406, 0.009491256, 0.00982696, 0.004150517, -0.005767998, -0.015045625, -0.017639698, -0.01004059, 0.005829035, 0.022339549, 0.029389325, 0.020142216, -0.003967406, -0.031800287, -0.047120579, -0.037171545, -0.001464888, 0.044801172, 0.075899533, 0.068147832, 0.014587848, -0.066286203, -0.133732109, -0.140140996, -0.053224281, 0.125614185, 0.355967895, 0.570360424, 0.699392682, 0.699392682, 0.570360424, 0.355967895, 0.125614185, -0.053224281, -0.140140996, -0.133732109, -0.066286203, 0.014587848, 0.068147832, 0.075899533, 0.044801172, -0.001464888, -0.037171545, -0.047120579, -0.031800287, -0.003967406, 0.020142216, 0.029389325, 0.022339549, 0.005829035, -0.01004059, -0.017639698, -0.015045625, -0.005767998, 0.004150517, 0.00982696, 0.009491256, 0.004791406, -0.000976592, -0.004821924, -0.005432295, -0.003418073, -0.000396741, 0.002075259, 0.003173925, 0.002899258, 0.001922666, 0.000915555, 0.000244148, -6.1037E-05]

RX DATAPATH API FUNCTIONS

Table 82. List of Rx Datapath API Functions

API Method Name	Comments
adi_ADRV903x_RxOrxDataCaptureStart()	Capture the Rx data in an internal RAM. Use if JESD link is not brought up.
adi_ADRV903x_RxNcoShifterSet()	Set Rx NCO.
adi_ADRV903x_RxNcoShifterGet()	Get Rx NCO settings.
adi_ADRV903x_RxTestDataSet()	Set up a test signal to be sent out the framer instead of the ADC output.
adi_ADRV903x_RxTestDataGet()	Get the test signal being sent out the framer instead of the ADC output.

TRANSMITTER SIGNAL PATH

Each transmitter has an independent signal path including separate digital filters, DACs, analog low-pass filters, and I/Q mixers that drive the signal outputs. Data is input to the Tx signal path via the JESD high-speed serial data interface at the transmitter profile's IQ data rate. The serial data is converted to parallel format through the JESD deframer into I and Q components. The data is processed through digital filtering and signal correction stages and input to I/Q DACs.

Each transmitter DAC is based on a current segmentation architecture providing a differential complementary current output. It uses the quad-switch structure and requires each pair of switches to be clocked on alternative clock edges to eliminates the code dependent glitches.

DIGITAL FILTER CONFIGURATION

It also allows the two interleaved input data streams to be updated on both rising and falling edges of the clock to effectively double the DAC sample rate without doubling the clock frequency.

Figure 138 shows the signal path for the Tx0, Tx1, Tx2, Tx3, Tx4, Tx5, Tx6, and Tx7 signal chain. Greyed-out blocks are not described in this section but in their relevant sections of the user guide.

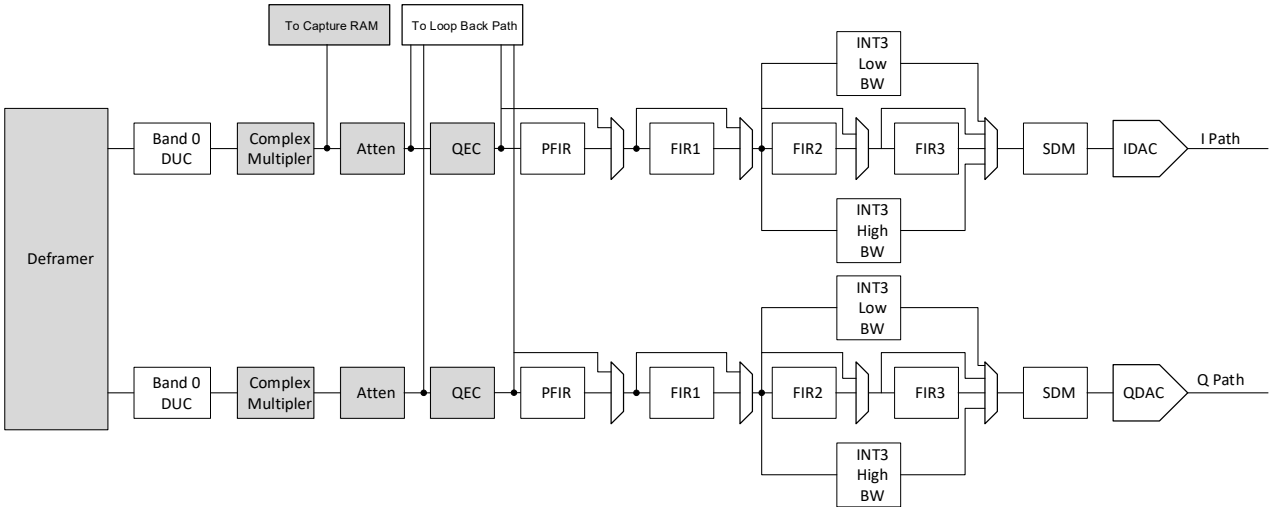


Figure 138. Tx Signal Path Diagram

Interpolation By 3 High Bandwidth Filter (INT3 High BW)

Either the INT3 High BW, INT3 Low BW or any combination of FIR2 and FIR3 are used in the Tx digital path. The INT3 High BW interpolates by a factor of 3.

INT3 High BW filter coefficients: [0.000976801, 0.001953602, 0.001953602, -0.0004884, -0.004151404, -0.005616606, -0.001221001, 0.007326007, 0.011721612, 0.004884005, -0.011233211, -0.021489621, -0.011965812, 0.015873016, 0.036141636, 0.024420024, -0.020512821, -0.058363858, -0.045909646, 0.024908425, 0.094261294, 0.085958486, -0.028327228, -0.165079365, -0.180952381, 0.030769231, 0.422222222, 0.808791209, 0.968009768, 0.808791209, 0.422222222, 0.030769231, -0.180952381, -0.165079365, -0.028327228, 0.085958486, 0.094261294, 0.024908425, -0.045909646, -0.058363858, -0.020512821, 0.024420024, 0.036141636, 0.015873016, -0.011965812, -0.021489621, -0.011233211, 0.004884005, 0.011721612, 0.007326007, -0.001221001, -0.005616606, -0.004151404, -0.0004884, 0.001953602, 0.001953602, 0.000976801]

Interpolation By 3 Low Bandwidth Filter (INT3 Low BW)

Either the INT3 Low BW, INT3 High BW or any combination of FIR2 and FIR3 are used in the Tx digital path. The INT3 Low BW interpolates by a factor of 3.

INT3 Low BW filter coefficients are: [0.019607843, 0.011764706, 0, -0.101960784, -0.098039216, 0, 0.407843137, 0.756862745, 0.996078431, 0.756862745, 0.407843137, 0, -0.098039216, -0.101960784, 0, 0.011764706, 0.019607843]

Finite Input Response 3 (FIR3)

The FIR3 is a fixed coefficient half-band interpolating filter. FIR3 can interpolate by a factor of two or it can be bypassed.

FIR3 filter coefficients are: [0.142857143, 0.571428571, 0.857142857, 0.571428571, 0.142857143,]

Finite Input Response 2 (FIR2)

The FIR2 is a fixed coefficient half-band interpolating filter. FIR2 can interpolate by a factor of two or it can be bypassed.

DIGITAL FILTER CONFIGURATION

FIR2 filter coefficients are: [-0.006349206, 0, 0.038095238, 0, -0.140659341, 0, 0.607570208, 0.997557998, 0.607570208, 0, -0.140659341, 0, 0.038095238, 0, -0.006349206]

Finite Input Response 1 (FIR1)

The FIR1 is a fixed coefficient half-band interpolating filter. FIR1 can interpolate by a factor of two or it can be bypassed.

FIR1 filter coefficients are: [0.000610426, 0.001465023, 0.000122085, -0.001831278, -0.000122085, 0.00280796, 0.000122085, -0.003906727, 0, 0.005371749, -0.000122085, -0.007203028, 0.000122085, 0.009278476, -0.000122085, -0.011842266, 0.000122085, 0.015138567, 0, -0.019045294, 0, 0.023928702, 0, -0.030032963, 0, 0.037846417, 0, -0.048101575, 0, 0.062385545, 0, -0.084116713, 0, 0.12196313, 0, -0.208033207, 0, 0.631424734, 0.993285313, 0.631424734, 0, -0.208033207, 0, 0.12196313, 0, -0.084116713, 0, 0.062385545, 0, -0.048101575, 0, 0.037846417, 0, -0.030032963, 0, 0.023928702, 0, -0.019045294, 0, 0.015138567, 0.000122085, -0.011842266, -0.000122085, 0.009278476, 0.000122085, -0.007203028, -0.000122085, 0.005371749, 0, -0.003906727, 0.000122085, 0.00280796, -0.000122085, -0.001831278, 0.000122085, 0.001465023, 0.000610426]

Tx Programmable Finite Impulse Response (PFIR)

The PFIR is used to compensate for roll off caused by the post-DAC analog low pass filter and the DAC sinc response. The PFIR has 24 filter taps. The PFIR also has programmable gain settings of +6 dB, 0 dB or -6 dB. The Tx PFIR filter has no interpolating ability, and it can be bypassed if desired.

Sigma Delta Modulator (SDM)

The TX DAC SDM is used to modulate the LSB's of the transmit data words going to the DAC which would normally be truncated. For normal traffic signals, truncation is sufficient. When broadcasting DC (i.e. unchanging) signals, the truncation of the LSB's leads to systematic biases and resulting TXLOL above specification.

The TX DAC SDM operates in four modes:

- Off: Clocks are disabled and the input to the SDM is muxed directly to the output with no modification.
- Bypass: Clocks are enabled, but the SDM accumulators are held in reset. The input is pipelined to the output without modification but matching the pipeline depth when the SDM is enabled.
- Enabled: Clocks are enabled and the SDM is running. The input is pipelined to match the SDM stages (2 pipes), truncated, and added to the SDM output. A 1/2 LSB round bit is subtracted at the same point to compensate for the 1/2 LSB round found before the DAC truncation.
- Automatic: Depending on the output of the TX DC Detection block, the SDM will operate in different modes. If the DC is below a threshold, the SDM is in bypass. If the DC is above the threshold, the SDM is in enabled mode.

DIGITAL FILTER CONFIGURATION

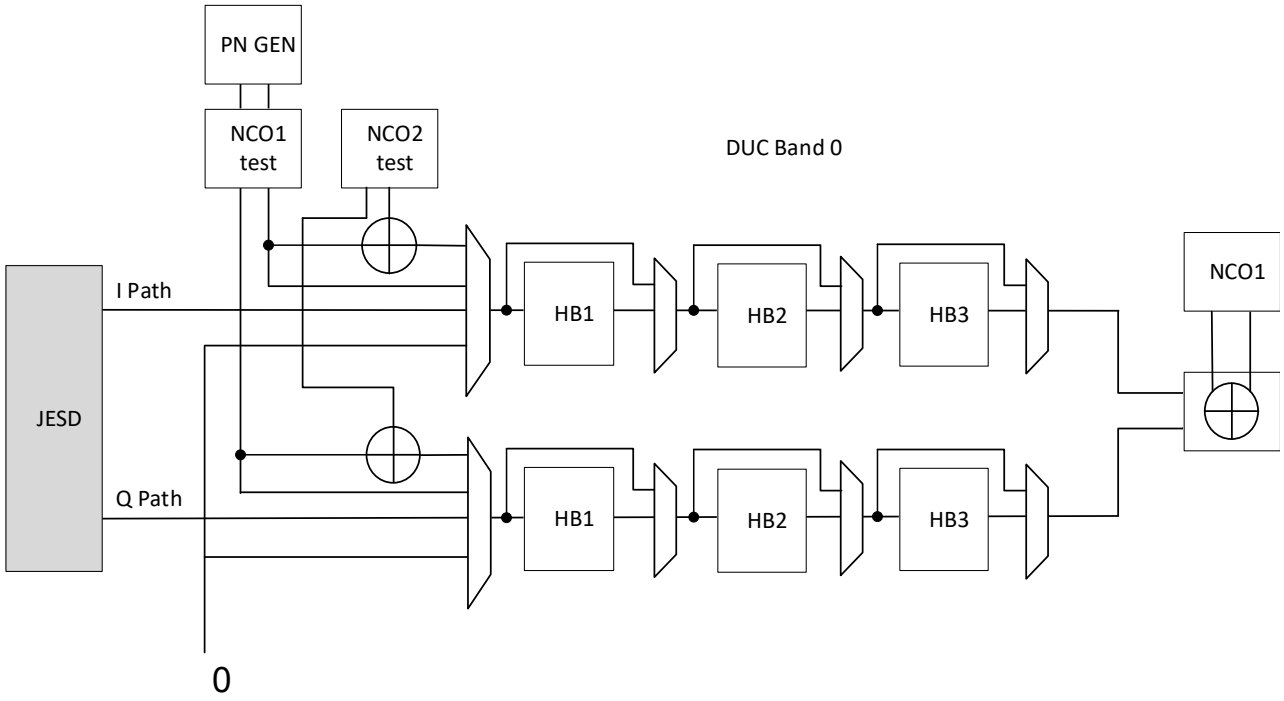


Figure 139. DUC Band 0

PN Generator (PN GEN)

The pseudo-random sequence generator can be muxed into the input of the Band0 DUC as shown in Figure 139. This PN sequence generator can run up to 1 GHz.

Test NCO (NCO1 Test and NCO2 Test)

The two test NCO's in the digital up convertor stage are identical and have a frequency tuning word with 20-bit resolution. Both of these NCO can run up to 1 GHz (±500 MHz).

Digital Up-Conversion Half Band 1 (DUC HB1)

The DUC HB1 filter is a fixed coefficient interpolating filter. The DUC HB1 interpolates by a factor of two or it can be bypassed.

DUC HB1 filter coefficients are: [0.000366256, 0, -0.000854597, 0, 0.002075449, 0, -0.004150897, 0, 0.007325113, 0, -0.012330607, 0, 0.019777805, 0, -0.030521304, 0, 0.046270297, 0, -0.070321084, 0, 0.111341717, 0, -0.201196435, 0, 0.628494689, 0.992308631, 0.628494689, 0, -0.201196435, 0, 0.111341717, 0, -0.070321084, 0, 0.046270297, 0, -0.030521304, 0, 0.019777805, 0, -0.012330607, 0, 0.007325113, 0, -0.004150897, 0, 0.002075449, 0, -0.000854597, 0, 0.000366256]

Digital Up-Conversion Half Band 2 (DUC HB2)

The DUC HB2 filter is a fixed coefficient interpolating filter. The DUC HB2 interpolates by a factor of two or it can be bypassed.

DUC HB2 filter coefficients are: [-0.005005494, 0, 0.034061775, 0, -0.134782078, 0, 0.601635942, 0.991820291, 0.601635942, 0, -0.134782078, 0, 0.034061775, 0, -0.005005494]

Digital Up-Conversion Half Band 3 (DUC HB3)

The DUC HB3 filter is a fixed coefficient interpolating filter. The DUC HB3 interpolates by a factor of two or it can be bypassed.

DIGITAL FILTER CONFIGURATION

DUC HB3 filter coefficients are: [0.013190034, 0, -0.101612115, 0, 0.586223742, 0.995603322, 0.586223742, 0, -0.101612115, 0, 0.013190034]

DUC NCO1

The DUC NCO has a frequency tuning word (FTW) with 48-bit resolution and is programmable over SPI via the API. This NCO is used to place bands at the appropriate frequency using a complex multiplier. The maximum operating frequency of this NCO is 1 GHz.

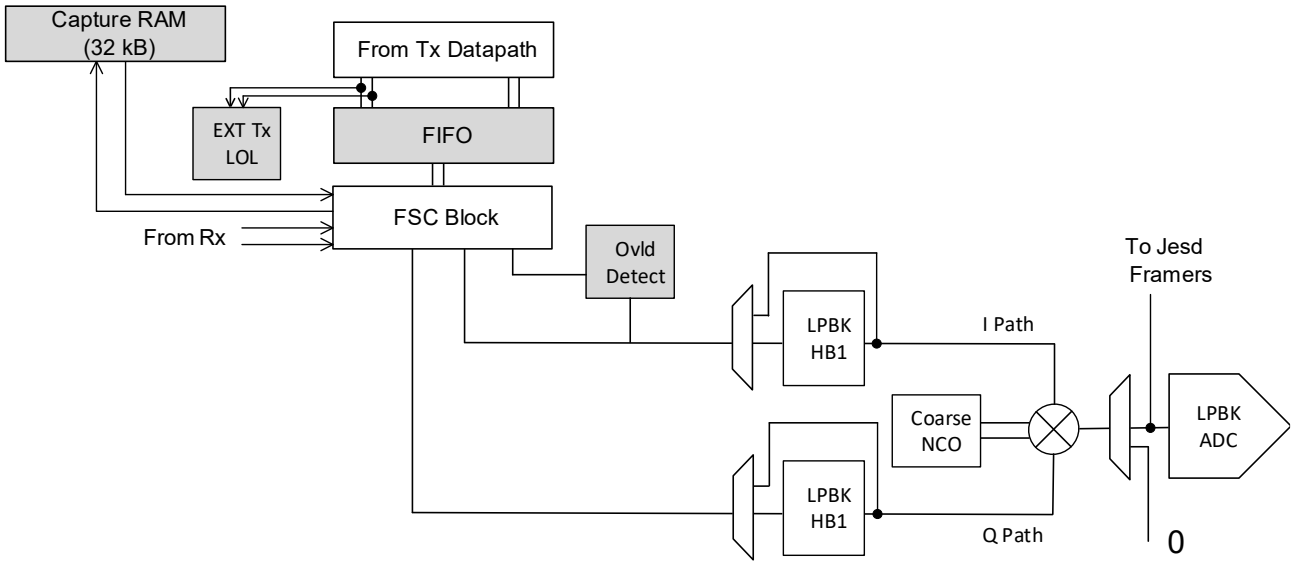


Figure 140. Tx Internal Loopback Path

Loopback Half-Band 1 Filter (LPBK HB1)

This filter is a decimating half-band filter with a stopband rejection of 80dBc. It can be bypassed if required.

The LPBK HB1 filter runs at the ADC clock rate divided by 2.

LPBK HB1 filter coefficients are: [-0.002685547, 0, 0.017578125, 0, -0.068359375, 0, 0.302734375, 0.498596191, 0.302734375, 0, -0.068359375, 0, 0.017578125, 0, -0.002685547]

Coarse NCO

The Coarse NCO in the loopback path has a frequency tuning word (FTW) with 3-bit resolution. This is used to shift the input data in frequency steps of ADC_Fs/8. The coarse NCO runs at the loopback path ADC rate.

TX DATAPATH API FUNCTIONS

Table 83. Tx Datapath API Functions

API Method Name	Comments
adi_ADRV903x_TxTestToneSet()	Set the specified Tx Test NCO to the given frequency, phase, attenuation, and enable state.
adi_ADRV903x_TxTestToneGet()	Get the specified Tx Test NCO's frequency, phase, attenuation, and enable state.
adi_ADRV903x_TxNcoShifterSet()	Set Tx NCO.
adi_ADRV903x_TxNcoShifterGet()	Get Tx NCO settings.

OBSERVATION RECEIVERS SIGNAL PATH

The device has two observation receivers (ORx0 and ORx1) that can be used to capture data for digital pre-distortion (DPD) algorithms and other measurements/calibration that require monitoring the transmitter outputs. The observation receiver can serve as an external loopback path to loop back the output of a PA, provided input level to the ORx is below the full-scale level of the ADC.

DIGITAL FILTER CONFIGURATION

The observation receiver path is implemented as direct RF conversion as opposed to the main receive path which is implemented as a zero-IF architecture. It eliminates the need for analog down conversion and supports large Nyquist zones suited for advanced multi-bands DPD applications. The ADC include a wide band S/H (sample and hold) buffer amplifier which runs at full sample rates and is based on an interleave architecture consisting of parallel pipelines ADCs with their sampling instances offset from each other to maintain uniform sampling of the input signal. The scalable and reconfigurable architecture balances high performance and low power consumption. The ORx ADC is implemented as four slices, each slice can run up to 2 GHz. This gives an effective sample rate of up to 8 GHz for the ORx signal path.

When the ORx is running at 8 GHz, the first Nyquist zone is from 0 GHz to 4 GHz and the second Nyquist zone is from 4 GHz to 8 GHz. The Nyquist zones scale with ADC rate, that is, if the ADC runs at 6 GHz the first Nyquist zone is from 0 GHz to 3 GHz and the second Nyquist zone is from 3 GHz to 6 GHz and then the third Nyquist zone would span from 6 GHz to 9 GHz. For ORx sampling there cannot be a bandwidth of interest spanning a Nyquist zone crossover point. The minimum distance a bandwidth of interest must start or end from a Nyquist zone crossover point is 100 MHz.

The assumption is that the user only supplies energy in the Nyquist zone through effective filtering as the device does not filter out other Nyquist zones. Hence if unwanted energy exists in those Nyquist zones it may fold down and corrupt the bandwidth of interest.

The diagram in Figure 141 shows the signal path for an ORx0 and ORx1 signal chain. Blocks that are not discussed in this section are greyed out.

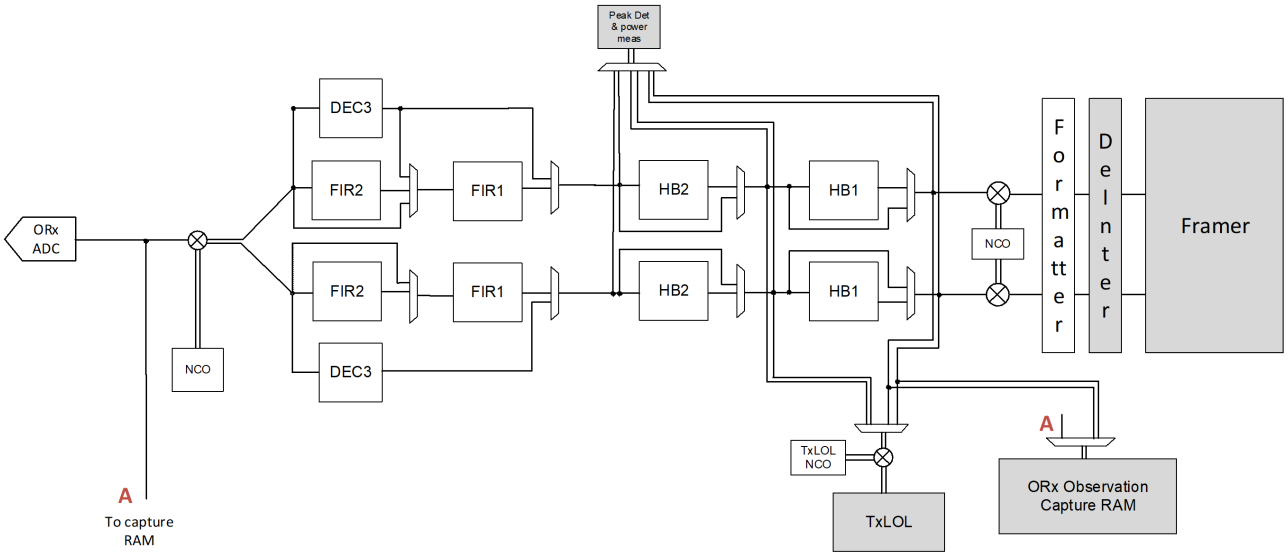


Figure 141. ORx Signal Path

As with standard discrete time theory the digital data is represented as the wanted signal and images of the wanted signal and can be seen in (a) of Figure 142. The goal is to shift the wanted signal down around DC to send only the necessary data of the ORx band to the SERDES interface. In an ideal configuration the wanted signal would be shifted down around DC and the image filtered out. However, the filters are not ideal, (b) in Figure 142, and hence we first shift the boundary of the wanted signal to the corner frequency of the filter path for maximum rejection of the image, (c) in Figure 142. Once the filtering is complete, (d) in Figure 142, the datapath uses the NCO at the end of the datapath to shift the wanted signal so that it is back centered around DC.

DIGITAL FILTER CONFIGURATION

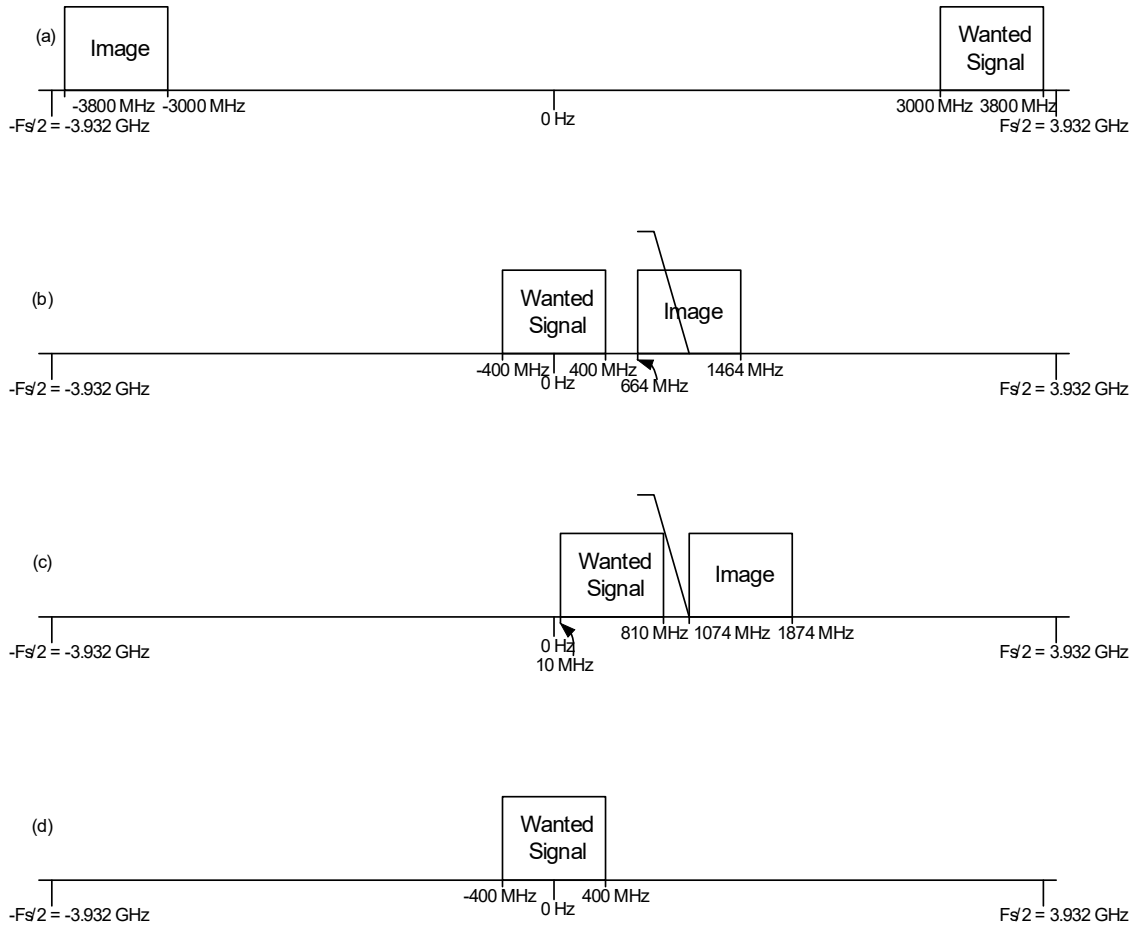


Figure 142. ORx Datapath NCO & Filtering Methodology

Decimation Stages

The signal path can be configured so that either the decimate-by-3 filter (DEC3) or FIR2 filter can be used in combinations with the FIR1, HB2 and HB1 filters in the ORx digital path. The DEC3 decimates by a factor of three while the other filter combination can be configured to decimate by factors of 2, 4, or 8. Each decimation stage can be bypassed as required.

Fine NCO

The observation path fine NCO has an FTW with 48-bit resolution. This NCO is used to place bands at the appropriate frequency using a complex multiplier. The maximum operating frequency of this NCO is 1 GHz.

DEC3

DEC3 filter coefficients are: [0.000976563, 0.000976563, -0.001953125, -0.012207031, -0.017578125, -0.006347656, 0.034179688, 0.065917969, 0.045898438, -0.063476563, -0.181152344, -0.176757813, 0.088867188, 0.55859375, 1.03515625, 1.2265625, 1.03515625, 0.55859375, 0.088867188, -0.176757813, -0.181152344, -0.063476563, 0.045898438, 0.065917969, 0.034179688, -0.006347656, -0.017578125, -0.012207031, -0.001953125, 0.000976563, 0.000976563]

Finite Impulse Response 2 (FIR2)

The FIR2 filter is a fixed coefficient decimating filter. The FIR2 decimates by a factor of two or it may be bypassed.

DIGITAL FILTER CONFIGURATION

FIR2 filter coefficients are: [0.013671875, 0, -0.103515625, 0, 0.58984375, 1, 0.58984375, 0, -0.103515625, 0, 0.013671875]

Finite Impulse Response 1 (FIR1)

The FIR1 filter is a fixed coefficient decimating filter. The FIR1 decimates by a factor of two or it may be bypassed.

FIR1 filter coefficients are: [-0.000976563, 0, 0.005859375, 0, -0.021484375, 0, 0.0625, 0, -0.165039063, 0, 0.619140625, 1, 0.619140625, 0, -0.165039063, 0, 0.0625, 0, -0.021484375, 0, 0.005859375, 0, -0.000976563]

Observation Receive Half Band 2 (HB2)

The HB2 filter is a fixed coefficient decimating filter. The HB2 decimates by a factor of two or it may be bypassed.

HB2 filter coefficients are: [0.000610352, 0, -0.00100708, 0, 0.00177002, 0, -0.00289917, 0, 0.004455566, 0, -0.006561279, 0, 0.009368896, 0, -0.013061523, 0, 0.017791748, 0, -0.023956299, 0, 0.031982422, 0, -0.042755127, 0, 0.057769775, 0, -0.080413818, 0, 0.119384766, 0, -0.20690918, 0, 0.632843018, 0.996490479, 0.632843018, 0, -0.20690918, 0, 0.119384766, 0, -0.080413818, 0, 0.057769775, 0, -0.042755127, 0, 0.031982422, 0, -0.023956299, 0, 0.017791748, 0, -0.013061523, 0, 0.009368896, 0, -0.006561279, 0, 0.004455566, 0, -0.00289917, 0, 0.00177002, 0, -0.00100708, 0, 0.000610352]

Observation Receive Half Band 1 (HB1)

The HB1 filter is a fixed coefficient decimating filter. The HB2 decimates by a factor of two or it may be bypassed.

HB2 filter coefficients are: [0.000610352, 0, -0.00100708, 0, 0.00177002, 0, -0.00289917, 0, 0.004455566, 0, -0.006561279, 0, 0.009368896, 0, -0.013061523, 0, 0.017791748, 0, -0.023956299, 0, 0.031982422, 0, -0.042755127, 0, 0.057769775, 0, -0.080413818, 0, 0.119384766, 0, -0.20690918, 0, 0.632843018, 0.996490479, 0.632843018, 0, -0.20690918, 0, 0.119384766, 0, -0.080413818, 0, 0.057769775, 0, -0.042755127, 0, 0.031982422, 0, -0.023956299, 0, 0.017791748, 0, -0.013061523, 0, 0.009368896, 0, -0.006561279, 0, 0.004455566, 0, -0.00289917, 0, 0.00177002, 0, -0.00100708, 0, 0.000610352]

TxLOL NCO

The TxLOL NCO has an FTW with 48-bit resolution. This NCO is used to place bands at the appropriate frequency for the Tx LOL calibration using a complex multiplier. The maximum operating frequency of this NCO is 1 GHz.

NCO

The observation path NCO has an FTW with 48-bit resolution. This NCO is used to place bands at the appropriate frequency using a complex multiplier at the end of the observation signal path. The maximum operating frequency of this NCO is 1 GHz.

ORX DATAPATH API FUNCTIONS

Table 84. List of ORx Datapath API Functions

API Method Name	Comments
adi_ADRV903x_RxOrxDataCaptureStart()	Capture the ORx data in an internal RAM. Use if JESD link is not brought up.
adi_ADRV903x_OrxNcoSet()	Set ORx NCO.
adi_ADRV903x_OrxNcoGet()	Get ORx NCO settings.

NCO FREQUENCY CHANGE PROCEDURE

On Tx/Rx/ORx data path there are separate NCOs that can be used to place the desired bands at the appropriate frequency using a complex multiplier. ADRV903x supports changing these NCOs frequencies during runtime and it needs to follow an appropriate procedure as described below.

Tx DUC NCO

As shown in Figure 139, NCO1 is the Tx DUC NCO that can be used to convert the desired bands at the appropriate frequency, the user can call adi_ADRV903x_TxNcoShifterSet() to change Tx NCO frequency at any time.

DIGITAL FILTER CONFIGURATION

RX DDC NCO

As shown in Figure 136, Rx DDC NCO can be used to convert the desired bands at the appropriate frequency, the user can take the procedure below to change its frequency,

1. Disable Rx QEC tracking if it is running.
2. Call adi_ADRV903x_RxNcoShifterSet() to change Rx DDC NCO frequency.
3. Enable Rx QEC tracking.

ORx NCO

On ORx channel as shown in Figure 141 there are two NCOs, one (ORx ADC NCO) is between ORx ADC and digital decimation filters that is used to convert the desired RF bands to the baseband that falls into digital filter bandwidth, the other (ORx datapath NCO) is just prior to ORx data formatter. As described in the Observation Receivers Signal Path section the datapath NCO is used to shift the wanted signal or band so that it is back centered around DC. Therefore, it is recommended for the users to utilize the datapath NCO to place the desired bands at any wanted frequency if the RF bands don't need to change. If the applications need to change the RF bands along with RF LO frequency change, it is recommended to take the same procedure that is described in this section above by adding ORx NCO frequency change on step 3 by calling API adi_ADRV903x_OrxNcoSet().

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

The device features 24 digital GPIO pins and 16 analog GPIO pins. These pins can be configured to provide more direct real time control and feedback from the ADRV903x than what is possible over the SPI interface. The distinction of analog GPIO does not imply analog behavior of the pin.

DIGITAL GPIO OPERATION

The device has 24 digital GPIO pins. These pins can be used in a variety of functions both for input and output modes. The digital GPIO pins use the VIF power supply as a reference for its logic high level.

Digital GPIO Input Modes

The digital GPIO input modes include the stream processor trigger and Tx attenuation control pins.

The stream processor executes streams based on triggers from various events. For example, the rising edge of a TRXn_CTRL pin and the ARM may directly invoke the execution of a stream, refer to the Stream Processor and System Control section. A stream is a programmed sequence of events executed within the chip to serve a distinct purpose. In the case of the GPIO based streams each GPIO can execute a unique stream based on a rising edge trigger and another unique stream for a falling edge trigger of the same GPIO. The primary use of the GPIO stream processor input is to facilitate Tx-to-ORx mapping which uses some number of GPIO pins in order to inform the ARM which Tx channel is currently looped back to the ORx. This is necessary to ensure proper operation of the External LO Leakage tracking calibration loop. Other stream capabilities will be defined over time. If there are specific requests for stream functionality based on GPIO trigger events, please request the functionality to an ADI Applications Engineer.

Tx attenuation control may use GPIO based latching schemes if desired. This refers to the case where the user does not want the Tx attenuation to update immediately after the SPI command to set the Tx attenuation is completed and instead wants the Tx attenuation to update immediately after a GPIO rising edge is observed on a programmed pin. Multiple Tx channels can use the same pin for attenuation update.

A second mode of Tx attenuation control is the S0/S1 attenuation feature. In this mode, the Tx attenuation is set as either S0 (State 0) or S1 (State 1) and the level of a GPIO pin determines whether the Tx channel is set into either S0 or S1. Multiple Tx channels can use the same pin for S0/S1 state control.

Table 85 summarizes the input digital GPIO features.

Table 85. Summary of Digital GPIO Input Features

Feature	Description	GPIO Pins Available for Feature
Stream GPIO Trigger	Executes a stream based on rising edge or falling edge pulses on a GPIO pin	GPIO pins [0:23]
Tx Attenuation Latch	Delays update of Tx attenuation until a GPIO pulse is detected on an assigned GPIO pin.	GPIO pins [0:23]
Tx Attenuation S0/S1 Control	Selects Tx attenuation state between S0 state and S1 state based on the level of a GPIO pin	GPIO pins [0:23]

Digital GPIO Output Modes

The digital GPIO output modes include access to the control out mux. The control out mux is an extremely flexible hardware logic that allows signals internal to the Tx, Rx, ORx, and other portions of the chip to be sent to the GPIO pins. An example of some signals that can be sent along GPIO pins include overload detector status from the Rx datapath, Rx slicer information, or status information from the PLLs. The flexibility of the control out mux allows any internal signal to be routed to any GPIO pin which affords a high level of flexibility in application debug. Please work with ADI Applications Engineering if specific signals are required for observation in an application.

DIGITAL GPIO API FUNCTIONS

Table 86. Digital GPIO API Functions

API Method Name	Comments
adi_ADRV903x_GpioStatusRead()	Returns the full status of all digital and analog GPIOs.
adi_ADRV903x_GpioConfigAllGet()	Returns the currently routed signal and channel Mask for all 24 digital GPIO pins
adi_ADRV903x_GpioConfigGet()	Returns the currently routed signal and channel Mask for a selected digital GPIO pin
adi_ADRV903x_GpioManualInputDirSet()	Allocates and configures selected digital GPIO pins for Manual Input Mode
adi_ADRV903x_GpioManualInputPinLevelGet()	Returns input read levels for all digital GPIO pins configured as Manual Inputs

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

Table 86. Digital GPIO API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_GpioManualOutputDirSet()	Allocates and configures selected digital GPIO pins for Manual Output Mode
adi_ADRV903x_GpioManualOutputPinLevelGet()	Returns output drive levels for all digital GPIO pins configured as Manual Outputs
adi_ADRV903x_GpioManualOutputPinLevelSet()	Sets output drive levels for all digital GPIO pins configured as Manual Outputs
adi_ADRV903x_GpioMonitorOutRelease()	This API function releases a GPIO if it is currently routing a monitor output function
adi_ADRV903x_GpioMonitorOutSet()	This API function configures a monitor output function for a GPIO
adi_ADRV903x_StreamGpioConfigSet()	Sets the GPIO pin assignments that trigger streams.
adi_ADRV903x_StreamGpioConfigGet()	Gets the GPIO pin assignments that trigger streams.

ANALOG GPIO OPERATION

The device has 16 analog GPIO pins. The voltage reference level of the analog GPIOs is 1.8 V. The reference supplies for these pins are VANA0_1P8 for GPIO_ANA_0 through GPIO_ANA_7 and VANA1_1P8 for GPIO_ANA_8 through GPIO_ANA_15. The main purpose of the GPIO_ANA pins is to serve as control pins for an external control element, such as a Digital Step Attenuator (DSA) or Low Noise Amplifier (LNA). A high-level overview of the GPIO_ANA features are provided below.

Table 87. Summary of GPIO_ANA Features

Feature	Description	GPIO Pins Available for Feature
Rx Gain Table External Control Word Output	The Rx gain table includes a column for 2-bit control of an external gain element. Each Rx channel has two fixed GPIO_ANA pins associated with it. The 2-bit value expressed on the pins depends on the gain index and gain table column	GPIO_ANA_[1:0]: Rx0 External Control Word GPIO_ANA_[3:2]: Rx1 External Control Word GPIO_ANA_[5:4]: Rx2 External Control Word GPIO_ANA_[7:6]: Rx3 External Control Word GPIO_ANA_[9:8]: Rx4 External Control Word GPIO_ANA_[11:10]: Rx5 External Control Word GPIO_ANA_[13:12]: Rx6 External Control Word GPIO_ANA_[15:14]: Rx7 External Control Word

Gain Table External Control Word

For proper use of this feature, a custom gain table must be created that uses the external control column. When a gain index with a non-zero value in the external control column of the gain table is selected, the value of the external control column will be output on a pair of GPIO_ANA pins. The configuration of the GPIO pins for gain table external control word is performed with the API.

ANALOG GPIO API FUNCTIONS

Table 88. List of Analog GPIO API Functions

API Method Name	Comments
adi_ADRV903x_GpioStatusRead()	Returns the full status of all digital and analog GPIOs.
adi_ADRV903x_GpioAnalogConfigAllGet()	Returns the currently routed signal and channel Mask for all 16 Analog GPIO pins
adi_ADRV903x_GpioAnalogConfigGet()	Returns the currently routed signal and channel Mask for a selected Analog GPIO pin
adi_ADRV903x_GpioAnalogManualInputDirSet()	Allocates and configures selected analog GPIO pins for Manual Input Mode
adi_ADRV903x_GpioAnalogManualInputPinLevelGet()	Returns input read levels for all analog GPIO pins configured as Manual Inputs
adi_ADRV903x_GpioAnalogManualOutputDirSet()	Allocates and configures selected analog GPIO pins for Manual Output Mode
adi_ADRV903x_GpioAnalogManualOutputPinLevelGet()	Returns output drive levels for all analog GPIO pins configured as Manual Outputs
adi_ADRV903x_GpioAnalogManualOutputPinLevelSet()	Sets output drive levels for all analog GPIO pins configured as Manual Outputs

GENERAL PURPOSE INTERRUPT

The device features two General Purpose Interrupt pins, GP_INT0 and GP_INT1. The GP_INT pins provide an interface that allows the device to inform the BBIC of an error in normal operation. Examples of the interrupt sources include PLL unlock events, SERDES link status, a stream processor error, or ARM exception error. A full list of interrupt sources is provided in Table 89. An example use of the two GP_INT pins is GP_INT1 pin can act as the high priority interrupt pin for Tx/Rx4-7 and GP_INT0 can act as the high priority interrupt pin for Tx/Rx0-3. All non-channel specific interrupts could be routed to both pins. The pins can be configured with independent bitmasks that control which signals can assert GP_INT1 or GP_INT0. GP_INT0 and GP_INT1 pins represent a bitwise OR of all unmasked GP_INT sources. Set the bit to 0 to unmask it and set the bit to 1 to mask it.

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

A description of the interrupt sources and their bit positions within the 48-bit Lower Word general purpose interrupt mask is provided in Table 89. The description of the 48-bit Upper Word of the general purpose interrupt mask is provided in Table 90.

Table 89. GP_INTERRUPT Bitmask Description Lower Word

Bit Position	Brief Description	Component
D51	Currently Unused	Open
D50	Currently Unused	Open
D49	Currently Unused	Open
D48	Currently Unused	Open
D47	Currently Unused	Open
D46	Currently Unused	Open
D45	Currently Unused	Open
D44	Currently Unused	Open
D43	Currently Unused	Open
D42	Currently Unused	Open
D41	Currently Unused	Open
D40	Core Stream Processor Error	Stream Proc
D39	ORx1 Stream Processor Error	Stream Proc
D38	ORX0 Stream Processor Error	Stream Proc
D37	Tx7 Stream Processor Error	Stream Proc
D36	Tx6 Stream Processor Error	Stream Proc
D35	Tx5 Stream Processor Error	Stream Proc
D34	Tx4 Stream Processor Error	Stream Proc
D33	Tx3 Stream Processor Error	Stream Proc
D32	Tx2 Stream Processor Error	Stream Proc
D31	Tx1 Stream Processor Error	Stream Proc
D30	Tx0 Stream Processor Error	Stream Proc
D29	Rx7 Stream Processor Error	Stream Proc
D28	Rx6 Stream Processor Error	Stream Proc
D27	Rx5 Stream Processor Error	Stream Proc
D26	Rx4 Stream Processor Error	Stream Proc
D25	Rx3 Stream Processor Error	Stream Proc
D24	Rx2 Stream Processor Error	Stream Proc
D23	Rx1 Stream Processor Error	Stream Proc
D22	Rx0 Stream Processor Error	Stream Proc
D21	Deframer IRQ [11] - Deframer 1 - Asserted with SYNC meaning the link dropped	Deframer
D20	Deframer IRQ [10] - Deframer 1 - Sysref phase error, unexpected Sysref phase received	Deframer
D19	Deframer IRQ [9] - Deframer 1 - PCLK is running too fast in relation to the sample/convertor clock	Deframer
D18	Deframer IRQ [8] - Deframer 1 - PCLK is running too slow in relation to the sample/convertor clock	Deframer
D17	Deframer IRQ [7] - Deframer 1 - 204B link layer error	Deframer
D16	Deframer IRQ [6] - Deframer 1 - 204C link layer error. CRC error count has exceeded threshold	Deframer
D15	Deframer IRQ [5] - Deframer 0 - Asserted with SYNC meaning the link dropped	Deframer
D14	Deframer IRQ [4] - Deframer 0 - Sysref phase error, unexpected Sysref phase received	Deframer
D13	Deframer IRQ [3] - Deframer 0 - PCLK is running too fast in relation to the sample/convertor clock	Deframer
D12	Deframer IRQ [2] - Deframer 0 - PCLK is running too slow in relation to the sample/convertor clock	Deframer
D11	Deframer IRQ [1] - Deframer 0 - 204B link layer error	Deframer
D10	Deframer IRQ [0] - Deframer 0 - 204C link layer error. CRC error count has exceeded threshold	Deframer
D9	Framer IRQ [9] - Reserved	Framer
D8	Framer IRQ [8] - Framer 2 - SYSREF phase error, unexpected Sysref phase received	Framer
D7	Framer IRQ [7] - Framer 2 - PCLK is running too fast in relation to the sample/convertor clock	Framer
D6	Framer IRQ [6] - Framer 2 - PCLK is running too slow in relation to the sample/convertor clock	Framer

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

Table 89. GP_INTERRUPT Bitmask Description Lower Word (Continued)

Bit Position	Brief Description	Component
D5	Framer IRQ [5] - Framer 1 - SYSREF phase error, unexpected Sysref phase received	Framer
D4	Framer IRQ [4] - Framer 1 - PCLK is running too fast in relation to the sample/convertor clock	Framer
D3	Framer IRQ [3] - Framer 1 - PCLK is running too slow in relation to the sample/convertor clock	Framer
D2	Framer IRQ [2] - Framer 0 - SYSREF phase error, unexpected Sysref phase received	Framer
D1	Framer IRQ [1] - Framer 0 - PCLK is running too fast in relation to the sample/convertor clock	Framer
D0	Framer IRQ [0] - Framer 0 - PCLK is running too slow in relation to the sample/convertor clock	Framer

Table 90. GP_INTERRUPT Bitmask Description Upper Word

Bit Position	Brief Description	Component
D47	Currently Unused	Open
D46	Currently Unused	Open
D45	Currently Unused	Open
D44	Currently Unused	Open
D43	Currently Unused	Open
D42	Currently Unused	Open
D41	Currently Unused	Open
D40	SPI1 Paging Error	SPI
D39	SPI0 Paging Error	SPI
D38	SPI1 Core Error	SPI
D37	SPI0 Abort	SPI
D36	SPI1 Abort	SPI
D35	SPI Clock Read abort	SPI
D34	Source Reducer Error Indication [0]	SPI
D33	Source Reducer Error Indication [0]	SPI
D32	RF0 PLL Unlock	PLL
D31	RF1 PLL Unlock	PLL
D30	Clock PLL Unlock	PLL
D29	RF0 PLL Charge Pump Overrange	PLL
D28	RF1 PLL Charge Pump Overrange	PLL
D27	Clock PLL Overrange Error	PLL
D26	SERDES PLL unlock	PLL
D25	Tx7 PA Protection – Slew Rate Limiter	Transmitter
D24	Tx7 PA Protection – Average OR Peak Power	Transmitter
D23	Tx6 PA Protection – Slew Rate Limiter	Transmitter
D22	Tx6 PA Protection – Average OR Peak Power	Transmitter
D21	Tx5 PA Protection – Slew Rate Limiter	Transmitter
D20	Tx5 PA Protection – Average OR Peak Power	Transmitter
D19	Tx4 PA Protection – Slew Rate Limiter	Transmitter
D18	Tx4 PA Protection – Average OR Peak Power	Transmitter
D17	Tx3 PA Protection – Slew Rate Limiter	Transmitter
D16	Tx3 PA Protection – Average OR Peak Power	Transmitter
D15	Tx2 PA Protection – Slew Rate Limiter	Transmitter
D14	Tx2 PA Protection – Average OR Peak Power	Transmitter
D13	Tx1 PA Protection – Slew Rate Limiter	Transmitter
D12	Tx1 PA Protection – Average OR Peak Power	Transmitter
D11	Tx0 PA Protection – Slew Rate Limiter	Transmitter
D10	Tx0 PA Protection – Average OR Peak Power	Transmitter
D9	ARM0 Force GP Interrupt	ARM
D8	ARM0 Watchdog Timer Timeout	ARM

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

Table 90. GP_INTERRUPT Bitmask Description Upper Word (Continued)

Bit Position	Brief Description	Component
D7	ARM0 Calibration Error	ARM
D6	ARM0 System Error	ARM
D5	ARM0 Memory ECC Error	ARM
D4	ARM1 Force GP Interrupt	ARM
D3	ARM1 Watchdog Timer Timeout	ARM
D2	ARM1 Calibration Error	ARM
D1	ARM1 System Error	ARM
D0	ARM1 Memory ECC Error	ARM

GP INTERRUPT API FUNCTIONS

Table 91. GP Interrupt API

API Method Name	Comments
adi_ADRV903x_GpIntPinMaskCfgGet()	Retrieves the General Purpose (GP) Interrupt Pin Mask Config for GP Int pins: GP_INT0, GP_INT1, or both
adi_ADRV903x_GpIntPinMaskCfgSet()	Sets the General Purpose (GP) Interrupt Pin Mask Config for GP Int pins: GP_INT0, GP_INT1, or both
adi_ADRV903x_GpIntStatusClear()	Clears the General Purpose (GP) Interrupt status bits selected in the gpIntClear word.
adi_ADRV903x_GpIntStatusGet()	Reads the General Purpose (GP) Interrupt status bits and can be used to determine what caused a GP Interrupt pin to be asserted
adi_ADRV903x_GpIntStickyBitMaskGet()	Gets the General Purpose (GP) Interrupt Type (pulse/edge (sticky) vs level (non-sticky) triggered) for all interrupt sources
adi_ADRV903x_GpIntStickyBitMaskSet()	Sets the General Purpose (GP) Interrupt Type (pulse/edge (sticky) vs level (non-sticky) triggered) for all interrupt sources

How to Use GP_INT

The setup and usage for the GP_INT command is as follows:

1. Initialize device (can optionally set GP_INTERRUPT masks in .json file)
2. Setup the GP_INTERRUPT masks for GP_INT1 and/or GP_INT0 using the API. GP_INT pins should be low, indicating that no interrupt source has asserted.
3. Decide if you want certain Interrupt sources to be sticky or non-sticky using the GpIntStickyBitMaskSet() API. If the bits are made sticky you will have to manually clear the error after reading the status.
4. Operate device. The BBIC should monitor the GP_INT1 and/or GP_INT0 for rising edges indicating an interrupt has occurred.
5. If the GP_INT1 or GP_INT0 pins assert, call the GpIntStatusGet() API to determine which interrupt has caused the pin to go high.
6. Perform recovery action.
7. Call GpIntStatusClear() to clear any sticky interrupts.

The BBIC should monitor the status of the GP_INT1/GP_INT0 pins after configuring the mask bits. If either pin asserts, this indicates that the ADRV903x has run into a problem that may require user intervention to resolve. The bits in the status register can be made sticky, and the pin will just follow the status register.

It is recommended to keep the PCLK Interrupt bits and the Charge Pump OverRange bits masked. Also, if only using a single internal LO, then keep the unused PLL Unlock bit masked off also to prevent this from asserting the GP_INT pin unessarily.

JTAG BOUNDARY SCAN

The ADRV903x supports JTAG boundary scan standards IEEE 1149.1.

Boundary scan mode is enabled by pulling the TEST_EN pin (R14) high and setting the GPIO[2:0] pins. See Table 92 below for the available modes.

Table 92. Boundary Scan Modes

TEST_EN	GPIO[2:0]	Boundary Scan Mode
0	XXX	No Boundary Scan
1	000	Boundary Scan LVDS Mode
1	001	Boundary Scan CMOS Mode

Note that the TEST_EN does not latch the GPIO levels. The boundary scan modes are set by combinational logic, so it doesn't matter if TEST_EN goes high first, and then GPIOs toggle, or if GPIOs go first and then TEST_EN goes high.

When the ADRV903x is in JTAG, the following pins are used for JTAG access.

Table 93. JTAG Pin Interface

JTAG Pin	ADRV903x Pin	Description
TCK	GPIO_7	Test Clock.
TMS	GPIO_6	Test Mode State. Sampled at the rising edge of TCK to determine the next state.
TDI	GPIO_5	Test Data In. Sampled at the rising edge of TCK.
TDO	GPIO_4	Test Data Out. Valid on the falling edge of TCK.
TRST	GPIO_3	Test Reset. An option pin which can rest the TAP controller's state machine.

THERMAL CONSIDERATIONS

In the ADRV903x data sheet the thermal resistance is provided using the table below.

Table 94. Thermal Resistance Values from Datasheet

Package Type	θ_{JA} (°C/W)	θ_{JCtop} (°C/W)	θ_{JB} (°C/W)	Ψ_{JT} (°C/W)	Ψ_{JB} (°C/W)
BP-506-1	11.98	0.78	2.98	0.69	2.82

This section describes how each item in Table 94 is defined based on JEDEC specs.

- θ_{JA} is the junction to ambient thermal resistance
- θ_{JCtop} is the junction to case thermal resistance
- θ_{JB} is the junction to board thermal resistance
- Ψ_{JT} is the junction to top thermal characterization number
- Ψ_{JB} is the junction to board thermal characterization number
- T_J is the maximum junction temperature (°C)
- T_{Amb} is the ambient temperature (°C)
- T_{Board} is the board temperature measured on the mid-point of the longest side of the package no more than 1mm from the edge of the package body (°C)
- T_{Top} is the temperature measured at the top-center of the package (°C)
- P_{Flow} is the portion of chip power that flows from junction to case (%)
- P_{Diss} is the total power dissipation in the chip (W)

θ_{JA} Junction to Ambient Thermal Resistance: The convection thermal resistance is the chip junction-to-ambient air thermal resistance in a 1ft cube JEDEC environment & PCB. It is the ability of a device to dissipate heat from the surface of the die to the ambient without using external heat sinks.

$$\theta_{JA} = \frac{T_J - T_{Amb}}{P_{Diss}} \quad (23)$$

θ_{JCtop} Junction to Case Thermal Resistance: The JEDEC conduction thermal resistance from junction-to-case. It is the ability of a device to dissipate heat from the surface of the die to the top or bottom surface of the package when an external heat sink is attached to the package.

$$\theta_{JCtop} = \frac{T_J - T_{Top}}{P_{Flow}} \quad (24)$$

θ_{JB} Junction to Board Thermal Resistance: The junction-to-board thermal resistance where T_{Board} is the temperature measured on or near component lead (within 1mm of package body). θ_{JB} is simulated based on JEDEC defined environment and PCB.

$$\theta_{JB} = \frac{T_J - T_{Board}}{P_{Flow}} \quad (25)$$

Ψ_{JT} Junction to Top Thermal Characterization Numbers: Simulated in the same environment as θ_{JA}. Ψ_{JT} is a characterization parameter between junction and package top.

$$\Psi_{JT} = \frac{T_J - T_{Top}}{P_{Diss}} \quad (26)$$

Ψ_{JB} Junction to Board Thermal Characterization Number: Ψ_{JB} is simulated in the same environment as θ_{JA}. Ψ_{JB} is the characterization parameter between junction and board.

$$\Psi_{JB} = \frac{T_J - T_{Board}}{P_{Diss}} \quad (27)$$

From the definitions about the thermal resistance, each of them has the specific environment that compliance to JEDEC definition, and for each case the unique heat dissipation path is presumed.

As stated in JEDEC51-12, Ψ_{JT} and Ψ_{JB} should only be used when no heat sink/heat spreader is present. When a heat sink/heat spreader is added, estimating/calculating junction temperature can be achieved by using appropriate θ_{JCtop} and θ_{JB}.

THERMAL CONSIDERATIONS

DELPHI COMPACT MODEL

Each thermal resistance is defined as one resistance between two nodes and also with unique boundary condition defined in JEDEC. It can be used for the thermal estimation if the boundary is the same as the definition in the JEDEC. However, the thermal environment on the board-level or system-level is more complex in general, and the heat fluxes over a wide spectrum of environments, such as PCB, heatsink or some convection cooling. A DELPHI compact model is created from the detailed model using a step-by-step simulation and statistical optimization process over a wide spectrum of environments. It has high degree of boundary condition independence (BCI) and is also computationally efficient which make it the most suitable for board-level or system-level simulation involving a large number of packages in which accurate temperature and heat flux data is needed.

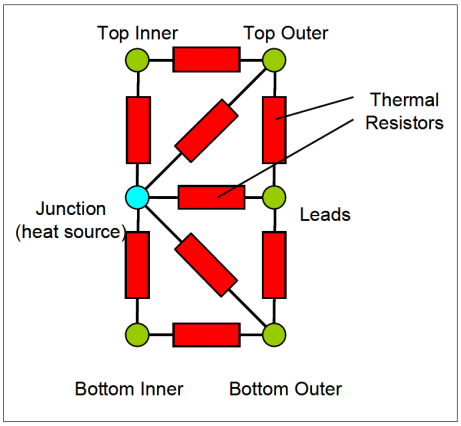


Figure 143. Network Compact Model

The DELPHI thermal resistance network is comprised of a limited number of nodes connected to each other by thermal resistor links (see Figure 143). In effect, the complex 3D heat flow within a real package is represented by a series of links.

Network nodes are, by definition, each associated with a single temperature only. The nodes can be either surface or internal. Surface nodes are associated with a physical region of the package surface defining the area of the node. In such a case, the node temperature represents the average temperature of the area allocated to the node in the actual package. Also, surface nodes must always have a direct one-to-one association with the corresponding physical areas on the actual package. Therefore, it is critical that they communicate with the environment in the same manner as the package.

Internal nodes lie within the package body and may or may not correspond to a physical region within the package. The predicted node temperature has no physical meaning for those internal nodes that do not correspond to actual regions within a package.

Surface nodes communicate with internal nodes as well as the surrounding environment. Internal nodes do not communicate with the environment directly, but they may have a heat source associated with them.

DELPHI PDML model is provided by ADI for the ADRV903x that is compatible with Simcenter FloTHERM, but only valid for steady state thermal simulation. The model includes the actual power dissipation for the die, the user should check with ADI before changing the power values in the DELPHI model.

MAXIMUM JUNCTION TEMPERATURE

As stated in ADRV903x data sheet the maximum junction temperature for continuous operation is 110°C. All the thermal models provided by ADI can be used to do a thermal simulation based on the customers' board level boundary. The thermal resistance obtained from the simulation can be used for the calculation of the package top temperature with the exactly the same boundary in the simulation. The junction temperature obtained from the simulation should be compliant to the data sheet spec (110°C for the continuous operation) to ensure the performance and lifetime specified in the data sheet are guaranteed.

THERMAL API FUNCTIONS

Table 95. List of Thermal API Functions

API Method Name	Comments
adi_ADRV903x_TemperatureGet()	Get the temperature from up to 12 sensors on die.

THERMAL CONSIDERATIONS

Table 95. List of Thermal API Functions (Continued)

API Method Name	Comments
adi_ADRV903x_TemperatureEnableGet()	Get the temperature sensors that are enabled.
adi_ADRV903x_TemperatureEnableSet()	Set the temperature sensors that are to be enabled.

POWER MANAGEMENT CONSIDERATIONS

The ADRV903x family of devices requires four different power supply domains.

1. 0.8 V Digital: this supply is connected to the device through the six VDIG_0P8 pins. This is the supply that feeds all digital processing and clock generation. Care should be taken to properly isolate this supply from all analog signals on the PCB to avoid digital noise coupling to sensitive signals. This supply input can have a tolerance of ±5%, but note that the total tolerance must include the tolerance of the supply device added to the voltage drop of the PCB. This supply is a high-current input, so it is strongly recommended that all six inputs be connected to a common power plane to minimize mismatch and I × R drop. Another recommended technique to maintain the correct supply voltage is to use a power supply with a remote sense connected directly to the ADRV903x supply input pins.
2. 1.0 V Analog: these supplies are collectively referred to as the VANA_1P0 supply. Each input should be treated as a noise-susceptible input, meaning proper decoupling and isolation techniques should be followed to avoid crosstalk between channels. The tolerance on these supply inputs is ±2.5%. This power domain is further divided into two different sub-domains. The static domain includes the supply inputs that maintain a steady current supply during all modes of operation. The dynamic domain are described as those supply inputs that experience current load steps when operating in TDD mode. The individual supply pins and their domain designations are listed in Table 96.
3. 1.8 V Analog: these supplies are collectively referred to as the VANA_1P8 supply. Each input should be treated as a noise-susceptible input, meaning proper decoupling and isolation techniques should be followed to avoid crosstalk between channels. The tolerance on these supply inputs is ±5%. This power domain is further divided into two different sub-domains. The static domain includes the supply inputs that maintain a steady current supply during all modes of operation. The dynamic domain are described as those supply inputs that experience current load steps when operating in TDD mode. The individual supply pins and their domain designations are listed in Table 96.
4. Interface Supply: The VIF_1P8 supply is a separate power domain shared with the BBIC interface. The nominal input voltage on this supply is 1.8 V with a tolerance of ±5%. This input serves as the voltage reference for the digital interface (SPI), GPIO, and digital control inputs.

POWER SUPPLY DOMAIN CONNECTIONS

Table 96 lists the pin number, the pin name, and a brief description of the block it powers in the ADRV903x. Recommendations reflect guidelines used in the design of the customer evaluation board decoupling layout. In general, it is recommended that designers determine current requirements for their desired use case and select PCB routing trace sizes that are appropriate to minimize resistive losses in the PCB power supply traces.

Table 96. Power Supply Pins and Functions

Pin Name	Pins	Type	Domain	Voltage	Recommended Routing/Notes	Description
VDIG_0P8	K12, L12, M12, N12, P12, R12	Digital	N/A	0.8 V	Route as a single power plane or sub-plane to minimize resistance. If using a power supply with remote sense, connect directly to the ADRV903x pins.	Digital core supply
VCLKSYN_1P0	U16	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Clock synthesizer supply
VCLKGEN_1P0	V16	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Clock generator supply
VRXLO0_1P0	A6	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	LO supply
VRXLO1_1P0	A17	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	LO supply
VTXLO0_1P0	A8	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs	LO supply

POWER MANAGEMENT CONSIDERATIONS

Table 96. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain	Voltage	Recommended Routing/Notes	Description
					with a ferrite bead if necessary. Very sensitive to aggressors.	
VTXLO1_1P0	A15	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	LO supply
VLO0_1P0	A10	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	LO supply
VLO1_1P0	A13	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	LO supply
VSYN0_1P0	E8	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Synthesizer supply
VSYN1_1P0	E15	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Synthesizer supply
VSERSYN_1P0	AA11	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Synthesizer supply
VDEV_1P0	F10	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Device clock supply
VSER_1P0	AB11, AC11	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	JESD serializer supply
VDES_1P0	AB12, AC12	Analog	Static	1.0 V	Star connect from the 1.0 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	JESD deserializer supply
VCONV0_1P0	H7	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	ADC/DAC supply
VCONV1_1P0	R7	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	ADC/DAC supply
VCONV2_1P0	H16	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as	ADC/DAC supply

POWER MANAGEMENT CONSIDERATIONS

Table 96. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain	Voltage	Recommended Routing/Notes	Description
					much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	
VCONV3_1P0	R16	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	ADC/DAC supply
VORX0_1P0	N7	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	ORX0 channel ADC supply
VORX1_1P0	N16	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	ORX1 channel ADC supply
VSCLK0_1P0	L7, M7	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Decoupling capacitors should be connected directly between L7/M7 and L8/M8.	ORX0 channel ADC sample clock supply
VSCLK1_1P0	L16, M16	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary. Decoupling capacitors should be connected directly between L16/M16 and L15/M15.	ORX1 channel ADC sample clock supply
VBB0_1P0	C7	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	Baseband supply
VBB1_1P0	C16	Analog	Dynamic	1.0 V	Star connect from the 1.0 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.0 V inputs with a ferrite bead if necessary.	Baseband supply
VANA0_1P8	C6	Analog	Static	1.8 V	Star connect from the 1.8 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Analog 1.8 V supply
VANA1_1P8	C17	Analog	Static	1.8 V	Star connect from the 1.8 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Analog 1.8 V supply
VCLKVCO_1P8	V15	Analog	Static	1.8 V	Star connect from the 1.8 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Input supply for clock VCO LDO
VSYS_1P8	F13	Analog	Static	1.8 V	Star connect from the 1.8 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	SYSREF clock supply

POWER MANAGEMENT CONSIDERATIONS

Table 96. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain	Voltage	Recommended Routing/Notes	Description
VVCO0_1P8	A3	Analog	Static	1.8 V	Star connect from the 1.8 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Input supply for VCO LDO
VVCO1_1P8	A20	Analog	Static	1.8 V	Star connect from the 1.8 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Input supply for VCO LDO
VSERVCO_1P8	V9	Analog	Static	1.8 V	Star connect from the 1.8 V static domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	Input supply for SERDES VCO LDO
VCONV0_1P8	J7	Analog	Dynamic	1.8 V	Star connect from the 1.8 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	ADC/DAC 1.8 V supply
VCONV1_1P8	P7	Analog	Dynamic	1.8 V	Star connect from the 1.8 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	ADC/DAC 1.8 V supply
VCONV2_1P8	J16	Analog	Dynamic	1.8 V	Star connect from the 1.8 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	ADC/DAC 1.8 V supply
VCONV3_1P8	P16	Analog	Dynamic	1.8 V	Star connect from the 1.8 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	ADC/DAC 1.8 V supply
VORX0_1P8	K7	Analog	Dynamic	1.8 V	Star connect from the 1.8 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	ORX ADC 1.8 V supply
VORX1_1P8	K16	Analog	Dynamic	1.8 V	Star connect from the 1.8 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	ORX ADC 1.8 V supply
VTX0_1P8	W3	Analog	Dynamic	1.8 V	Star connect from the 1.8 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	TX upconverter supply
VTX1_1P8	W20	Analog	Dynamic	1.8 V	Star connect from the 1.8 V dynamic domain. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	TX upconverter supply
VIF_1P8	U12	Digital	N/A	1.8 V	Connect to the power supply used by the baseband processor interface circuits to ensure it is common. This supply input is not susceptible to noise, so special routing techniques are not required.	SPI, GPIO, and control signal supply
VCLKVCO_1P0	V14	Analog	N/A	1.0 V	Connect a 4.7 µF capacitor between this pin and VSSA (X7R ceramic is recommended).	Internal VCO LDO output capacitor connection pin
VSERVCO_1P0	V8	Analog	N/A	1.0 V	Connect a 4.7 µF capacitor between this pin and VSSA (X7R ceramic is recommended).	Internal VCO LDO output capacitor connection pin

POWER MANAGEMENT CONSIDERATIONS

Table 96. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain	Voltage	Recommended Routing/Notes	Description
VVCO0_1P0	A4	Analog	N/A	1.0 V	Connect a 4.7 µF capacitor between this pin and VSSA (X7R ceramic is recommended).	Internal VCO LDO output capacitor connection pin output
VVCO1_1P0	A19	Analog	N/A	1.0 V	Connect a 4.7 µF capacitor between this pin and VSSA (X7R ceramic is recommended).	Internal VCO LDO output capacitor connection pin

IMPORTANT:

During operation, some supply currents can vary significantly, especially during calibration cycles when multiple blocks can be enabled at the same time. Each supply needs to have adequate capacity to provide the necessary current so that performance criteria over all process and temperature variations are maintained. Add the margins in Table 97 to measured current values at room temperature using nominal silicon and nominal voltage to ensure proper operation under all conditions.

Table 97. Power Supply Margins

Supply	Margin
0.8 V	15 % + 900 mA
1.0 V	20 %
1.8 V	10 %

POWER SUPPLY SEQUENCE

The device requires a specific power-up sequence to avoid undesirable power-up currents. In the optimal sequence, the VDIG_0P8 supply should come up first. After the VDIG_0P8 source is enabled, the VANA_1P0 supplies should be enabled next, followed by the VANA_1P8 supplies. Note that the VIF_1P8 supply can be enabled at any time without affecting the other circuits in the device. In addition to this sequence, it is also recommended to toggle the $\overline{\text{RESET}}$ signal after power has stabilized prior to initializing the device.

The power-down sequence recommendation is similar to power-up. All supplies should be disabled in reverse order (or all together) before VDIG_0P8 is disabled. If such a sequence is not possible, then all supplies should have their sources disabled simultaneously to ensure no back feeding to circuits that have been powered down.

POWER SUPPLY ARCHITECTURE

The diagram in Figure 144 outlines the power supply inputs on the ADRV903x. This configuration follows the recommendations outlined in Table 96 for groupings of static and dynamic supply inputs. This diagram includes the ferrite beads in series with each analog supply to provide RF isolation from other blocks that could couple through the supply lines. It is recommended that each input be evaluated separately to determine if a ferrite bead is required or if it can be replaced by a short (PCB trace or a 0 Ω resistor).

POWER MANAGEMENT CONSIDERATIONS

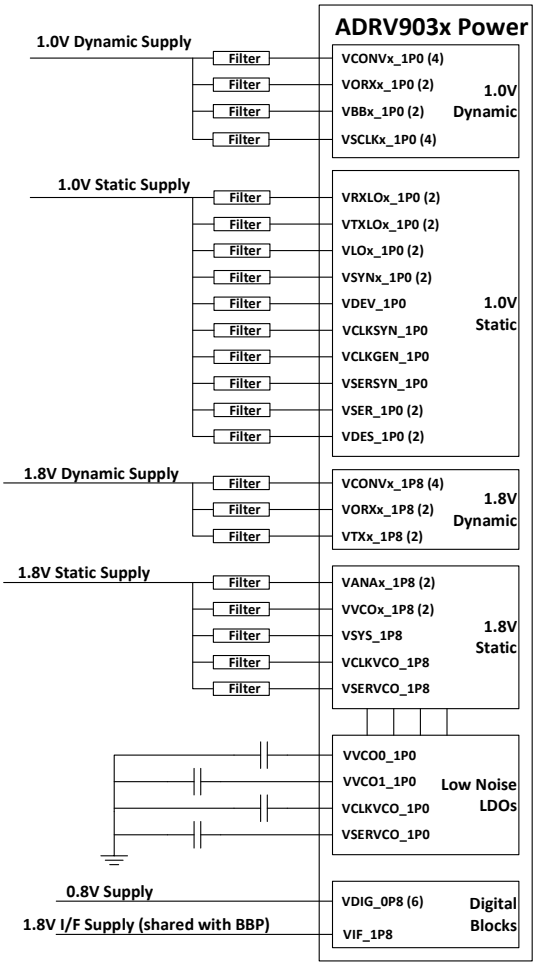


Figure 144. ADRV903x Power Supply Input Distribution

RBIAS SETUP

All internal currents and voltages are based off a controller reference. There are two separate controller references – one for each half of the device. The controller current value is set by resistors connected to the RBIAS0 and RBIAS1 pins (U6 and U17). These resistors should be 4.99 kΩ with a tolerance of 0.1% to ensure that the correct reference is used for internally generated currents and voltages. Care should also be taken in the PCB layout to ensure the trace from each resistor to the corresponding RBIAS pin is properly shielded from noise that could couple into the internal reference generator circuits and degrade performance.

POWER SAVING MODES

There are several power saving modes available on the ADRV903x device which have varying trade-offs of power consumption saved vs. the recovery time required to bring the radio back into an active mode. These are described below in order of increasing power consumption saved but with associated longer recovery times.

DTx Mode

Discontinuous transmission (DTX) is a power saving mode which quickly powers down some components of the ADRV903x when there is no data to transmit.

DTX mode will gate off the digital clocks from TX datapath, ramp down the TX Atten block and pause the TX QEC and TX LOL tracking calibrations. There are three distinct modes supported by the DTX function, Auto, SPI, or Pin modes. In Auto mode, DTX looks for a

POWER MANAGEMENT CONSIDERATIONS

programmable number of consecutive zero samples in the Tx datapath and when detected will deactivate the blocks to save power. In SPI mode, DTx is triggered by the DtxForceSet() API and in PIN mode DTx is triggered by a GPIO going high. This is configured using the DtxCfgSet() API. The number of consecutive zeros to trigger DTx in Auto mode is also configured using this API. Recovery from this mode takes ~1−1.5 µs.

Table 98. DTx Modes

DTx Mode Enumerator	Operation
ADI_ADRV903x_DTXMODE_DISABLE	DTX is in Disable Mode
ADI_ADRV903x_DTXMODE_AUTO	DTX is in Automatic Mode
ADI_ADRV903x_DTXMODE_SPI	DTX is in SPI Control Mode
ADI_ADRV903x_DTXMODE_PIN	DTX is in PIN Control Mode

Channel Standby Mode

The user can save power, for example in TDD mode, by using the Radio Control feature to quickly powerdown and up individual or groups of RF channels. It is possible to put all RF channels in standby by disabling all active channels either using RadioControl PIN Mode or SPI Mode as described in the System Control section. In this mode, most of the RF and analog blocks are powered down, the converters are put in standby and the digital clocks are halted. Recovery from this mode takes in the order of ~2 µs.

Chip Standby Mode

Chip Standby Mode is implemented on the ADRV903x device and saves more power than the DTX and channel standby mode but with a longer recovery back to active mode. In this mode, most of the blocks inside the ADRV903x are powered off, including JESD blocks (framers, deframers, and Serdes lanes), PLLs for clock and LO, and the ARM is put into a lower power mode and clocked off the DEVCLK directly. The tradeoff with powering off more blocks is the recovery time with more time needed to turn those blocks back on including re-running calibrations and also re-running of multi-chip synchronization, which is also lost in standby mode. This mode is designed to fit the scenario where the network traffic load is extremely low, for example as a night time power saving mode. In this case the radio could be put into chip standby for a couple of hours and the environment, especially the ambient temperature, may change significantly during the standby time, which requires some of the calibrations to be re-run on exit of chip standby. The API commands to put the transceiver into chip standby mode and a procedure to exit it and recover the performance is given below.

The user calls the adi_ADRV903x_StandbyEnter(…) to enter the chip standby mode. The StandbyEnter() API function will put the radio control mode into SPI mode, and the active channels before entering standby mode are stored in the data structure adi_ADRV903x_StandbyRecover_t, for the recovery afterwards when adi_ADRV903x_StandbyExit(…) function is called. In addition the software saves off information on the active tracking calibrations. It is important to note that the radio control mode won't automatically switch to PIN mode even it was enabled before entering Chip standby, so the user needs to call the API - adi_ADRV903x_RadioCtrlCfgSet(…), to change the radio control mode if PIN mode is needed again.

The basic Software Procedure for entering and exiting Chip Standby mode is:

- Enter Chip Standby Mode
  - This stage powers down the specific circuitries inside Palau.
  - The user calls adi_ADRV903x_StandbyEnter(…) function to enter the chip standby mode.
- Recovery procedure when exiting from chip Standby Mode
  - Re-enable the blocks which were disabled by the chip standby by calling adi_ADRV903x_StandbyRecover(…).
  - MCS (Multi-Chip Synchronization) is needed because the clock PLL was powered off and the synchronization was lost in standby mode, so the user can call app_program_Phase7Mcs (shown in the src/app/example folder of the API package) which is for the bootup as well to synchronize the clocks.
  - RX ADC and ORX ADC Init Cal are required for the performance recovery. However, the user doesn't need to change the data structure adi_ADRV903x_PostMcsInit_t or calMask. The API call to adi_ADRV903x_PostMcsInit in this procedure will assign the specific calMask to adi_ADRV903x_InitCalsRun. What the user needs to do is to call app_program_Phase8PostMcsInit.
  - Bring up JESD by calling app_program_Phase9DataInterfaceBringup.
  - Call adi_ADRV903x_StandbyExit(…) function which enables the active TX/RX/ORX channels and tracking calibrations which are stored in the data structure adi_ADRV903x_StandbyRecover_t. In parallel, the software puts TX LB ADC into fast attack mode.

POWER MANAGEMENT CONSIDERATIONS

- Check the status of the LB ADC fast attack status by calling API adi_ADRV903x_StandbyExitStatusGet(…). Note, you should not send any data to the transmitter before the fast attack has completed.
- Turn radio control to PIN mode if needed by calling API adi_ADRV903x_RadioCtrlCfgSet(…)
- Start the traffic/send UserData. TX QEC and RX QEC require the traffic data to converge to the reasonable performance which may take 1~2 seconds.

It is possible to reduce the power consumption compared to Active Mode by ~80 to 90% when using Chip Standby mode. Some example measurements are given below. These are given for guidance only. Please check the power saving numbers when using your specific UseCase.

Table 99. Example power saving when using Chip Standby Mode

Profile	Active Mode Power	Chip Standby Mode Power	Chip Standby Entry time (note1)	Chip Standby Recovery time (note2)
UC59	10.7 W	1.3 W	0.2 sec	9.1 sec
UC78	12.1 W	1.6 W	0.37 sec	10.9 sec

Note 1: the standby entry time may vary from run to tun because the entering API needs to wait for the tracking calibration to finish the current iteration.

Note 2: the recovery time doesn't include the QEC convergence time which as mentioned above needs the traffic data to converge and usually takes about 1 second.

Chip Power Down Using RESETB

You can powerdown all components on the ADRV903x by simply asserting the RESETB pin low. In addition to powering down all analog and digital blocks and associated calibration information, this will also reset the ARM and associated memory, meaning all programmed firmware is lost. After de-asserting RESETB, a full chip re-program is required as described in Programming the Device section. This mode will achieve the highest power saving at the expense of longest recovery time. The exact recovery time depends on the UseCase but typically is > 10s.

POWER SAVING MODES API FUNCTIONS

Table 100. List of Power Saving Modes Related API Functions

API Method Name	Comments
adi_ADRV903x_DtxCfgGet()	Reads the Discontinuous Transmit (DTX) settings for the requested Tx channel
adi_ADRV903x_DtxCfgSet()	Configures the Discontinuons Transmit (DTX) mode. You can configure DTx to be in Automatic mode, where the DTx will trigger on detection of a programmable number of zeros in the Tx datapath, or a GPIO triggered mode or SPI triggered mode.
adi_ADRV903x_DtxForceSet()	If you configure DTX to be in SPI mode, you can set or unset DTX using the dtxForce bit for the requested Tx channel
adi_ADRV903x_DtxGpioCfgSet()	If you configure the DTX to be in GPIO mode, you need to configure the GPIO which will trigger DTx per Tx channel. If the GPIO is already being used by another resource the API will return an error.
adi_ADRV903x_DtxGpioCfgGet()	If you configure the DTX to be in GPIO mode, you can readback the GPIO you have configured to trigger DTx per channel
adi_ADRV903x_DtxStatusGet()	Reads the Discontinuous Transmit (DTX) status for the requested Tx channel
adi_ADRV903x_StandbyEnter()	Call this API when entering into Chip Standby mode. You need to pass a pointer to the Standby Recover data structure which will store information on which channels are enabled, what serdes lanes are used, which cals are enabled etc. so these can be re-applied on exit of Chip Standby.
adi_ADRV903x_StandbyRecover()	Call this API when you want to recover or come out of Chip Standby mode. You need to pass a pointer to the Standby Recover data structure which was saved off when entering Chip Standby.
adi_ADRV903x_StandbyExit()	This API runs the Chip Standby Exit Sequence, which enables the channels and runs the fast attack cals before clearing the Standby state. It must be called after StandbyRecover and after MCS/JESD link bring up.
adi_ADRV903x_StandbyExitStatusGet()	You can poll the status of the Chip Standby Exit procedure to check when it has completed. Note, you should not send any data to the transmitter before the StandbyExit has completed.

RF PORT IMPEDANCE MATCHING

This section describes the recommended RF transmitter and receiver interfaces to obtain optimal device performance. Some reference is also provided regarding board layout techniques and balun selection guidelines.

The ADRV903x a highly integrated transceiver with transmit, receive and observation receive signal chains. External impedance matching networks are required on transmitter and receiver ports to achieve performance levels indicated on the data sheet. Analog Devices Inc. recommends the utilization of simulation tools in the design and optimization of impedance matching networks. To achieve best correlation from simulation to PCB, accurate models of the board environment, SMD components (for example, baluns, and filters), and device port impedances are required.

RF PORT IMPEDANCE DATA

This section provides the port impedance data for all transmitters and receivers in the device. Note the following:

- Z_o is defined as 100 Ω for Tx/Rx/ORx (Differential).
- The reference plane for this data is the device ball pads.
- Single ended mode port impedance data is not available. However, a rough assessment is possible by taking the differential mode port impedance data and dividing both the real and imaginary components by 2.
- Contact Analog Devices Applications Engineering for the impedance data in Touchstone format.

ADS SETUP USING DATA ACCESS COMPONENT AND SEDZ FILE

The port impedances are supplied as an *.s1p Series Equivalent Differential Z (impedance) file. This format allows simple interface to ADS by using the Data Access Component. The Pi network on the single ended side and the double differential Pi configuration on the differential side allow maximum flexibility in designing matching circuits and is suggested for all design layouts as it can step the impedance up or down as needed with appropriate component selection.

Operation is as follows:

1. 1. The DAC block reads the rf port *.s1p file. This is the device rf port reflection coefficient.
2. 2. The two equations convert the RF port reflection coefficient to a complex impedance. The end result is the RX_SEDZ variable.
3. 3. The RF port calculated complex impedance (RX_SEDZ) is utilized to define the impedance.

For highest accuracy, use EM modeling results of the PCB artwork and S parameters of the matching components and balun in the simulations.

TRANSMITTER BIAS AND PORT INTERFACE

This section considers the dc biasing of the transmitter (Tx) outputs and how to interface to each Tx port. The transmitters operate over a range of frequencies. The Tx outputs are dc biased to a 1.8 V supply voltage using either RF chokes (wire-wound inductors) or a transformer center tap connection.

Careful design of the dc bias network is required to ensure optimal RF performance levels. When designing the dc bias network, select components with low dc resistance (R_{DCR}) to minimize the voltage drop across the series parasitic resistance element with either of the suggested dc bias schemes suggested in Figure 145. The red resistors (R_DCR) indicate the parasitic elements. As the impedance of the parasitics increase, the voltage drop (ΔV) across the parasitic element increases, causing the transmitter RF performance (for example, $P_{O,1dB}$ and $P_{O,MAX}$) to degrade. Select the choke inductance (L_C) high enough relative to the load impedance such that it does not degrade the output power.

The recommended dc bias network is shown in Figure 146. This network has fewer parasitics and fewer total components.

RF PORT IMPEDANCE MATCHING

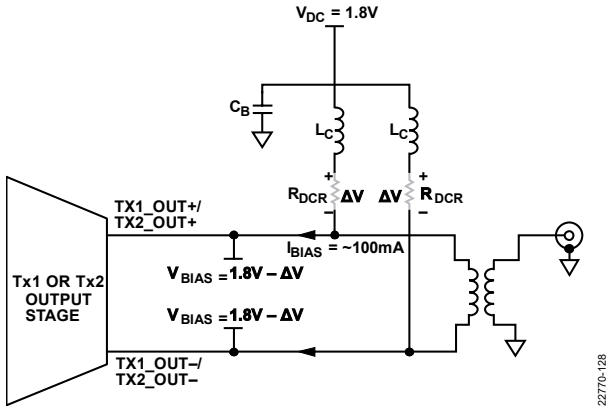


Figure 145. RF DC Bias Configurations Depicting Parasitic Losses Due to Wire Wound Chokes

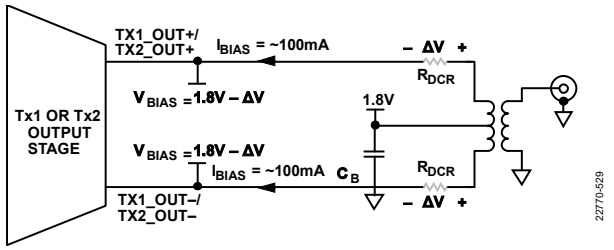


Figure 146. RF DC Bias Configurations Depicting Parasitic Losses Due to Center Tapped Transformers

Figure 147 to Figure 150 identify four basic differential transmitter output configurations. Impedance matching networks (balun single-ended port) are most likely required to achieve optimum device performance from the device. Also, the transmitter outputs must be ac-coupled in most applications due to the dc bias voltage applied to the differential output lines of the transmitter.

The recommended RF transmitter interface featuring a center tapped balun is shown in Figure 147. This configuration offers the lowest component count of the options presented.

Brief descriptions of the Tx port interface schemes are provided as follows:

- Center tapped transformer passes the bias voltage directly to the transmitter outputs
- RF chokes are used to bias the differential transmitter output lines. Additional coupling capacitors (C_C) are added in the creation of a transmission line balun
- RF chokes are used to bias the differential transmitter output lines and connect into a transformer
- RF chokes are used to bias the differential output lines that are ac-coupled into the input of a driver amplifier.

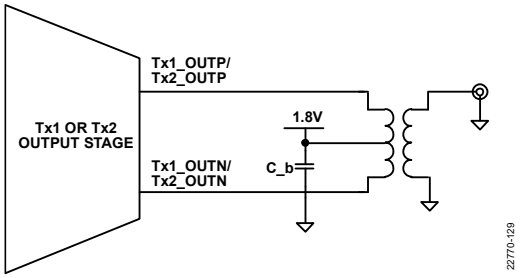


Figure 147. ADRV903x RF Transmitter Interface Configuration A

RF PORT IMPEDANCE MATCHING

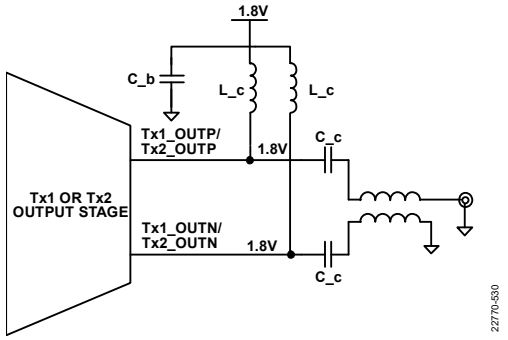


Figure 148. ADRV903x RF Transmitter Interface Configuration B

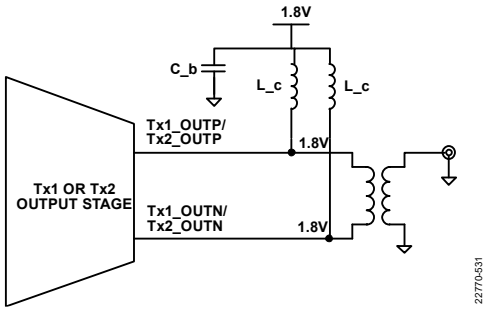


Figure 149. ADRV903x RF Transmitter Interface Configuration C

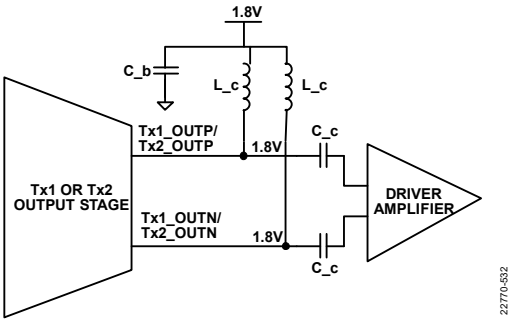


Figure 150. ADRV903x RF Transmitter Interface Configuration D

If a Tx balun is selected that requires a set of external dc bias chokes, careful planning is required. It is necessary to find the optimum compromise between the choke physical size, choke dc resistance (R_{DCR}) and the balun low frequency insertion loss. In commercially available dc bias chokes, resistance decreases as size increases. However, as choke inductance increases, resistance increases. Therefore, it is undesirable to use physically small chokes with high inductance as they exhibit the greatest resistance. For example, the voltage drop of a 500 nH, 0603 choke at 100 mA is roughly 50 mV. The maximum current here is below 100 mA.

Table 101. Sample Wire-Wound DC Bias Choke Resistance vs. Size

Inductance (nH)	Resistance (Size: 0603)	Resistance (Size: 1206)
100	0.10	0.08
200	0.15	0.10
300	0.16	0.12
400	0.28	0.14
500	0.45	0.15
600	0.52	0.20

RF PORT IMPEDANCE MATCHING

GENERAL RECEIVER PATH INTERFACE

The device has two types of receivers. These receivers include eight main receive pathways (Rx0, Rx1, Rx2, Rx3, Rx4, Rx5, Rx6, and Rx7) and two observation receivers (ORx0, ORx1). The Rx and ORx channels are designed for differential use only.

The receivers support a wide range of operation frequencies. In the case of the Rx and ORx channels, the differential signals interface to an integrated mixer. The mixer input pins have a dc bias of approximately 0.9 V present on them and may need to be ac-coupled depending on the common mode voltage level of the external circuit.

Important considerations for the receiver port interface are as follows:

- Device to be interfaced: filter, balun, T/R switch, external LNA, and external PA. Determine if this device represents a short to ground at dc.
- Rx and ORx maximum safe input power is +18 dBm (peak).
- Rx and ORx optimum dc bias voltage is 0.9 V bias to ground.
- Board Design: reference planes, transmission lines, and impedance matching.

Figure 151 shows possible differential receiver port interface circuits. The options in Figure 151 and Figure 152 are valid for all receiver inputs operating in differential mode, though only the Rx1 signal names are indicated. Impedance matching may be necessary to obtain data sheet performance levels.

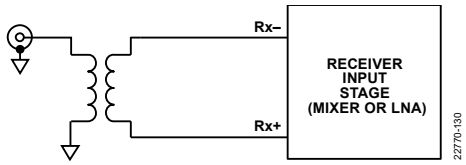


Figure 151. Differential Receiver Input Interface Circuits

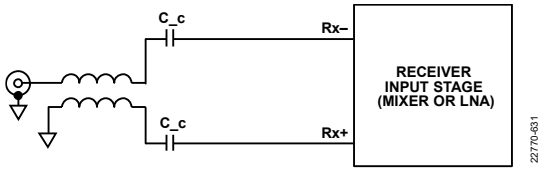


Figure 152. Differential Receiver Input Interface Circuits

Given wide RF bandwidth applications, SMD balun devices function well. Decent loss and differential balance are available in a relatively small (0603, 0805) package.

PCB LAYOUT CONSIDERATIONS

PCB LAYOUT OVERVIEW

The ADRV903xis a highly integrated RF agile transceiver with significant signal conditioning integrated onto one chip. Due to this high level of complexity and its high pin count, careful printed circuit board (PCB) layout is important to obtain optimal performance. This section provides a checklist of issues to look for and general guidelines on how to optimize the PCB to mitigate performance issues. The goal of this document is to help achieve the best performance from the ADRV903x while reducing board layout effort. This section assumes that the reader is an experienced analog/RF engineer who understands RF PCB layout as well as RF and high-speed transmission lines.

The ADRV903x evaluation board represents one of the most complex implementations of the device. All RF inputs and outputs, JESD serial data lanes, and digital control and monitoring signals are implemented in this design. Advanced PCB technology is used to achieve maximum device performance while seeking to maintain a high level of performance in the face of constraints presented by the routing density. Depending on the intended application, users may not require all signals to be routed and can, therefore, use alternate PCB layout techniques to reach their design goals. These include but are not limited to a traditional ball grid array (BGA) fanout, fewer layers, through hole vias only, and lower grade PCB materials.

This section discusses the following issues and provides guidelines for system designers to get the best performance out of the ADRV903x device:

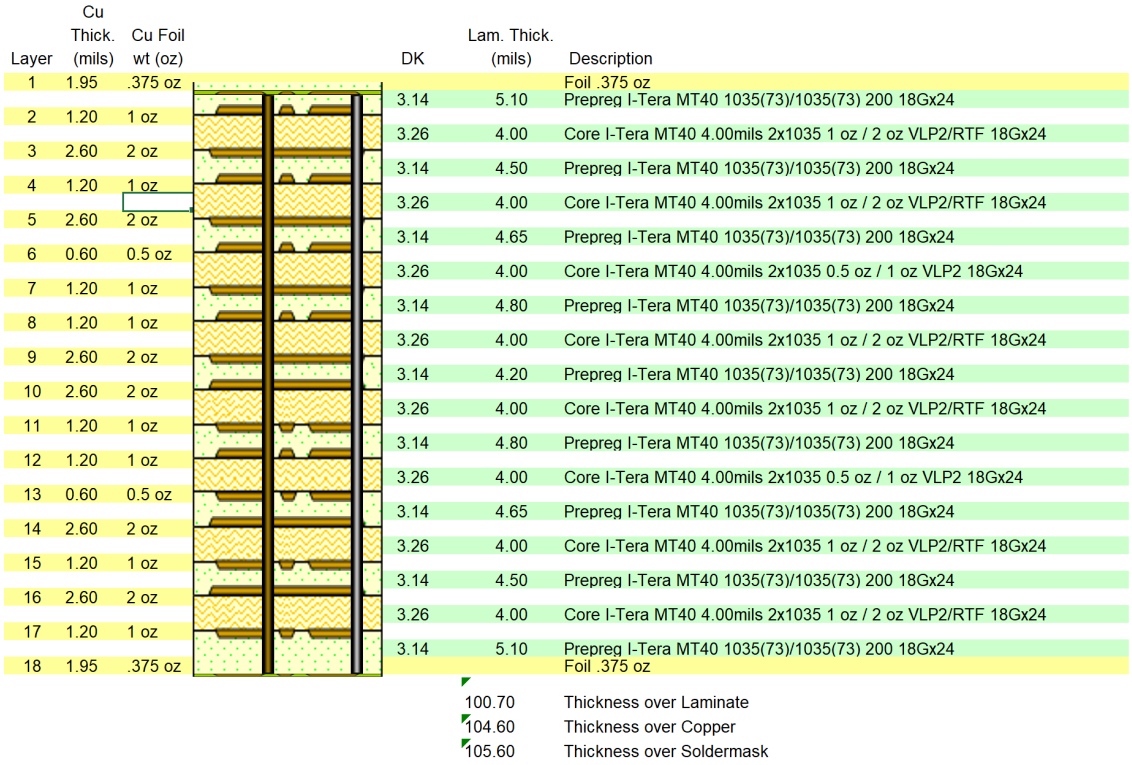
- PCB material and stack up selection
- Fanout and trace space layout guidelines
- Component placement and routing priorities
- RF and JESD transmission line layout
- Isolation techniques used on the ADRV903x customer evaluation board
- Power management routing considerations
- Analog signal routing recommendations
- Digital signal routing recommendations
- Unused pin instructions

PCB MATERIAL AND STACK UP SELECTION

The ADRV903x evaluation board utilizes Isola I-Tera® MT-40 dielectric material. This material was selected for its low loss tangent and low dielectric constant characteristics. It was also highly recommended by our PCB vendor as one of the most reliable materials and easiest to manufacture. On previous evaluation systems, Analog Devices has chosen a combination of low loss, RF capable dielectric for the outer edge layers and standard FR4-370 HR dielectric for interior layers. RF signal routing on these boards was confined to the top and bottom layers. Therefore, the material mix was a good compromise to obtain optimum RF performance and low overall board cost. Given the need to route RF and high speed, digital data lanes on multiple layers due to the increased number of RF channels and JESD lanes, I-Tera material was chosen for all layers on this board. There are several other material options on the market from other PCB material vendors that are also valid options for use with the ADRV903x device. The key comparison metric for these materials is the dielectric constant and the loss tangent. Designers must also be careful to ensure that the thermal characteristics of the material are adequate to handle high reflow temperatures for short durations and expected operating temperatures for extended durations.

Figure 153 shows the PCB stack up used for the ADRV903x evaluation board. Layer 1 and Layer 18 are primarily used for RF IO signal routing, so the prepreg material was selected to support the required controlled impedance traces. Layer 2 and Layer 17 have uninterrupted ground copper flood beneath all RF routes on Layer 1 and Layer 18. Layer 2 is also used in combination with Layers 4, 15, and 17 to route high speed digital JESD lanes. These signal layers use the layers above and below them as reference ground planes. Clean reference planes are important to maintain signal integrity on sensitive RF and high-speed digital signal paths. Layers 3, 4, and 5 are used for the 1.0 V analog power domains and Layer 10 is used for the 1.8 V analog power domains. Layers 14 and 15 are reserved for digital 0.8 V power domain routing to minimize impedance on the supply while keeping it isolated by adjacent ground layers from the analog signals on the PCB. Layers 2, 6, and 9 are solid ground planes designed to help isolate sensitive analog signal and power layers from potentially noisy digital signals routed in the lower half of the PCB. The remaining layers are used to route all power, digital control, GPIO, and clock distribution circuits.

PCB LAYOUT CONSIDERATIONS



Layer	Cu Thick. (mils)	Cu Foil wt (oz)	DK	Lam. Thick. (mils)	Description
1	1.95	.375 oz			Foil .375 oz
			3.14	5.10	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
2	1.20	1 oz			
			3.26	4.00	Core I-Tera MT40 4.00mils 2x1035 1 oz / 2 oz VLP2/RTF 18Gx24
3	2.60	2 oz			
			3.14	4.50	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
4	1.20	1 oz			
			3.26	4.00	Core I-Tera MT40 4.00mils 2x1035 1 oz / 2 oz VLP2/RTF 18Gx24
5	2.60	2 oz			
			3.14	4.65	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
6	0.60	0.5 oz			
			3.26	4.00	Core I-Tera MT40 4.00mils 2x1035 0.5 oz / 1 oz VLP2 18Gx24
7	1.20	1 oz			
			3.14	4.80	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
8	1.20	1 oz			
			3.26	4.00	Core I-Tera MT40 4.00mils 2x1035 1 oz / 2 oz VLP2/RTF 18Gx24
9	2.60	2 oz			
			3.14	4.20	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
10	2.60	2 oz			
			3.26	4.00	Core I-Tera MT40 4.00mils 2x1035 1 oz / 2 oz VLP2/RTF 18Gx24
11	1.20	1 oz			
			3.14	4.80	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
12	1.20	1 oz			
			3.26	4.00	Core I-Tera MT40 4.00mils 2x1035 0.5 oz / 1 oz VLP2 18Gx24
13	0.60	0.5 oz			
			3.14	4.65	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
14	2.60	2 oz			
			3.26	4.00	Core I-Tera MT40 4.00mils 2x1035 1 oz / 2 oz VLP2/RTF 18Gx24
15	1.20	1 oz			
			3.14	4.50	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
16	2.60	2 oz			
			3.26	4.00	Core I-Tera MT40 4.00mils 2x1035 1 oz / 2 oz VLP2/RTF 18Gx24
17	1.20	1 oz			
			3.14	5.10	Prepreg I-Tera MT40 1035(73)/1035(73) 200 18Gx24
18	1.95	.375 oz			Foil .375 oz
				100.70	Thickness over Laminate
				104.60	Thickness over Copper
				105.60	Thickness over Soldermask

Figure 153. PCB Material Stack Up Diagram

Table 102 describes the drill table for via structures used in the evaluation board to route all signals from the transceiver. Note that the metal and dielectric thicknesses have been balanced to ensure that the thickness of each half of the PCB is relatively equal to avoid uneven flexing or deforming under pressure or temperature changes.

Via structures selection is based on signal routing requirements and manufacturing constraints. Ground planes are full copper floods with no splits except for vias, through-hole components, and isolation structures. Ground and power planes are all routed to the edge of the PCB with a 10 mil pullback from the edge to decrease the risk of layer to layer shorts at the exposed board edge.

Table 102. Drill Table

Start Layer	End Layer	Drill Type	Plate Type	Via Fill	Drill Depth	Do Not Break Layers
1	18	Mechanical	PTH	Not applicable	100.40	
1	8	Mechanical	Via	Nonconductive via fill	100.40	
18	5	CDD	Back drill	Nonconductive via fill	76.10	4
1	3	CDD	Back drill	Nonconductive via fill	12.20	4
18	17	Laser	Micro Via	CuVF_Button pattern	5.55	
1	2	Laser	Micro Via	Nonconductive via fill	5.55	

Controlled impedance traces, single ended and differential, are required to obtain best RF performance. Impedances of 50 Ω and 100 Ω are required for RF, high speed digital, and clock signals. Table 103 describes details about trace impedance controls used in the ADRV903x evaluation board and types of line structures used to obtain desired impedance and performance on, and for, given layers and impedances.

Table 103. PCB Trace Impedance Table

Layer	Structure Type	Target Impedance	Impedance Tolerance	Target Line Width	Edge Coupled Pitch	Reference Layers	Modeled Line Width	Modeled Impedance	Coplanar Space
1	Edge coupled differential	100.00 Ω	±10 Ω	8.00 mils	14.25 mils	2	8.00 mils	100.64 Ω	10.00 mils
1	Single ended	50.00 Ω	±5 Ω	10.50 mils	0.00 mils	2	10.50 mils	50.36 Ω	10.00 mils
2	Edge coupled differential	100.00 Ω	±10 Ω	4.25 mils	11.00 mils	1 and 3	4.25 mils	99.89 Ω	12.00 mils

PCB LAYOUT CONSIDERATIONS

Table 103. PCB Trace Impedance Table (Continued)

Layer	Structure Type	Target Impedance	Impedance Tolerance	Target Line Width	Edge Coupled Pitch	Reference Layers	Modeled Line Width	Modeled Impedance	Coplanar Space
4	Edge coupled differential	100.00 Ω	±10 Ω	4.25 mils	13.00 mils	3 and 5	4.25 mils	99.89 Ω	12.00 mils
6	Edge coupled differential	100.00 Ω	±10 Ω	4.50 mils	12.00 mils	5 and 7	4.50 mils	99.77 Ω	12.00 mils
8	Edge coupled differential	50.00 Ω	±10 Ω	4.25 mils	11.75 mils	7 and 9	4.25 mils	99.90 Ω	
11	Edge coupled differential	50.00 Ω	±10 Ω	4.25 mils	11.75 mils	10 and 12	4.25 mils	99.90 Ω	
12	Single ended	50.00 Ω	±5 Ω	4.50 mils	0.00 mils	11 and 13	4.50 mils	50.35 Ω	
12	Edge coupled differential	100.00 Ω	±10 Ω	4.50 mils	12.00 mils	11 and 13	4.25 mils	100.22 Ω	
13	Edge coupled differential	100.00 Ω	±10 Ω	4.50 mils	12.00 mils	12 and 14	4.50 mils	99.75 Ω	
13	Single ended	50.00 Ω	±5 Ω	4.75 mils	0.00 mils	12 and 14	4.75 mils	49.83 Ω	
15	Edge coupled differential	100.00 Ω	±10 Ω	4.25 mils	13.00 mils	14 and 16	4.25 mils	99.89 Ω	12.00 mils
17	Edge coupled differential	100.00 Ω	±10 Ω	4.25 mils	11.00 mils	16 and 18	4.25 mils	99.89 Ω	12.00 mils
18	Edge coupled differential	100.00 Ω	±10 Ω	8.00 mils	14.25 mils	17	8.00 mils	100.64 Ω	10.00 mils
18	Single ended	50.00 Ω	±5 Ω	10.50 mils	0.00 mils	17	10.50 mils	50.36 Ω	10.00 mils

FANOUT AND TRACE SPACING GUIDELINES

The ADRV903x uses a 506-ball BGA, 19 mm × 19 mm package. The pitch between the pins is 0.8 mm. This small pitch makes it impractical to route all signals on a single layer. RF and high-speed data pins have been placed on the perimeter rows of the BGA to minimize complexity of routing these critical signals. Via in pad technology is used to route all other signals to inner layers on which they are routed. The recommended via size includes an 8-mil drill hole with a 12-mil capture pad. A combination of micro vias between layers 1 and 2 and between layers 17 and 18, and through vias are used to connect signals between the different PCB layers.

Figure 154 illustrates the fanout of RF differential channels from the device on the right and left sides of the device footprint. The top layer of the PCB is used for this fanout. Note that each signal pair is designed with the required characteristic impedance and isolation to minimize crosstalk between channels. The isolation structures include a series of ground balls around each RF channel and the digital interface section of the device. Connect these ground balls by traces to form a wall around each section, and then fill the area to make the ground as continuous as possible underneath the device. Note that the receiver channels are in the eight ground boxes in the interior of the ball grid array. Figure 155 shows a similar fanout on the bottom layer of the PCB for the receiver channels. These channels are routed on the bottom layer to optimize isolation from each other and from the transmitter and observation receiver channels.

PCB LAYOUT CONSIDERATIONS

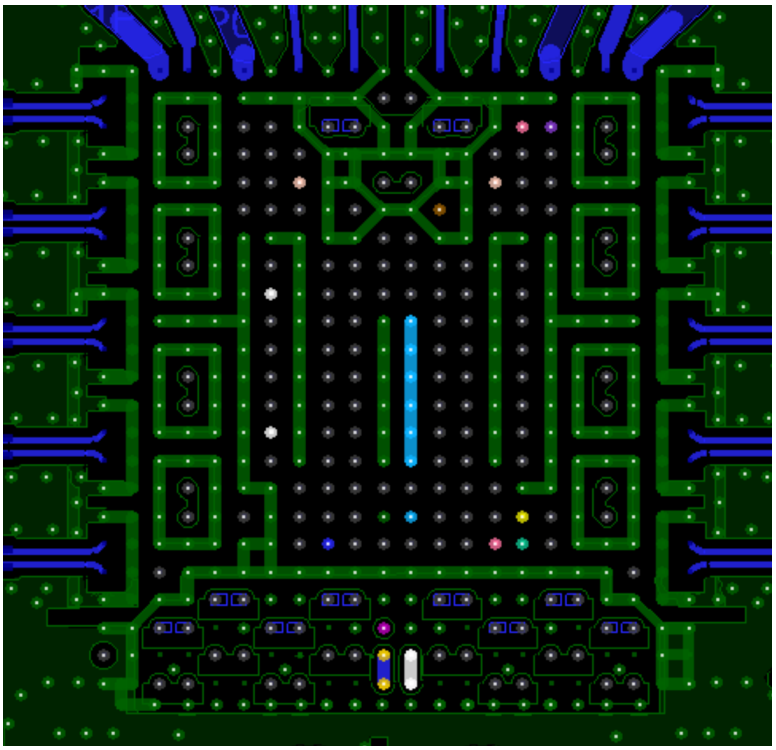


Figure 154. ADRV903x Customer Evaluation Board RF Observation Receiver and Transmitter Fanout and Layout

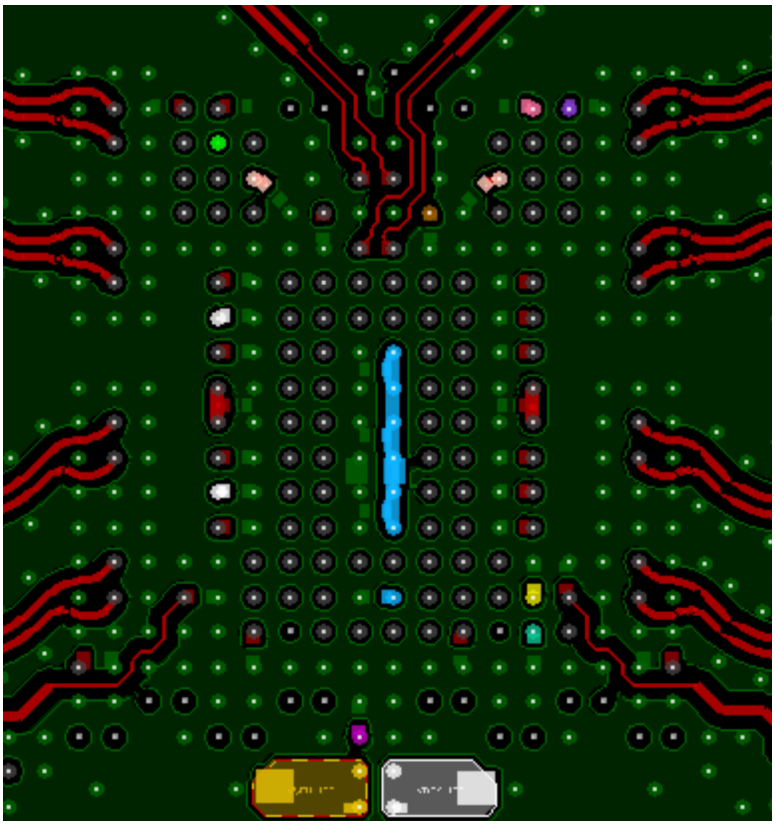


Figure 155. ADRV903x Customer Evaluation Board RF Receiver and Fanout and Layout

PCB LAYOUT CONSIDERATIONS

COMPONENT PLACEMENT AND ROUTING GUIDELINES

The ADRV903x transceiver requires few external components to function. Those that are required must be carefully placed and routed to optimize performance. This section provides a checklist for properly placing and routing some of those critical signals and components.

Signals With Highest Routing Priority

RF inputs and outputs, clocks, and high-speed digital signals are the most critical for optimizing performance and must be routed with the highest priority. Figure 156 shows the general directions in which each of the signals must be routed to effectively isolate them from aggressor signals. Red arrows show the recommended fanout for the RF channels, the purple arrows show the recommended direction for the DEVCLK and SYSREF signals, and the blue arrows show the recommended routing direction for the JESD interface signals. It will be extremely difficult to keep all RF channels on a single PCB layer due to the channel density of this device. In such cases, it is recommended to route the observation receiver and transmitter channels on the top PCB layer with adequate channel-to-channel isolation and the receivers on the bottom layer. Ensure that the trace impedance is properly designed to 100 Ω differential including the vias needed to transfer the signals between PCB layers.

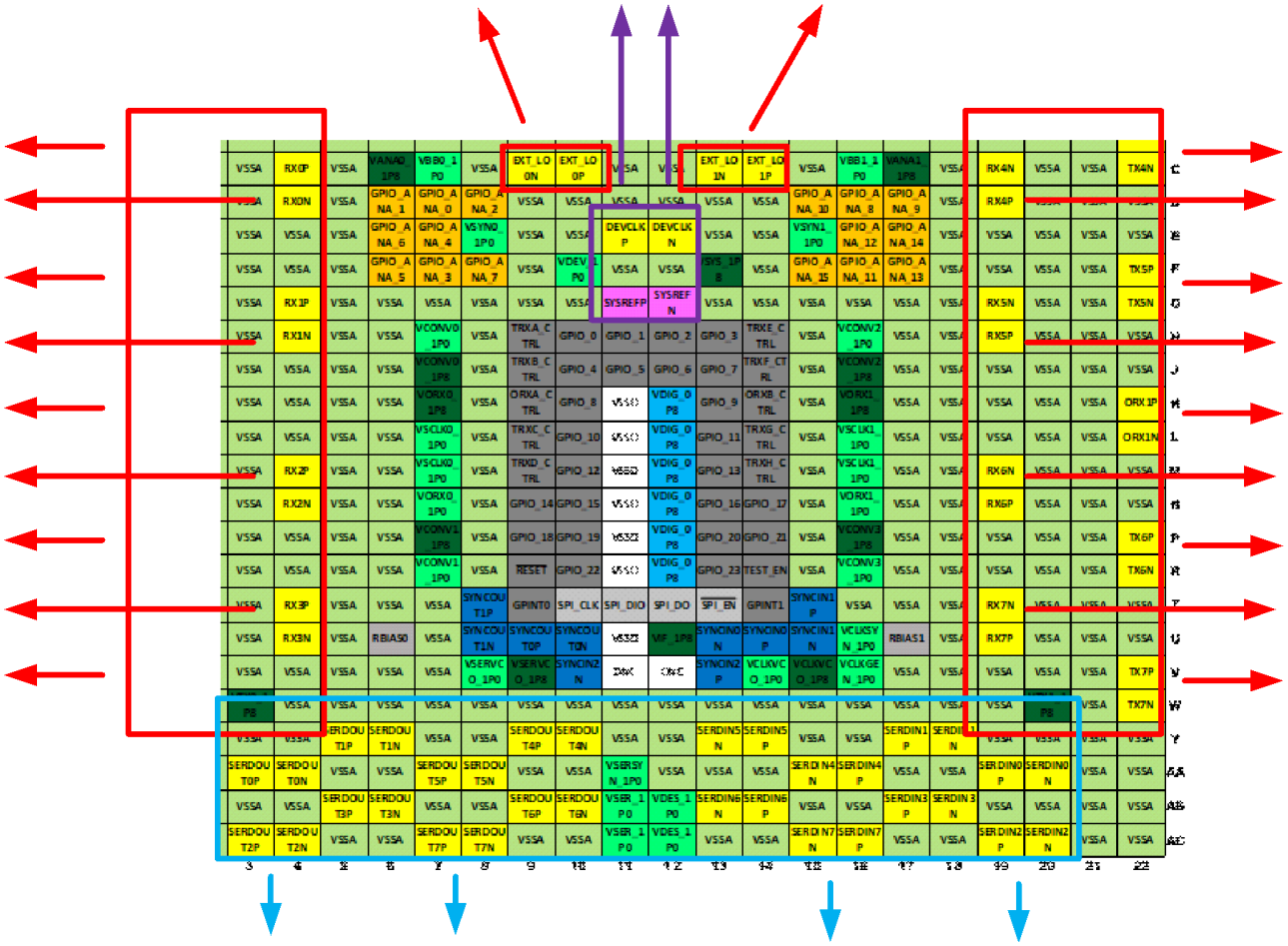


Figure 156. RF Channel, DEVCLK, SYSREF, and JESD Interface Routing Direction Guidelines

Transmitter, receiver, and observation receiver routing (also referred to as trace routing), physical design (trace width/spacing), matching network design, and balun placement significantly impact RF transceiver performance. To avoid performance degradation, optimize path design, component selection, and placement. The RF Routing Guidelines section describes proper matching circuit placement and routing in greater detail. To achieve desired levels of isolation between RF signal paths, use the considerations and techniques described in the Isolation Techniques section in designs.

PCB LAYOUT CONSIDERATIONS

Connect external clock inputs to DEVCLK_P and DEVCLK_N through 0.1 µF ac coupling capacitors. Place a 100 Ω termination across the input near pin E10 and pin E11, as shown in Figure 157. Shield traces by ground planes above and below with vias staggered along the edges of the differential pair routing. This shielding is important because it protects the reference clock inputs from spurious signals that can transfer to different clock domains within the device. The SYSREF differential signal should be routed in the same manner as the DEVCLK signal with the exception that ac coupling capacitors are not needed for this signal.

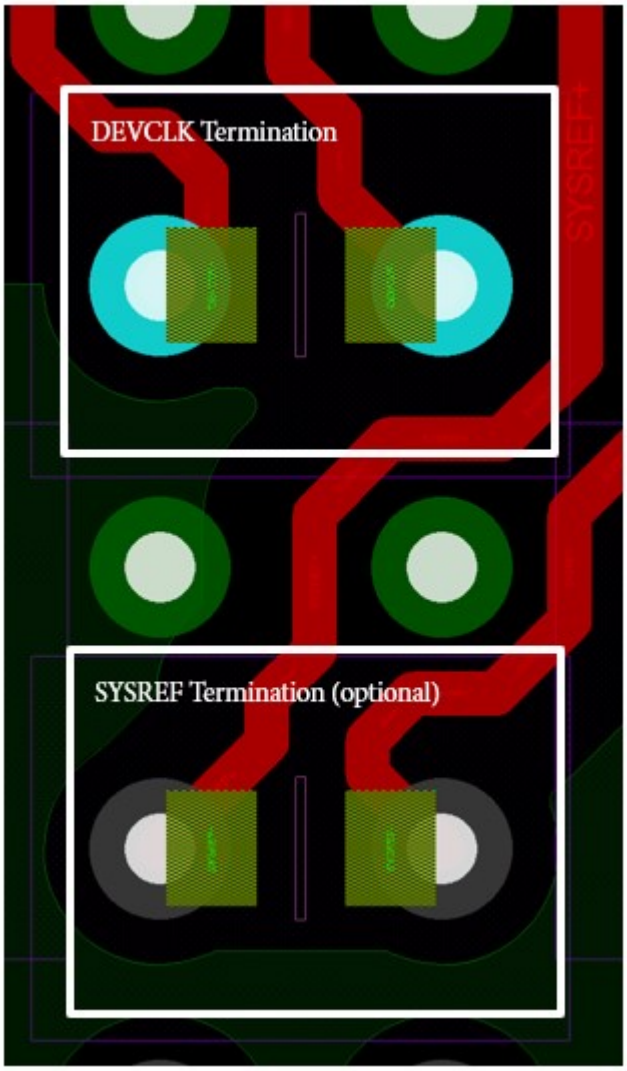


Figure 157. DEVCLK and SYSREF Termination

Route JESD204B high speed digital interface traces at the beginning of the PCB design process with the same priority as the RF signals. The JESD204B/JESD204C Routing Recommendations section outlines launch and routing guidelines for these signals. Provide adequate isolation between interface differential pairs.

Signals With Second Routing Priority

Power supply routing and quality has a direct impact on overall system performance. The Power Management Layout Design section provides recommendations for how to best route power supplies to minimize loss as well as interference between RF channels. Follow recommendations provided in the Power Management Layout Design section to optimize RF and isolation performance.

PCB LAYOUT CONSIDERATIONS

Signals With Lowest Routing Priority

Route remaining low frequency digital general-purpose inputs and outputs (GPIOs) and SPI signals. It is important to route all digital signals bounded between row g and row W and column 7 and column 16 down and away from sensitive analog signals on PCB signal layers (see Figure 156 for the ball diagram). Route these bounded signals using a solid ground layer that shields other sensitive signals from potentially noisy digital signals. Analog GPIO signal traces are routed on layers separated from RF I/O and high speed digital, but still on the analog side of the PCB. These signals are typically used for static control for external RF components that are referenced to the same 1.8 V power rail as the ADRV903x.

RF AND JESD TRANSMISSION LINE LAYOUT

RF Routing Guidelines

The ADRV903X evaluation boards use both surface coplanar waveguide and surface edge coupled coplanar waveguide transmission lines for transmitter, receiver, and observation receiver RF signals. In general, Analog Devices does not recommend using vias to route RF traces unless a direct route on the same layer as the device is not possible.

- Keep balanced lines as short as possible for differential mode signaling between the device and the RF balun.
- Keep the length of the single-ended transmissions lines for RF signals as short as possible.
- Keeping signal paths as short as possible reduces susceptibility to undesired signal coupling and reduces the effects of parasitic capacitance, inductance, and loss on the transfer function of the transmission line and impedance matching network system.

The routing of these signal paths is the most critical factor in optimizing performance and, therefore, must be routed prior to any other signals and maintain the highest priority in the PCB layout process.

All 18 RF ports are impedance matched using pi matching networks, both differential and single ended. In the case of the receivers and transmitters, two stages of differential pi networks are used to allow us to impedance match for wide bandwidths. This makes our evaluation over the entire frequency range of the device easier to accomplish. Typical customer applications will not require two pi network stages for the differential matching network.

Figure 158 shows the routing for the Rx0 receiver on the customer evaluation board. Note that the single-ended portion is kept as short as possible between the connector and the balun. The differential side is split into two sections: a pin network matching circuit near the balun and another pin network near the ADRV903x. If the matching circuit is designed for a single channel bandwidth, it may be possible to eliminate one of the differential pi networks.

If using an ADRV903x variant that includes external local oscillator inputs, these signals should follow the same routing guidelines as the RF receiver inputs to ensure proper isolation is maintained.

PCB LAYOUT CONSIDERATIONS

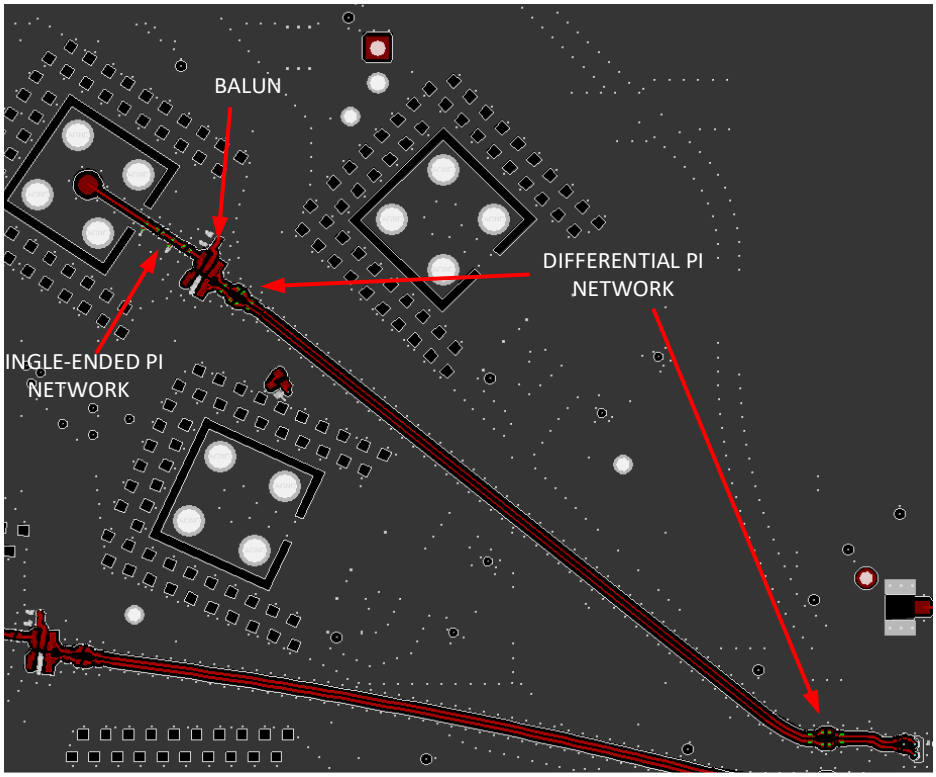


Figure 158. Receiver RF Routing and Matching Network

Figure 159 depicts the path from device to external connector that routes Tx1 on the customer evaluation board. Matching components locations are highlighted to illustrated proper component placement. All the RF signals must have a solid ground reference under each path to maintain the desired impedance. Ensure that none of the critical traces run over a discontinuity in the ground reference.

PCB LAYOUT CONSIDERATIONS

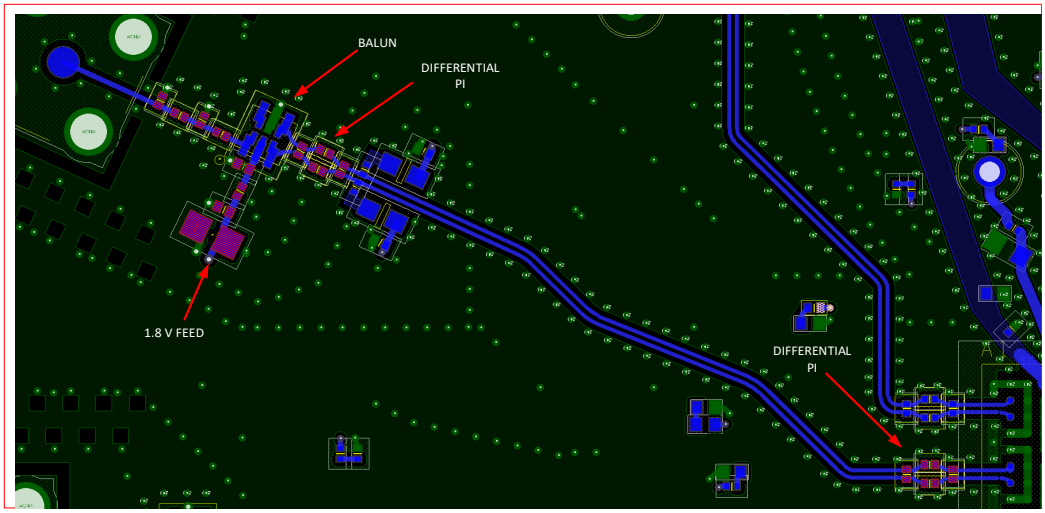


Figure 159. Transmitter RF Routing and Matching Network

Transmitter Bias Supply Guidelines

Each transmitter requires approximately 125 mA supplied through an external connection. Bias voltages are supplied at the dc feed of a center tapped balun in the RF signal path on the ADRV903X customer evaluation board as shown in Figure 160.

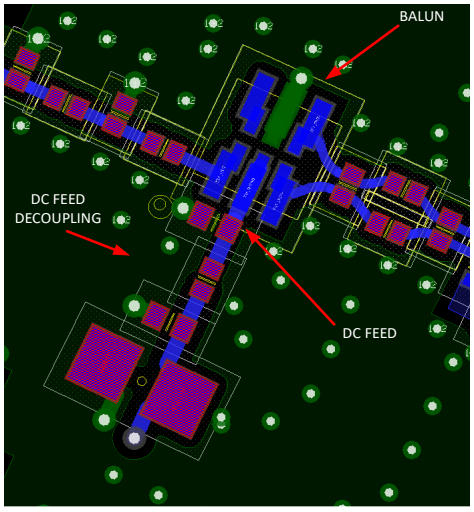


Figure 160. 1.8 V Transmitter Bias Routing at Balun

To reduce switching transients caused by attenuation setting changes, power the balun dc feed directly from the 1.8 V supply plane. Design the geometry of the plane to isolate each transmitter from the others. It is strongly recommended that each transmitter have a power finger that connects to the main 1.8 V supply but isolates this supply from the other transmitters. The finger width is designed to minimize impedance keeping voltage drop due to transmitter current at a minimum.

PCB LAYOUT CONSIDERATIONS

The ADRV903x evaluation board couples the 1.8 V supply into each transmitter via a center tapped balun, but the board is also provisioned for an external choke feed inductor dc supply option. When a balun is selected that does not have a dc feed input, RF choke inductors must be used to supply current to the transmitters. Chokes are connected from the 1.8 V supply to each transmitter output. Note that in this scenario, the transmitter balun must be ac-coupled. The RF chokes must also be decoupled by capacitors from the power feed to ground. Place the ground connections to these capacitors as close as possible to the transmitter output pins. To avoid peaking due to current transients, match both chokes and their layout carefully. Figure 161 shows an example of this arrangement.

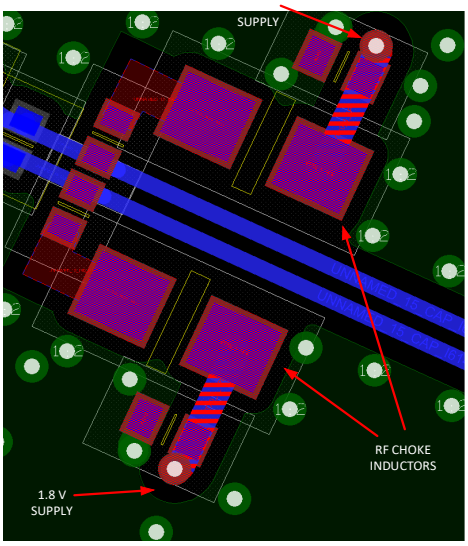


Figure 161. Transmitter Balun RF Choke DC Supply Connection

JESD204B/JESD204C Routing Recommendations

The ADRV903X uses a JESD204B/JESD204C serializer-deserializer (SERDES) high speed serial interface. Keep the differential traces for the SERDES lanes very short by placing the device as close as possible to the baseband processor and routing the traces as directly as possible between the devices. Using a PCB material with a low dielectric constant and loss tangent is also strongly recommended. For a specific application, loss must be modeled to ensure adequate drive strength is available in both the ADRV903x and the baseband processor.

Route the differential pairs on a single plane if possible using a solid ground plane as a reference on the layers directly above and/or below the signal layer. Reference planes for the impedance controlled traces must not be segmented or broken along the entire length of a trace. If routing on a single plane is not possible, try to minimize the number of vias needed and their length to make the connection inductance as low as possible.

All SERDES lane traces must be impedance controlled, targeting 100 Ω differential. Ensure that the pair is loosely coplanar, edge coupled. The ADRV903x customer evaluation board uses 4-mil wide traces and a separation of approximately 10 mil. This sizing varies depending on the stack up and selected dielectric material. Minimize the pad area for all the connector and passive components to reduce parasitic capacitance effects on the transmission lines, which can negatively impact signal integrity. Vias used to route these signals must be minimized as much as possible. Use blind vias wherever possible to eliminate via stub effects and use micro vias to minimize inductance. If using standard vias, use maximum length vias to minimize the stub size. For example, on an 8-layer board, use layer 7 for the stripline pair, thus reducing the stub length of the via to that of the height of a single layer. For each via pair, a pair of ground vias must be placed nearby to minimize the impedance discontinuity.

For SERDES signal traces, the recommendation is to route them on the top side of the board as a 100 Ω differential pair (coplanar edge coupled waveguide). In the case of the ADRV903x customer evaluation board, the SERDES signals are routed on inner layers 2, 4, 15, and 17. Capacitors (100 nF) are places in series near the FMC connector away from the chip to provide ac coupling.

PCB LAYOUT CONSIDERATIONS

Figure 162 and Figure 163 show the transition between ball and launch. Surrounding ground references, above and below the signal layer, are designed to tune the modal impedances ideal for the high speed signaling and according to the JESD204B/JESD204C standards.

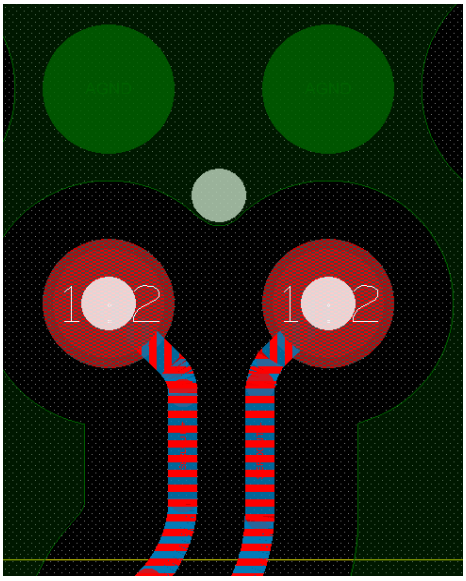


Figure 162. SERDES Signal Launch on Layer 2

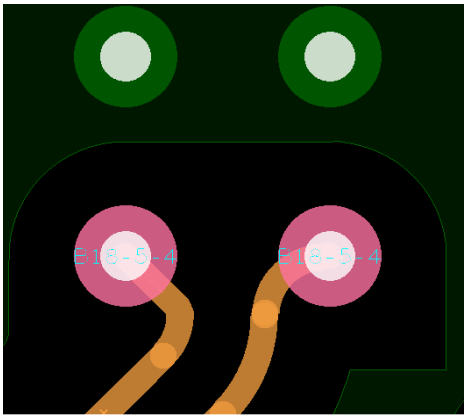


Figure 163. SERDES Signal Launch on Layer 4

ISOLATION TECHNIQUES

Given the density of sensitive and critical signals, significant isolation challenges are faced when designing a PCB for the ADRV903x device. Analytically determining aggressor-to-victim isolation in a system is very complex and involves considering vector combinations of aggressor signals and coupling mechanisms.

Isolation Between RF I/O Ports

The primary coupling mechanisms between RF I/O paths on the evaluation board are:

PCB LAYOUT CONSIDERATIONS

- Magnetic field coupling
- Surface propagation
- Cross domain coupling via ground

To reduce the impact of these coupling mechanisms on the ADRV903x customer evaluation board, several strategies were used, including evenly spaced ground vias, ground cutouts, and specialized routing techniques. A careful designer may notice various bends in the routing of differential paths. These routes were developed and tuned through iterative electromagnetic simulation to minimize magnetic field coupling between differential paths.

Additional shielding is provided by using connecting VSSA balls under the device to form a shield around RF I/O ball pairs. This ground provides a termination for stray electric fields. Ground vias are used along single-ended RF I/O traces. Optimal via spacing is 1/10 of a wavelength for the highest signal frequency, but that spacing can vary somewhat due to practical layout considerations.

$$Wavelength\ (\mathrm{m}) = \frac{300}{frequency\ (\mathrm{MHz}) \times \sqrt{\ _{r}}} \tag{28}$$

These techniques are illustrated in Figure 164.

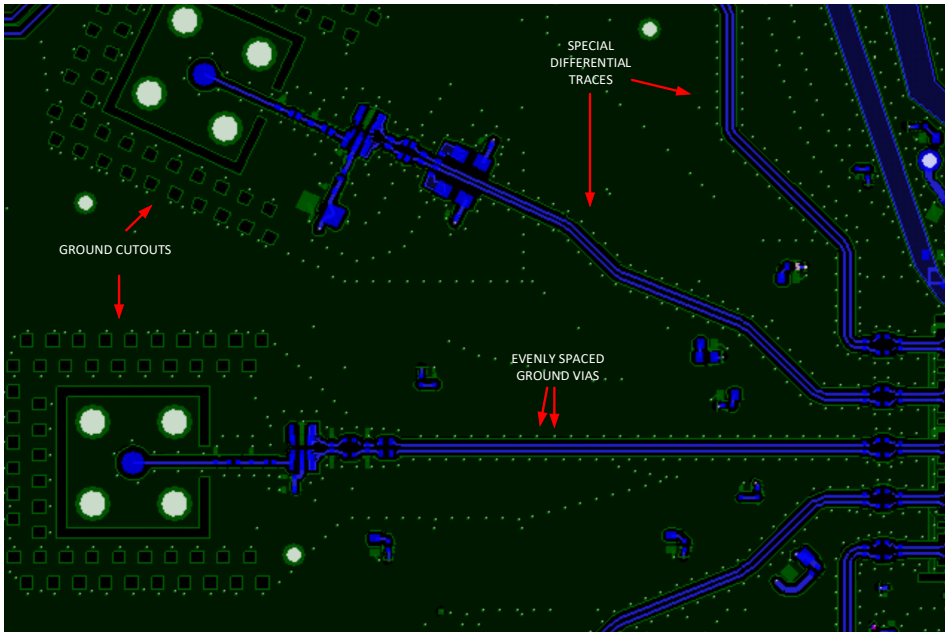


Figure 164. RF I/O Isolation Structures

RF I/O baluns are spaced and aligned to reduce magnetic coupling from the structures in the balun package. Care must also be taken to reduce crosstalk over shared grounds between baluns.

Isolation Between SERDES Lines

The JESD204B/JESD204C interface uses 16 lane pairs that can operate at rates up to 32.44 Gbps. During PCB layout, ensure those lines are routed following the rules described in the JESD204B/JESD204C Routing Recommendations section. To operate at such high data rates, additional routing techniques are needed to minimize crosstalk between lanes. S-shaped routing is used on several long differential pair routes on the ADRV903x evaluation board based on electro-magnetic simulation results that show this to be the best technique for minimizing crosstalk.

PCB LAYOUT CONSIDERATIONS

Figure 165 illustrates these isolation techniques. Ground vias are placed between each pair of traces to provide isolation and decrease crosstalk. Spacing between vias follows the rule provided in the wavelength calculation equation. Ground cutouts are also used to aid in lane isolation. For most accurate spacing of fencing vias, use electro-magnetic simulation software.

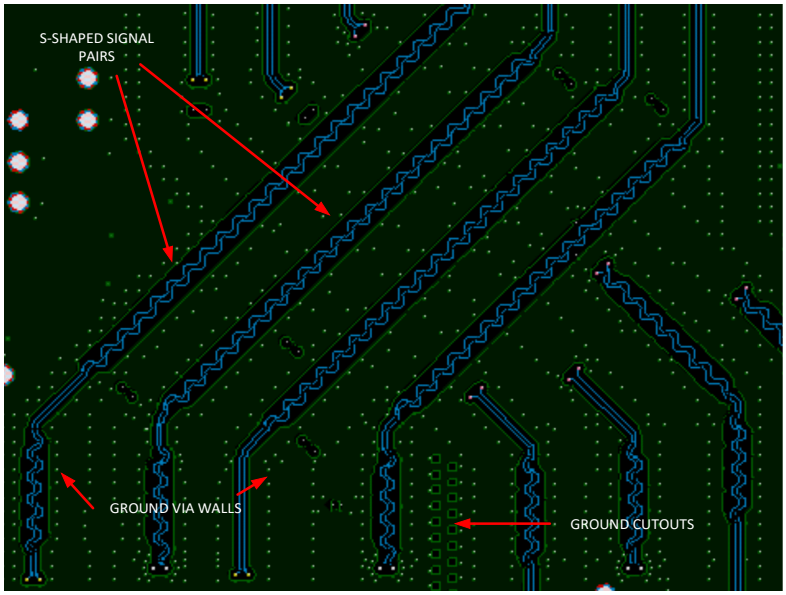


Figure 165. SERDES Lane Routing and Isolation

POWER MANAGEMENT LAYOUT DESIGN

Due to the complexity and high level of integration in the ADRV903x, power supply routing is critical to achieve optimum RF performance. The device is designed with 41 separate power supply input pins that are tied to four power supply rails: 1.8 V (analog, 1.0 V (analog), 0.8 V (digital), and VIF_1P8 (interface supply). The analog supplies are further divided into two supply rails: one that supplies low noise, constant-on functions such as those related to synthesizer and clock generation functions (called static), and one that supplies other functions that are gated by the transmitter or receiver enables during TDD operation (called dynamic). As described in the Power Management Considerations section, the device has four internal regulators that are used to buffer and regulate the supplies for the internal VCOs. All other supply voltages feed directly into the internal circuits from the PCB. This section describes the techniques used on the ADRV903x customer evaluation board to provide isolation between power domains and minimize I × R drops when current loading is highest.

Analog Power Ring Approach

The RF section is designed as two hemispheres with four transmitters, four receivers, and an observation receiver on each side. To reduce coupling between channels and keep each power supply input isolated from the others, a star connection approach is used. This approach involves connecting each power supply input to a common power supply bus, using an isolated trace designed specifically for the current requirements of the particular input. The ADRV903x evaluation board uses a power ring approach to provide the power supply bus for the 1.8 V and 1.0 V analog supplies. Figure 166 through Figure 168 illustrate the 1.0 V power supply rail routing. Note that the 1.0 V static and 1.0 V dynamic supplies are routed on multiple layers. This technique is used to reduce the amount of trace impedance inserted between the supplies and the loads. It also helps to control the directivity of the current so that it is routed outside the area that is RF signal areas, which also assists in maintaining supply isolation. Figure 169 illustrates similar supply trace routing for the 1.8 V static and dynamic supplies. Note that the 1.8 V supply currents are not as large as the 1.0 V currents, so only one layer is used for these supplies.

PCB LAYOUT CONSIDERATIONS

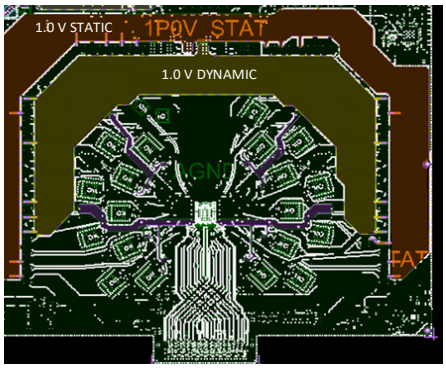


Figure 166. 1.0 V Analog Power Ring Layout Approach – Layer 3

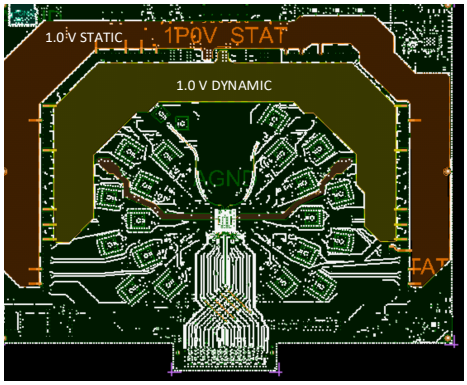


Figure 167. 1.0 V Analog Power Ring Layout Approach – Layer 4

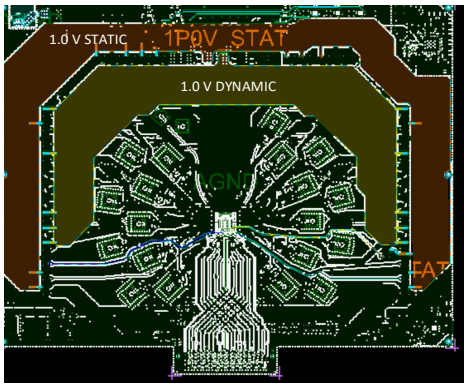


Figure 168. 1.0 V Analog Power Ring Layout Approach – Layer 5

PCB LAYOUT CONSIDERATIONS

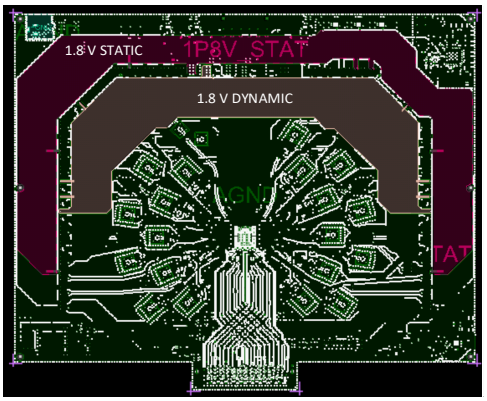


Figure 169. 1.8 V Analog Power Ring Layout Approach – Layer 10

Analog Power Star Connections

The analog power ring approach provides ample locations for creating the individual star connections. This approach enables the designer to control the current paths for each supply, as well as design individual traces that better control the effect of voltage drops on other circuits when large load current changes occur. Each individual power supply input is evaluated for its maximum current consumption value, and the star connection trace is then designed to minimize the voltage drop for that particular supply input while still providing isolation from the other inputs. Figure 170 illustrates how these star connections are made to the individual supply balls of the device. This figure shows two of the highest current-load supply inputs on the 1.0 V static rail. The thickness of the traces are determined based on the current load for each star connection and the metal thickness of the layer. Note that the traces are routed below the RF connectors to minimize interference with the RF signals.

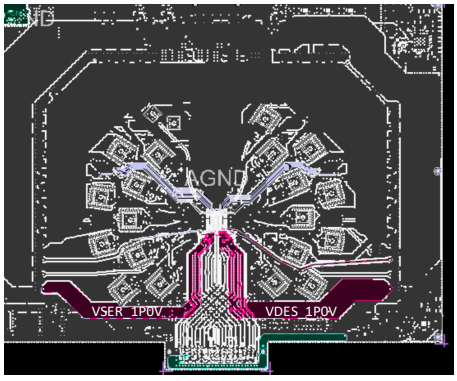


Figure 170. 1.8 V Supply Routing Using Star Connections

For the sensitive supplies connected to the static supply rails, all these input balls are on the top row of the ball grid. It is recommended that the supply star connections for these inputs be routed on the same layer as the ADRV903x to avoid the inductance of via connections and to isolate these noise sensitive supply inputs for other signals. For these supply inputs, the 0.1 µF bypass capacitors can be placed as close to the ADRV903x as possible.

Digital Power Routing (VDIG_0P8)

The digital 0.8 V supply is the noisiest supply in the system, so it is important to keep this supply shielded from the other supplies. It is also the highest current supply, so the thickness of the traces needs to be adequate to carry the load current to the device without experiencing significant voltage drops. There are six digital power input pins to the device to help distribute the current and minimize losses due to the ball

PCB LAYOUT CONSIDERATIONS

impedance. Figure 171 illustrates the approach used on the customer evaluation board to supply this current. A digital power channel is routed from the power supply to the device and the entire area is flooded with copper to provide a low resistance supply trace. This channel is shielded on all sides including several ground layers above and below the flooded area so that it is isolated from other signals.

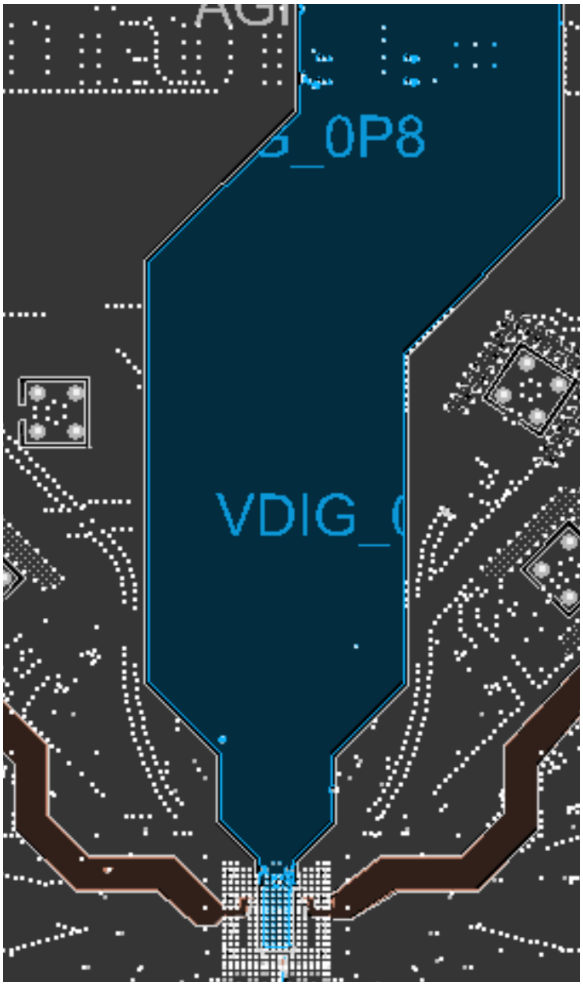


Figure 171. Digital Supply Routing

Figure 172 shows a zoomed-in view of the connection to the device. Note that all six of the input balls are connected directly to the flooded area to reduce the trace resistance. A cutout area is included around the device clock input pin vias to provide additional isolation between the digital supply and the reference clock input. Note that the digital supply is bounded on both sides by rows 8 and 15 of the ball grid array. These are the analog boundaries that separate the digital block of the device from the RF channels. It is important to maintain these boundaries when routing signals to avoid coupling digital noise into the radio channels.

PCB LAYOUT CONSIDERATIONS

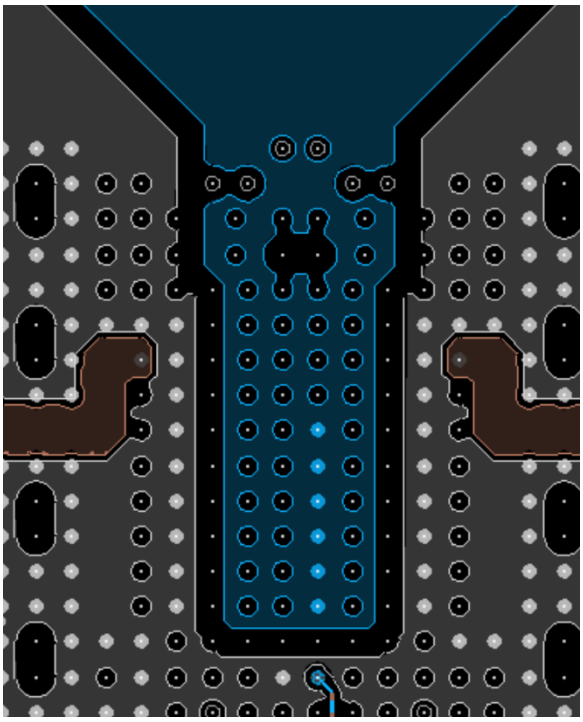


Figure 172. Digital Supply Connection to SIX VDIG_0P8V Input Pins

Interface Supply Input (VIF_1P8)

The interface supply (VIF_1P8) is a low current input that provides the supply reference for the SPI serial interface. This supply input can be routed as a signal trace with adequate thickness to minimize voltage drop when the device is active. Route this trace in the digital area (bottom rows of the device) and keep it isolated from other signals to ensure it is not corrupted by other active digital signals or by the JESD interface lanes.

Ground Returns

Another critical routing consideration is how to control the mixing of ground currents to avoid noise coupling between different power domains. One way to keep domains separated is to provide different ground return planes for each supply domain. This approach can complicate a dense PCB layout like that required for the ADRV903x. Another option is to connect all ground to the same plane system and use cutouts and channeling like those used in the RF sections to provide better channel to channel isolation. Creating such ground channels can provide the benefit of steering ground currents in a desired path without the complexity of trying to keep ground planes isolated from each other. The specifics of such designs are dependent on the PCB layout and the level of isolation desired.

Input Bypass Component Placement

There are subtle component placement techniques for placing power supply bypass components that can have a substantial impact on radio performance. Follow these guidelines when placing components on power supply inputs:

- Each power supply pin requires, at a minimum, a 0.1 µF bypass capacitor near the pin. For inputs that require a large current step, a 10 µF capacitor in parallel is recommended. Place the ground side of the bypass capacitor(s) so that ground currents flow away from other power pins and their bypass capacitors.
- Route power supply input traces to the bypass capacitor and connect the capacitor(s) as close to the supply pin as possible through a via to the component side of the PCB. It is recommended that the via be located inside the power supply pin pad to minimize trace inductance.
- Some power supplies require a ferrite bead in series with the supply line to prevent RF noise from coupling between different inputs, whereas others can do without the extra protection. It is recommended that each line be connected with a series component, either a ferrite bead or a 0 Ω place holder. Ensure that the device is sized properly to handle the current load for the particular power supply input of concern.

PCB LAYOUT CONSIDERATIONS

- Figure 173 illustrates how the 0.1 µF bypass capacitors should be connected to the device. This image shows each supply pin connected to the bottom of the PCB using an in-pad via. One of the supplies uses two input pins (in this case, VSCLK0_1P0). For input using two input pins, it is acceptable to connect them together and to a single bypass capacitor. The capacitors are 0201-size devices that fit directly between the via to the supply ball and the via to an adjacent ground return ball. It is important to follow this procedure to minimize noise coupling between supplies, especially for the supplies that are tied to the observation receivers (VORX0_1P0, VORX1_1P0, VSCLK0_1P0, and VSCLK1_1P0).
- If larger capacitors are used to provide low frequency decoupling and a larger startup current capability, they can be placed further away from the input pins without affecting overall performance.

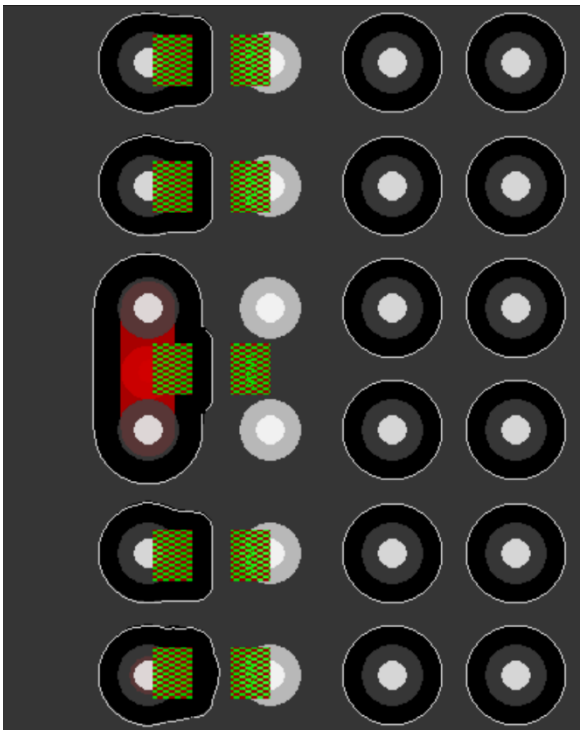


Figure 173. Power Supply Decoupling Capacitor Example

DIGITAL SIGNAL ROUTING CONSIDERATIONS

The digital signal routing (for example, SPI bus, enable controls, and GPIO) is the least sensitive area relative to routing for signal integrity. It is very important to isolate these signals from analog and power supply signals to avoid digital noise coupling into other circuits. On the evaluation board, these signals are routed from the bottom of the board up through the area where the SERDES signals are routed on layers 11, layer 12, and layer 13. Digital I/O signals use VIF_1P8 as their reference supply, so this technique keeps the ground return common with the reference supply. Most of these signals are static or infrequently change state, so once signals are routed out of the device, they can be fanned out to other areas of the PCB without concern of interfering with radio functions.

ANALOG GPIO SIGNAL ROUTING CONSIDERATIONS

The analog GPIO signals are available for providing control or monitoring or other analog ICs used in the same design as the ADRV903x. These signals use the 1.8 V analog supply as their reference, making them better suited for connecting to other devices such as power amplifiers that require coordinated control. These signals are typically static or infrequently change state, so routing for isolation is not as critical as for other analog signals. Care should be taken, however, to keep these traces away from digital signal routing to avoid coupling digital noise into the 1.8 V analog power supply inputs.

RBIAS ROUTING CONSIDERATIONS

There are two RBIAS current setting inputs on this device – one for each hemisphere of the device. Each ball must have a 4.99 kΩ, 0.1% resistor connected between it and ground. It is recommended that these components be placed on the same layer as the ADRV903x device

PCB LAYOUT CONSIDERATIONS

and the traces be routed as short as possible on either the same PCB layer or a nearby layer with proper shielding around them to avoid any noise coupling into the bias network.

UNUSED PIN INSTRUCTIONS

In some applications, the user may decide not to use all available inputs or outputs. In these cases, take care to follow the recommendations listed in Table 104 for unused pins.

Table 104. Recommendations for Unused Pins

Pin No.	Type	Mnemonic	When pins are not used:
B1, C1, B22, C22, F1, G1, F22, G22, P1, R1, P22, R22, V1, W1, V22, W22	O	TX0N, TX0P, , TX4P, TX4N, TX1N, TX1P, TX5P, TX5N, TX2N, TX2P, TX6P, TX6N, TX3N, TX3P, TX7P, TX7N	Do not connect.
C4, D4, C19, D19, G4, H4, G19, H19, M4, N4, M19, N19, T4, U4, T19, U19	I	RX0P, RX0N, RX4N, RX4P, RX1P, RX1N, RX5N, RX5P, RX2P, RX2N, RX6N, RX6P, RX3P, RX3N, RX7N, RX7P	Do not connect.
K1, L1, K22, L22	I	ORX0N, ORX0P, ORX1P, ORX1N	Do not connect.
D6, D7, D8, D15, D16, D17, E6, E7, E16, E17, F6, F7, F8, F15, F16, F17	I/O	GPIO_ANA_1, GPIO_ANA_0, GPIO_ANA_2, GPIO_ANA_10, GPIO_ANA_8, GPIO_ANA_9, GPIO_ANA_6, GPIO_ANA_4, GPIO_ANA_12, GPIO_ANA_14, GPIO_ANA_5, GPIO_ANA_3, GPIO_ANA_7, GPIO_ANA_15, GPIO_ANA_11, GPIO_ANA_3	Connect to VSSA with a 10 kΩ resistor or configure as outputs, drive low, and leave disconnected.
H9, H14, J9, J14, L9, L14, M9, M14, K9, K14	I	TRXA_CTRL, TRXE_CTRL, TRXB_CTRL, TRXF_CTRL, TRXC_CTRL, TRXG_CTRL, TRXD_CTRL, TRXH_CTRL, ORXA_CTRL, ORXB_CTRL	Connect to VSSA.
F7, F11, L7, L11	I	ORX_CTRL_C, ORX_CTRL_B, ORX_CTRL_D, ORX_CTRL_A	Connect to VSSA directly or with a 10 kΩ pull-down resistor.
C9, C10, C13, C14	I	EXT_LO0N, EXT_LO0P, EXT_LO1N, EXT_LO1P	Do not connect.
H10, H11, H12, H13, J10, J11, J12, J13, K10, K13, L10, L13, M10, M13, N9, N10, N13, N14, P9, P10, P13, P14, R10, R13	I/O	GPIO_0 to GPIO_23	Connect to VSSA with a 10 kΩ resistor or configure as outputs, drive low, and leave disconnected.
T9, T14	O	GPINT0, GPINT1	Do not connect.
T12	O	SPI_DO	Do not connect.
R14	I	TEST_EN	Connect to VSSA.
U13, U14, U15, T15, V10, V13	I	SYNCIN0N, ,SYNCIN0P, SYNCIN1N, ,SYNCIN1P, SYNCIN2N, ,SYNCIN2P	Connect to VSSA.
T8, U8, U9, U10	O	SYNCOUT1P, SYNCOUT1N, SYNCOUT0P, SYNCOUT0N	Do not connect.
Y5, Y6, Y9, Y10, AA3, AA4, AA7, AA8, AB5, AB6, AB9, AB10, AC3, AC4, AC7, AC8	O	SERDOUT1P, SERDOUT1N, SERDOUT4P, SERDOUT4N, SERDOUT0P, SERDOUT0N, SERDOUT5P, SERDOUT5N, SERDOUT3P, SERDOUT3N, SERDOUT6P, SERDOUT6N, SERDOUT2P, SERDOUT2N, SERDOUT7P, SERDOUT7N	Do not connect.
Y13, Y14, Y17, Y18, AA15, AA16, AA19, AA20, AB13, AB14, AB17, AB18, AC15, AC16, AC19, AC20	I	SERDIN5N, SERDIN5P, SERDIN1P, SERDIN1N, SERDIN4N, SERDIN4P, SERDIN0P, SERDIN0N, SERDIN6N, SERDIN6P, SERDIN3P, SERDIN3N, SERDIN7N, SERDIN7P, SERDIN2P, SERDIN2N	Do not connect.