ADRV9040 System Development User Guide

INTRODUCTION

This document is the main source of information for system engineers and software developers using the Analog Devices, Inc., ADRV904x software defined radio transceiver family. This document is organized to make it easy to find the relevant information. The document includes the following sections:

- The System Overview section explains the capability of the part and an introduction to all the subsystems and functions, including block diagrams and interfaces.
- The Software Architecture section explains software design approach using application programming interfaces (APIs) and all details needed to develop code to operate the device.
- The Software Integration section explains the structure of the Analog Devices API and how to integrate into the code.
- The Serial Peripheral Interface (SPI) section describes the main control interface between the baseband processor (BBP, also referred to as BBIC) and the device.
- The System Initialization section explains the sequence of steps needed at startup.
- The JESD204B/JESD204C Interface section describes the high-speed digital interface that transfers data to/from a baseband processor.
- The Synthesizer Configuration section describes the design, control, and versatility of the synthesizer subsystem.
- The Arm Processor and Device Calibrations section explains how the ARM schedules and controls the calibrations.
- The Stream Processor and System Control section explains the stream processor functions and how they are implemented.
- The transmitter overview and path control sections describe the operation of the Tx attenuation settings and available software API used for control.
- The PA Protection section 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.
- The Receiver Gain Control and Gain Compensation section describes automatic and manual gain control options.
- The Digital Filter Configuration section describes the digital processing portion of each receiver and transmitter and provides details on configuration options.
- The General Purpose Input/Output Configuration section describes the different general-purpose input/output (GPIO) capabilities and how to configure them for different functions.
- The General Purpose Interrupt section describes the various interrupt options that can be routed to the GPINT pins for monitoring purposes.
- The RF port interface overview section describes the radio frequency (RF) port impedance matching process and explains different topologies that can be used to achieve proper impedance matching.
- The Power Management Considerations section 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.
- The PCB Layout Considerations section provides guidelines for proper printed circuit board (PCB) layout and techniques for maximizing performance and minimizing channel to channel interference.
- The ACE software section explains setup and control or the device using the graphical user interface (GUI) software.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

SYSTEM OVERVIEW

The ADRV904x is a highly integrated, RF agile transceiver offering eight transmitters, two observation receivers for monitoring transmitter channels, eight receivers, integrated local oscillator (LO) and clock synthesizers, and digital signal processing functions to provide a complete transceiver solution. The device provides the high radio performance and low power consumption demanded by cellular infrastructure applications such as small cell base station radios, macro 3G/4G/5G systems, and massive multiple input and multiple output (MIMO) base stations.

The receiver (Rx) and transmitter (Tx) signal paths use a zero intermediate frequency (IF) (ZIF) architecture that provides wide bandwidth with dynamic range suitable for noncontiguous multicarrier base station applications. The ZIF architecture has the benefits of low-power and RF frequency and bandwidth agility. The lack of aliases and out of band images eliminates anti-aliasing and image filters, reducing system size and cost and making band independent solutions possible.

The device also includes two wide bandwidth observation path receiver subsystems for monitoring transmitter outputs. The complete transceiver subsystem includes automatic and manual attenuation control, DC offset correction, quadrature error correction (QEC), and digital filtering. GPIOs that provide an array of digital control options are also integrated.

The transceiver includes four fully integrated phase-locked loops (PLLs). A single PLL provides high performance, low-power fractional-N RF LO synthesis supporting single and multiband time division duplex (TDD) and frequency division duplex (FDD) operation with an instantaneous bandwidth (IBW) up to 600 MHz. An additional PLL provides a second RF LO in order to support multiband applications with spacing greater than 600 MHz or to enable unique transmitter/receiver LO frequencies for frequency planning flexibility. A dedicated PLL provides clock for the digital circuitry. An additional PLL provides clock for Serdes blocks. A multichip synchronization mechanism synchronizes the phases of all local oscillators and clocks between multiple chips. All voltage controlled oscillators (VCOs) and loop filter components are integrated and can be adjusted through the SPI interface.

The transceiver contains fully integrated digital front end (DFE) functionality which includes carrier digital upconversion (CDUC)/carrier digital downconversion (CDDC), crest factor reduction (CFR), digital predistortion (DPD), closed loop gain control (CLGC), and voltage standing wave ratio (VSWR) monitor.

The CDUC feature of the ADRV904x filters and places individual carrier components within the band of interest. The CDDC feature decomposes and filters individual carriers before sent through serial data interface. The CDUC and CDDC features enable efficient utilization of serial data interface.

The low-power CFR engine of the ADRV904x reduces the peak to average ratio of the input signal, enabling higher efficiency transmit line ups while reducing the processing load on baseband processors.

This device contains a fully integrated, low-power DPD engine for use in power amplifier (PA) linearization. DPD enables use of high- efficiency PAs, reducing the power consumption of base station radios while also reducing the number of SERDES lanes necessary to interface with baseband processors. The device includes an A55 processor to independently serve DPD, CLGC, and VSWR monitor features. The dedicated and powerful processor together with the DPD engine provides industry leading DPD performance.

The serial data interface consists of eight serializer lanes and eight deserializer lanes. The interface supports both the JESD204B and JESD204C standards, and it operates at data rates up to 19.66 Gbps for JESD204B and up to 32.44 Gbps for JESD204C. Both fixed and floating-point data formats are supported. The floating-point format allows internal automatic gain control (AGC) to be transparent to the baseband processor. Details of the format are available in a separate document.

The ADRV904x is powered directly from 0.8 V, 1.0 V, and 1.8 V regulators, and is controlled via a standard SPI serial port. Comprehensive power-down modes are included to minimize power consumption in normal use. The device is packaged in a 27 mm × 20 mm, 736-ball grid array.

Figure 1 is a high level view of the functions in the ADRV904x. Descriptions of each block with setup and control details are provided in subsequent sections of this document.

SYSTEM OVERVIEW

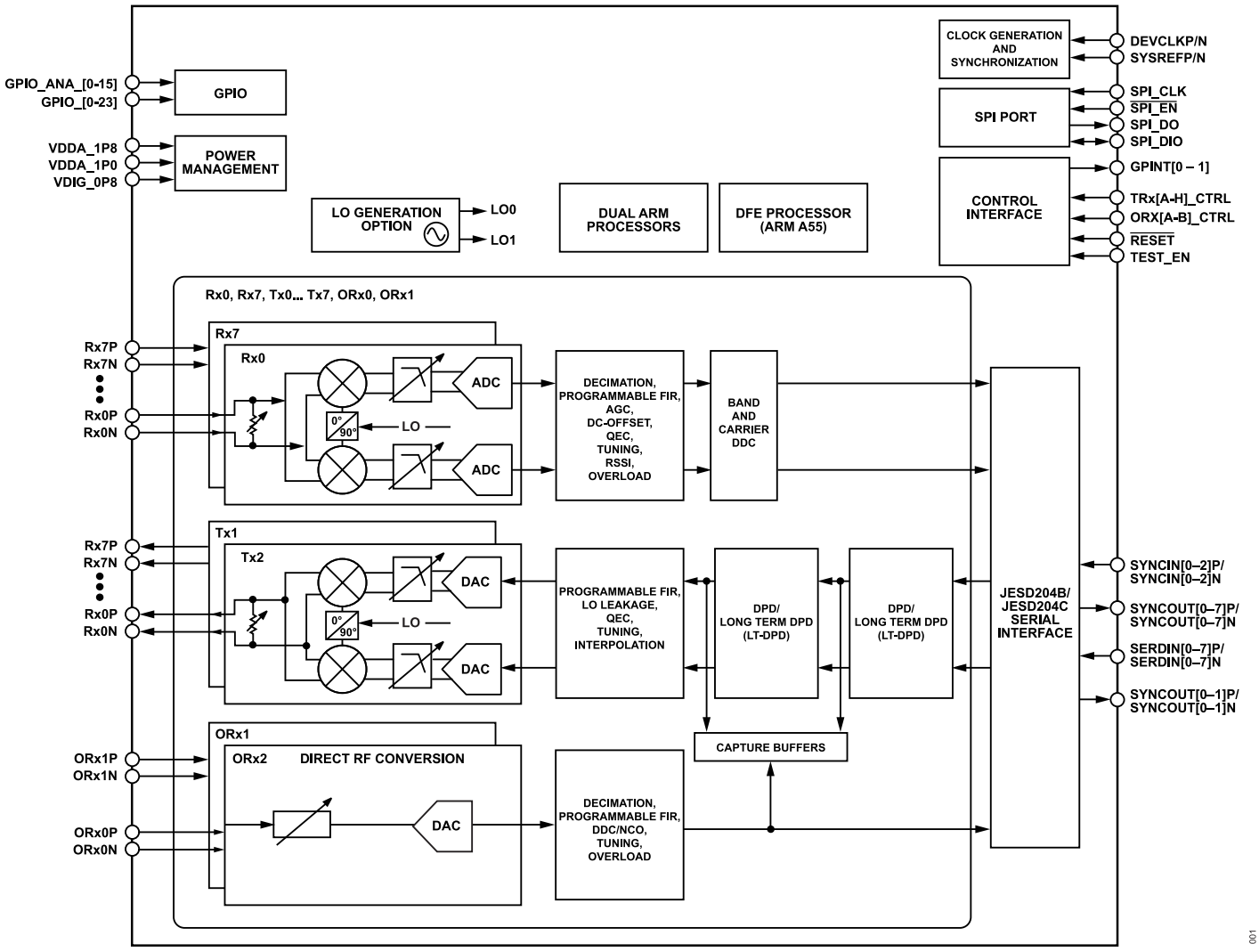


Figure 1. ADRV904x Functional Block Diagram

SOFTWARE INTEGRATION

This section provides information about the software deliverables from Analog Devices, including the Analog Devices developed 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 deliverable follows the structure in Figure 2. The software package consists of the following main folders:

- **api** contains the API C source code for controlling the ADRV904x family of transceiver devices which can be integrated into a user application running on a baseband processor. This folder also contains the API documentation.
- **firmware** contains precompiled binary for the dual core embedded ARM processor in the ADRV904x family of devices in the radio subdirectory, the precompiled binary for the quad core A55 DFE application processor, which hosts the DFE algorithms and the error reference documentation for debugging calibration errors.
- **gain_tables** contains the programmable gain table for the receivers. The transmitter attenuation table is hard coded into the ADRV904x and is not programmable.
- **gui** contains the ADRV904x transceiver evaluation GUI installation binaries.
- **profiles** contains standard use cases that define the transceiver data rates and settings in JSON format.

Note that the base product for this family is the ADRV9040. Therefore, all software developed will use this number as a prefix regardless of the family variant. Note that this is a preliminary organization of the ADRV9040 software deliverables, and the folder structure of the ADRV9040 software deliverables is subject to change before the first official software release.

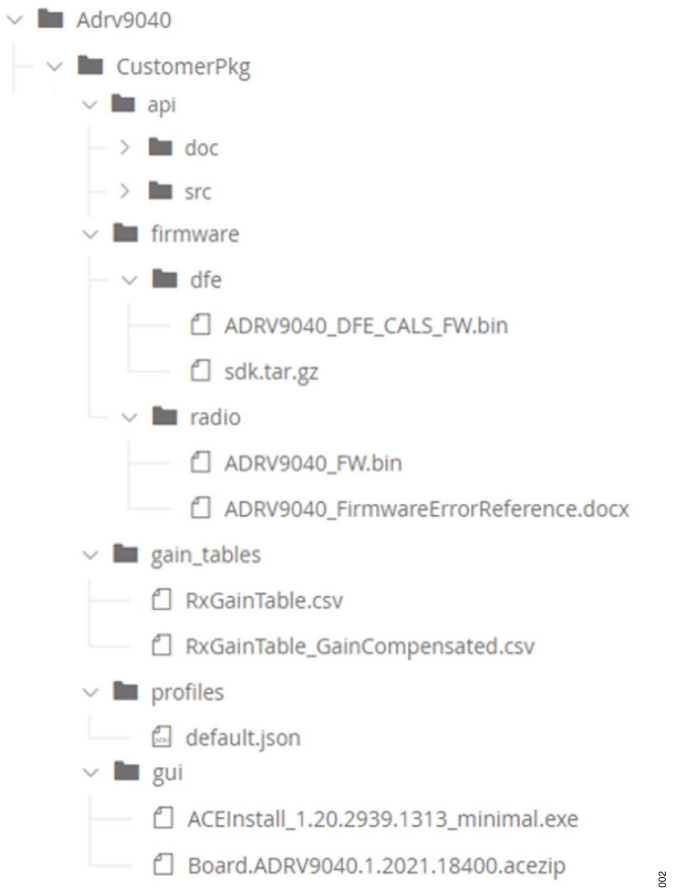


Figure 2. ADRV904x Software Deliverables

SOFTWARE INTEGRATION

Note that the stream processor binary and the profile binary resource files are required for programming the ADRV904x. The stream binary and the profile binary resource files targeting a specific use case are generated with an Analog Devices 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 3. A more detailed explanation of the integration process is provided in the sections to follow.

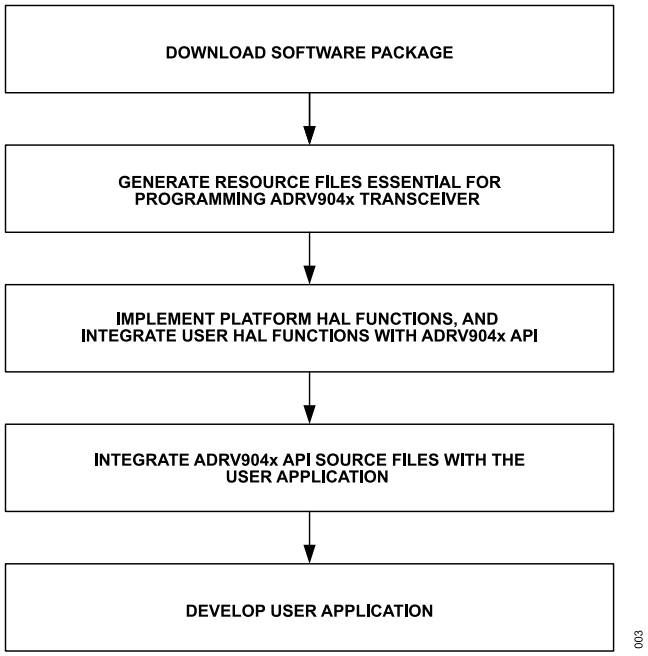


Figure 3. Software Integration Process Overview

SOFTWARE ARCHITECTURE

The ADRV904x contains dedicated signal processing blocks, analog-to-digital converters (ADCs), digital-to-analog converters (DACs), two ARM processor cores, and a coprocessor called the stream processor. A simplified logical partitioning of a typical software architecture designed with the ADRV904x firmware and API is in Figure 4. The firmware for the ARM cores is a precompiled binary. The stream coprocessor binary is user generated. The process to generate a stream binary is described in the Resource Files section.

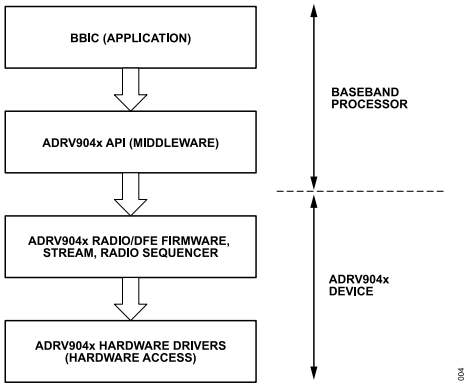


Figure 4. Logical Partitioning/Layering of ADRV904x Software Application

An ADRV904x user application commands the ADRV904x 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

SOFTWARE INTEGRATION

an operating system. Analog Devices recommends using a platform running an operating system such as Linux, which provides a sufficiently large memory footprint, and can take advantage of Analog Devices utility functions. Refer to the API Integration Checklist for information regarding integrating the ADRV904x API into a user application.

Figure 5 shows a more detailed ADRV904x-based system software architecture. A baseband processor running an ADRV904x-based application controls the transceiver through the ADRV904x API. The API integrated with the user application relies on a SPI interface in the baseband processor to interact with the ADRV904x. The ARM firmware running on a dual core embedded ARM processor consists of the algorithms for transceiver calibrations, controlled through the ADRV904x 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 advanced high performance bus (AHB) interface.

The stream coprocessor sets up and manages the transmit/receive chains of the ADRV904x on occurrence of certain events (such as transmitter/receiver/observation receiver (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 ADRV904x hardware resources through AHB memory mapped registers. There is one core stream processor and 18 slice stream processors, one each for the eight transmitter and receiver datapaths, and two for the observation receiver datapaths. The existence of individual slice stream processors for each datapath enables true real-time parallel operation of all unique transmitter and receiver datapaths.

The control commands issued to the embedded ARM processor and the stream coprocessor in the ADRV904x 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 general-purpose interrupt status pins.

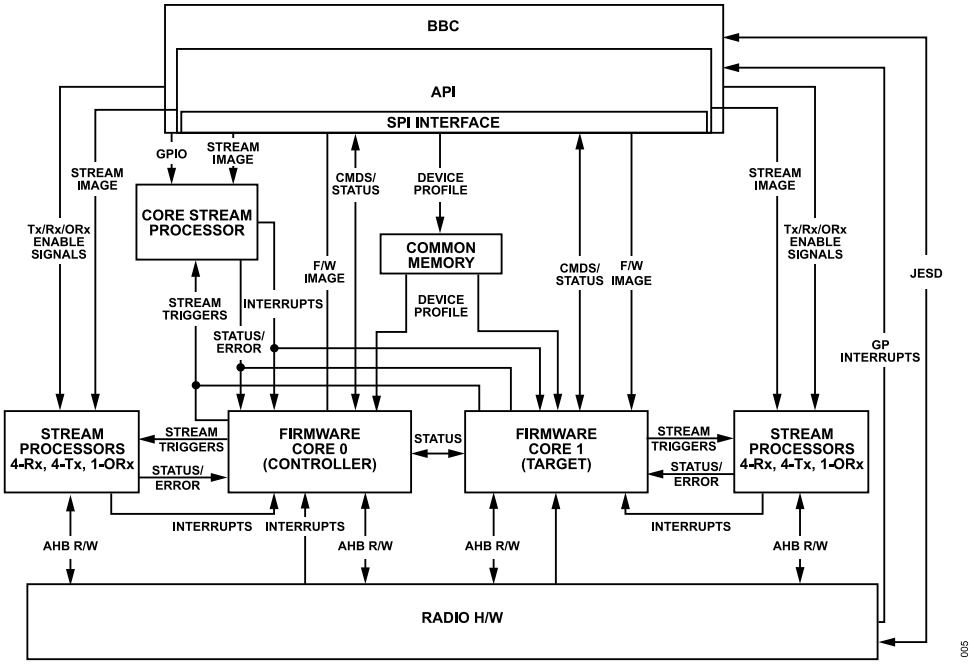


Figure 5. A Detailed Software Architectural View of an ADRV904x Software Application

Analog Devices Evaluation System Software Architecture

The software architecture of the ADRV904x Analog Devices evaluation system is in Figure 6. The software architecture of the Analog Devices 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 system on a chip field programmable gate array (SoC FPGA) host integrated with an embedded processor running a Linux operating system. The ADRV904x API is integrated with the baseband processor software to control the ADRV904x. The platform hardware abstraction layer (HAL) interface (SPI and timer) is implemented for the Analog Devices ADRV904x evaluation system, which in turn is used by the ADRV904x API for interacting with the ADRV904x.

SOFTWARE INTEGRATION

In a typical use case of the Analog Devices 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 ADRV904x through an 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.

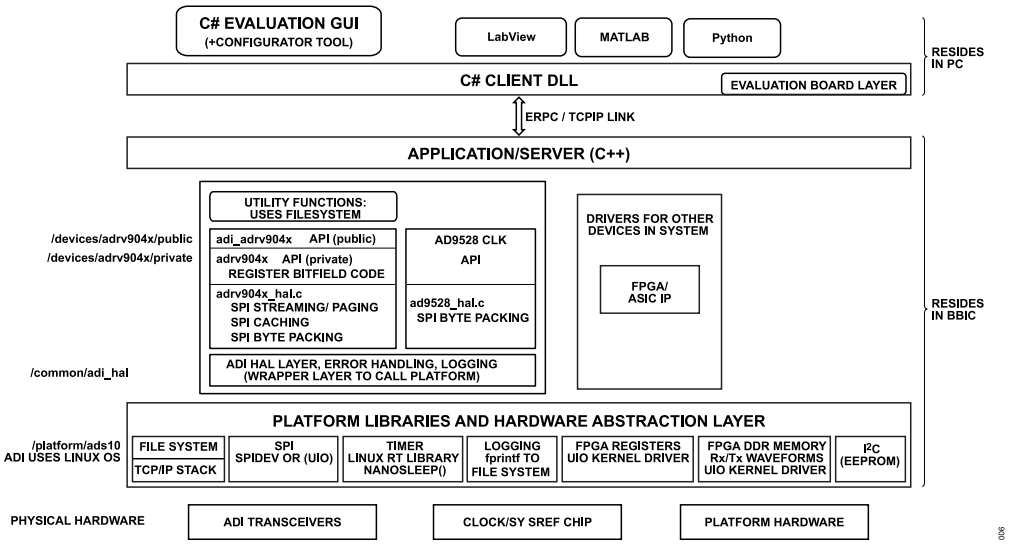


Figure 6. ADRV904x-Based Analog Devices Evaluation System Software Architecture

RESOURCE FILES

The following are the resources required for an ADRV904x software package:

- ADRV904x firmware binaries
- Stream coprocessor binary
- Profile binary
- Receiver gain tables

This section goes through the steps to generate resource files, which are essential for the ADRV904x to function. The resource files are programmed into the transceiver during the initialization phase. A brief description of the resource files are in Table 1.

Table 1. ADRV904x Platform Files

ADRV904x Platform File	Purpose	Generation Mechanism	Size
ADRV9040_FW.bin	The precompiled firmware binary for the embedded dual core ARM processors in the ADRV904x transceiver, which mainly consists of Analog Devices proprietary algorithms used to calibrate the transceiver.	Delivered as part of the ADRV904x software package.	641 kb
ADRV9040_DFE_CALS_FW.bin	The precompiled firmware binary for the embedded quad core A55 DFE processors in the ADRV904x transceiver, which mainly consists of Analog Devices proprietary DFE algorithms such as DPD, CLGC, and VSWR.	Delivered as part of the ADRV904x software package.	Variable (maximum of 8 MB)
stream_image.bin	The binary file for the stream coprocessor in the ADRV904x, which is mainly used for setting up and managing the transmit/receive chains in a time critical manner on occurrence of certain events, such as transmit/receive enable.	User generated with Analog Devices evaluation software. Please refer to the ADRV904x evaluation system user guide for steps to generate the stream binary.	100 kb
DeviceProfileTest.bin	The ADRV904x configuration for a particular use case is programmed through the profile binary. The profile consists of the filter	User generated with Analog Devices evaluation software. Please refer to the ADRV904x evaluation system	25 kb

SOFTWARE INTEGRATION

Table 1. ADRV904x Platform Files (Continued)

ADRV904x Platform File	Purpose	Generation Mechanism	Size
	coefficients, clock rates, signal blocks, DFE resources to enable/disable for a particular use case.	user guide for steps to generate the profile binary.	
RxGainTable.csv/ RxGaintTable_GainCompensated.csv	The front end gain look up tables for the ADRV904x receiver.	Default table delivered as part of the ADRV904x software package. User can generate custom gain tables.	Less than 10 kb

Note that 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.

Firmware Binaries

The firmware for dual core embedded ARM processors in the ADRV904x is delivered in the form of precompiled binary files. The firmware in the ADRV904x mainly consists of Analog Devices proprietary algorithms used for calibration, and drivers to access ADRV904x hardware resources.

The firmware binaries are delivered in the software package, see Figure 7. It is required to program both the firmware binaries as part of ADRV904x initialization. Refer to the programming section for information on initializing the ADRV904x.

Figure 7. ADRV904x Firmware Binaries Delivered in the ADRV904x Software Package

Stream Binary

The ADRV904x consists of a stream coprocessor that 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 18 slice stream processors, one each for the eight transmitter, receiver datapaths, and two for the observation receiver datapaths. The existence of individual slice stream processors for each datapath enables true real-time parallel operation of all unique transmitter and receiver datapaths.

Please refer to the resource file generation using ACE section of the ADRV904x evaluation system user guide for a procedure to generate a stream binary.

Profile Binary

The profile consists of the ADRV904x 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 ADRV904x during initialization.

The user is required to supply the parameters listed in Table 2 as input to an Analog Devices provided tool in order to generate a programmable profile binary. Refer to the software resource files section in the ADRV904x evaluation system user guide for information on how to generate profile binaries.

SOFTWARE INTEGRATION

Table 2. User Provided Inputs to Generate an ADRV904x Profile Binary

Parameter	Description
Tx/Rx Channels Enable	The ADRV904x 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 JESD204C Mode	The ADRV904x supports JESD204B and JESD204C protocols for datapath interface with the baseband processor. The user can configure the part to use either JESD204B or JESD204C depending on the lane rates required.
Transmitter Configuration	
Tx LO Frequency	RF frequency setting of the LO 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 ADRV904x.
Tx Sampling Rate	Transmit signal I/Q sample rate at the JESD deframer input of the ADRV904x.
Tx Synthesis Bandwidth	The transmit side analysis bandwidth for DPD implemented in the baseband processor.
Tx Synthesis Frequency Upper Edge	Upper frequency edge of the synthesis bandwidth.
Tx Synthesis Frequency Lower Edge	Lower frequency edge of the synthesis bandwidth.
ORx Sampling Rate	Observation receiver signal IQ sample rate at the JESD framer output of ADRV904x.
Receiver Configuration	
Rx LO Frequency	RF frequency setting of the LO 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 ADRV904x 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 ADRV904x for Band1.
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 DACs on the transmitter side
NP	DAC sample bit width.
L	Number of input lanes at the ADRV904x deserializer input.
F	Number of DAC bytes per frame of data.
K	Number of frames per multiframe.
S	Number of samples per clock per DAC.
E (JESD204C)	Extended multiblock setting for JESD204C use case, usually set to 1.
Serializer Lanes Enabled	8-bit mask that indicates the serializer lanes enabled on the ADRV904x 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 ADCs on the receiver side.
NP	ADC sample bit width.
L	Number of output lanes at the ADRV904x serializer output.
F	Number of ADC bytes per frame of data.
K	Number of frames per multiframe.
S	Number of samples per clock per DAC.
E (JESD204C)	Extended multiblock setting for JESD204C use case, usually set to 1.
Serializer Lanes Enabled	8-bit mask that indicates the serializer lanes enabled on the ADRV904x framer input.

SOFTWARE INTEGRATION

Consider an example shown in the Transmitter Configuration for Profile Generation section below for a use case with 100 MHz transmitter/receiver primary signal bandwidth, 245.76 MSPS sample rate for the transmitter data at the ADRV904x deframer input, and 245.76 MSPS sample rate for the receiver data at the ADRV904x framer output. In this example, the system supports a JESD lane rate of 9.8 Gbps, and JESD204B is used as the link protocol for the baseband processor to the ADRV904x data interface. The example derives the parameters to input to an Analog Devices provided tools to generate a programmable profile binary

Transmitter Configuration for Profile Generation

In this example, derive the transmitter side profile configuration for a 100 MHz carrier IBW signal transmitted on transmit channels Tx0 to Tx3 at +50 MHz offset with respect to the LO, and on transmit channels Tx4 to Tx7 at −50 MHz offset with respect to the LO as shown in Figure 8.

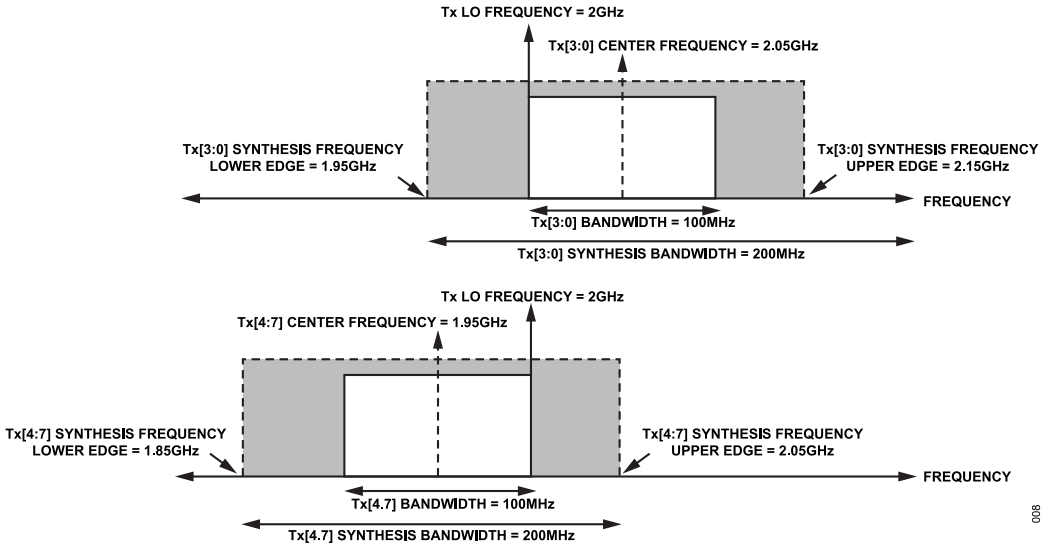


Figure 8. Example Transmit Carrier Configuration for Generating a Profile Binary

The transmitter 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 (see Table 3).

Table 3. Example Transmitter Configuration to Generate an ADRV904x 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

SOFTWARE INTEGRATION

To configure the JESD settings in the transmit path, assume that the system supports a lane rate of 9.8 Gbps in JESD204B mode. The ADRV904x 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 DACs (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 the ADRV904x is calculated as:

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

Proceed to calculate the number of DAC bytes per frame as follows:

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

The transmitter side JESD parameters for profile generation can be plugged into the configuration table (Table 4).

Table 4. JESD Deframer (Tx side) Profile Configuration Example

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

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

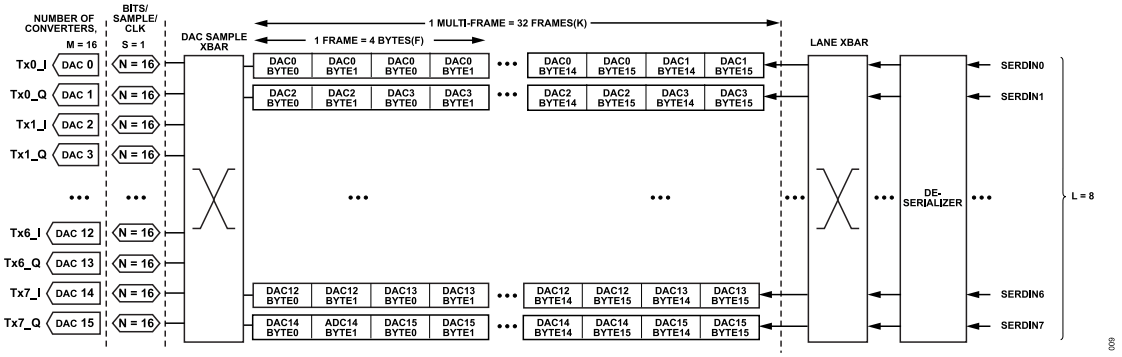


Figure 9. JESD Deframer (Tx Side) Configuration for the Example Profile

Receiver Configuration for Profile Generation

In this example, derive the receiver side profile configuration for a 400 MHz carrier IBW signal in Figure 10, which has the following characteristics:

- 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 downconverted separately and serialized in the ADRV904x before sending it across to the baseband processor. The received signal output from the ADRV904x framer is sampled at 122.88 MSPS. These values are included for the receiver configuration in Table 5.

SOFTWARE INTEGRATION

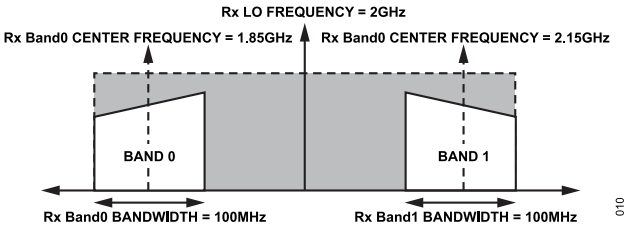


Figure 10. Wideband Receiver Carrier Configuration for an Example ADRV904x Profile Generation

Table 5. Receiver Configuration Example for ADRV904x 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 ADRV904x. The ADRV904x supports up to eight serializer output lanes, and 16 bits per ADC sample (Np = 16). This example has all eight receiver channels enabled. Therefore, a total of 16 ADCs (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 downconverters present in the ADRV904x receiver path. In this case, since the two bands are digitally downconverted individually in the ADRV904x receiver path, the data is framed, serialized, and sent to the baseband through the ADRV904x framer and JRx module. For each individual band, the digital downconverter in the ADRV904x outputs 16 channels of data corresponding to the 16 ADCs. The framer in the ADRV904x receives data from a total of 2 ×16 downconverted 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 ADRV904x is calculated as:

$$L = \frac{NumBands \times M \times S \times Np \times \left(\frac{10}{8}\right) \times RxSamplingRate}{Lane\ \ Rate} = \frac{2 \times 16 \times 1 \times 16\left(\frac{10}{8}\right) \times 122.88\ \ \text{MSPS}}{9830.4\ \ \text{MSPS}} = 8 \quad (3)$$

Proceed to calculate the number of ADC bytes per frame as follows:

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

The receiver side JESD parameters for profile generation can be plugged into the configuration table (Table 6).

Table 6. JESD Framer (Rx Side) Configuration Example for ADRV904x

JESD Framer (Rx Side) Configuration	
Framer Lane Xbar	Default (refer to the JESD204 Standard section)
DAC Sample Xbar	Default (refer to the JESD204 Standard section)
M	16
Np	16
L	8
F	4
K	32
S	1
Serializer Lanes Enabled	0xFF (SERDOUT0 to SERDOUT7)

SOFTWARE INTEGRATION

The JESD configuration for this profile is captured in Figure 11.

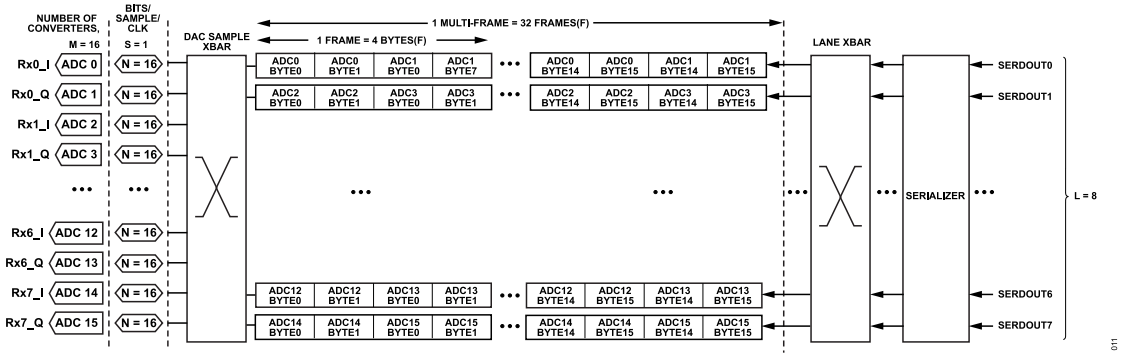


Figure 11. JESD Framer (Rx Side) Configuration Example for ADRV904x Profile Generation

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

Receiver Gain Table

The ADRV904x provides a 32 dB dynamic range for receiver gain control. The receiver gain is applied on the receiver front end through a programmable lookup table programmed into the ADRV904x 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 12.

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

012

Figure 12. Example Rx Gain Table Entries

The front-end attenuation is given by:

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

Analog Devices provides a default gain table in CSV format as part of the software package that can be programmed into the ADRV904x during initialization (see Figure 13). 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.

Figure 13. Default Rx Gain Table Provided as Part of the ADRV904x Software Package

SOFTWARE INTEGRATION

The resource files (firmware binary, stream binary, profile binary, and receiver gain tables) are provided as input to the API **adi_adrv904x_PreMcsInit( )** while programming the device through the data structure **adi_adrv904x_TrxFileInfo_t** defined in the file **api/src/c_src/devices/adrv904x/public/include/adi_adrv904x_utilities_types.h**. Refer to the Programming the Device section for the ADRV904x programming sequence. Shown below is the listing.

```
typedef struct adi_adrv904x_TrxFileInfo
{
adi_adrv904x_streamBinaryInfo_t stream; /*!< Stream File Settings and Path to stream binary*/
adi_adrv904x_cpuBinaryInfo_t cpu; /*!< CPU File Settings and Path to Firmware Binary */
adi_adrv904x_CpuProfileBinaryInfo_t cpuProfile; /*!< CPU Profile File Settings and Path to Profile Binary */
adi_adrv904x_RxGainTableInfo_t rxGainTable[ADI_ADRV904X_RX_GAIN_TABLE_ARR_MAX]; /*!< Rx Gain Table Settings */
} adi_adrv904x_TrxFileInfo_t;
```

API INTEGRATION

The ADRV904x control commands are implemented through the ADRV904x API. This section explains the steps needed to integrate the ADRV904x API with a user application. The ADRV904x API architecture is in Figure 14.

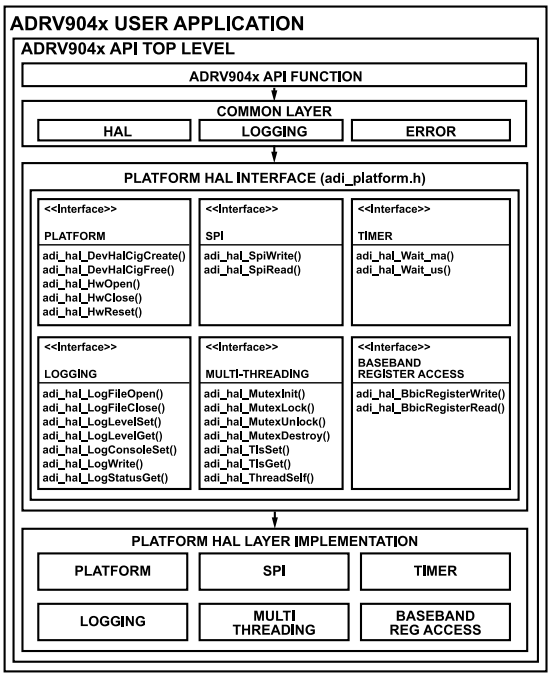


Figure 14. ADRV904x API Architecture

A brief description of each layer in the ADRV904x API is provided as follows:

- ADRV904x API Functions. This is the top-level interface to API functions called by the user application to configure and control the ADRV904x 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.
- Platform HAL Interface. Platform refers to a radio system that includes the baseband processor and the ADRV904x. Platform HAL refers to low-level hardware abstraction layer/device drivers in the platform that are essential for the ADRV904x 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 ADRV904x API Folder Structure section for an explanation on actions that a user needs to take in order to integrate the platform HAL interface.

SOFTWARE INTEGRATION

- 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 the ADRV904x API folder structure (see the ADRV904x API Folder Structure section) and integrate ADRV904x API files into a user application project.
2. Implement platform HAL layer functions described in the Platform HAL Interface section.
3. Implement multithreading HAL layer functions described in the Implementing the Platform HAL section.
4. Instantiate device data structure as described in the Instantiating the Device Data Structure section.
5. Instantiate error handling memory (see the Error Handling Memory section) and architect error handling (see the Error Handling section).
6. Configure the SPI bridge as described in the SPI API Functions section.

ADRV904x API Folder Structure

The top level folder structure of the API software delivered in the ADRV904x software package is in Figure 15. Each subfolder is explained in the following sections. Analog Devices understands that the developer may desire to use a different folder structure. While Analog Devices provides API source code releases for the ADRV904x family of devices in the folder structure shown in Figure 15, the developer may organize the ADRV904x API into a custom folder organization if required. Modifying the content of each ADRV904x API source file prevents easy updates to future ADRV904x API releases.

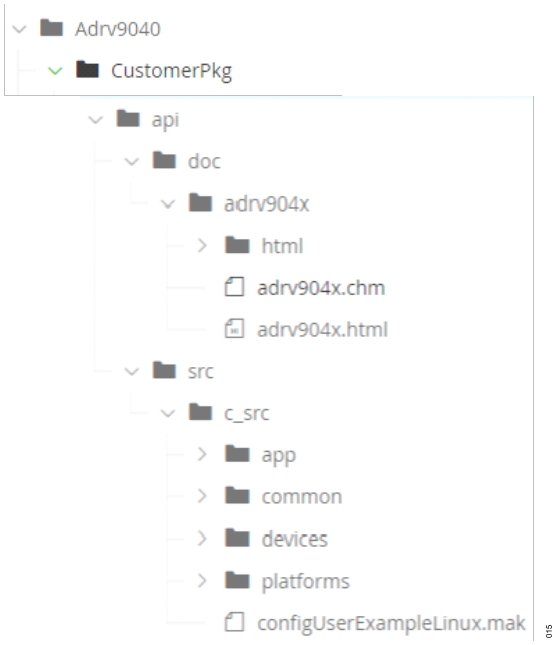


Figure 15. ADRV904x API Folder Structure

The user can follow the same folder structure described in Figure 15 to integrate the API into a user application. Shown in Figure 16 is the Analog Devices evaluation system project where the API source code is integrated into the Analog Devices evaluation system software project. The API folder structure is marked in red in Figure 16.

SOFTWARE INTEGRATION

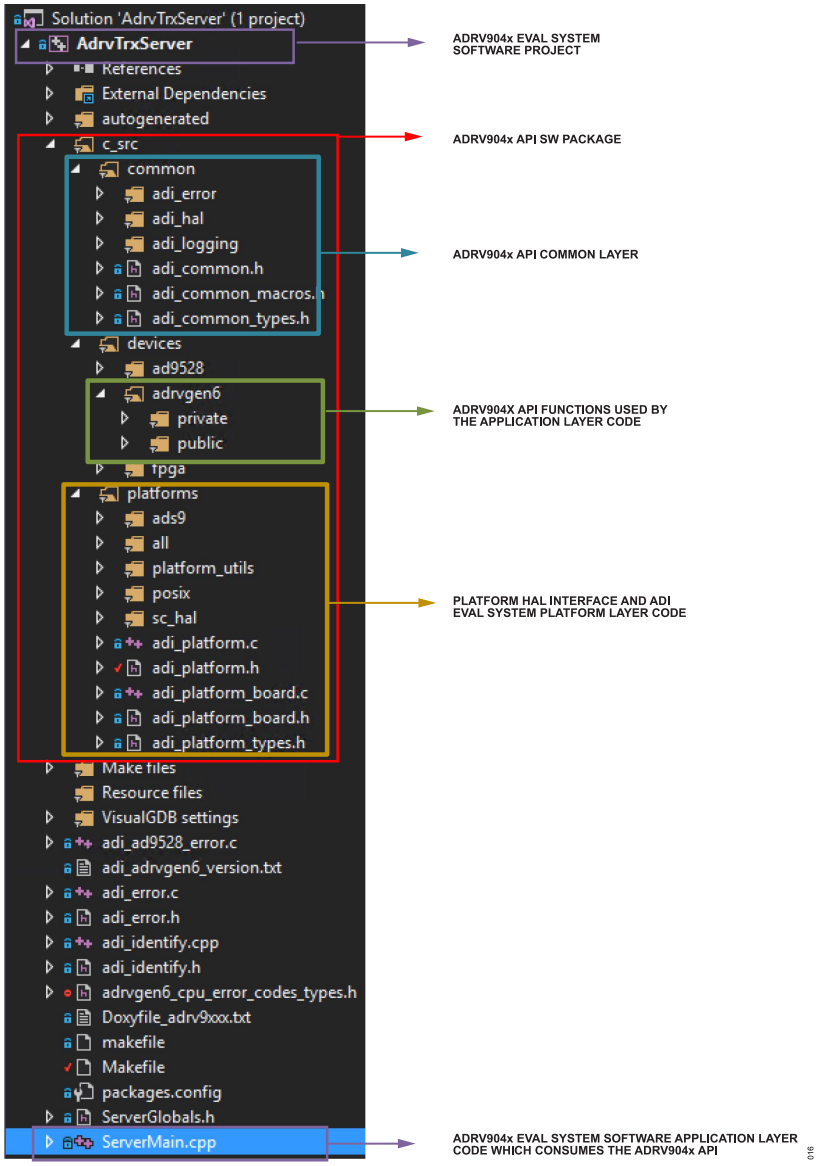


Figure 16. ADRV904x-Based Analog Devices Evaluation Software Project Integrated with ADRV904x API

Devices Directory

The devices directory (**Adi.Adrv904x.CustomerPkg/api/src/c_src/devices**) in the API folder structure contains the ADRV904x API that can be called by the user application for configuring and controlling the ADRV904x transceiver. An expanded view of the ADRV904x API folder is shown in Figure 17.

SOFTWARE INTEGRATION

Figure 17. ADRV904x Device API Folder Structure Expanded View

The **/include** folder inside private and public folders consists of C API function prototypes in the **adi_adrv904x_<API File>.h** header files, and type definitions consisting of user defined structures and enumerations in the **adi_adrv904x_<API File>_types.h** files. The **/src** folder consists of the C API function definitions of the prototypes declared in the **/include/adi_adrv904x_<API File>.h** header files.

The API files are logically partitioned, each file corresponds to one specific functionality of the ADRV904x. Table 7 consists of the list of API files and their corresponding functionality.

Table 7. List of Top Level ADRV904x API Files (Files with Functions) Delivered in the ADRV904x Software Package

API File	Description
adi_adrv904x_cals.h	API functions that control the ADRV904x calibrations. The user can enable/disable and retrieve calibration status through the API functions provided in this file.
adi_adrv904x_core.h	API functions used to initialize the ADRV904x.
adi_adrv904x_cpu.h	Low-level API commands for the dual core embedded ARM Cortex-M4 processor in the ADRV904x that are consumed by other API functions.
adi_adrv904x_dfe_cpu.h	Low-level API commands for the quad core embedded ARM Cortex-A55 DFE processor in the ADRV904x that are consumed by other DFE calibration API functions.
adi_adrv904x_dfe_cfr.h	API functions to configure and control the ADRV904x CFR engine.
adi_adrv904x_dfe_dpd.h	API functions to configure and control the Analog Devices DPD algorithm on the ADRV904x transceiver.
adi_adrv904x_error.h	Error reporting functions for the ADRV904x.
adi_adrv904x_hal.h	API functions that cover ADRV904x SPI interface features.
adi_adrv904x_radioctrl.h	API functions for ADRV904x radio control functions such as transmit/receive enable and LO frequency setting.
adi_adrv904x_rx.h	API functions to control the ADRV904x receiver chain functionality such as gain control, CDDC, and formatter.
adi_adrv904x_tx.h	API functions to control the ADRV904x transmitter chain functionality such as attenuation control, CDUC, and PA protection.
adi_adrv904x_utilities.h	Utility API functions that can be used to program the ADRV904x transceiver. The utility functions depend on a file system being available on the BBIC host (typically running Linux OS).
adi_adrv904x_user.h	Compile time constants/macros for the ADRV904x API.
adi_adrv904x_agc.h	API functions for the ADRV904x AGC functionality
adi_adrv904x_datainterface.h	API functions to control the ADRV904x JESD data interface

SOFTWARE INTEGRATION

Table 7. List of Top Level ADRV904x API Files (Files with Functions) Delivered in the ADRV904x Software Package (Continued)

API File	Description
adi_adrv904x_gpio.h	API functions to setup GPIO pins on the ADRV904x. This also includes the general-purpose interrupt functions that provide ADRV904x diagnostic data to the user.

Note that the user is strictly forbidden from modifying the contents in the devices folder. The only file that a user can edit is the **adi_adrv904x_user.h**, which contains compile time macros to adjust intended functionality of certain features, such as timeout values for commands.

Common Layer

The **api/src/c_src/common/** folder contains common layer functions highlighted in the API architecture (see Figure 14). The common layer functions are the set of service layer calls used by all API functions to abstract away low-level functions such as HAL 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 ADRV904x API package is shown in Figure 18.

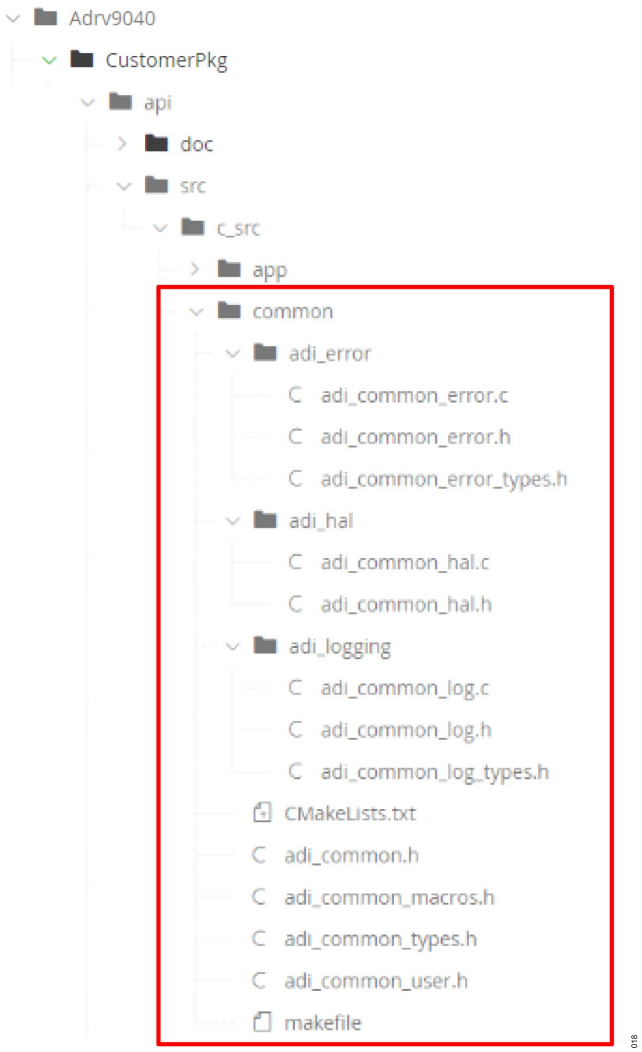


Figure 18. Expanded View of the ADRV904x Common Layer Code Delivered in ADRV904x Software Package

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 the **/common/adi_logging** folder. This feature requires a user implementation of the **adi_hal_LogWrite** interface function in the platform

SOFTWARE INTEGRATION

HAL implementation. The adi_hal_LogWrite() platform layer function is found in the file **api/src/c_src/platfrom/adi_platform.h**. Directions to implement the Analog Devices platform HAL functions are described in the Error Handling section. The APIs 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. The Analog Devices transceiver open hardware function **adi_hal_HwOpen** calls the function **adi_hal_LogLevelSet** to set the desired logging level. Available logging levels are described by the **adi_hal_LogLevel_e** enum value defined in **/api/src/c_src/platforms/adi_platform_types.h** as shown in Table 8.

Table 8. ADRV904x Logging Levels

adi_hal_LogLevel_e Enum Value	Description
ADI_HAL_LOG_NONE	All types of log messages not selected
ADI_HAL_LOG_MSG	Log message type
ADI_HAL_LOG_WARN	Warning message type
ADI_HAL_LOG_ERR	Error message type
ADI_HAL_LOG_API	API function entry for logging purposes
ADI_HAL_LOG_API_PRIV	Private API function entry for logging purposes
ADI_HAL_LOG_BF	Bit field (BF) function entry for logging purposes
ADI_HAL_LOG_HAL	Analog Devices HAL function entry for logging purposes
ADI_HAL_LOG_SPI	SPI transaction type
ADI_HAL_LOG_ALL	All types of log messages selected

The hierarchy of logging function calls are shown in Figure 19. 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.

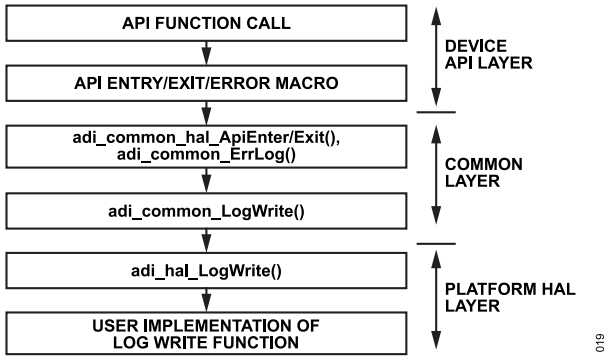


Figure 19. Hierarchy of ADRV904x Logging Function Calls

An illustration of logging function being invoked in the top level API function is shown in Figure 20. In this illustration, the RxGainSet() API calls the macro ADI_ADRV904X_API_ENTERY() to log the function entry, ADI_ADRV904X_API_EXIT() to log the function exit 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

```
ADI_API adi_adrv904x_ErrAction_e adi_adrv904x_RxTxEnableSet(adi_adrv904x_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_adrv904x_ErrAction_e recoveryAction = ADI_ADRV904X_ERR_ACT_CHECK_PARAM;

    ADI_ADRV904X_NULL_DEVICE_PTR_RETURN(device);

    ADI_ADRV904X_API_ENTRY(&device->common);

#if ADI_ADRV904X_RADIOCTRL_RANGE_CHECK > 0
    recoveryAction = adrv904x_RxTxEnableSetRangeCheck(device, orxChannelMask, orxChannelEnable, rxChannelMask, rxChannelEnable, txChannelMask, txChannelEnable);
    if (recoveryAction != ADI_ADRV904X_ERR_ACT_NONE)
    {
        ADI_API_ERROR_REPORT(&device->common, recoveryAction, "RxTxEnableSetRangeCheck Issue");
        goto cleanup;
    }
#endif

    /*Enable requested Rx Channel signal chains*/
    recoveryAction = adrv904x_RxEnableSet(device, rxChannelMask, rxChannelEnable);
    if (recoveryAction != ADI_ADRV904X_ERR_ACT_NONE)
    {
        ADI_API_ERROR_REPORT(&device->common, recoveryAction, "RxEnableSet Issue");
        goto cleanup;
    }

    /* Enable requested ORx Channel signal chains */
    recoveryAction = adrv904x_OrxEnableSet(device, orxChannelMask, orxChannelEnable);
    if (recoveryAction != ADI_ADRV904X_ERR_ACT_NONE)
    {
        ADI_API_ERROR_REPORT(&device->common, recoveryAction, "OrxEnableSet Issue");
        goto cleanup;
    }

    /* Enable requested Tx Channel signal chains */
    recoveryAction = adrv904x_TxEnableSet(device, txChannelMask, txChannelEnable);
    if (recoveryAction != ADI_ADRV904X_ERR_ACT_NONE)
    {
        ADI_API_ERROR_REPORT(&device->common, recoveryAction, "TxEnableSet Issue");
        goto cleanup;
    }

cleanup:
    ADI_ADRV904X_API_EXIT(&device->common, recoveryAction);
}
```

020

Figure 20. Illustration of Logging Functions Invoked from ADRV904x Top Level API

Error Handling

The error handling functions are found in the **api/src/c_src/common/adi_error** folder of the ADRV904x API software package. Each ADRV904x API function returns an enum adi_adrv904x_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 as follows:

- In the development/debug phase, the user is expected to debug errors through manual intervention by taking advantage of the error handbook provided by Analog Devices as part of the software package and other tools such as logging to develop error handling code in the application software.
- In the deployment 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 21.

SOFTWARE INTEGRATION

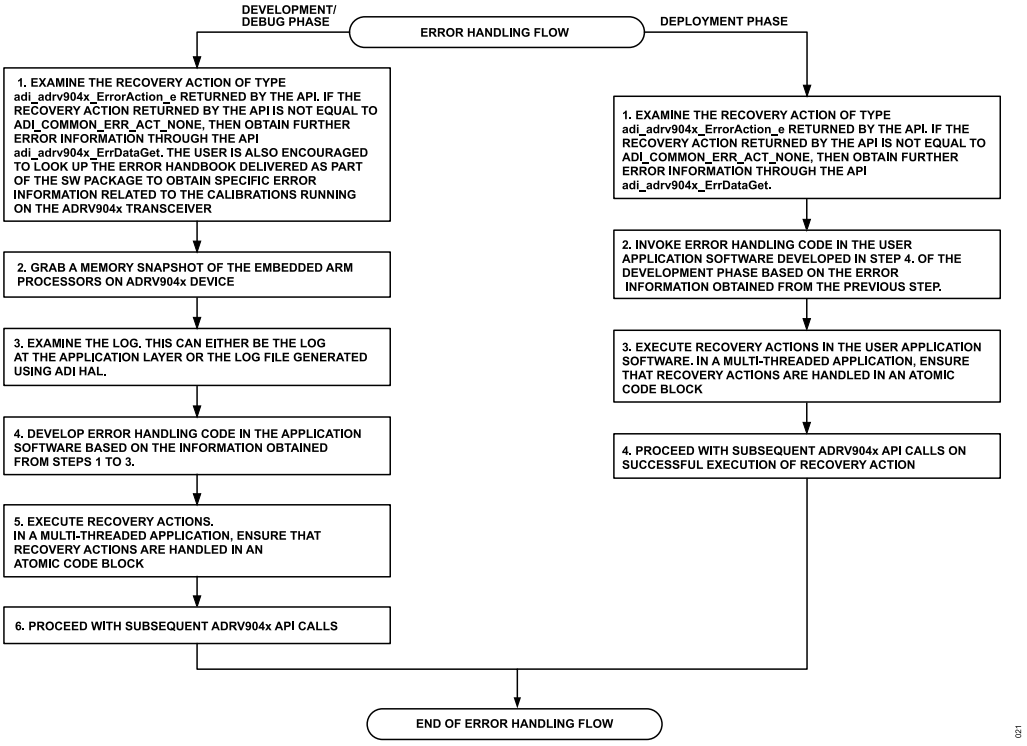


Figure 21. General Error Handling Strategy for ADRV904x

The set of all possible error recovery actions returned by each ADRV904x API are described in Table 9. Each ADRV904x API function call responds to the application layer with information on action that needs to be taken because of a possible error in the API function call. The error structure contains further information in order to take the required action.

Table 9. ADRV904x 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 an ADRV904x functionality such as GPIO, PA protection, GP interrupt, and gain control.
ADI_COMMON_ERR_ACT_RESET_INTERFACE	−300	API not OK—interface not working, device cannot be programmed or cannot access timer/I^2C/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.
ADI_COMMON_ERR_ACT_NONE	0	API function completed successfully—no error handling action is required.

Note that the recovery actions are still under construction and the actions listed in Table 9 are subject to change.

The API error structure is accessed via **adi_adrv904x_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 which the error was first encountered. An example context diagram of an error stack frame is shown in Figure 22. In this example, an error encountered in the ADRV904x 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_adrv904x_Device_t**).

SOFTWARE INTEGRATION

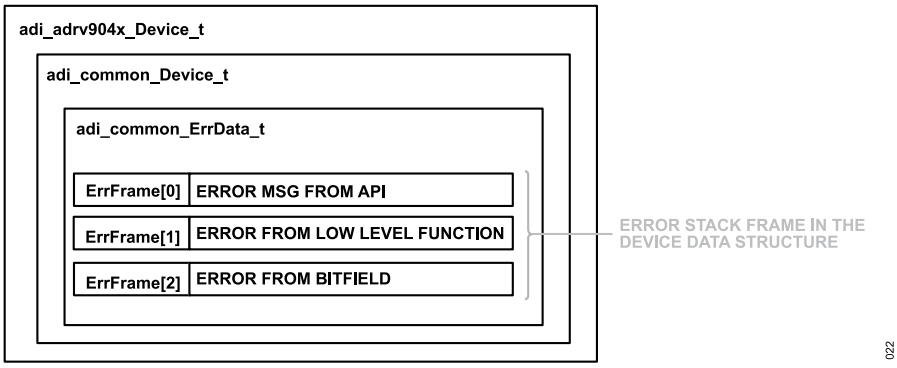


Figure 22. Example Context Diagram of Error Stack Frame Returned from the ADRV904x

Each error frame in the error stack trace contains the following members (which appear in the log when user prints the log by using the **adi_common_Errdata_t** function) described 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

Analog Devices delivers a firmware error handbook as part of the software package highlighted in Figure 23. The error handbook contains descriptions of all possible error codes reported by the ADRV904x firmware and is intended to be used as reference documentation during development/debug phase.

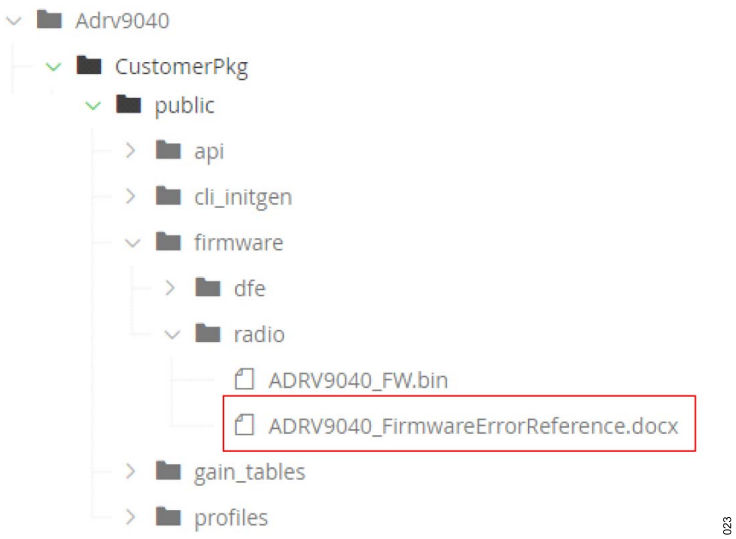


Figure 23. Firmware Error Reference Handbook Delivered as Part of Software Package

SOFTWARE INTEGRATION

The entries in the firmware error handbook are organized as shown in Figure 24 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_adrv904x_ErrDataGet()** contained in the file **/api/src/c_src/devices/adrv904x/public/include/adi_adrv904x_error.h**. The API **adi_adrv904x_ErrDataGet()** is a service function to lookup the error table entry for handling error programmatically.

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

024

Figure 24. 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 25. 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

025

Figure 25. Example Entry in the Error Handbook with Multiple Recovery Actions Associated with It

Debugging Commonly Occurring Recovery Actions

The following sections describe some debug steps that a user can take on occurrence of errors described in Table 9.

API Recovery Action—ADI_COMMON_ACT_NO_ACTION

The ADI_COMMON_ACT_NO_ACTION API recovery action is returned when an ADRV904x 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 ADRV9040 API parameter range check or null parameter check failed. If this recovery action is returned, the ADRV9040 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 ADRV9040 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 ADRV9040 API function to complete.

For debug, the developer may access further information located in the error structure, like an 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 software 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.

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 because of the SPI hardware being busy or used by another thread, the ADRV9040 API attempts to retry the SPI operation once. If the SPI transaction fails again, the ADRV9040 API reports this recovery action. This action is also returned if an ADIHAL error is returned because of the inability to access the driver.

SOFTWARE INTEGRATION

For the suggested application layer action, take the following steps:

1. Call to determine the specific ADIHAL error code and verify that ADIHAL is the error source.
2. Log error code and source.
   - If the ADIHAL error is a timeout, the ADRV9040 API function may be retried.
   - If the ADIHAL error is not a timeout, the application should try resetting the SPI driver and retrying the function call.
3. If a recovery action persists, verify SPI communication with other SPI devices and assess the need for a BBIC system reset.

There are conditions detected by an ADRV9040 API function that only the BBIC can determine if it is a true error or not. An example is a data interface error counter threshold overflow. If a data interface counter overflows once an hour or once a month, only the BBIC can determine if the counter overflow constituted an actual error condition.

For the suggested application layer action, take the following steps:

1. Record the error.
2. Perform any BBIC determined recovery actions.

If an ADRV9040 API function has detected a condition in where the data interface (JESD) then further information can be attained by the error structure.

For the suggested application layer action, take the following steps:

1. Record the error.
2. 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. Apart from resetting the feature, it must be reconfigured to the state needed as well.

API Recovery Action—ADI_COMMON_ACT_ERR_RESET_MODULE

The ADI_COMMON_ACT_ERR_RESET_MODULE API recovery action is returned if the ADRV9040 API detects an issue in any of the modules.

For the ARM processor module, it requires a complete reset and reload of the ARM firmware. This type of action might be required if the communication interface to the ADRV904x ARM processor fails or the ARM watchdog timer reports an error. These events are not expected in production code but are fail-safe mechanisms in the event of a catastrophic error.

In case of an error, take the following steps:

1. Issue **adi_adrv904x_RxTxEnableSet()** to disable transmitter to keep hardware in a safe state. If this fails, a full ADRV904x reset is required.
2. Set PA and other RF front-end components in powered down/init state.
3. Call **adi_common_ErrorInfoGet()** to determine the specific error code and verify the error source. Log error code and source.
4. Dump the ADRV904x ARM memory if necessary for debug.
5. Dump the ADRV904x SPI registers if necessary for debug.
6. Reload the ADRV904x stream processor and ARM binary firmware files.
7. Continue with normal init sequence to run init calibrations and enable tracking calibrations.

Platform Layer—Adi.Adrv904x.CustomerPkg/api/src/c_src/platforms

The folder structure of ADRV904x platform layer code is shown in Figure 26. Here, the platform refers to the radio system that includes the baseband processor and the ADRV904x. The platform HAL refers to low-level device drivers in the baseband processor essential for the transceiver API to function.

SOFTWARE INTEGRATION

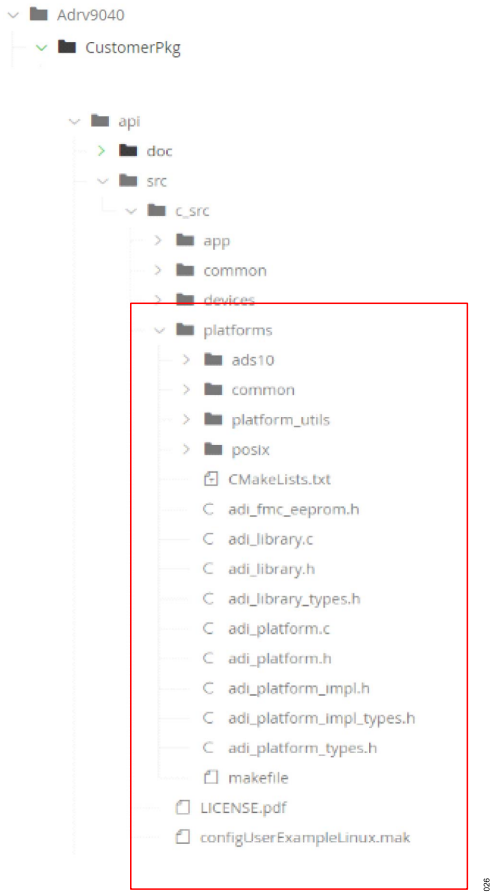


Figure 26. Expanded View of the ADRV904x Platform Layer Code Delivered in ADRV904x Software Package

The user must take the following actions to integrate the platform HAL with the ADRV904x API:

1. Examine the platform HAL interface used by the ADRV904x API.
2. Configure the platform HAL interface.
3. Implement the platform HAL interface.

The following sections 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 ADRV904x API functions.

The platform HAL interface is implemented as C function pointers, which 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 ADRV904x software package are shown in Figure 27.

SOFTWARE INTEGRATION

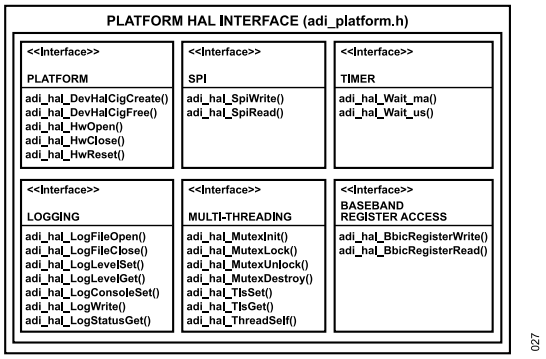


Figure 27. ADRV904x Platform HAL Interface Functions

A brief description of the platform HAL interface functions is provided in Table 10.

Table 10. ADRV904x Platform HAL Interface Function Description

Function Name	Purpose
adi_hal_HwOpen	Opens and initializes all platform drivers/resources and peripherals required to control the ADRV904x (SPI, timer, and logging).
adi_hal_HwClose	Closes any resources opened by **adi_hal_HwOpen**.
adi_hal_HwReset	Toggles the hardware reset signal for the ADRV904x.
adi_hal_SpiWrite	Writes 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	Reads an array of data bytes from a targeted SPI device (address bytes are provided by a transmitter data array which are packed into the byte array before calling this function).
adi_hal_Wait_us	Performs a wait/thread sleep in units of microseconds.
adi_hal_Wait_ms	Performs a wait/thread sleep in units of milliseconds.
adi_hal_LogFileOpen	Opens a file for logging.
adi_hal_LogLevelSet	Masks to set the severity of information to write to the log (error/warning/message).
adi_hal_LogLevelGet	Gets the current log level setting.
adi_hal_LogWrite	Logs a debug message (message, warning, error) from the API to the platform log.
adi_hal_LogFileClose	Closes the log file.
adi_hal_DevHalCfgCreate	Allows the platform to allocate and configures the **devHalCfg** structure. The **devHalCfg** structure is described in the Configuring the Platform HAL section.
adi_hal_DevHalCfgFree	Allows the platform to deallocate the **devHalCfg** structure
adi_hal_BbicRegisterRead, **adi_hal_BbicRegisterWrite**, **adi_hal_BbicRegistersRead**, and **adi_hal_BbicRegistersWrite**	Communicates with the baseband processor (typically FPGA).
adi_hal_MutexInit	Initializes the mutex into an unlocked state for multithreading applications.
adi_hal_MutexDestroy	Signas 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
adi_hal_TlsGet	Retrieves the value in the calling **thread's** thread local storage slot for the specified index.
adi_hal_ThreadSelf	Returns the ID of the current thread.

SOFTWARE INTEGRATION

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 Analog Devices evaluation platform is shown in the following code block. All the substructures used in **adi_hal_Cfg_t** are defined in **api/src/c_src/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 deallocate the memory, the **adi_hal_DevHalCfgDestroy()** function can be called. Since the HAL configuration is opaque to Analog Devices 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 as follows:

- **adi_hal_DevHalCfgCreate()**—interfaces defined are hardware/feature specific
- **adi_hal_HwOpen()**—enable all selected HAL interfaces and 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 10 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 platform HAL function is shown in Figure 28.

SOFTWARE INTEGRATION

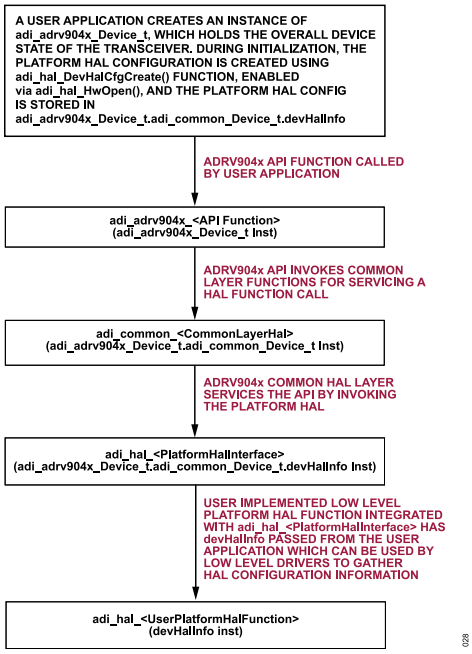


Figure 28. Flow of Platform HAL Configuration From a User Application to the Platform HAL Layer Function

Implementing the Platform HAL

Users who develop code to target custom hardware platforms use different drivers for the peripherals such as the SPI and timer specific to their platform. Users can refer to to the drivers developed for the Analog Devices evaluation platform as reference. Implementation of platform HAL must conform the function prototypes defined in **api/src/c_src/platforms/adi_platform.h**. A user implementation of all the functions defined in Table 10 is mandatory to guarantee the functionality of the ADRV904x.

The Analog Devices HAL interface is a library of functions that the ADRV904x API uses when it needs to access the target platform hardware as described in the Configuring the Platform HAL section. The software package delivered by Analog Devices contains drivers specific to the Analog Devices evaluation platform (ADS10-V1EBZ). The ADS10-V1EBZ platform HAL functions can be found in **api/src/c_src / platforms/ads10** folder shown in Figure 29.

SOFTWARE INTEGRATION

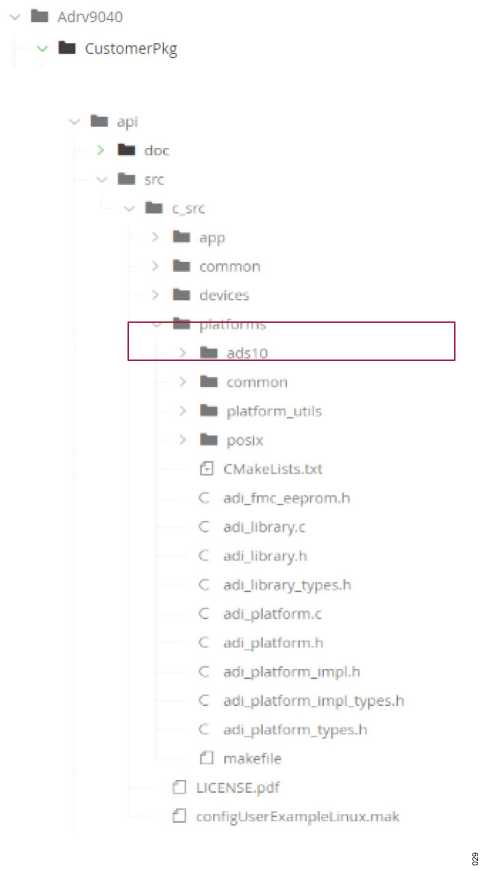


Figure 29. ADS10-V1EBZ Folder Containing Platform HAL Functions for Analog Devices Evaluation Platform

The user implements functions that must 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 Analog Devices evaluation system platform drivers are assigned to the platform HAL interface functions listed in Table 10.

The following code block is a snippet from **adi_hal_PlatformSetup()** function where the Analog Devices 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*/
```

SOFTWARE INTEGRATION

```
adi_hal_BbicRegisterRead = ads10_BbicRegisterRead;
adi_hal_BbicRegisterWrite = ads10_BbicRegisterWrite;
adi_hal_BbicRegistersRead = ads10_BbicRegistersRead;
adi_hal_BbicRegistersWrite = ads10_BbicRegistersWrite;
/* Multi-Threading support functions example */
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
```

Multithreading

The purpose of the multithreading feature is to support multithreaded applications using the ADRV904x API. It means that use of the API by a multithreaded 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 multithreading feature is independent of the transceiver itself; therefore, it is not configured at init time or run time as such. If an application is single threaded and, therefore, does not require the multithreading 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 30. Multithread support allows a single customer application process to divide tasks logically among several threads without having to 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 that if a customer has their own multithreading management environment and do not wish to use Analog Devices multithreading HAL functions, then the multithreading HAL functions in **adi_hal_PlatformSetup()** can be implemented as empty function stubs.

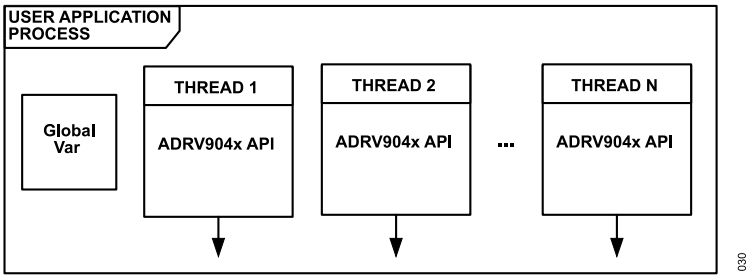


Figure 30. Multithreaded User Application Process Using ADRV904x API

The multithreading 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 10.

Each ADRV904x has 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_adrv904x_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

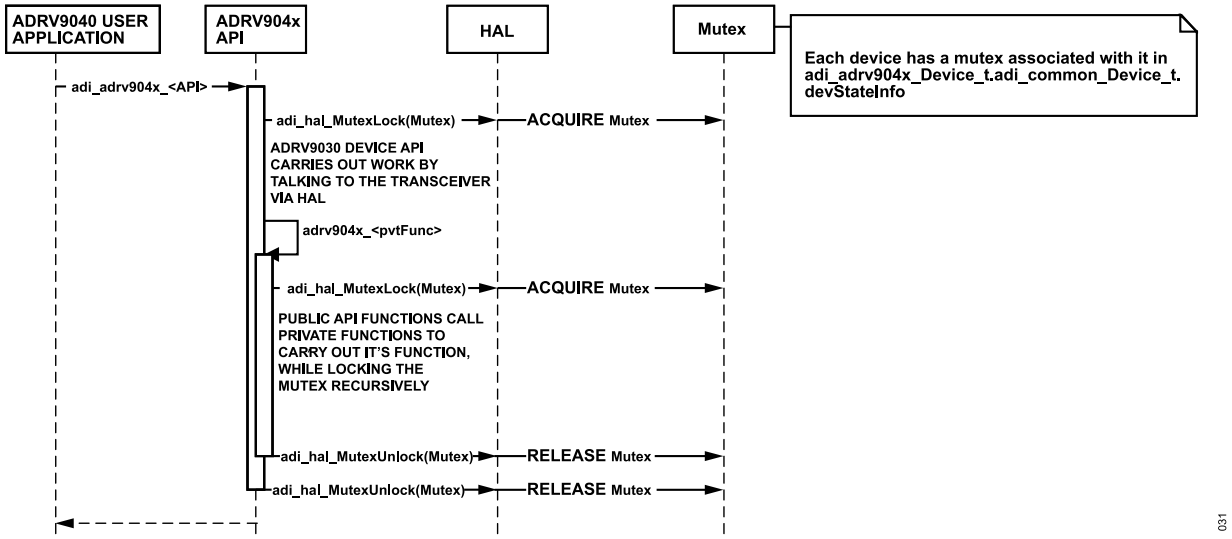


Figure 31. ADRV904x Mutex Operation Sequence

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

1. Examine the multithreading platform HAL functions in Table 10.
2. Implement a recursive mutex that can be used to lock shared state information recursively.
3. Integrate the user implemented platform HAL with Analog Devices multithreading functions in **api/src/c_src/platforms/adi_hal_platforms.c**.
4. Integrate the thread safe API functions into multiple threads in a user application. The API invokes the user implemented platform HAL mutex functions to acquire and release shared state information.

Error Handling Memory

Error handling was previously introduced in the Error Handling section of this document. In the context of a multithreading 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()** for thread safe error handling. This is illustrated in Figure 32.

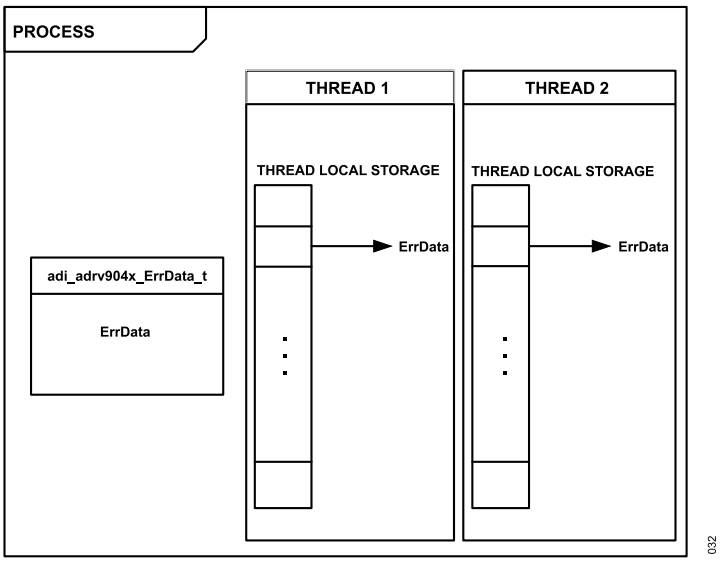


Figure 32. Illustration of Thread Local Storage

SOFTWARE INTEGRATION

In a typical use case, the user calls the **adi_hal_TlsSet()** function in a user application to associate the error handling memory (**adi_adrv904x_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 the **adi_hal_TlsGet()** function, and updates the error value from the thread in which the error has occurred.

Figure 33 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_ADRV9030_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;
}
```

033

Figure 33. Code Snippet for Error Handling Code and Error Data Retrieval

DEVELOPING AN APPLICATION

Once the ADRV904x 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 ADRV904x in order to run an application. The user can also refer to application examples supplied by Analog Devices in the directory **api/src/c_src/app/examples** that demonstrates how to configure and control the ADRV904x transceiver through the ADRV904x device API supplied by Analog Devices.

Instantiating the Device Data Structure

Device data structure holds all the state information of the ADRV904x. The device data structure is defined in **/c_src/api/public/adi_adrv904x_types.h**. The device data structure instance is passed to each ADRV904x API function call. An overview of the device data structure content is shown in Figure 34.

SOFTWARE INTEGRATION

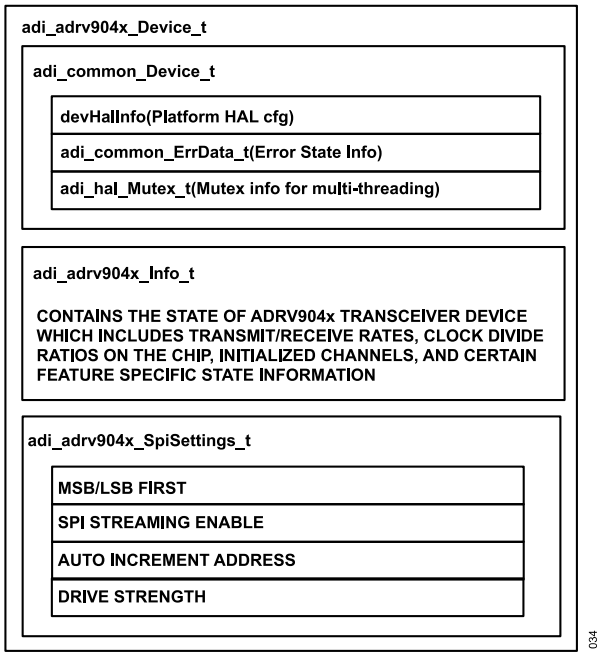


Figure 34. ADRV904x Device Data Structure

To support multiple devices the application layer code must instantiate **multiple adi_adrv904x_Device_t** structures to describe each physical ADRV904x device.

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 Configuring the Platform HAL section). For example, **devHalInfo** might contain information such as the SPI chip select to be used for the physical ADRV904x 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 ADRV904x is targeted (SPI chip select) when a particular ADRV904x API function is called. The developer may need to store other hardware information unique to a particular ADRV904x in this structure such as timer instances, log file information to allow for multithreading.

The **devStateInfo** member is of type **adi_adrv904x_Info_t** and is a run time state container for the ADRV904x API. The application layer must allocate memory for this structure, but only the ADRV904x 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 or ARM loaded), as well as a debug store for any run time data, such as error codes and error sources. It is not intended for the application layer to access the **devStateInfo** member directly, as API functions are 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 ADRV904x. The error structure is defined in Error Handling Memory. The user is also expected to initialize the ADRV904x SPI settings in the substructure **adi_adrv904x_SpiConfigSettings_t** in the device data structure.

Programming the Device

Before attempting to program the device, take the following steps:

1. Examine and integrate the ADRV904x API source code with the user application, implement platform HAL functions as described in the Platform HAL Interface section. Associate the Analog Devices platform HAL interface functions in **adi_hal_PlatformSetup( )** defined in the source file **api/src/c_src/platforms/adi_platform.c**.
2. Generate ADRV904x resource files as described in the Resource Files section, instantiate a variable of type **adi_adrv904x_TrxFileInfo_t** and initialize it with path to resource files and resource file settings.
3. Instantiate an ADRV904x 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.

SOFTWARE INTEGRATION

6. Instantiate a HAL configuration structure of type **adi_hal_Cfg_t** and initialize HAL configurations as described in the section Configuring the Platform HAL.
7. Configure the BBIC to the ADRV904x SPI bridge through the structure **adi_adrv904x_SpiConfigSettings_t** as described in the section SPI API Functions.
8. Ensure that a valid clock is being supplied to the ADRV904x transceiver.

After instantiating the device data structure, the next essential step is to program the ADRV904x. To program the device, the resource files described in the Error Handling Memory section are necessary. The programming mainly involves five stages as follows:

1. Creating the ADRV904x data structure, creating platform HAL configuration, and enabling the platform HAL.
2. In the pre multichip synchronization (MCS) initialization stage, the ADRV904x hardware blocks are initialized, the resource files (firmware binary, stream binary, profile, and receiver gain tables) are loaded and the embedded ARM processor boot up sequence is triggered.
3. In the MCS stage, the device requests system reference signal (SYSREF) pulses for synchronization from a clock device in the platform.
4. In the post MCS stage, the ADRV904x init time calibrations are performed, the LO frequencies are setup and the radio control mode is programmed.
5. In the JESD bring up stage, the JESD link between the baseband processor and the ADRV904x is initialized and the link is reset.

The sequence of events involved in programming the ADRV904x is shown in Figure 35.

SOFTWARE INTEGRATION

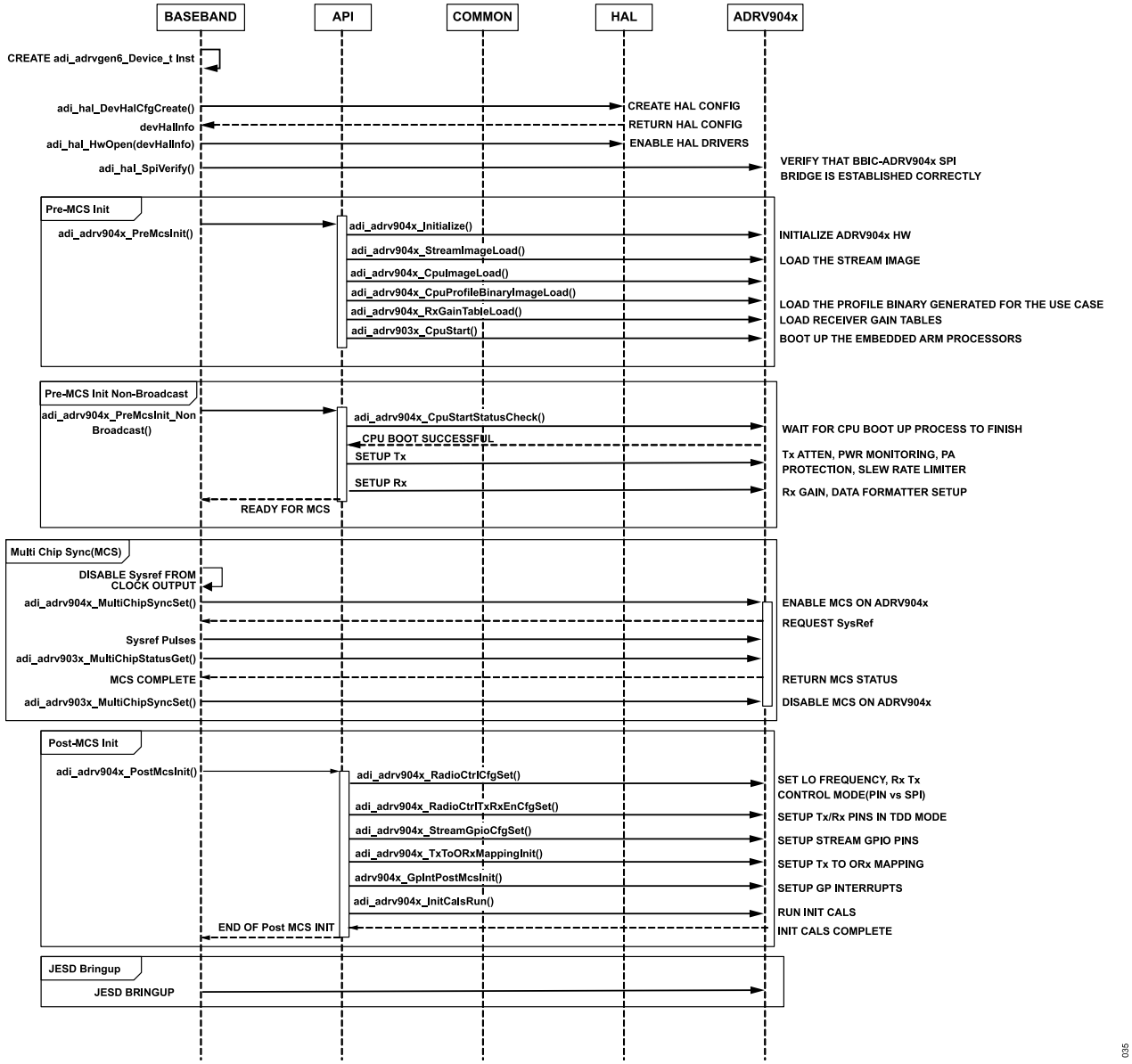


Figure 35. ADRV904x Programming Sequence

Refer to Figure 36 for the JESD bring up sequence.

SOFTWARE INTEGRATION

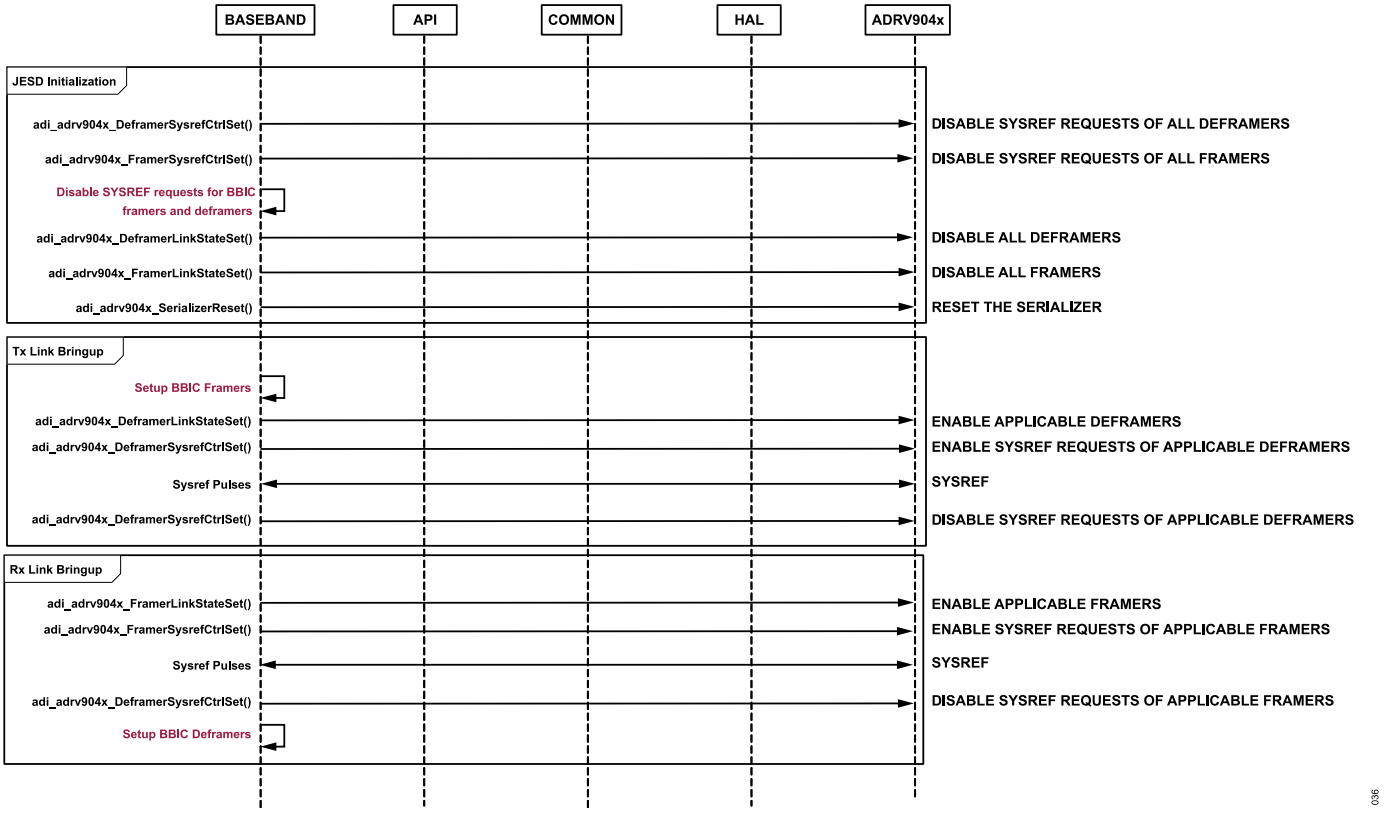


Figure 36. JESD Bring Up Sequence with ADRV904x

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 ADRV904x diagnostic data, such as PLL unlock, which can be monitored on a dedicated GP interrupt pin output from the ADRV904x.

The user can create an interface that detects the GP interrupts in the hardware, invoke GP interrupt handlers provided by Analog Devices that reside in **/devices/adrv904x/adi_adrv904x_gpio.h**. 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 of the Analog Devices 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 ADRV904x application project. A typical compilation flow of an ADRV904x application using Analog Devices provided makefiles is shown in Figure 37.

SOFTWARE INTEGRATION

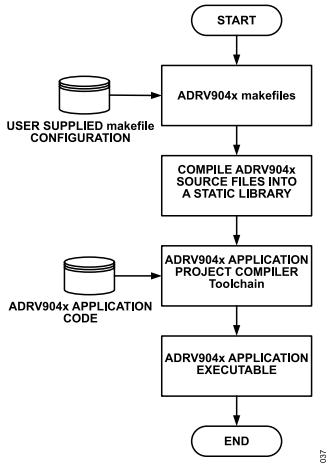


Figure 37. Compilation Flow Using ADRV904x Makefiles

Makefiles

There are four makefiles that are delivered as part of the Analog Devices software package shown in Table 11, which can be used to compile the Analog Devices source files into a static library for consumption in an ADRV904x application project. The static libraries are generated in an output directory whose name is specified by the user in the makefile configuration.

Table 11. ADRV904x Makefiles for Compilation

Target ADRV904x Source File	Makefile Path Relative to API Directory	Static Library Output File
ADRV904x Device API	**/src/c_src/devices/adi_adrv904x**	**/src/c_src/devices/adi_adrv904x/<OutputDir>/ libadi_adrv904x.a**
ADRV904x Device Common Layer	**/src/c_src/common**	**/src/c_src/common/<OutputDir>/libadi_common.a**
ADRV904x Device Platform Layer	**/src/c_src/platform**	**/src/c_src/platform/<OutputDir>/libadi_platform.a**
ADRV904x Bitfield Layer	**/src/c_src/devices/adi_adrv904x/private/bf**	**/src/c_src/devices/adi_adrv904x/private/bf<OutputDir>/ libadi_bf.a**

Note that **<OutputDir>** is specified by the user in the makefile configuration. The makefile configuration can be found in the Configuring the Makefiles section.

In order to compile the makefile, the user can use the command in the following code block.

```
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 ADRV904x source files. The mandatory configurations required by ADRV904x makefiles are listed in Table 12. The ADRV904x makefiles use C compilers by default. As part of the software package, an example makefile configuration file is **/api/src/c_src/ configUserExample.mak**. The example configuration file is based on the GNU compiler collection (GCC) and the configuration values used in the example configuration file are found in Table 12.

Table 12. Configurations Required by ADRV904x Makefile

Configuration	Description	Value Used in Analog Devices Example Configuration Files
BINARYDIR	The output folder name for the static library to be created in the same directory as the location of the makefile	-[1]
USE_ADS_10	This configuration is only applicable for compiling source codes for the Analog Devices evaluation platform; a value of 1 enables compilation for the Analog Devices evaluation platform	0

SOFTWARE INTEGRATION

Table 12. Configurations Required by ADRV904x Makefile (Continued)

Configuration	Description	Value Used in Analog Devices Example Configuration Files
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	-[1]
LDFLAGS	Linker flags	-Wl,-gc-sections -lgcov

[1] Nothing in the file.

In addition to the mandatory configurations required by the ADRV904x makefiles, the user can include additional makefile configurations specific to the project. The complete example makefile configuration listing is shown in Figure 38. The additional configurations used in the example ADRV904x project are included in the red box shown in Figure 38.

```
#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
```

038

Figure 38. ADRV904x Makefile Configuration Example

SERIAL PERIPHERAL INTERFACE (SPI)

The SPI bus allows a BBIC to control the ADRV904x. 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

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 $\overline{\text{SPI\_EN}}$ deasserting 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 Analog Devices 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 SPI_DIO 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 pin and SPI_DIO pin 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 multibyte I/O operation is allowed. This device cannot be used to control other devices on the bus—it only operates as a subordinate.

SERIAL PERIPHERAL INTERFACE (SPI)

The following are the two phases of 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 multibyte data transfer.

Phase 1 Instruction Format

Table 13 shows the information contained in the 16-bit control field.

Table 13. SPI Transaction Phase 1 Instruction Format

Bit Position	Shorthand Interpretation
D15	R/$\overline{W}$
D14:D0	A[14:0] (address word)

The bit positions are explained as follows:

- 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 I/O 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 I/O 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, $\overline{SPI\_EN}$ 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 the least significant bit (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 the next LSB to the most significant bit (MSB). The final bit in Phase 1 instruction signifies if the operation is read (set) or write (clear). If the operation is a write, the BBIC transmits the next eight bits LSB to MSB. If the operation is a read, the device transmits the next eight bits LSB to MSB.

If msbFirst = 1, then this is MSB first mode. In this mode, the first bit transmitted is the R/$\overline{W}$ 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 transmits the next eight bits MSB to LSB. If the operation is a read, the device transmits the next eight bits MSB to LSB.

Single-byte data transfer can occur where a single address/data transaction occurs between the $\overline{SPI\_EN}$ transition from low to high. This is called single instruction mode and is shown in Figure 39.

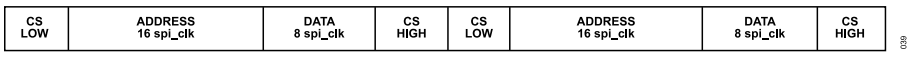


Figure 39. SPI Transactions Under Single Instruction Operation

Alternatively, it is possible to make multiple address/data transactions between the $\overline{SPI\_EN}$ transition from low to high. This is called single instruction buffered mode. This uses the transfer format of address followed by data (byte order: A A D A A D …) until the $\overline{SPI\_EN}$ signal is driven high. The address must be written for each data byte transfer when using this mode. This is shown in Figure 40.

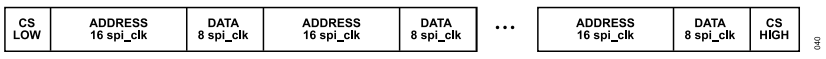


Figure 40. SPI Transactions Under Single Instruction Buffered Operation

SERIAL PERIPHERAL INTERFACE (SPI)

Multibyte Data Transfer (SPI Streaming)

Multibyte 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 multibyte 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 14. List of SPI API Functions

API Method Name	Comments
adi_adrv904x_SpiCfgSet()	Sets the configuration of the SPI interface.
adi_adrv904x_SpiCfgGet()	Reads the configuration of the SPI interface.

Configure the SPI interface by calling **adi_adrv904x_SpiCfgSet()**. The **adi_adrv904x_HwOpen()** command is the first command called before any other API interaction with the device. The **adi_adrv904x_HwOpen()** command calls **adi_adrv904x_HwReset()**, which issues a reset pin toggle followed by the configuration of the SPI interface via **adi_adrv904x_SpiCfgSet()**.

Users can configure SPI settings for the device with different SPI controller configurations by configuring member values of the **adi_adrv904x_SpiSettings_t data** structure. The **adi_adrv904x_SpiSettings_t data** structure is defined in the following code block.

```
typedef struct adi_adrv904x_SpiSettings
{
uint8_t                         msbFirst;
uint8_t                         enSpiStreaming;
uint8_t                         autoIncAddrUp;
uint8_t                         fourWireMode;
adi_adrv904x_CmosPadDrvStr_e    cmosPadDrvStrength;
} adi_adrv904x_SpiSettings_t;
```

The parameters for this structure are listed in Table 15.

Table 15. SPI Bus Setup Parameters

Structure Member	Value	Function	Default
msbFirst	0x00	LSB first.	0x01
	0x01	MSB 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	Autoincrement. Functionality intended to be used with SPI streaming. Sets address autoincrement -> next addr = addr + 4	0x01
	0x01	Autodecrement. Functionality intended to be used with SPI streaming. Sets address autodecrement -> next addr = addr − 4	
fourWireMode	0x00	SPI hardware implementation. Use 3-wire SPI (SPI_DIO pin is bidirectional). Figure 41 shows example of SPI 3-wire mode of operation.	0x01
	0x01	SPI hardware implementation. Use 4-wire SPI. Figure 42 and Figure 43 show examples of SPI 4-wire mode of operation. Note that the default mode for the Analog Devices FPGA platform is 4-wire mode.	
cmosPadDrvStrength	0x00 ADI_ADRV904X_CMOSPAD_DRV_WEAK	Applies to all output CMOS pins (GPIO, SPI_SDO, GPINT). 5 pF load @ 75 MHz.	0x01
	0x01 ADI_ADRV904X_CMOSPAD_DRV_STRONG	Applies to all output CMOS pins (GPIO, SPI_SDO, GPINT). 100 pF load @ 100 MHz.	

SERIAL PERIPHERAL INTERFACE (SPI)

Table 16 lists the timing specifications for the SPI bus. The relationship between these parameters is in Figure 41. 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, therefore, the bus ready parameter after the data is driven from the device (t_{HZS}) is not shown in the diagram.

Table 16. SPI Bus Timing Constraint Values

Parameter	Min	Typical	Max	Description
t_{CP}	20 ns			SPI_CLK cycle time (clock period)
t_{MP}	5 ns			SPI_CLK pulse width
t_{SC}	0.5 ns			$\overline{\text{SPI\_EN}}$ setup time to first SPI_CLK rising edge
t_{HC}	0.5 ns			Last SPI_CLK falling edge to $\overline{\text{SPI\_EN}}$ hold
t_S	1 ns			SPI_DIO data input setup time to SPI_CLK
t_H	1 ns			SPI_DIO data input hold time to SPI_CLK
t_{CO}[1]	4 ns		6 ns	SPI_CLK falling edge to output data delay (3-wire mode or 4-wire mode)
t_{HZM}	$t_{CO\ (min)}$		$t_{CO\ (max)}$	Bus turnaround time after BBIC drives the last address bit
t_{HZS}	$t_{CO\ (min)}$		$t_{CO\ (max)}$	Bus turnaround time after device drives the last data bit

[1] t_{co} value is mentioned in the above tables requires the highest drive strength settings to achieve 4 ns to 6 ns.

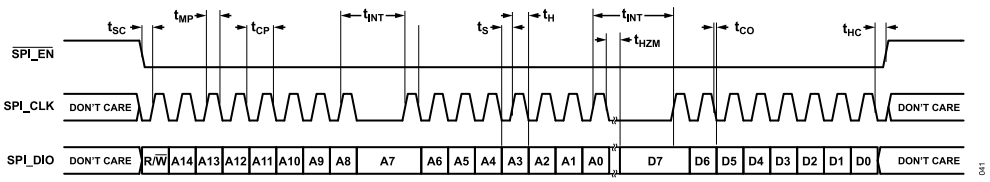


Figure 41. 3-Wire SPI Timing with Parameter Labels

TIMING DIAGRAM EXAMPLES

The diagrams in Figure 42 and Figure 43 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 are performed with a 3-wire bus, the SPI_DO line in Figure 42 is eliminated, and the SPI_DIO and SPI_DO signal in Figure 43 are 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.

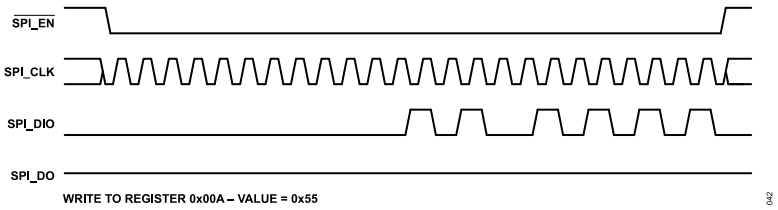


Figure 42. Nominal Timing Diagram, SPI Write Operation

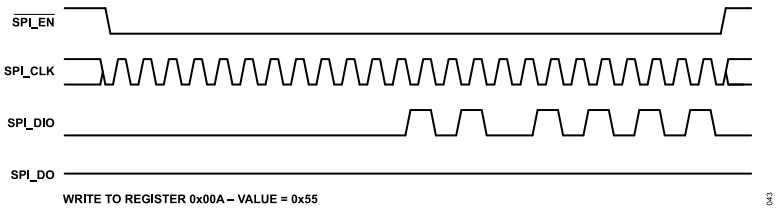


Figure 43. Nominal Timing Diagram, SPI Read Operation

AUXILIARY SPI OVERVIEW

The ADRV904x features an optional secondary SPI Subordinate 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 in Table 17.

Table 17. Allowed Pins for Auxiliary Interface

Auxiliary 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 run-time interactions. When the auxiliary SPI interface is enabled, the selected GPIO pins are reserved only for auxiliary SPI utilization.

AUXILIARY SPI API FUNCTIONS

Table 18. List of Auxiliary SPI Related API Functions

API Method Name	Comments
adi_adrv904x_AuxSpiCfgSet()	Sets the enable/disable and configuration of the auxiliary SPI interface.
adi_adrv904x_AuxSpiCfgGet()	Returns the enable/disable status and current configuration of the auxiliary SPI interface.

JESD204B/JESD204C INTERFACE

The ADRV904x uses a SERDES high-speed serial interface based on the JESD204B/JESD204C 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 ADRV904x and the BBIC.

The ADRV904x supports Device Subclass 0 and Device Subclass 1 operation modes. In Devices Subclass 0, deterministic latency is not supported. In Device 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 44. 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 electromagnetic interference (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 (JESD204B and JESD204C), and 66-bit characters in the case of 64B/66B encoding (JESD204C only). The physical layer is responsible for the transmission and reception of characters at the lane bit rate.

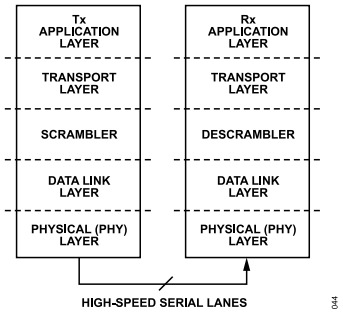


Figure 44. Key Blocks of the JESD204B/JESD204C Standard

Figure 45 and Figure 46 illustrate how the JESD204 transmit and receive protocols are implemented.

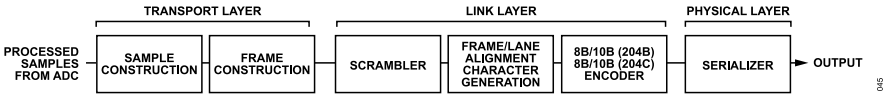


Figure 45. JESD204B/JESD204C Framer (JTX)

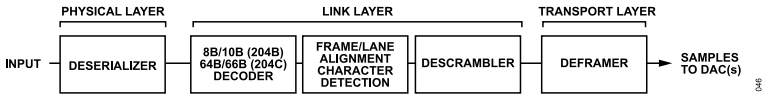


Figure 46. JESD204B/JESD204C Deframer (JRX)

The data interface blocks in the ADRV904x can operate in either JESD204B or JESD204C modes. Fewer number of lanes may be needed when operating in JESD204C, which results in simpler PCB layout and lower power consumption.

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

JESD204B/JESD204C INTERFACE

deterministic latency. In Device 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 Device 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)
- Forward error correction (FEC) signal (error detection and correction, only supported on ADRV904x framers)
- Command channel (transmitting commands and status information)

Refer to the JESD204B and JESD204C specifications for further details.

JESD204B/JESD204C 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 ADRV904x 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 (receiver/observation receiver data time multiplexed on the same lane according to the receiver to transmitter frame timing) for up to eight lanes. To assist in debugging, they contain internal data generators, allowing several test patterns and pseudo random binary sequence (PRBS) patterns to be sent across the link.

Figure 47 provides a high level overview of the framers present in the ADRV904x. The data samples available to the framers come from the main RF receiver datapaths and from the observation receiver datapaths. Further details are provided in Figure 47.

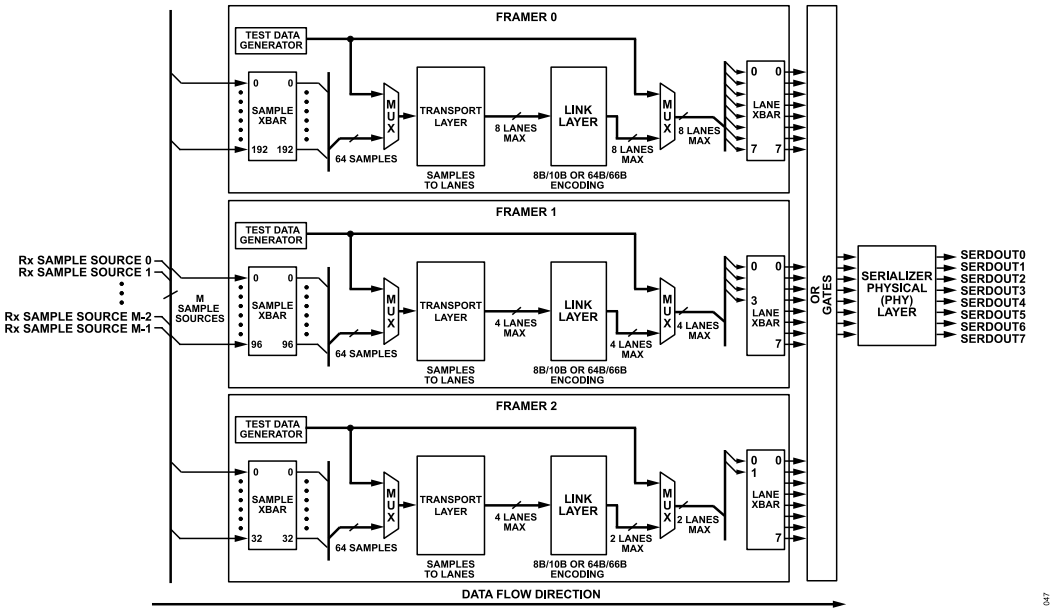


Figure 47. High Level Overview of the JESD204B/JESD204C Framers

JESD204B/JESD204C INTERFACE

JESD Terminology

CGS—Code group synchronization.

CF—Determines if control bits are appended to a sample or mapped into a separate control word and appended at the end of each frame cycle. It is the number of control words in a frame.

CS—Determines the number of control bits needed to be transferred per sample in frame cycle. Control bits are appended at the end of each sample if CF = 0 or at the end of the frame cycle if CF = 1. CF is always 0 for Analog Devices JESD204B capable ADC devices.

E—The number of multiblocks in an extended multiblock.

F—Number of octets per lane in a frame cycle.

HD—Parameter controls if sample information is contained within a lane or if it can be divided over more lanes.

K—Number of frames per multiframe or extended multiblock.

L—Number of Lanes to encode data into per frame cycle.

LMFC—Local multiframe clock.

LEMC—Local extended multiblock clock.

M—Number of converter devices.

N—Converter resolution.

N'—The converter resolution plus any control and tail bits for padding. It is an integer multiple of 4.

NS—Number of samples per converter per converter clock cycle.

RD—Running disparity.

S—Number of samples per converter per frame cycle.

SERDES—Serializer/deserializer.

JESD204B/JESD204C Framers Parameters

Table 19 provides a list of the supported framer parameter values. Note that not all combinations of those framer parameter values are supported.

Table 19. List of JESD204B/JESD204C Framers Parameters

Parameter	Parameter Description	Possible Parameter Values[1]	Other Values
M	Number of converter devices.	2 to 192[1] (Framer 0) 2 to 96[1] (Framer 1) 2 to 32[1] (Framer 2)	
L	Number of lanes to transmit and encode data on.	0 to 7	Power two values are only valid (1,- 2, 4, 8)
F	JESD204B—number of octets per lane in a frame cycle; JESD204C— used in conjunction with K to set the extended multiblock period. F is calculated as in JESD204B mode.	JESD204C 2 to 32. Valid configuration: F = M × Np × S ÷ (8 × L)	JESD204B can only support F values given below 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 multi frame; JESD204C—used in conjunction with F to set the extended multiblock period.	1 to 32. F × K must be a multiple of four and (20 ≤ F × K ≤ 256)K × F = 256 × E, K ≤ 256	
E	JESD204C only—number of blocks in an extended multiblock.	1, 3, 5	
CS	Number of control bits per converter sample.	0 to 3	

JESD204B/JESD204C INTERFACE

Table 19. List of JESD204B/JESD204C Framers Parameters (Continued)

Parameter	Parameter Description	Possible Parameter Values[1]	Other Values
HD	High density mode.	0, 1	

[1] M - Converter max if CDUC/CDDC bypassed is 32.

Due to the digital downconversion (DDC) bands and CDDC blocks in the receiver datapaths, there are more data sample sources than the number of 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. Therefore, the possible M values can 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 ADRV904x to allow the selection and mapping of data samples from 128 data sample sources. Figure 48 illustrates the possible data sample sources available to each sample crossbars.

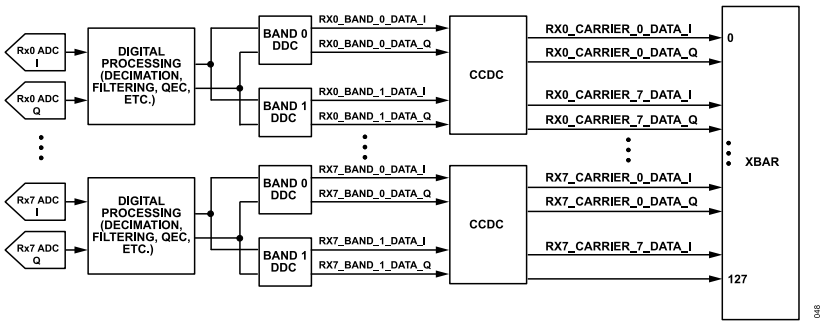


Figure 48. Sample Data Sources Available to Each Sample Crossbar

The data sample sources can be categorized as data samples from one of the CDDC bands within a receiver datapath (eight receiver datapaths, one component carrier per receiver datapath), and 16 data samples going through each CDDC datapaths, resulting in 128 possible data sample sources.

The number of virtual converters on the receive side depends on the sampling rate. Each carrier is mapped to a virtual pair of 30.72 MSPS DAC converters. For example, if each receiver is receiving three carriers at 30.72 MSPS, then the total number of virtual converters = 8 channels × 3 carriers/channel × 2 converters/carrier = 48 virtual converters. To consider another example, if each receiver is receiving three carriers at 61.44 MSPS, then the total number of virtual converters = 8 channels × 3 carriers/channel × 2 converters/carrier × 2 = 96 virtual converters, where the last factor of two is sample rate (61.44 MSPS)/virtual converter rate (30.72 MSPS).

Table 20 provides a list of all the sample crossbar input value along with the corresponding sample data sources.

Table 20. Sample Crossbar Input Selection Values

Sample Crossbar Input Value	Data Sample Source
0	RX0_CDDC_C0_DATA_I
1	RX0_CDDC_C0_DATA_Q
2	RX0_CDDC_C1_DATA_I
3	RX0_CDDC_C1_DATA_Q
4	RX0_CDDC_C2_DATA_I
5	RX0_CDDC_C2_DATA_Q
6	RX0_CDDC_C3_DATA_I
7	RX0_CDDC_C3_DATA_Q
8	RX0_CDDC_C4_DATA_I
9	RX0_CDDC_C4_DATA_Q
10	RX0_CDDC_C5_DATA_I
11	RX0_CDDC_C5_DATA_Q
12	RX0_CDDC_C6_DATA_I

JESD204B/JESD204C INTERFACE

Table 20. Sample Crossbar Input Selection Values (Continued)

Sample Crossbar Input Value	Data Sample Source
13	RX0_CDDC_C6_DATA_Q
14	RX0_CDDC_C7_DATA_I
15	RX0_CDDC_C7_DATA_Q
16	RX1_CDDC_C0_DATA_I
17	RX1_CDDC_C0_DATA_Q
18	RX1_CDDC_C1_DATA_I
19	RX1_CDDC_C1_DATA_Q
20	RX1_CDDC_C2_DATA_I
21	RX1_CDDC_C2_DATA_Q
22	RX1_CDDC_C3_DATA_I
23	RX1_CDDC_C3_DATA_Q
24	RX1_CDDC_C4_DATA_I
25	RX1_CDDC_C4_DATA_Q
26	RX1_CDDC_C5_DATA_I
27	RX1_CDDC_C5_DATA_Q
28	RX1_CDDC_C6_DATA_I
29	RX1_CDDC_C6_DATA_Q
30	RX1_CDDC_C7_DATA_I
31	RX1_CDDC_C7_DATA_Q
32	RX2_CDDC_C0_DATA_I
33	RX2_CDDC_C0_DATA_Q
34	RX2_CDDC_C1_DATA_I
35	RX2_CDDC_C1_DATA_Q
36	RX2_CDDC_C2_DATA_I
37	RX2_CDDC_C2_DATA_Q
38	RX2_CDDC_C3_DATA_I
39	RX2_CDDC_C3_DATA_Q
40	RX2_CDDC_C4_DATA_I
41	RX2_CDDC_C4_DATA_Q
42	RX2_CDDC_C5_DATA_I
43	RX2_CDDC_C5_DATA_Q
44	RX2_CDDC_C6_DATA_I
45	RX2_CDDC_C6_DATA_Q
46	RX2_CDDC_C7_DATA_I
47	RX2_CDDC_C7_DATA_Q
48	RX3_CDDC_C0_DATA_I
49	RX3_CDDC_C0_DATA_Q
50	RX3_CDDC_C1_DATA_I
51	RX3_CDDC_C1_DATA_Q
52	RX3_CDDC_C2_DATA_I
53	RX3_CDDC_C2_DATA_Q
54	RX3_CDDC_C3_DATA_I
55	RX3_CDDC_C3_DATA_Q
56	RX3_CDDC_C4_DATA_I
57	RX3_CDDC_C4_DATA_Q
58	RX3_CDDC_C5_DATA_I
59	RX3_CDDC_C5_DATA_Q

JESD204B/JESD204C INTERFACE

Table 20. Sample Crossbar Input Selection Values (Continued)

Sample Crossbar Input Value	Data Sample Source
60	RX3_CDDC_C6_DATA_I
61	RX3_CDDC_C6_DATA_Q
62	RX3_CDDC_C7_DATA_I
63	RX3_CDDC_C7_DATA_Q
64	RX4_CDDC_C0_DATA_I
65	RX4_CDDC_C0_DATA_Q
66	RX4_CDDC_C1_DATA_I
67	RX4_CDDC_C1_DATA_Q
68	RX4_CDDC_C2_DATA_I
69	RX4_CDDC_C2_DATA_Q
70	RX4_CDDC_C3_DATA_I
71	RX4_CDDC_C3_DATA_Q
72	RX4_CDDC_C4_DATA_I
73	RX4_CDDC_C4_DATA_Q
74	RX4_CDDC_C5_DATA_I
75	RX4_CDDC_C5_DATA_Q
76	RX4_CDDC_C6_DATA_I
77	RX4_CDDC_C6_DATA_Q
78	RX4_CDDC_C7_DATA_I
79	RX4_CDDC_C7_DATA_Q
80	RX5_CDDC_C0_DATA_I
81	RX5_CDDC_C0_DATA_Q
82	RX5_CDDC_C1_DATA_I
83	RX5_CDDC_C1_DATA_Q
84	RX5_CDDC_C2_DATA_I
85	RX5_CDDC_C2_DATA_Q
86	RX5_CDDC_C3_DATA_I
87	RX5_CDDC_C3_DATA_Q
88	RX5_CDDC_C4_DATA_I
89	RX5_CDDC_C4_DATA_Q
90	RX5_CDDC_C5_DATA_I
91	RX5_CDDC_C5_DATA_Q
92	RX5_CDDC_C6_DATA_I
93	RX5_CDDC_C6_DATA_Q
94	RX5_CDDC_C7_DATA_I
95	RX5_CDDC_C7_DATA_Q
96	RX6_CDDC_C0_DATA_I
97	RX6_CDDC_C0_DATA_Q
98	RX6_CDDC_C1_DATA_I
99	RX6_CDDC_C1_DATA_Q
100	RX6_CDDC_C2_DATA_I
101	RX6_CDDC_C2_DATA_Q
102	RX6_CDDC_C3_DATA_I
103	RX6_CDDC_C3_DATA_Q
104	RX6_CDDC_C4_DATA_I
105	RX6_CDDC_C4_DATA_Q
106	RX6_CDDC_C5_DATA_I

JESD204B/JESD204C INTERFACE

Table 20. Sample Crossbar Input Selection Values (Continued)

Sample Crossbar Input Value	Data Sample Source
107	RX6_CDDC_C5_DATA_Q
108	RX6_CDDC_C6_DATA_I
109	RX6_CDDC_C6_DATA_Q
110	RX6_CDDC_C7_DATA_I
111	RX6_CDDC_C7_DATA_Q
112	RX7_CDDC_C0_DATA_I
113	RX7_CDDC_C0_DATA_Q
114	RX7_CDDC_C1_DATA_I
115	RX7_CDDC_C1_DATA_Q
116	RX7_CDDC_C2_DATA_I
117	RX7_CDDC_C2_DATA_Q
118	RX7_CDDC_C3_DATA_I
119	RX7_CDDC_C3_DATA_Q
120	RX7_CDDC_C4_DATA_I
121	RX7_CDDC_C4_DATA_Q
122	RX7_CDDC_C5_DATA_I
123	RX7_CDDC_C5_DATA_Q
124	RX7_CDDC_C6_DATA_I
125	RX7_CDDC_C6_DATA_Q
126	RX7_CDDC_C7_DATA_I
127	RX7_CDDC_C7_DATA_Q

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 49 illustrates the transport layer in a framer. The transport layer also handles the SYSREF signal that is used to achieve deterministic latency.

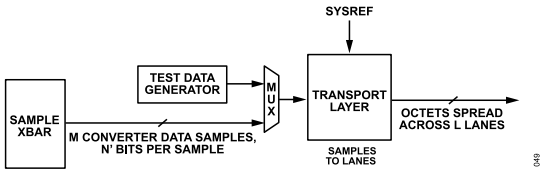


Figure 49. High Level Overview of the Transport Layer in a Framer

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 50 and Figure 51 provide two examples of converter data samples mapping by the transport layer for two different sets of M, N', S, and L parameters.

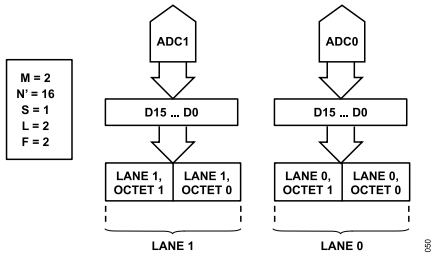


Figure 50. Converter Data Samples Mapping Example with M = 2, N' = 16, S = 1, and L = 2

JESD204B/JESD204C INTERFACE

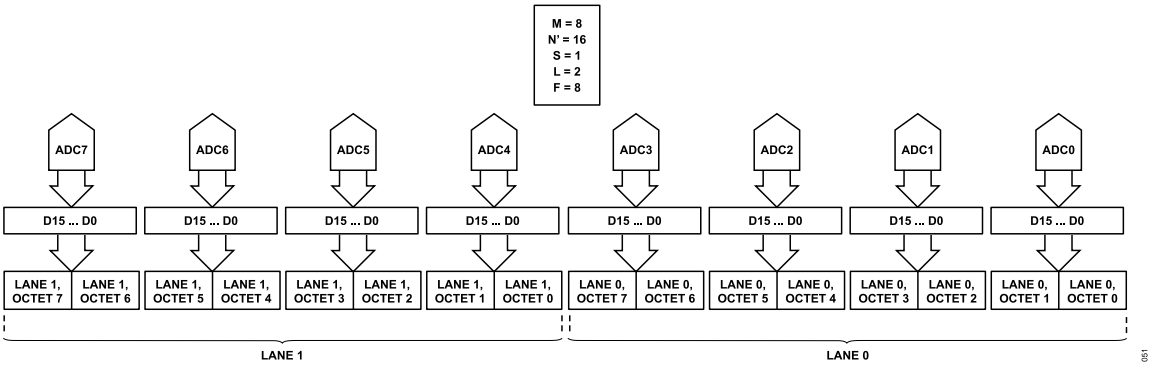


Figure 51. Converter Data Samples Mapping Example with M = 8, N' = 16, S = 1, and L = 2

Figure 52 is an example mapping for 128 converters.

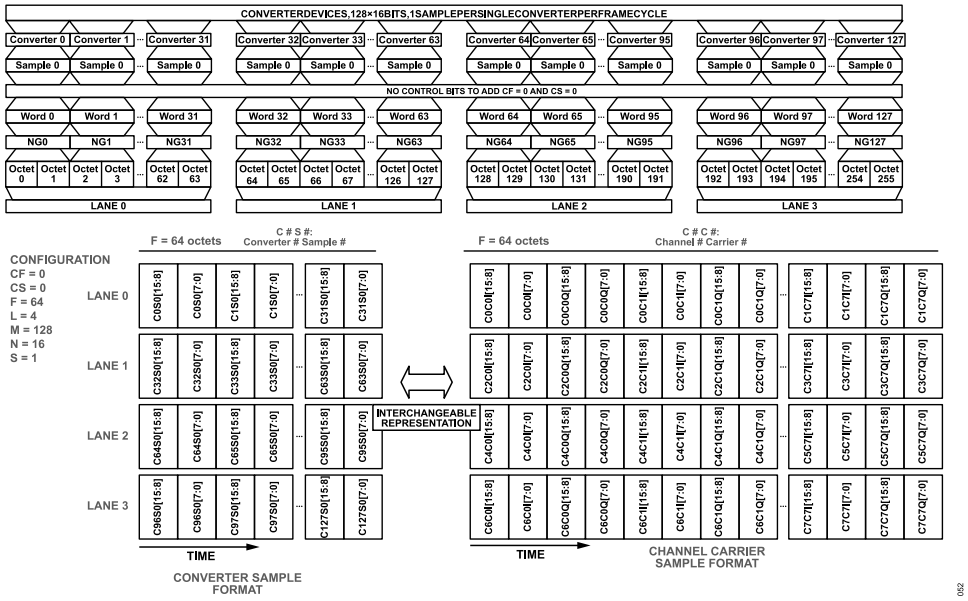


Figure 52. Converter Data Samples Mapping Example with M = 8, N' = 16, S = 1 and L = 2

SYSREF Signal

The SYSREF signal provides a timestamp used to achieve deterministic latency and multichip synchronization. For all three framers in the ADRV904x, 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. The link layer handles CGS, generates ILAS when required, and 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 cyclic redundancy check (CRC) bits or FEC bits.

Figure 53 provides a high level overview of the link layer in a framer.

JESD204B/JESD204C INTERFACE

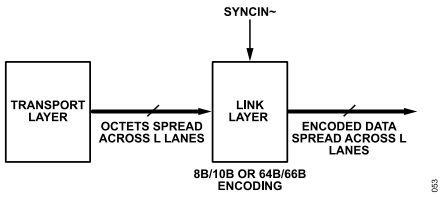


Figure 53. High Level Overview of the Link Layer in a Framer

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 ADRV904x, the SYNCIN~ signal is handled by the link layer in JESD204B mode. The ADRV904x 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 54.

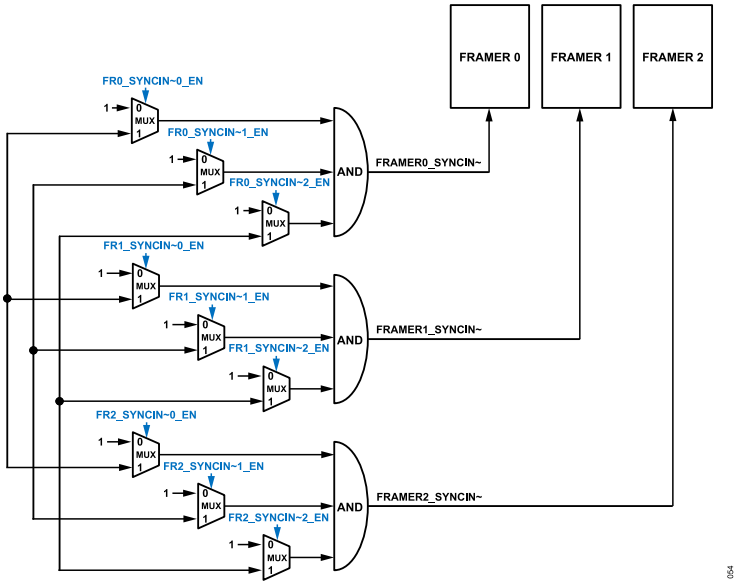


Figure 54. SYNCIN~ Crossbar

Lane Crossbars

Each of the ADRV904x 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 and two logical output lanes to any of its eight output lanes. This is illustrated in Figure 55 for Framer 0, Figure 56 for Framer 1 and Figure 57 for Framer 2.

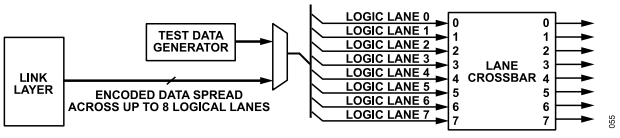


Figure 55. Framer 0 Lane Crossbar

JESD204B/JESD204C INTERFACE

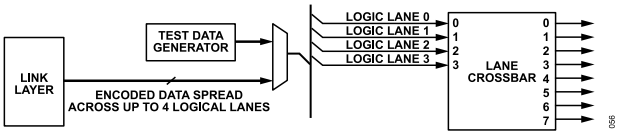


Figure 56. Framer 1 Lane Crossbar

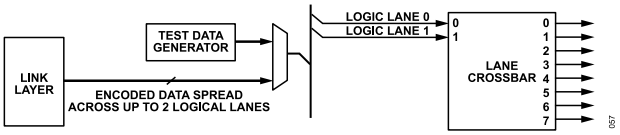


Figure 57. Framer 2 Lane Crossbar

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 58 illustrates this for Lane 0.

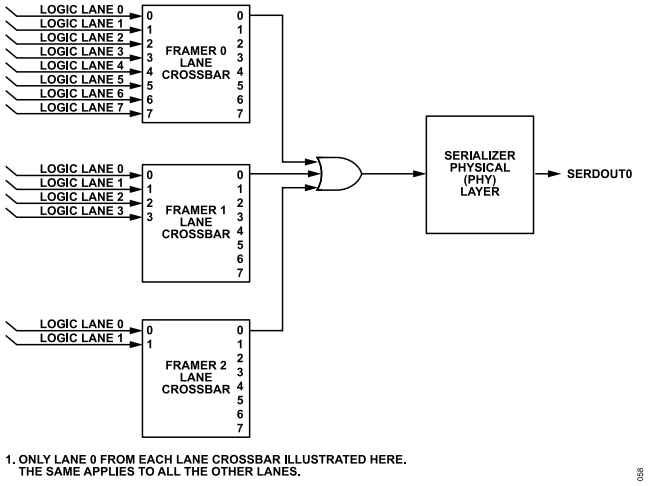


Figure 58. OR Gate Example for Lane 0

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:

- At the input of the transport layer (enables checking of transport layer, link layer, and serializer PHY all together).
- 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.

JESD204B/JESD204C INTERFACE

- 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.

I/Q Sample Rate and Output Lane Data Rate Relationship Between Framers

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

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

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

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

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

The three framers can operate at different I/Q sample rates. The ratio of the I/Q 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. Receiver data is put on the lanes in the receive time slot and observation receiver data is put on the same lanes during transmit time slots.

Figure 59 shows an example where Framer 0 is used to handle the receiver data and the observation receiver data in link sharing mode. The same four physical lanes are used to transmit the receiver data during the receive time slots, and the observation receiver data during the transmit time slots.

Link sharing is supported on Framer 0 and Framer 1 only. For link sharing in JESD204B mode , F/L must be same for both receiver and observation receiver. Similarly for JESD204C, F can be different but E/L must be same for receiver and observation receiver.

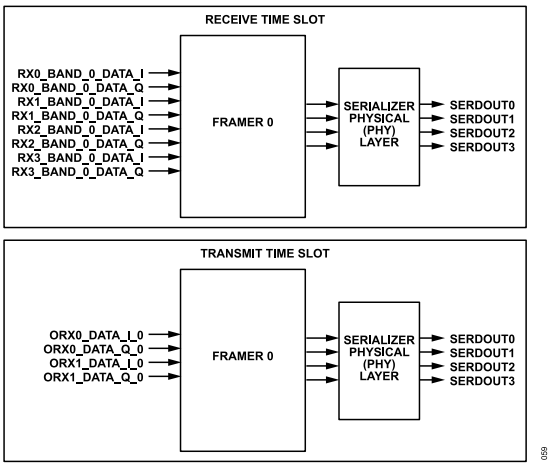


Figure 59. Link Sharing Example

JESD204B/JESD204C INTERFACE

JESD204B/JESD204C 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 DACs of transmitter paths.

The ADRV904x 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 signal integrity.

Lane crossbar settings is between the deserializer and the deframer in the transmitter path. Sample crossbar settings between the deframer output and the DAC input.

The number of virtual digital mapping to the analog converters on the transmitter side is explained as follows.

Each carrier is mapped to a virtual pair of 30.72 MSPS DAC converters. For example, if each transmitter is transmitting three carriers at 30.72 MSPS, then the total number of virtual converters = 8 channels × 3 carriers/channel × 2 converters/carrier = 48 virtual converters. To consider another example, if each transmitter is transmitting three carriers at 61.44 MSPS, then the total number of virtual converters = 8 channels × 3 carriers/channel × 2 converters/carrier × 2 = 96 virtual converters, where the last factor of 2 is sample rate (61.44 MSPS)/virtual converter rate (30.72 MSPS).

Figure 60 provides a high level overview of the deframers in ADRV904x. Further details are provided in the sections that follow. Note that the flow of data in Figure 60 and Figure 61 is from right to left.

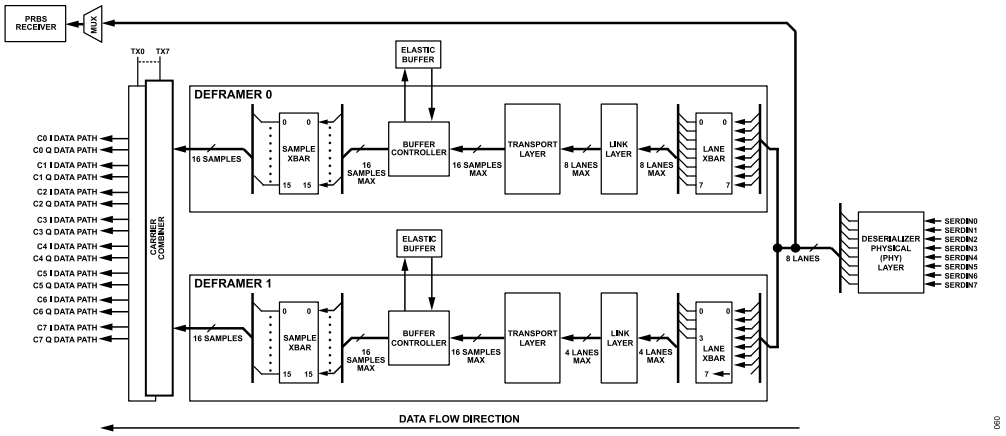


Figure 60. High Level Overview of Deframers

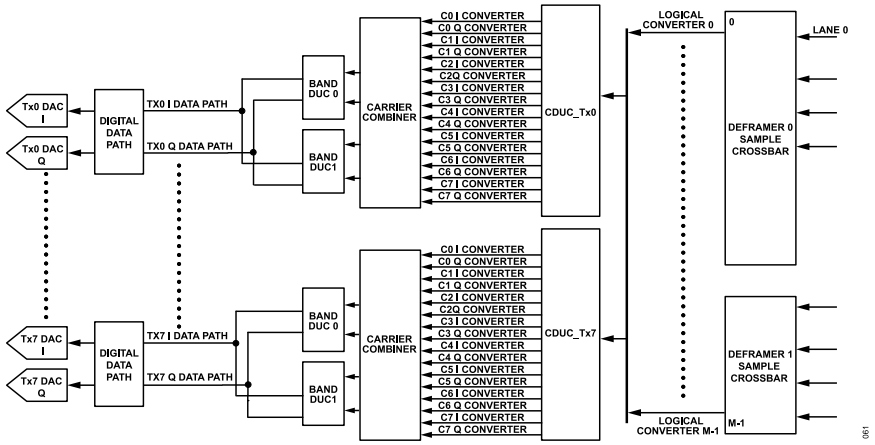


Figure 61. ADRV904x Digital Path Overview

JESD204B/JESD204C INTERFACE

JESD204B/JESD204C Deframers Parameters

Table 21 provides a list of the supported deframer parameter values. Note that not all combinations of those deframer parameter values are supported.

Table 21. List of JESD204B/C Deframers Parameters

Parameter	Parameter Description	Possible Parameter Values[1]
M	Number of converter devices.	1 to 192 (Deframer 0), 1 to 96 (Deframer 1)
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. 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, 322, 3, 4, 6, 8, 12, 16, 32
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. JESD204C—used in conjunction with F to set the extended multiblock period.	1 to 32. F × K must be a multiple of four and (20 ≤ F × K ≤ 256)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 ADRV904x 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. 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 62 shows the lane crossbars in detail.

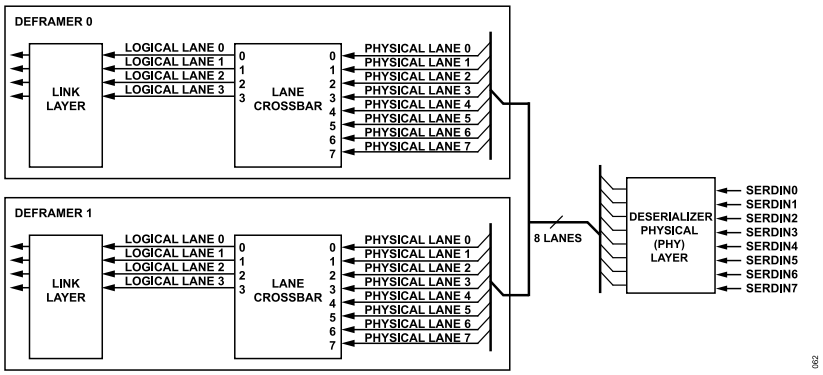


Figure 62. Deframer 0 and Deframer 1 Lane Crossbars

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 CGS, receives and verifies ILAS and performs 8B/10B decoding on the data received from the SERDES input lanes. It also performs character reinsertion 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 63 provides a high level overview of the link layer in a framer.

JESD204B/JESD204C INTERFACE

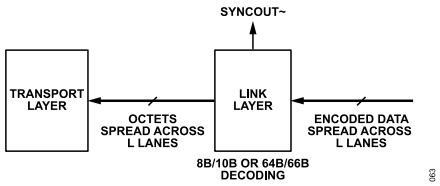


Figure 63. High Level Overview of the Link Layer in a Deframer

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 ADRV904x 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 64.

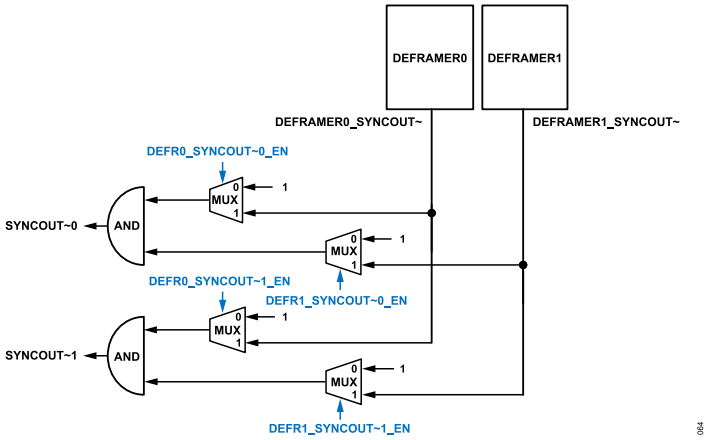


Figure 64. Deframer to SYNCOUT~ Output Options

Transport Layer

The transport layer of a deframer is responsible for deskewing 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 65 shows the transport layer in a deframer.

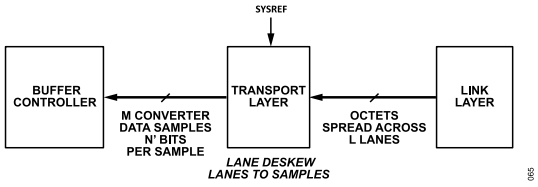


Figure 65. High Level Overview of the Transport Layer in a Deframer

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 are packed and encoded before being sent across the SERDES link.

The PRBS receiver can be set in the following modes:

JESD204B/JESD204C INTERFACE

- 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.

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 ADRV904x during initialization.

JESD PHY LAYER

The ADRV904x 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 the ADRV904x is 32,440.32 Mbps.

The JESD physical layer establishes a reliable channel between the transmitter and the receiver. A high level view of a typical JESD serial link is shown in Figure 66.

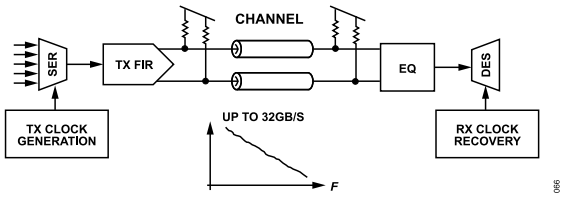


Figure 66. Overview of a Typical High Speed Serial Link

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 (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. Therefore, a clock data recovery block is needed on the receiver side to find the optimum sampling phase of the clock. In Figure 66, note the frequency response of the channel. At higher data rates (see Figure 66) large losses are seen across the channel causing the received signal to be attenuated resulting in large intersymbol 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 67 shows an impulse response (data=..010000..) at the end of a lossy transmission line. Each color represents a different bit period. All the energy must 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 precursor, whereas the time slots after the impulse are referred to as first, second, and so on post cursors.

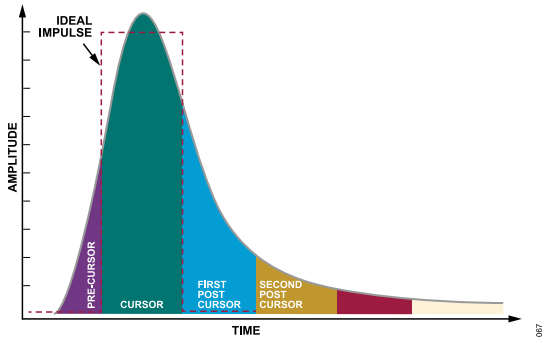


Figure 67. Intersymbol Interference of an Impulse Response

JESD204B/JESD204C INTERFACE

In order to overcome insertion loss caused by ISI, a preemphasis and deemphasis (or equalization) circuits are used. An equalizer on the receiver side helps with deemphasis, which subtracts the energy in the precursors and post cursors. Therefore, only the energy at the cursor remains after the signal travels through the channel.

For both equalization and preemphasis, 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 (preemphasis) or at the output of the channel (equalization), shown in Figure 68. The top diagram shows an impulse as it goes into the channel which acts as a low-pass filter (LPF). 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.

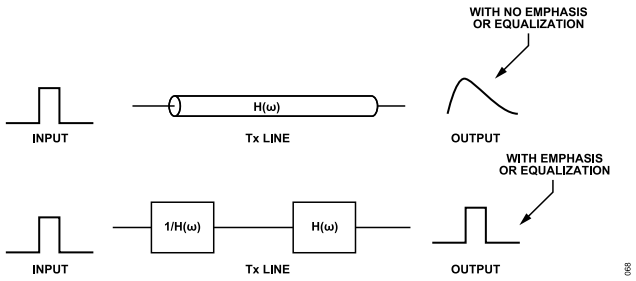


Figure 68. Theoretical Diagram of the Operation of Preemphasis and Deemphasis

The serializer implements a 3-tap finite impulse response (FIR) filter, whereas the deserializer implements continuous time linear equalization (CTLE), programmable gain amplifier (PGA), and decision feedback equalizer (DFE) 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/JESD204C transmitter mask requires a differential amplitude greater than 360 mV and less than 770 mV. The default amplitude is 0 (maximum amplitude setting).

Table 22. 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 must be verified using an internal eye monitor after the equalizer circuit of the receiver as this shows the actual eye that the receiver circuit receives.

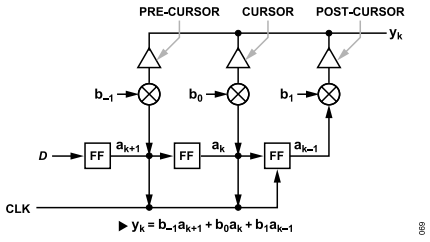


Figure 69. Serializer Emphasis implementation

A three tap FIR equalizer is implemented in the serializer as shown in Figure 69. Here, the cursor, or largest tap weight multiplying a_k is in the center. There is a precursor 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 preemphasis is used as it is easier to realize bit delays with flip-flops than trying to implement analog delays at the receiver.

The serializer preemphasis circuit allows boosting of the amplitude anytime the serial bit changes state. If no bit transition occurs, the amplitude is left unchanged. Preemphasis 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 preemphasis is controlled by setting precursor and post cursor amplitude settings that are listed in Table 23 and Table 24. Use the API function **adi_adrv904x_SerLaneCfgSet()** to set these parameters.

Table 23. Precursor Amplitude Settings

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

Table 24. 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 transmitter output via the JESD PHY. The ADRV904x has eight deserializer lanes that can be used to connect to the BBIC serializer for transmitting the baseband data. The ADRV904x 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 GHz to 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 ADRV904x JESD deserializer PHY is in Figure 70.

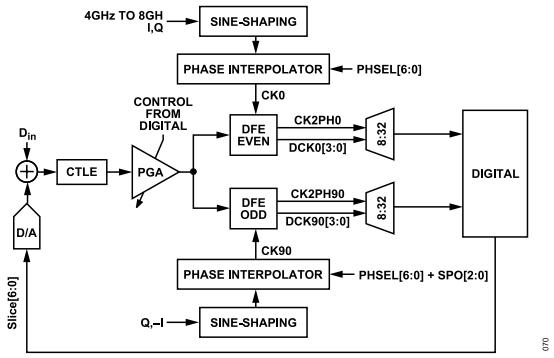


Figure 70. Block Diagram of ADRV904x Deserializer PHY

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 block. 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 block 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 CTLE output. 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 to rail the incoming data to $+V_{REF}(1)$ or $-V_{REF}(0)$. The output of the PGA block has strong 1s or 0s, 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 the ADRV904x. This eases the dependency on the CTLE block, which attenuates the incoming signal and trades amplitude for a good eye margins 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

JESD204B/JESD204C INTERFACE

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 that 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 help to get the data and edge sampling instances of the incoming data to run clock data recovery (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.

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

- Full Rate Mode.

  In this mode, the ADRV904x has lane rates within the range of 4 Gbps to 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 71, the serial input data goes through the CTLE block to clean up the signal before sampling. The CTLE block 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.

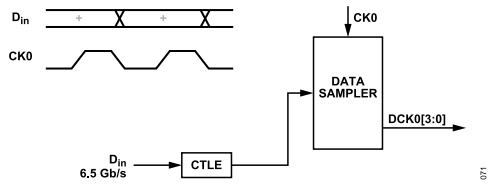


Figure 71. Simplified Block Diagram of DES in Full Rate Mode with Timing Diagram

  Note that SERDES calibrations are not needed for this mode.

- Half Rate Mode.

  In this mode, the ADRV904x has lane rates within the range of 8 Gbps to 16 Gbps. The sampling clock is now half as fast as the data rate. From the timing diagram in Figure 72, 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; therefore, only Slice 0 is used in full rate, but Slice 1 is used in half rate mode. 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 block is configured in a similar fashion to clean up the signal and max out the gain to drive the samplers.

JESD204B/JESD204C INTERFACE

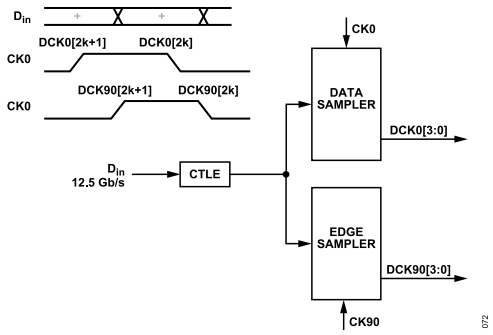


Figure 72. Simplified Block Diagram of DES in Half Rate Mode with Timing Diagram

- Quarter Rate Mode.

In this mode, the ADRV904x has lane rates within the range of 16 Gbps to 24.33 Gbps. Looking closely at Figure 73, 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.

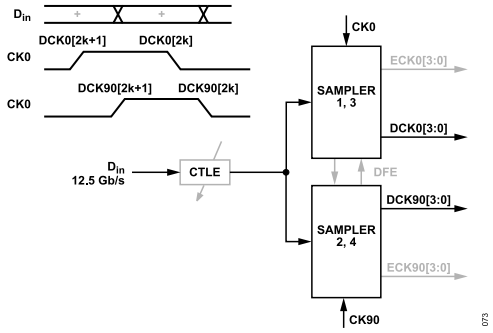


Figure 73. Simplified Block Diagram of DES in Quarter Rate Mode with Timing Diagram

In both half rate and quarter rate modes, the CTLE block alone cannot provide enough equalization. The DFE block is needed in addition to CTLE.

For DFE to work properly, the system needs to be linearly amplified by PGA up until the sampling point. The DFE block looks at the impulse response and subtracts the energy in the post cursors in order that zero ISI is seen for next sampling instants. This helps in achieving higher signal fidelity for high lane rates (>8 Gbps).

Figure 74 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 precursors 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 block, up to the sampling instant.

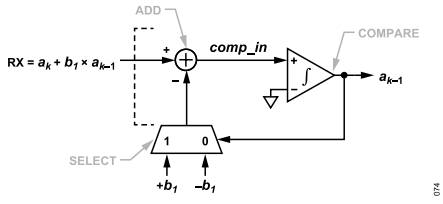


Figure 74. High-Level Block Diagram of a Single Tap DFE

When a sample of the comparator input (comp_in) is taken, a comparison is made to generate the output sample. Referring to the next sample, the sample taken is a_{k-1}. This previous sample is fed back, weighed by the value of the first post cursor, and subtracted out (or added negatively). Based on the comparators result, select either $+b_1$ or $-b_1$. This subtracts the energy in the post cursor and reduce ISI. Note that PGA is set up such that incoming signal is linear to accommodate this analog subtraction via DFE.

JESD204B/JESD204C INTERFACE

In quarter rate mode, the complete process of the DFE – Compare -> Select -> Add, is fast enough in order that it is completed within a single UI. The ADRV904x DFE uses three post cursors (b_1, b_2, and b_3) to get the best equalized performance for data rates up to 24.33 Gbps.

The ADRV904x 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 transmitter 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 half rate and quarter rate modes. The Arm optimizes the CTLE and DFE parameters in quarter rate mode to maintain signal integrity over the entire operating temperature. The calibration routines set all the offset correction bits and DFE tap weights. The tracking calibration is designed to withstand a temperature change rate of 0.2°C/sec.

Table 25 summarizes the design targets for the amount of insertion loss that the ADRV904x can support for different lane rates.

Table 25. Insertion Loss (Over Operating Temperature) Support for SERDES Lane Rate (Gbps) on ADRV904x

Data Rate (Gbps)	(Preliminary) Min Insertion Loss (dB)	(Preliminary) Worst-Case Insertion Loss (dB)
4 to 8	2	14 (no DFE)
8 to 16	3	23
16 to 24	4	25
24.33	4.5	25

The ADRV904x can tolerate an insertion loss deviation of ~4 dB up to ¾ of the max data rate. This is according to the JESD specifications.

The ADRV904x SERDES block integrates the following two different methods to verify link integrity:

- The user can test the signal integrity of each of the lanes using a built-in horizontal eye monitor and a 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. The horizontal eye monitor and vertical eye monitor can be run using the following APIs:
  - The horizontal eye monitor can be run using the RunEyeSweep_v2() API.
  - The horizontal + vertical eye monitor can be run using the RunVerticalEyeSweep_v2() API.
- An internal PRBS checker can be used to verify the link integrity over temperature and ensure meeting the bit error target for long duration tests.

Deserializer Configuration

The deserializer includes a nonadaptive, programmable equalizer. This helps in compensating for signal integrity distortions for each channel due to PCB trace length and impedance.

The **adi_adrv904x_DesCfg_t** data structure contains the information required to properly configure the deserializer. Details of each member can be found in API documentation. The transceiver evaluation software has the option to output example data structures with values chosen from the configuration tab of the software.

```
typedef struct adi_adrv904x_DesCfg
{
“highBoost”: 0,
“configOption1”: 0,
“configOption2”: 0,
“configOption3”: 0,
“configOption4”: 0,
“configOption5”: 0,
“configOption6”: 0,
“configOption7”: 0,
“configOption8”: 0,
“configOption9”: 0,
“configOption10”: 0,
“desInvertLanePolarity”: 0
```

```
} adi_adrv904x_DesCfg_t;
```

The parameters in the desCfg (see the code block) can be modified based on the insertion loss of the PCB trace that is being used. These parameters modify the CTLE LPF and other SERDES calibration settings. For high insertion loss cases (>8 dB), it is recommended to set **highBoost** to 1 and all remaining latest **configOptions** to 0. For other cases, where insertion loss is less than 8 dB, recommendations are typically made available in the profiles that come with the software. Contact Analog Devices for the recommended config Option settings that can be used for optimum robust SERDES operation of required 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:

JESD204B

For the deframer side, follow these steps:

1. Initialize and bring up the baseband processor framer side.
2. The deframer is held in reset state until the INIT command, then deframer issues a synchronization request by asserting the SYNC signal.
3. The framer starts sending K28.5 characters, then the deframer is brought out of reset.
4. The 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 in IronPython Code for PRBS Testing section).
6. After the deframer link is up, the user needs to enable the SERDES tracking cal. 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. 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 data links.

FIRST TIME SYSTEM BRINGUP—CHECKING LINK INTEGRITY

1. For ease of debug during bringup, 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_adrv904x_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 from baseband processor.
5. Call the API **adi_adrv904x_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_adrv904x_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_adrv904x_DeframerSysrefCtrlSet()** API in order that the external SYSREF signal at the pin can be gated off internally and the deframer does not see a potential invalid SYSREF pulse before it is configured correctly.
8. After bringing up of the JESD204B link or for debugging the deframer, the baseband processor can check the status of the deframer using **adi_adrv904x_DeframerStatusGet()**.

SAMPLE IN IRONPYTHON CODE FOR PRBS TESTING

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

```
'''
Koror IronPython Programming Template Version 0.2
Generated with:
ACE GUI Version: 1.25.3231.1410
ACE Plug-in Vesion: 1.2022.21101
API Client Version: 0.9.0.6
API Server Version: 0.9.0.6
FPGA Version: 0.3.3.0
ARM Version: 0.9.0.6
StreamVersion: 0.9.0.6


Note: ACE does not need to be disconnected from ADRV9040 command server before running this script
anymore.
'''




import clr
import System
from System import Array, Boolean, SByte, Byte, Int16, UInt16, Int32, UInt32, Int64, UInt64
import sys
import os
import glob # for finding plug-in's installation path


ACE_DIRECTORY = os.environ['ACEEXEDIR'] or os.getcwd()
""" str: path to ACE
Example: 'C:\Program Files\Analog Devices\ACE'
"""


CLIENT_DIRECTORY = os.environ['ADRV9040LIBPATH'] or (glob.glob(r'C:\ProgramData\Analog Devices\ACE\Plu▶
gins\Board.ADRV904*')[0] + '\lib')
""" str: path to client DLLs included in ADRV9040 Board plugin's installation
Example: 'C:\ProgramData\Analog Devices\ACE\Plugins\Board.ADRV9040.1.2021.24400\lib'
"""


DEFAULT_IPADDRESS = '192.168.1.10'
DEFAULT_PORT = '5000'
```

JESD204B/JESD204C INTERFACE

```
# Add ACE and client to path
sys.path.append(ACE_DIRECTORY)
sys.path.append(CLIENT_DIRECTORY)


# Verify ADRV9040 plugin version used
print 'Loading client DLLs from', CLIENT_DIRECTORY
clr.AddReference("AnalogDevices.EvalClient")
clr.AddReference("AnalogDevices.EvalClient.Installers")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Ad9528.Device")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Board")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Device")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Fpga.Device")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Platform")
clr.AddReference("Adrv904x.Configurator")
clr.AddReference("StreamGen")


from AnalogDevices.EvalClient import *
from AnalogDevices.EvalClient.Installers import *
from AnalogDevices.EvalClient.Adrvgen6.Board import *
from AnalogDevices.EvalClient.Adrvgen6.Board.BaseClasses import *
from AnalogDevices.EvalClient.AD9528 import *
from AnalogDevices.EvalClient.Adrvgen6.Ad9528.Device import *
from AnalogDevices.EvalClient.Adrvgen6.Fpga.Device import *
from AnalogDevices.EvalClient.FPGAGEN6 import *
from AnalogDevices.EvalClient.Device import *
from AnalogDevices.EvalClient.AdiCommon import *
from AnalogDevices.EvalClient.Adrvgen6 import *
from AnalogDevices.Adrvgen6.Platform import *
from AnalogDevices.Adrv904x.Configurator import *
from Adi.Design import *


def connect(ipAddress="", portNumber=""):
"""
Connect IronPython to the command server.
If TCP connection exceptions are seen, please verify that
ACE is disconnected from the ADRV9040 command server (see header).
"""
ipAddress = ipAddress or DEFAULT_IPADDRESS
portNumber = portNumber or DEFAULT_PORT
print 'Attempting to connect to {}:{}...'.format(ipAddress, portNumber)
EvalClientManager.Instance.Initialize(CLIENT_DIRECTORY)
transport = Transports.CreateDefaultTcpTransport(ipAddress + ':' + portNumber)
context = ExecutionContext(transport)
context.ErrorRetriever = ErrorRetriever()

platform = EvalClientManager.Instance.PlatformBuilder.CreatePlatform('', context)
platform.Timeout = 5000
platform.OutputFilesBasePath = '.'
configGen = ConfigGen()
return platform
```

JESD204B/JESD204C INTERFACE

```
def spiWriteByte(address, data):
"""
>>> spiWriteByte(0xA, 0x55)
spiWriteByte 0x000A 0x55
"""
data = Array[Byte]([data])
numBytes = len(data)
adrv904x.hal.RegistersByteWrite(None, address, data, numBytes)


print 'spiWriteByte\t0x{:04X}\t0x{:02X}'.format(address, int(data[0]))

def spiReadByte(address, numBytes=1):
"""
>>> spiWriteByte(0xA, 0x55);
>>> spiReadByte(0xA)
spiReadByte 0x000A 0x55
85
"""
data = Array[Byte]([0])
adrv904x.hal.RegistersByteRead(None, address, data, None, numBytes)
data = int(data[0]) # get first element of returned data array


print 'spiReadByte\t0x{:04X}\t0x{:02X}'.format(address, data)
return data




def generatePrbsInFpga():
print "generatePrbsInFpga() is running"
prbsmode = adi_fpgagen6_PrbsTestModes_e.ADI_FPGAGEN6_PRBS_31
fpgaDev.prbs.PrbsSerializerEnable(0xff,prbsmode)
print '====== FPGA Status ====== '
print '\t FPGA PRBS Mode: %s' % str(prbsmode)
#print '\t OnBoardPRBSChecker() is running'




if __name__ == '__main__':


# Connect IronPython to command server
platform = connect()

# Get device objects
board = platform.Boards[0]
ad9528Dev = board.DeviceGet('ad9528', 0)
fpgaDev = board.DeviceGet('fpga', 0)
try:
adrv904x = board.DeviceGet('adrvgen6', 0)
except SystemError: # SW0.5 onwards
adrv904x = board.Adrvgen6DeviceGet(0)
```

JESD204B/JESD204C INTERFACE

```
# API documentation: System Explorer -> ADRV904X -> Open Help File


# Check SPI functionality by writing/reading from scratchpad SPI register


address = 0xA
data = 0x55
spiWriteByte(address, data)
spiReadByte(address)


# Your code goes here!
print 'Hello World!'

generatePrbsInFpga()


#Deframer checker location
print 'Deframer checker location '
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_SAMPLEDATA
DfrmPrbsCfg_t.checkerLocation = adi_adrvgen6_DeframerPrbsCheckLoc_e.ADI_ADRVGEN6_PRBSCHECK_LANEDATA
#DfrmPrbsCfg_t.checkerLocation = adi_adrv904x_DeframerPrbsCheckLoc.ADI_ADRV904X_PRBSCHECK_SAMPLEDATA #
Check PRBS at output of deframer (JESD204b deframed sample)
adrv904x.dataInterface.DfrmPrbsCheckerStateSet(DfrmPrbsCfg_t)

#Reading the Deframer PRBS test mode and Checker location
print 'Reading the Deframer PRBS test mode and Checker location'
adrv904x.dataInterface.DfrmPrbsCheckerStateGet(DfrmPrbsCfg_t)
#adrvgen6Device_0.dataInterface.DfrmPrbsCheckerStateGet(DfrmPrbsCfg_t)

print'====== DfrmPrbsCheckerStateGet ====== '
print'\t DfrmPrbsChecker PolyOrder: %s' % str(DfrmPrbsCfg_t.polyOrder)
print'\t DfrmPrbsChecker CheckerLocation: %s' % str(DfrmPrbsCfg_t.checkerLocation)



adrv904x.dataInterface.DfrmPrbsCountReset()
time.sleep(0.5)


#DfrmPrbsErrCountGet
DfrmPrbsErrCounters_t = adi_adrvgen6_DfrmPrbsErrCounters_t()
adrv904x.dataInterface.DfrmPrbsErrCountGet(DfrmPrbsErrCounters_t)
print
for i in range(0,4):
print "==================================== "
print '\t PRBS error of lane %d' %i
#print 'DfrmPrbsErrCounters_t. sampleErrors', DfrmPrbsErrCounters_t.sampleErrors[i] # only at the same▶
ple mode, 0b: sample error count, 1b: invalid error flag, 2b: error flag
print '\t DfrmPrbsErrCounters_t. errorStatus', DfrmPrbsErrCounters_t.errorStatus[i] # Array contains
error status
```

```
print '\t DfrmPrbsErrCounters_t. laneErrors', DfrmPrbsErrCounters_t.laneErrors[i] # Lane 0 contains
error counters if in sample mode, otherwise errors are maintained per lane
```

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

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 the 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 pair traces are length matched.
   - Verify that the differential impedance of the traces is 100 Ω.
2. Check the board (powered) for the following:
   - 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 for the following:
   - 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 receiver vs. transmitter 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 the 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. To check serial data, take the following steps:
   a. Verify the transmitter data rate and the receiver expected rate are the same.
   b. 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.
   c. Verify /K/ characters with high impedance probe. (If /K/ characters are correct, the transmitter side of the link is working properly. If /K/ characters are not correct, the transmitter device or the board lanes signal has an issue.
   d. Verify the transmitter CML differential voltage on the data lanes.
   e. Verify the receiver CML differential voltage on the data lanes.
   f. 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.
   g. 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. Check for link parameter conflicts by taking the following steps:
   a. Verify ILAS multiframes are transmitting properly. Verify the link parameters on the transmitter device, the receiver device, and those transmitted in ILAS second multiframe.
   b. Calculate the 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), check for the following causes for link reset.

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 JESD204B 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 reestablishment 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 deskew 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 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 rin Figure 75 and Figure 76.

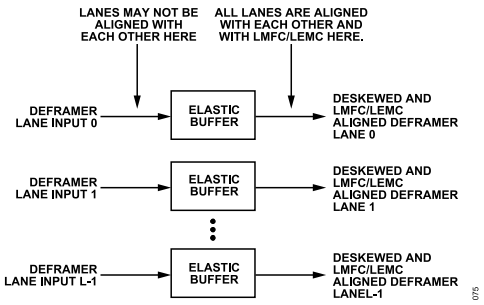


Figure 75. Elastic Buffers in the Deframers

JESD204B/JESD204C INTERFACE

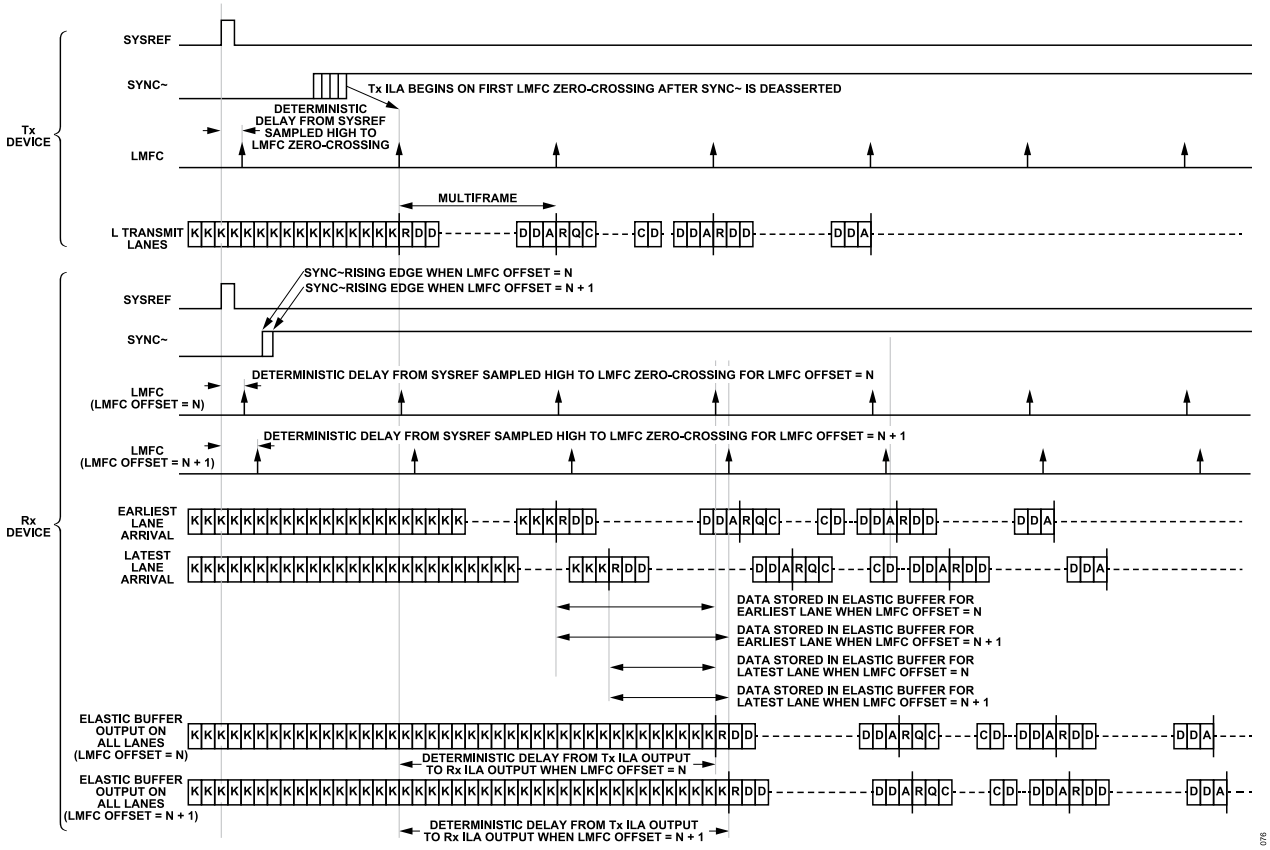


Figure 76. Impact of LMFC Offset on Elastic Buffer Depth in JESD204B Mode

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 deskew 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 75 and Figure 77.

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 two, the elastic buffer is not able to store enough data for some LEMC offset values.

JESD204B/JESD204C INTERFACE

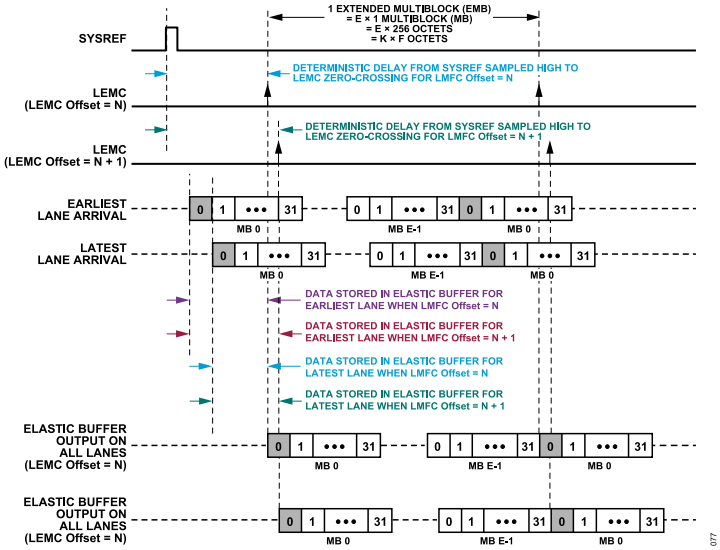


Figure 77. Impact of LEMC Offset on Elastic Buffer Depth in JESD204C Mode

Programming the LMFC Offset for a Deframer

The following are three ways to program the LFMC offset for a given deframer:

- By modifying the profile file being used.
- By using the adi_adrv904x_DeframerCfg data structure.
- 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 JESD204B mode, and to the LEMC offset in JESD204C 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 **ADRV904X_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 78).

```
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,
```

078

Figure 78. Deframer 0 deframer_lmfc_offset Field for the ADRV904X_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.

Setting the LMFC/LEMC Offset in the adi_adrv904x_DeframerCfg_t Data Structure

An alternative way of programming the LMFC/LEMC offset consists in using the **lmfcOffset** field of the **adi_adrv904x_DeframerCfg** data structure for the relevant deframer (see Figure 79). Note that the device must be reprogrammed after changing the LMFC/LEMC offset for a given deframer in the **adi_adrv904x_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 makes the programming process quicker.

```
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 assc
    uint8_t sysrefNShotEnable;        /*!< 1 = Enable SYSREF NShot (ability to ignore first rising ec
    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;
```

079

Figure 79. LMFC Offset Field in adi_adrv904x_DeframerCfg Data Structure

JESD204B/JESD204C INTERFACE

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 Deframer 0 or Deframer 1 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 that you must reestablish the link between the JESD framer and the transceiver JESD deframer after changing the LMFC/LEMC offset. To do this, you must toggle the **DeframerLinkStateSet()** and also toggle the **DeframerSysrefCtrlSet()**. A new Sysref is not required..

Reading Back the Buffer Depths for Each Deframer Lanes

The API function **adi_adrv903x_CoreBufDepthGet** can be used to read back the deframer FIFO depth for each of the deframers.

In JESD204B mode, the unit of the values read back in those registers is four octets. In other words, an increment of the buffer depth value read back by one unit corresponds to an actual increment by four 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 eight octets. In other words, an increment of the buffer depth value read back by one unit corresponds to an actual increment by eight 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.

Therefore, it is 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 26. List of JESD Related API Functions

API Method Name	Comments
adi_adrv904x_AdcSampleXbarSet()	Sets the ADC sample crossbar for the specified ADRV904x framer.
adi_adrv904x_AdcSampleXbarGet()	Gets the ADC sample crossbar converter configuration map for the chosen JESD204B/JESD204C framer converter.
adi_adrv904x_DacSampleXbarSet()	Sets the DAC sample crossbar for the specified ADRV904x deframer.
adi_adrv904x_DacSampleXbarGet()	Gets the DAC sample crossbar for the specified ADRV904x deframer.
adi_adrv904x_FramerCfgGet()	Gets the ADRV904x Framer configuration.
adi_adrv904x_DeframerCfgGet()	Gets the ADRV904x Deframer configuration.
adi_adrv904x_FramerLinkStateGet()	Gets the link states for all the framers.
adi_adrv904x_FramerLinkStateSet()	Sets the link states for all the framers.
adi_adrv904x_DfrmPrbsCountReset()	Reset deframer PRBS count error.
adi_adrv904x_DeframerLinkStateGet()	Gets the link states for all the deframers.
adi_adrv904x_DeframerLinkStateSet()	Sets the link states for all the deframers.
adi_adrv904x_DfrmPrbsCheckerStateSet()	Sets the lane/sample PRBS checker configuration.
adi_adrv904x_DfrmPrbsCheckerStateGet()	Gets the lane/sample PRBS checker configuration.

JESD204B/JESD204C INTERFACE

Table 26. List of JESD Related API Functions (Continued)

API Method Name	Comments
adi_adrv904x_FramerSysrefCtrlSet()	Enables or disables the external SYSREF signal to the framers.
adi_adrv904x_FramerSysrefCtrlGet()	Get the status of the external SYSREF signal to the framers.
adi_adrv904x_DeframerSysrefCtrlSet()	Enables or disables the external SYSREF signal to the deframers.
adi_adrv904x_DeframerSysrefCtrlGet()	Get the status of the external SYSREF signal to the deframers.
adi_adrv904x_FramerTestDataSet()	Sets the PRBS type and enables/disables framer PRBS generation.
adi_adrv904x_FramerTestDataGet()	Gets the PRBS framer test mode and inject points.
adi_adrv904x_DfrmPrbsErrCountGet()	Gets the deframer sample PRBS error counts.
adi_adrv904x_SerializerReset()	Resets the serializer.
adi_adrv904x_FramerLmfcOffsetSet()	Sets LMFC offset for selected framer.
adi_adrv904x_FramerLmfcOffsetGet()	Gets LMFC offset for selected framer.
adi_adrv904x_DfrmLmfcOffsetSet()	Sets LMFC offset for selected deframer.
adi_adrv904x_DfrmLmfcOffsetGet()	Gets LMFC offset for selected deframer.
adi_adrv904x_DfrmPhaseDiffGet()	Gets phase differential value for selected deframer.
adi_adrv904x_FramerStatusGet()	Get framer status for the selected framer.
adi_adrv904x_DeframerStatusGet()	Get deframer status for the selected framer.
adi_adrv904x_DeframerStatusGet_v2()	Get deframer status for the selected framer.
adi_adrv904x_DfrmErrCounterStatusGet()	Get deframer status for the selected deframer. JESD204B only.
adi_adrv904x_DfrmErrCounterReset()	Clear the error counters selected deframer. JESD204B only.
adi_adrv904x_Dfrm204cErrCounterStatusGet()	Get deframer status for the selected deframer. JESD204C only.
adi_adrv904x_Dfrm204cErrCounterReset()	Clear the error counters selected deframer. Support for only JESD204C.
adi_adrv904x_DfrmLinkConditionGet()	Get deframer link condition.
adi_adrv904x_DfrmFifoDepthGet()	Get the deframer FIFO depth.
adi_adrv904x_DfrmCoreBufDepthGet()	Get the deframer core buffer depth.
adi_adrv904x_DfrmIlasMismatchGet()	Compares received ILAS to programmed ILAS. JESD204B only.
adi_adrv904x_DfrmIlasMismatchGet_v2()	Compares received ILAS to programmed ILAS. JESD204B only.
adi_adrv904x_FramerSyncbModeSet()	Set the framer JESD204B syncb signal mode to SPI or PIN mode.
adi_adrv904x_FramerSyncbModeGet()	Get the framer JESD204B syncb signal mode to SPI or PIN mode.
adi_adrv904x_FramerSyncbStatusSet()	Get the framer JESD204B syncb signal status.
adi_adrv904x_FramerSyncbStatusGet()	Set the framer JESD204B syncb signal status.
adi_adrv904x_FramerSyncbErrCntGet()	Get the framer JESD204B syncb signal error counter.
adi_adrv904x_FramerSyncbErrCntReset()	Reset the framer JESD204B syncb signal error counter.
adi_adrv904x_DeframerSyncbErrCntGet()	Get the deframer JESD204B syncb signal error counter.
adi_adrv904x_DeframerSyncbErrCntReset()	Reset the deframer JESD204B syncb signal error counter.
adi_adrv904x_DfrmIrqMaskGet()	Gets the JESD204B IRQ clear register. There is one register for all deframers
adi_adrv904x_DfrmIrqMaskSet()	Sets the JESD204B IRQ clear register. There is one register for all deframers.
adi_adrv904x_DfrmIrqSourceReset()	Reset the JESD204B IRQ clear register. There is one register for all deframers.
adi_adrv904x_DfrmIrqSourceGet()	Get the JESD204B IRQ source registers for the specified deframer.
adi_adrv904x_DfrmErrCntrCntrlSet()	Configure or reset the JESD204B deframer error counters.
adi_adrv904x_DfrmErrCntrCntrlGet()	Get the configuration for the JESD204B deframer error counters.
adi_adrv904x_RunEyeSweep()	Run horizontal eye sweep.
adi_adrv904x_RunEyeSweep_v2()	Run horizontal eye sweep.
adi_adrv904x_RunVerticalEyeSweep()	Run vertical eye sweep.
adi_adrv904x_RunVerticalEyeSweep_v2()	Run vertical eye sweep.
adi_adrv904x_FramerTestDataInjectError()	Injects an error into the framer test data by inverting the data
adi_adrv904x_SerLaneCfgSet()	Set the serializer lane configuration parameters for the chosen lane.
adi_adrv904x_SerLaneCfgGet()	Get the serializer lane configuration parameters for the chosen lane.
adi_adrv904x_RxTestDataSet()	Set up a test signal to be sent out the framer instead of the ADC output.

JESD204B/JESD204C INTERFACE

Table 26. List of JESD Related API Functions (Continued)

API Method Name	Comments
adi_adrv904x_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 ADRV904x that performs a series of configuration tasks based on an event. When requested the stream processor performs a series of predefined actions that 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 the following:

- Transmitter datapath enable/disable
- Receiver datapath enable/disable
- Observation receiver datapath enable/disable

The stream processor image changes with different configurations. For example, the stream processor that enables the receivers are different depending on the JESD configuration. Therefore, it is 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 as follows:

- The framer choices for observation receiver and receiver
- Link sharing profiles use different stream files to non link sharing profiles
- If floating point formatting is being used on receiver and observation receiver 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 transmitter datapaths, eight receiver datapaths, and two for the observation receiver datapaths. These observation receiver datapaths are not shared with the internal transmitter channel loopback paths that facilitate data collection during the various transmitter calibrations, however for external local oscillator leakage (LOL) calibrations the observation receiver path is used. The existence of individual slice stream processors for each datapath enables true real-time parallel operation of all individual transmitter, receiver, and observation receiver datapaths.

Each slice stream processor may only access the digital register sub maps corresponding to its specific functionality. For example, the transmitter slice stream processors can only access the transmitter 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 27. Stream Processor API Functions

API Method Name	Comments
adi_adrv904x_StreamVersionGet()	Reads back the version of the stream binary loaded.
adi_adrv904x_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, through pin control mode or radio sequencer. 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 receiver and transmitter signal chains are controlled using eight transreceiver control (TRXn_CTRL) pins. Any one of the TRXn_CTRL pins can control any of the eight receiver and/or eight transmitter paths. A TRXn_CTRL pin can even control multiple paths at the same time. Using the configurator, define which transmitter

STREAM PROCESSOR AND SYSTEM CONTROL

and receiver channels must be controlled by the TRXn_CTRL pins. When these TRXn_CTRL pins are toggled high, the predetermined signal chain(s) are enabled. When these pins are toggled low, the predetermined signal chain(s) are disabled.

The observation receiver 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.

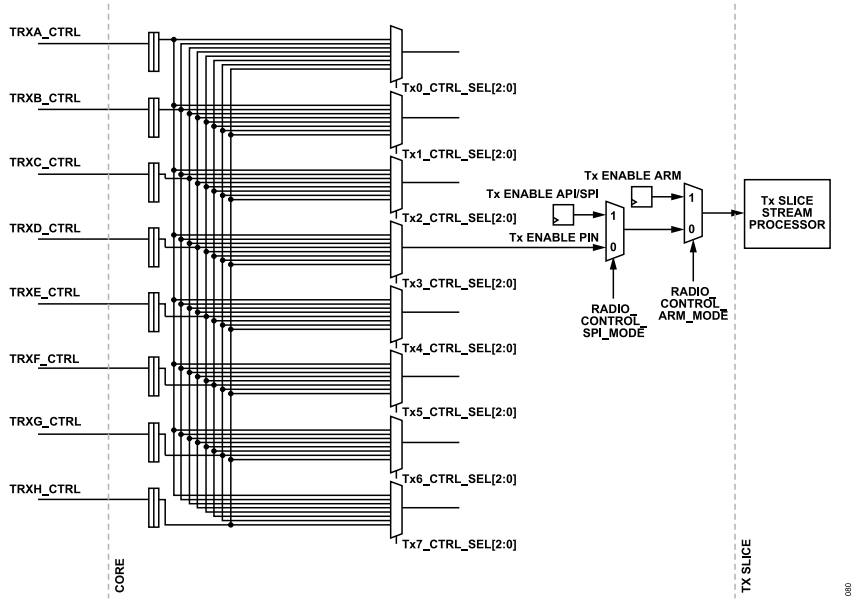


Figure 80. TRXn_CTRL Pins for Transmitter Slice

Figure 80 shows that each TRXn_CTRL pin can be wired to control any transmitter datapath or multiple transmitter datapaths at the same time. The value of Tx#_CTRL_SEL[2:0] in Table 28 dictates what TRXn_CTRL pin a given transmitter slice is pointing to. The mux RADIO_CONTROL_SPI_MODE chooses whether the TRXn_CTRL pins or the API (via the SPI) controls the enable/disable of the transmitter datapaths. Further on in the mux chain the RADIO_CONTROL_ARM_MODE chooses whether the Arm processor or the API/TRXn_CTRL pins are able to control the transmitter datapaths. This is because the Arm processor needs to have priority over when it can control the transmitter datapaths for calibrations. Therefore, when the Arm processor takes control of the signal paths it cannot be stopped.

Table 28. 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
TRXH_CTRL	111

STREAM PROCESSOR AND SYSTEM CONTROL

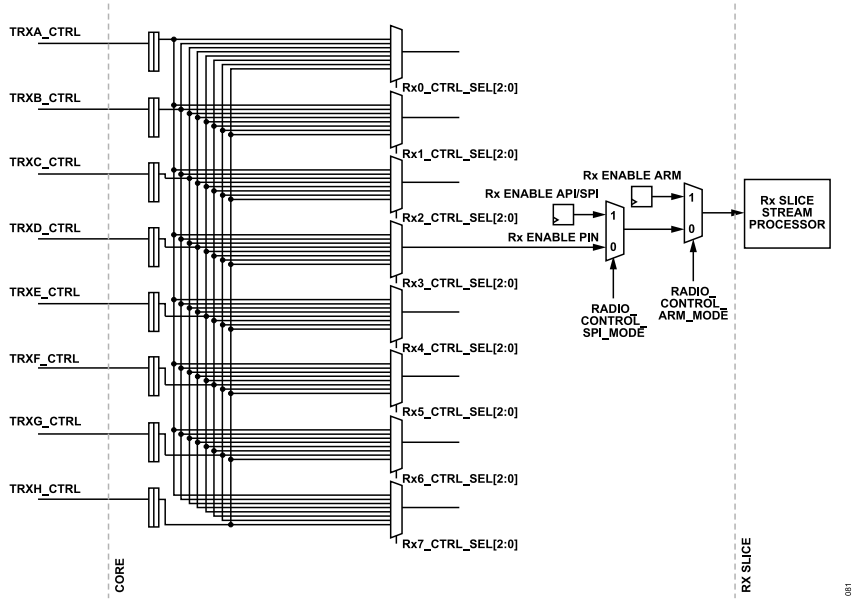


Figure 81. TRXn_CTRL Pins for Rx Slice

Figure 81 shows each TRXn_CTRL pin can be wired to control any receiver datapath or multiple receiver datapaths at the same time. The value of Rx#_CTRL_SEL[2:0] as stated in Table 29 dictates what TRX_EN pin a given receiver slice is pointing to. The mux Rx#_CTRL_SEL[2:0] is set to point at a given TRXn_CTRL pin. The mux RADIO_CONTROL_SPI_MODE chooses whether the TRX_EN pins or the API (via the SPI) controls the enable/disable of the receiver datapaths. However, note that further on in the mux chain the RADIO_CONTROL_ARM_MODE chooses whether the Arm or the API/TRXn_CTRL pins are able to control the receiver datapaths. This is because the Arm needs to have priority over when it can control the receiver datapaths for calibrations. Therefore, when the Arm takes control of the signal paths it cannot be stopped.

Table 29. 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

Configuring the TRX[A-H]_CTRL Pins for Transmitter and Receiver Pin Control

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

```
"txEnMapping": [
15,
```

STREAM PROCESSOR AND SYSTEM CONTROL

```
240,
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 in the following code block to control the Rx0 to Rx2 channels when the TRXE_CTRL pin is triggered, and the Rx3 to Rx7 channels when the TRXG_CTRL pin is triggered. The Rx0 to Rx2 channels are Bit[0:2], which is 00000111b in binary and 7 in decimal. The Rx3 to Rx7 channels are Bit[3:7], which is 11111000b in binary and 248 in decimal. 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 30. Pin Mode Enable API Function

API Method Name	Comments
adi_adrv904x_RadioCtrlCfgSet()	Sets the enable/disable of SPI or pin mode for radio control. The device defaults to SPI mode on power-up. Therefore, this API needs to be called to enable pin mode if necessary.
adi_adrv904x_RadioCtrlCfgGet()	Reads the radio control mode. SPI or pin mode.
adi_adrv904x_RadioCtrlTxRxEnCfgSet()	Sets the Tx/Rx channels to be turned on and off by the TRX[A-H]_CTRL pins. The API method also needs to be considered for antenna cal.
adi_adrv904x_RadioCtrlTxRxEnCfgGet()	Reads the Tx/Rx channels that are controlled by the TRX[A-H]_CTRL pins.
adi_adrv904x_RxTxEnableSet()	Enables or disables all Rx, ORx, and Tx channels that are in SPI control mode (see adi_adrv904x_RadioCtrlCfgSet). Has no effect on channels that are not set to SPI control mode. Furthermore, channels that are not initialized are not 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_adrv904x_RxTxEnableGet()	Reads SPI mode enable/disable status of Rx/Tx/ORx.
adi_adrv904x_ChannelEnableGet()	Reads the enable/disable status of each channel. Works in pin and SPI control mode.

TRANSMITTER TO OBSERVATION RECEIVER MAPPING

The transmitter-to-observation receiver mapping concept is vital to obtaining the best performance from any calibration that uses an external loop back path for data acquisition. For calibrations that utilize an external loop back path, such as transmitter LOL tracking, the Arm processor must be informed which transmitter channel it is currently observing at each observation receiver channel to apply corrections to the appropriate channel. The means to provide this information to the Arm processor is called transmitter-to-observation receiver mapping.

The following are the two interfaces that can be used to indicate the transmitter-to-observation receiver mapping condition:

- RCI Mode.

  API command interface—uses an API command adi_adrv904x_TxToOrxMappingSet(…) to indicate which transmitter channel is currently input to either observation receiver channel.

- Advantage—no GPIO pins are required.
- Disadvantage—to synchronize that the external path connection matches the transmitter to the observation receiver mapping state indicated to the Arm processor is more challenging than the pin control method. This leads to a possible scenario wherein the external path connection is in one state while the transmitter-to-observation receiver 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 the higher overhead to indicate transmitter-to-observation receiver mapping to the Arm processor.

Pin control interface—uses from two to eight GPIO pins to indicate the transmitter to observation receiver mapping state.

- Advantage—much easier to achieve synchronization between the external path connection and transmitter-to-observation receiver mapping state as the signaling lines that control an external switch can be used to also indicate the transmitter-to-observation receiver mapping state provided that logic levels for the transceiver input pins are met.
- Disadvantage—GPIO pins are required and these pins are required to connect back the ADRV904x GPIO.

- DFE Mode (RCI Hybrid Mode)
  - DFE mode—in this mode, the ADRV904x firmware controls transmitter to observation receiver mapping GPIOs, where transmitter, receiver, and observation receiver enable still come from BBIC/FPGA.
    - Advantage—removes the need for the user to control transmitter to observation receiver mapping and the need to keep track of which channel and calibration to run next.

The following sections go into detail regarding different modes for transmitter-to-observation receiver mapping.

TRANSMITTER TO OBSERVATION RECEIVER MAPPING—RCI MODE

RCI API mode—for FDD systems, transmitter to observation receiver mapping is controlled via SPI (API mode) and VSWR forward reverse switching via GPIO (VSWR direction).

RCI PIN mode—for the TDD system, transmitter-to-observation receiver mapping must be controlled by GPIO (pin mode). Apart from the GPIO required for observation receiver switching, two additional GPIOs are required for VSWR, one for VSWR enable and one for VSWR direction. Enabling VSWR and disabling GPIO are part of pin mapping (3-pin mode and 4-pin mode). VSWR direction GPIO can be selected from the digital GPIO mapping under stream in the Analog Devices Analysis, Control, Evaluation (ACE) GUI.

Transmitter-to-observation receiver mapping is controlled by the stream processor. This invokes the streams required to set a transmitter to observation receiver mapping configuration and also sets the VSWR enable function. VSWR direction can be set using pin mode or API mode and separate GPIOs are used.

External switches are required for mapping four transmitters to one observation receiver as the ADRV904x has eight transmitter channels and only two observation receiver channels. The Tx0 to Tx3 channels need to be mapped to ORx0 and the Tx4 to Tx7 channels must be mapped to ORx1 for DFE cals to run.

Transmitter to Observation Receiver Mapping—Pin Interface

In pin mode, GPIOs from BBIC must be used for switch control and to inform the ADRV904x on which the transmitter channel is connected to which observation receiver by giving the same signal via the ADRV904x GPIO. In pin mode, VSWR enabled is part of the transmitter-to-observation receiver mapping configuration. VSWR direction uses separate GPIO assigned from the stream settings—digital GPIO mapping in the ACE GUI. Two options are available: use common VSWR direction using a single GPIO, or use a separate GPIO for the direction of ORx0 and ORx1.

Three aspects of stream processor configuration must be input to generate the proper stream binary for a desired transmitter to observation receiver mapping. In these modes, the GPIO input pins send a signal to the stream processor which indicates the mapping to the ARM processor so that it may properly run the calibrations. These aspects are as follows:

- Transmitter observability. The user must configure a transmitter observability attribute for each transmitter channel. This defines which observation receiver channel a specific transmitter channel can be observed by a single transmitter channel can only be observable to one observation receiver channel. It is invalid for any transmitter to be observed at both ORx0 and ORx1.
- GPIO mapping mode. For modes featuring fewer pins, there are additional constraints upon the user in terms of the configuration of the feedback path. The available modes are as follows:
  - 2-pin mode

- 3-pin mode
- 4-pin mode
- 6-pin mode
- 8-pin mode

- GPIOs for the transmitter to observation receiver mapping. The user must define which GPIO pins must be used to indicate the transmitter to observation receiver mapping. Any of the 24 digital GPIO pins can be used to indicate the mapping.

2-pin mode does not support VSWR, all other pin modes support VSWR.

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 transmitter to observation receiver mapping state—these can be assigned to any GPIO pin. Changing any of the GPIOs in the TxToOrxMap word latches the specified transmitter to observation receiver mapping.

Pin Interface—2-Pin Mode

The 2-pin mode represents the minimum GPIO requirement for transmitter to observation receiver mapping. In this mode, ORx1 and ORx2 share the same control from BBIC for transmitter to observation receiver mapping.

This mode does not support VSWR.

The two GPIOs (Bit D0, Bit D1) must be connected to the two SP4T switches and the same GPIOs must be connected to the assigned Koror GPIO pins. Table 31 is an example and the user must modify the mapping as per the SP4T switch connection and pin logic.

The user can define the transmitter channel mapping assignment from the ACE GUI, JSON, or via an API.

Table 31. Transmitter to Observation Receiver Mapping 2-Pin Mode

D1	D0	ORx0 Mapping	ORx1 Mapping
0	0	Tx0-DPD	Tx4-DPD
0	1	Tx1-DPD	Tx5-DPD
1	0	Tx2-DPD	Tx6-DPD
1	1	Tx3-DPD	Tx7-DPD

Pin Interface—3-Pin Mode

The 3-pin mode extends the 2-pin mode by adding a third GPIO pin to indicate VSWR enable. In 3-pin mode, Bit D0, Bit D1, and Bit D2 are used. Pin D0 and Pin D1 usage is similar to 2-pin mode but the third Pin D2 is used for VSWR enable. Once VSWR is enabled, no other DFE tracking cals will run.

Table 32. Transmitter to Observation Receiver Mapping 3-Pin Mode

D2	D1	D0	ORx0 Mapping	ORx1 Mapping
0	0	0	Tx0-DPD	Tx4-DPD
0	0	1	Tx1-DPD	Tx5-DPD
0	1	0	Tx2-DPD	Tx6-DPD
0	1	1	Tx3-DPD	Tx7-DPD
1	0	0	Tx0-VSWR	Tx4-VSWR
1	0	1	Tx1-VSWR	Tx5-VSWR
1	1	0	Tx2-VSWR	Tx6-VSWR
1	1	1	Tx3-VSWR	Tx7-VSWR

Mapping can be generated from the stream settings in the ACE GUI directly as shown in Figure 82. To do this, take the following steps:

1. Select transmitter to observation receiver mapping as 3-pin mode.
2. Select digital GPIO mapping for Bit D0, Bit D1, and Bit D2. Also, select GPIO for VSWR direction. Independent GPIOs can be selected for each observation receiver or a common GPIO selected for ORx0 and ORx1.

STREAM PROCESSOR AND SYSTEM CONTROL

3. Assign channels in the pin table. Index 0 to Index 7 correspond to Bit D0, Bit D1, and Bit D2 in Table 32. Here, Index 0 (Bit D0 = 0, Bit D1 = 0, Bit D2 = 0) corresponds to transmitter Tx0 and Tx4 channels. If as per the physical routing or switch connection, you must change, for example, to Tx1 and Tx5. You can change the channel assignment in GUI and generate the required binaries.

   Users can also find the corresponding assignment in **TxToORxMappingPinTable** in JSON file as shown in Figure 83. Enum dec value and corresponding channels are shown in Table 33.

Table 33. Tx to ORx Mapping Pin

Channel	Enum Dec Value
Tx0	0
Tx1	1
Tx2	2
Tx3	3
Tx4	4
Tx5	5
Tx6	6
Tx7	7
Tx0-VSWR	8
Tx1-VSWR	9
Tx2-VSWR	10
Tx3-VSWR	11
Tx4-VSWR	12
Tx5-VSWR	13
Tx6-VSWR	14
Tx7-VSWR	15

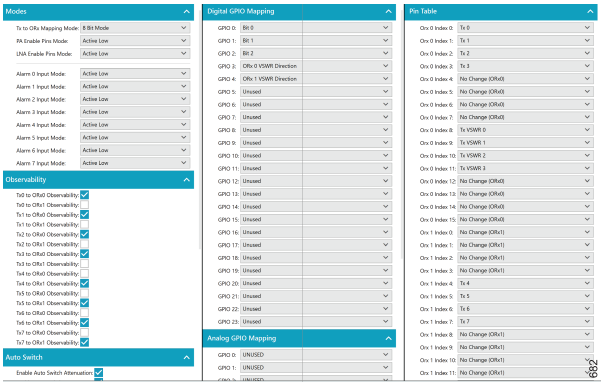


Figure 82. Stream Settings in ACE GUI

The JSON file corresponding to the configuration in Figure 82 is shown in Figure 83. Users can also modify JSON (profile file) directly.

STREAM PROCESSOR AND SYSTEM CONTROL

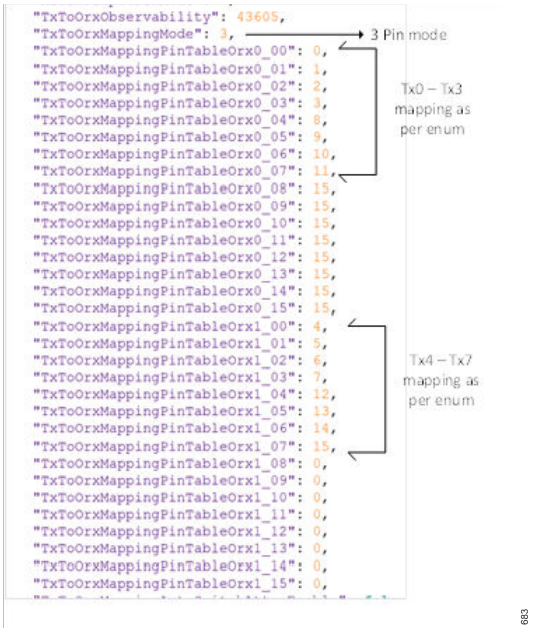


Figure 83. JSON File

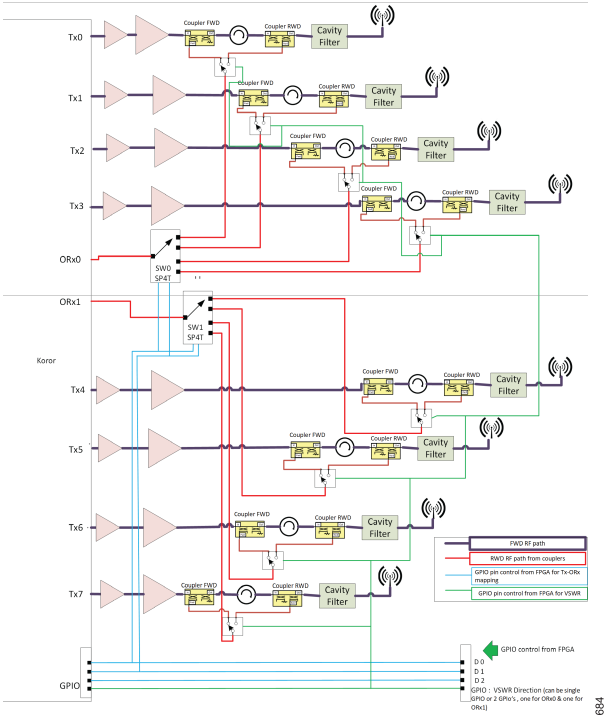


Figure 84. 3-Pin Mode with VSWR (Four GPIOs)

Pin Interface—4-Pin Mode

In 4-pin mode, Bit D0 and Bit D1 control transmitter to observation receiver mapping as in 2-pin mode. ORx0 and ORx1 can be controlled independently. Bit D2 is used for VSWR enable and Bit D3 decides which observation receiver to be enabled. Mapping details are shown in Table 34.

Table 34. Transmitter to Observation Receiver Mapping 4-Pin Mode

D3	D2	D1	D0	ORx0 Mapping	ORx1 Mapping
0	0	0	0	Tx0-DPD	N/A[1]
0	0	0	1	Tx1-DPD	N/A[1]
0	0	1	0	Tx2-DPD	N/A[1]
0	0	1	1	Tx3-DPD	N/A[1]
0	1	0	0	Tx0-VSWR	N/A[1]
0	1	0	1	Tx1-VSWR	N/A[1]
0	1	1	0	Tx2-VSWR	N/A[1]
0	1	1	1	Tx3-VSWR	N/A[1]
1	0	0	0	N/A[1]	Tx4-DPD
1	0	0	1	N/A[1]	Tx5-DPD
1	0	1	0	N/A[1]	Tx6-DPD
1	0	1	1	N/A[1]	Tx7-DPD
1	1	0	0	N/A[1]	Tx4-VSWR
1	1	0	1	N/A[1]	Tx5-VSWR
1	1	1	0	N/A[1]	Tx6-VSWR
1	1	1	1	N/A[1]	Tx7-VSWR

[1] Not applicable. In this case, the firmware does not take any action, and the current mapping will be maintained.

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 transmitter to observation receiver mapping on either side of the chip is enabled. Here, Bit D0 to Bit D2 control ORx0, and Bit D3 to Bit D5 are used for ORx1. Bit D0 to Bit D1 and Bit D3 to Bit D4 are used for transmitter select, and Bit D2 and Bit D5 are used for VSWR enable. The bit mapping is described in Table 35.

ORx0 and ORx1 can be controlled independently or together in this mode. Table 35 is a representation showing Bit D0 to Bit D2 and Bit D3 to Bit D5 independently for better understanding. In the actual application, Bit D0 to Bit D6 have values at the same time and one of the transmitter channels is be connected to ORx0, the same way one channel is connected to ORx1.

Table 35. Transmitter to Observation Receiver Mapping 6-Pin Mode

D5	D4	D3	D2	D1	D0	ORx0 Mapping	ORx1 Mapping
-	-	-	0	0	0	Tx0-DPD	N/A[1]
-	-	-	0	0	1	Tx1-DPD	N/A[1]
-	-	-	0	1	0	Tx2-DPD	N/A[1]
-	-	-	0	1	1	Tx3-DPD	N/A[1]
-	-	-	1	0	0	Tx0-VSWR	N/A[1]
-	-	-	1	0	1	Tx1-VSWR	N/A[1]
-	-	-	1	1	0	Tx2-VSWR	N/A[1]
-	-	-	1	1	1	Tx3-VSWR	N/A[1]
0	0	0	-	-	-	N/A[1]	Tx4-DPD
0	0	1	-	-	-	N/A[1]	Tx5-DPD
0	1	0	-	-	-	N/A[1]	Tx6-DPD
0	1	1	-	-	-	N/A[1]	Tx7-DPD
1	0	0	-	-	-	N/A[1]	Tx4-VSWR
1	0	1	-	-	-	N/A[1]	Tx5-VSWR
1	1	0	-	-	-	N/A[1]	Tx6-VSWR
1	1	1	-	-	-	NA[1]	Tx7-VSWR

[1] Not applicable. In this case, the firmware does not take any action, and the current mapping will be maintained.

STREAM PROCESSOR AND SYSTEM CONTROL

Pin Interface—8-Pin Mode

The 8-pin mode is the same as the 3-pin mode with independent control for ORx0 and Orx1, Bit D0 to Bit D1 and Bit D4 to Bit D5 are used for transmitter selection. Bit D2 and Bit D6 are used for observation receiver selection independently. Bit D3 and Bit D7 are used for VSWR enable or DFE cals independently. The bit mapping is described in Table 36.

Table 36. Transmitter to Observation Mode Mapping 8-Pin Mode

D7	D6	D5	D4	D3	D2	D1	D0	ORx0 Mapping	ORx1 Mapping
-	-	-	-	0	0	0	0	Tx0-DPD	N/A[1]
-	-	-	-	0	0	0	1	Tx1-DPD	N/A[1]
-	-	-	-	0	0	1	0	Tx2-DPD	N/A[1]
-	-	-	-	0	0	1	1	Tx3-DPD	N/A[1]
0	1	0	0	-	-	-	-	N/A[1]	Tx4-DPD
0	1	0	1	-	-	-	-	N/A[1]	Tx5-DPD
0	1	1	0	-	-	-	-	N/A[1]	Tx6-DPD
0	1	1	1	-	-	-	-	N/A[1]	Tx7-DPD
-	-	-	-	1	0	0	0	Tx0-VSWR	N/A[1]
-	-	-	-	1	0	0	1	Tx1-VSWR	N/A[1]
-	-	-	-	1	0	1	0	Tx2-VSWR	N/A[1]
-	-	-	-	1	0	1	1	Tx3-VSWR	N/A[1]
1	1	0	0	-	-	-	-	N/A[1]	Tx4-VSWR
1	1	0	1	-	-	-	-	N/A[1]	Tx5-VSWR
1	1	1	0	-	-	-	-	N/A[1]	Tx6-VSWR
1	1	1	1	-	-	-	-	N/A[1]	Tx7-VSWR

[1] Not applicable. In this case, the firmware does not take any action, and the current mapping will be maintained.

Transmitter to Observation Receiver Mapping API Functions

Table 37. Transmitter to Observation Receiver Mapping API Functions

API Method Name	Comments
adi_adrv904x_TxToOrxMappingConfigGet()	Retrieves the Tx to ORx mapping configuration setup.
adi_adrv904x_TxToOrxMappingSet()	Sets the Tx to ORx external signal routing for Tx calibrations that use the ORx for observation.
adi_adrv904x_TxToOrxMappingGet()	Retrieves the Tx to ORx external signal routing for Tx calibrations that use the ORx for observation.
adi_adrv904x_TxToOrxMappingPresetAttenSet()	Sets ORx preset attenuation value for selected Tx channel(s) to be used when mapped to an ORx channel.
adi_adrv904x_TxToOrxMappingPresetAttenGet()	Retrieves ORx preset attenuation value for selected Tx channel to be used when mapped to an ORx channel.
adi_adrv904x_TxToOrxMappingPresetAttenGet_v2()	Retrieves ORx preset attenuation value for selected Tx channel to be used when mapped to an ORx channel.
adi_adrv904x_TxToOrxMappingPresetNcoSet()	Sets ORx preset numerically controlled oscillator (NCO) values for selected Tx channel(s) to be used when mapped to an ORx channel.
adi_adrv904x_TxToOrxMappingPresetNcoGet()	Retrieves ORx preset NCO values for selected Tx channel to be used when mapped to an ORx channel.
adi_adrv904x_TxToOrxMappingPresetNcoGet_v2()	Retrieves ORx preset NCO values for selected Tx channel to be used when mapped to an ORx channel.
adi_adrv904x_TxToOrxMappingAndVswrDirSet()	Sets the Tx to ORx mapping and VSWR direction.
adi_adrv904x_TxToOrxMappingAndVswrDirGet()	Retrieves the Tx to ORx mapping and VSWR direction.

STREAM PROCESSOR AND SYSTEM CONTROL

TRANSMITTER TO OBSERVATION RECEIVER MAPPING—DFE MODE

In RCI Hybrid or DFE mode, VSWR enable via GPIO is not required as the scheduler schedules the VSWR calibration. The observation receiver switch control and the control signal for forward/reverse switch control are generated from the ADRV904x. The user needs to route preassigned GPIO for switch controls.

In DFE mode, the user can only support up to 5 GPIOs per ORX channel. GPIOs can be assigned based on the required bit configuration for customer hardware (SP4T and SP2T switches).

The user/customer can select the GPIOs in the 24-bit digital GPIO pin selection and 16 analog GPIO pin selection.

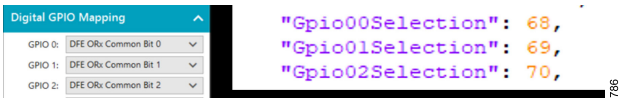
Refer to the enums section in the adrv904x_radioctrl_types.h from the software package for more details.

```
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX0_MAP_BIT0          = 60U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX0_MAP_BIT1          = 61U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX0_MAP_BIT2          = 62U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX0_MAP_BIT3          = 63U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX1_MAP_BIT0          = 64U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX1_MAP_BIT1          = 65U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX1_MAP_BIT2          = 66U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX1_MAP_BIT3          = 67U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX_CMN_MAP_BIT0       = 68U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX_CMN_MAP_BIT1       = 69U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX_CMN_MAP_BIT2       = 70U,
ADRV904X_STREAM_GPIO_DFE_TX_TO_ORX_CMN_MAP_BIT3       = 71U,
```

785

Figure 85. GPIOs Enum

The below Figure 86 is an example of how 68 to 70 for a GPIO will be assigned in the JSON when the DFE ORX common bits are assigned.



786

Figure 86. GPIOs Assignment in the JSON

In DFE mode, the mapping and the direction are configured in ADRV904x through GUI stream settings or JSON update. The GPIOs assigned in stream settings are utilized to control the external switches. This ensures that both the ADRV904x and the switch are synchronized for the same mapping and FWD/REV direction.

The Tx to Orx mapping mode should be selected as DFE control mode (enum 32 in JSON) as shown below.

"TxToOrxMappingMode": 32

STREAM PROCESSOR AND SYSTEM CONTROL

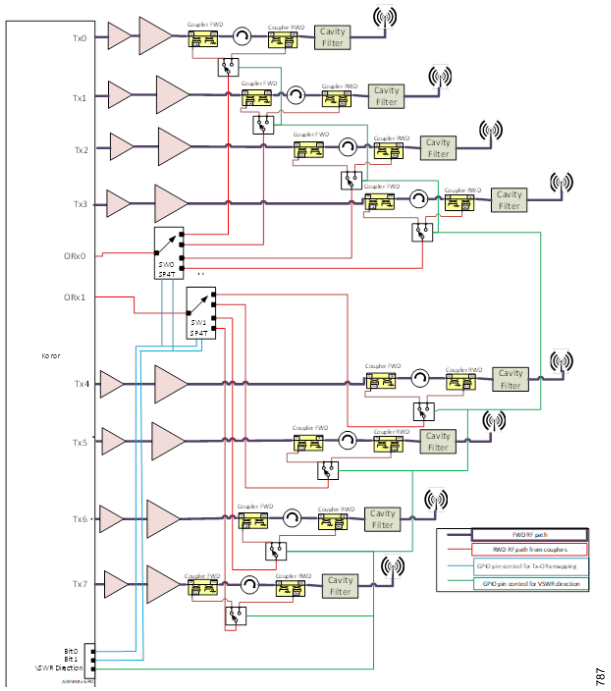


Figure 87. Tx-ORx Mapping Mode and GPIO Mapping

Table 38 represents the physical mapping for the example use case shown in the Figure 87.

Table 38. Tx-ORx Mapping, DFE Mode

VSWR Direction	SP4T Control			
D2	D1	D0	ORx0 Mapping	ORx0 Mapping
1	0	0	Tx0 – DPD / VSWR FWD	Tx4 – DPD / VSWR FWD
1	0	1	Tx1 – DPD / VSWR FWD	Tx5 – DPD / VSWR FWD
1	1	0	Tx2 – DPD / VSWR FWD	Tx6 – DPD / VSWR FWD
1	1	1	Tx3 – DPD / VSWR FWD	Tx7 – DPD / VSWR FWD
0	0	0	Tx0 – VSWR REV	Tx4 – VSWR REV
0	0	1	Tx1 – VSWR REV	Tx5 – VSWR REV
0	1	0	Tx2 – VSWR REV	Tx6 – VSWR REV
0	1	1	Tx3 – VSWR REV	Tx7 – VSWR REV

This is mapped to meet the example hardware requirement of Table 38. Users can customize the pin table in ACE GUI based on the external circuitry. If separate controls are required for ORx0 and ORx1, the user should use DFE ORx Bit 0 to Bit 2 for ORx0 and Bit 4 to Bit 6 for ORx1. (Note that this is just an example, any GPIO can be selected.)

RADIO SEQUENCER

This section provides an overview of the radio sequencer (RS) features to be made available in the ADRV904x transceiver. The document includes a hardware architecture overview, an overview of how to generate binary for the radio sequencer, and a summary of API software interface functions to configure the radio sequencer.

Radio System Level Overview

Radio sequencer is like a main timing controller in a transceiver that is in sync with system SSB sync. SSB sync is the timing reference from the distribution unit (DU) to make sure the frame timing across base stations is aligned over air to avoid interference between colocated base stations. Figure 88 shows only the control flow in transceiver.

STREAM PROCESSOR AND SYSTEM CONTROL

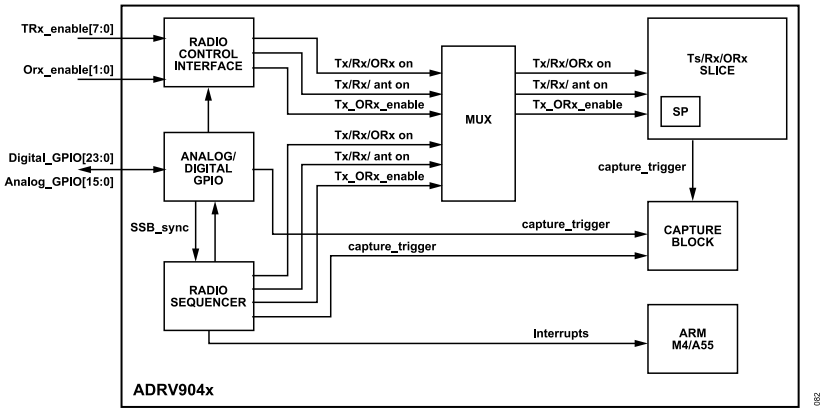


Figure 88. Radio Sequencer Block in ADRV904x Transceiver

Radio Sequencer Feature

In previous generation transceivers, all the controls for the transmitter/receiver/observation receiver channel enables and external triggers are from the baseband processor through an on-chip GPIO pin. This required many GPIO pins from the BBP to the transreceiver and take a lot of board space. Generally, in any base station, SSB_SYNC output the absolute reference for air timing, is available and used to generate all the control signals that drive transceivers, external switches, and power amplifier (PA) controls.

The ADRV904x transceiver provides legacy radio control interface along with radio sequencer. Moreover, radio sequencer takes the SSB_SYNC as input and generates all the control signals required internally to the transceiver, external switches, and PA controls. The radio sequencer is designed with a lot of flexibility to make sure customers can program complex sequences and save considerable number of pins as BBP needs to provide only SSB_SYNC and a way to switch pattern using an API.

Radio sequencer is in general a pattern generator with programmable period and timing. It supports both 5G and 4G timing including extended CP case. The radio sequencer introduces a concept of multiframes for convenience to the user. The actual SSB sync signal within a 5G frame (or the SSB_SYNC pin) might not correspond to a frame boundary and the period might span multiframes, it may be more useful to have pattern repetition aligned to frame boundaries as standard defines the timing in term of frames. A multiframe boundary is defined with the same period as SSB_SYNC or can be configured with the 3GPP frame time of 10 ms, but with an arbitrary phase offset to be chosen by the user. This helps to synchronize radio sequencer engines between multiple ADRV904xs by using on-chip multichip synchronization (MCS). Analog Devices recommends configuring same as 3GPP frame time of 10 ms as this helps in enabling switching patterns on 10 ms frame boundary, and since radio sequencer switches pattern on its internal multiframe boundary, it is faster to start a pattern or switch between patterns. Maximum multiframe boundary allowed is 40 ms. The radio sequencer binary loading, initialization, and GPIO configuration are done in MCS period. The radio sequencer feature has two modes to generate SSB sync signal by configuring few parameters as explained in the following sections.

Radio Sequencer Modes of Operation

Internal SSB_SYNC Mode

In this mode, SSB sync is generated internally depending on the configuration done for symbol period, 4G or 5G standard, and normal or extended CP. However, the external SSB sync signal is required to measure the skew, and this needs to be read back by BBP and adjusted to align multiple ADRV904xs. The internal SSB_SYNC mode provides automatic monitoring of subsequent SSB_SYNC pulse. For every SSB_SYNC pulse, the skew is latched and can be read back.

However, there is no specific phase error calculation in hardware for the internal SSB_SYNC mode. Also, there is no tracking where the counter or phase is automatically adjusted. BBP is required to read this periodically and take corrective action. In internal SSB_SYNC mode, there is no status reporting for phase error between external and internal SSB sync. Analog Devices recommends using this mode because the timing is synced to the system master clock and any variation in SSB sync does not get propagated to airtime. As mentioned, BBP can read the phase offset and decide to correct or not depending on the system deployment scenarios. The procedure to adjust the phase offset is given in phase offset correction procedure.

STREAM PROCESSOR AND SYSTEM CONTROL

External SSB_SYNC Tracking Mode

In this mode, every external SSB sync is used to track and shift the internal SSB sync to make sure the skew across the ADRV904x is maintained without intervention of BBP. The external SSB_SYNC tracking mode is meant to keep the radio timing aligned to the global timing reference if the timing drifts due to temperature changes and other system wide considerations (for example, global time updates). Since the SSB_SYNC pin is not captured synchronously across multiple devices, exact determinism in frame timing is not possible. However, small changes in phase from the resynchronized SSB_SYNC pin should not cause detrimental effects in the sequencer.

Radio sequencer also has a feature to switch pattern in run time, and the new pattern is applied on the next SSB sync. This can be used to switch to customer specific patterns for factory testing, calibration, and field antenna calibration

Radio Sequencer Hardware Overview

Radio sequencer is a radio timing pattern generator based on a simple very long instruction word (VLIW) instruction set. There are 16 identical sequence controllers driving 16 controls from each, giving 256 controls that are multiplexed to all internal and external controls in order that any sequencer output can be routed to any control. This very flexible feature to assign any sequencer output to any of the GPIO pins can be used for debugging and verification.

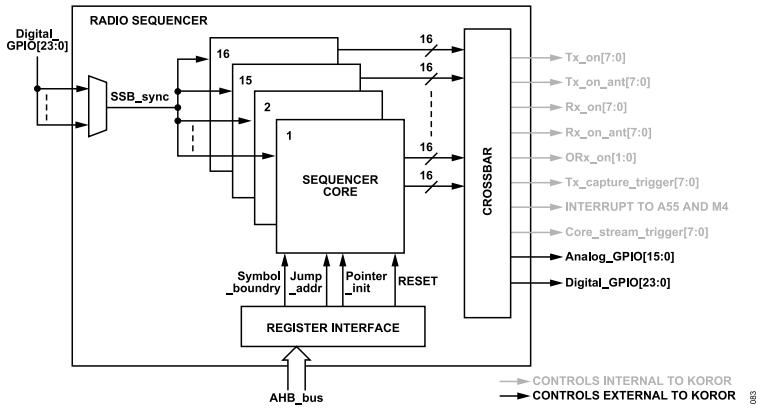


Figure 89. Simplified Block Diagram of Radio Sequencer

The sequencer executes instructions on symbol boundary and is designed to take care of normal and extended CP symbol time internally depending on the configuration done through parameters in the JSON file, which is used to generate the profile using a configurator. The parameters available in JSON file, which corresponds to radio sequencer, are listed as follows:

```
“radioCtrlPreInit”: {
“radioSequencerSsbSyncGpioCtrl”: 2, // GPIO input assigned for timing reference signal SSB_Sync
“radioSequencerGpioDigOut”: [ 0,2, ---,23], // Digital GPIO assigned for front-end modules and switches
“radioSequencerGpioAnaOut”: [ 0,2, --,15] // Analog GPIO assigned for front-end modules and switches
“radio_seq”: {
“enable”: 1, // parameter to enable RS or use legacy mode of controlling the TRx
“numerology”: “RADIO_SEQ_NUMEROLOGY_2”, // 5G numerology and μ = 0 numerology represents
 15 kHz which is same as LTE.
“ssb_sync_mode”: 0, // mode used to sync the internal SSB with external SSB 0 - internal mode 1 -
external tracking mode
“ssb_period_in_ms”: 10 # // period to generate SSB sync internally, ADI recommend using 10ms
(3GPP frame time)
}
```

The enum for numerology and the valid values are as follows:

```
enum RadioSeqNumerology_e
{
RADIO_SEQ_NUMEROLOGY_0 = 0,
```

```
RADIO_SEQ_NUMEROLOGY_1 = 1,
RADIO_SEQ_NUMEROLOGY_2 = 2,
RADIO_SEQ_NUMEROLOGY_2_ECP = 3,
RADIO_SEQ_NUMEROLOGY_3 = 4,
RADIO_SEQ_NUMEROLOGY_4 = 5
};
```

Radio Sequencer Organization

There are 16 RSs and to minimize the use of separate sequencers for various controls, there is a predefined set of patterns executing on a dedicated RS, which reduces the complexity of the generator and the user interface. The predefined RS patterns are as follows:

- Null pattern—this is utilized by all the RS to start the initial pattern after reset is released. This is taken care by the CLI/GUI tool.
- Capture/measurement patterns—two RSs are allocated for running capture/measurement related sequencer patterns, one for each side of the chip (ORx0 and ORx1 side), these patterns are defined by Analog Devices and are not user-programmable.
- Frame timing pattern—there are 10 RSs allocated for frame timing to independently generate transmitter/receiver/observation receiver and additional controls as required. These RSs are user-programmable. All unused RSs dedicated to frame timing are loaded with null pattern by the CLI/GUI tool.

Table 39.

Description	Run Under Sequencer
Null pattern (Address 0x0)	All
Frame timing pattern	0
Frame timing pattern	1
Frame timing pattern	2
Frame timing pattern	3
Frame timing pattern	4
Frame timing pattern	5
Frame timing pattern	6
Frame timing pattern	7
Frame timing pattern	8
Frame timing pattern	9
ORx0 side of the chip—DPD and VSWR capture pattern not requiring RS capture trigger	10
ORx0 side of the chip—CLGC pattern	10
ORx1 side of the chip—DPD and VSWR capture pattern not requiring RS capture trigger	11
ORx1 side of the chip—CLGC pattern	11
Reserved	12 to 15

Radio Sequencer GUI

This GUI is used to configure required patterns for all the internal and external controls, and then generate a binary that is loaded during initialization, configuration, and control by the baseband.

As noted in the Radio Sequencer Organization section, the sequencer fetches the instruction on symbol boundaries. The GUI converts the timing in terms of symbol to generate a binary sequence. The steps for generating an RS binary using GUI is given in the Radio Sequencer Binary Generation Using GUI section.- For generating an RS binary using GUI by passing the JSON file, the JSON format is as follows:

```
{
"ConfigData": {
"GenerateSource": false, // If true, dump configuration and source files to current dir
"SourceFiles": [], // Path to RS image file to generate. Can be either relative to the working dir, or
absolute.
"XbarSettings": {} //
},
```

STREAM PROCESSOR AND SYSTEM CONTROL

```
// Signal timing information. A set of signal timing declarations organized into “patterns” per radio
sequencer.
// Each pattern is composed of one or more “sequences”. A sequence contains one or more sets of
signals and their associated timing specifications.
// All timing information is specified in terms of 5G symbols, unless specified.
“SignalTimingData”: {
“Sequencers”: {
“FRAME_TIMING_0”: { // sequencer 0
“Patterns”: [ // List of patterns in sequencer 0
{
“Sequences”: [ // Set of sequences within pattern 0 When the last sequence in a pattern completes, it
loops back to the first sequence
{ // Specifies a sequence with a duration of 70 symbols that loops forever before moving to the next
sequence. If the user doesn’t specify 70 symbols’ worth of timing information below, the sequence will
be “padded” such that its interval is 70 symbols.
 “Iterations”: 0, // An infinite sequence is created by setting iterations : 0
“Duration”: 70, // total duration of sequence in terms of symbols.
 “Signals”: [ // Specifies the set of signals controlled in this sequence, and when they are ena▶
bled/disabled. Only one set is allowed at this time.
{
“Controls”: [ // Control TX0 to TX7 enable
 “DIG_GPIO_0_EN”,
“TX_0_CH_EN”,
“TX_1_CH_EN”,
“TX_2_CH_EN”,
“TX_3_CH_EN”,
“TX_4_CH_EN”,
“TX_5_CH_EN”,
“TX_6_CH_EN”,
“TX_7_CH_EN”
],
“Timing”: [ // Enable at symbol #0 for 48 symbols
{
“Enable”: 0,
“Duration”: 48,
“EnableFineUs”: 0.0, // This is the delay from the symbol boundary before the assertion takes place.
 This cannot be for the first symbol. i.e if “Enable”=0
“DisableFineUs”: 2 // This is the delay from symbol boundary before a desertion takes place. This
cannot be for the last symbol of total duration.
},
 { // Enable again at 69 for 1 symbol, with a fine delay of 29.68us
// The resolution of enableFineUs is 1/122.88MHz = 8.14ns.
“Enable”: 69,
“Duration”: 1,
“EnableFineUs”: 29.68
“DisableFineUs”: 0.0
}
]
}
]
}
]
}
]
}
}
```

STREAM PROCESSOR AND SYSTEM CONTROL

```
}
}
```

To understand how the JSON parameters are configured, consider the example shown in Figure 91. Here, the frame periodicity is 2.5 ms and the JSON example given is only for the transmitter enable signal as captured in the code snippet in Figure 92. As mentioned earlier, the RS is VLIW-based and executes instructions on symbol boundary. Analog Devices recommends configuring the controls in terms of symbols to define the timing for a full frame or at least for the frame periodicity to ensure there is no residue that can create a time shift over time. The fine time shift which is required to adjust the delays and skew between controls can be set using **DisableFineUs** and **EnableFineUs** in JSON.

In 3GPP, there are various slot formats defined in TS 38.213, as shown in Figure 90. Depending on the deployment, the customer configures the slot format. For example, we have taken the slot sequence DDDSU (D = downlink, S = special, and U = uplink) for 2.5 ms periodicity for Numerology 1 as shown in Figure 90.

TABLE 11.1.1-1: SLOT FORMATS FOR NORMAL CYCLIC PREFIX

FORMAT	SYMBOL NUMBER IN A SLOT													
	0	1	2	3	4	5	6	7	8	8	10	11	12	13
0	D	D	D	D	D	D	D	D	D	D	D	D	D	D
1	U	U	U	U	U	U	U	U	U	U	U	U	U	U
45	D	D	D	D	D	D	F	F	U	U	U	U	U	U

084

Figure 90. 3GPP Slot Format as Defined in TS 38.213

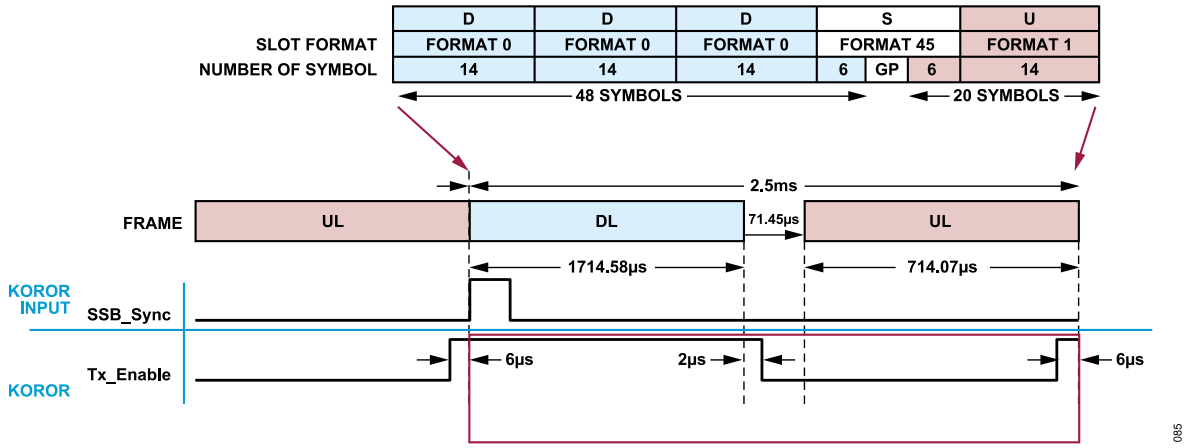


Figure 91. Mapping of Transmitter Enable to Frame Timing

STREAM PROCESSOR AND SYSTEM CONTROL

Figure 92. Snippet from JSON

The example timing taken is for total of 70 symbols—48 downlink (DL) symbols, two guard periods (GPs), and 20 uplink (UL) symbols. In the snippet from JSON, control is for transmitter enable, and Symbol 0 of frame is when SSB_SYNC trigger is initiated. Symbol 0 is set to **Enable** and the duration as 48 symbols. The fine delay of 2 µs is set using **DisableFineUs** and is calculated using **SampleClkfreqkhz**; the default sample clock frequency is 12.88 MHz. For the left over time of 22 symbols, transmitter enable is low, but at the end there is an enable time of 6 µs. Therefore, Symbol 69 is enabled and enable is delayed using **EnableFineUs**, which is calculated as the difference of symbol time and the time when the signal needs to be low (symbol time − 6 µs).

Radio Sequencer API

Table 40. List of Radio Sequencer Related API Functions

API Method Name	Description
adi_adrv904x_RadioSequencerPhaseSet()	Sets the phase offset of the internal generated SSB sync signal.
adi_adrv904x_RadioSequencerPhaseGet()	Retrieves the phase error between the external SSB sync and internal SSB sync.
adi_adrv904x_RadioSequencerPreMcsCfg()	This function is used during the preMcsInit sequence in order to load the radio sequencer configuration.
adi_adrv904x_RadioSequencerPostMcsCfg()	This function is used during the postMcsInit sequence in order to load the radio sequencer configuration.
adi_adrv904x_RadioSequencerRadioStart()	Starts radio operation using the default frame timing pattern (Pattern 0) for each of the frame timing sequencers (0 to 9). If a sequencer does not have a frame timing pattern defined, it executes the null pattern.
adi_adrv904x_RadioSequencerRadioStop()	Stops radio operation by setting all 10 frame timing sequencers to execute the null pattern. This operation occurs immediately, that is it does not wait for a multiframe boundary.

STREAM PROCESSOR AND SYSTEM CONTROL

Phase Offset Correction and Radio Sequencer Start Procedure

Analog Devices recommends using internal SSB sync mode to generate the sync with required period. Phase error can be adjusted at different scenarios, depending on the requirement customer can decide which one to use. Phase offset can be adjusted after initialization for every boot or one time as a factory calibration or during run time. In any scenario external SSB is required to get the phase error. The procedure to adjust the phase offset between the external SSB sync and the internally generated SSB sync is as follows:

1. Configure the SSB sync period and mode in JSON under the **"radio seq"** tag.

```
"radio_seq": {
"enable": 1, // parameter to enable RS or use legacy mode of controlling the TRx
"numerology": "RADIO_SEQ_NUMEROLOGY_2", // 5G numerology and μ = 0 numerology represents 15 kHz
which is same as LTE.
"ssb_sync_mode": 0, // mode used to sync the internal SSB with external SSB
"ssb_period_in_ms": 10 // period to generate SSB sync internally, ADI recommend using 10ms (3GPP
frame time)
```

2. Send external SSB sync after MCS is complete, and read the phase error using the **adi_adrv904x_RadioSequencerPhaseGet()** API function.
3. Correct the phase error using the **adi_adrv904x_RadioSequencerPhaseSet()** API function.
4. Configure the radio in pin control mode using the **adi_adrv904x_RadioCtrlCfgGet()** API function.
5. Once the phase is adjusted and radio is put in pin control mode the radio sequencer needs to be started using the **adi_adrv904x_RadioSequencerRadioStart()** API function.

For run time case stop the radio sequencer using **adi_adrv904x_RadioSequencerRadioStop()**, then run Step 2 and Step 3, and start the radio sequencer using **adi_adrv904x_RadioSequencerRadioStart()**.

Radio Sequencer Binary Generation Using GUI

Follow the below steps to generate an RS binary using the GUI:

1. First, build the JSON with the required timing as explained in JSON input to GUI. It is referred to as the RS timing JSON.
2. Open the existing use case profile JSON file at the link

   C:\ProgramData\AnalogDevices\ACE\Plugins\Board.ADRV9040.x\content\DeviceResources\Adrv9040\profiles as shown in Figure 93.

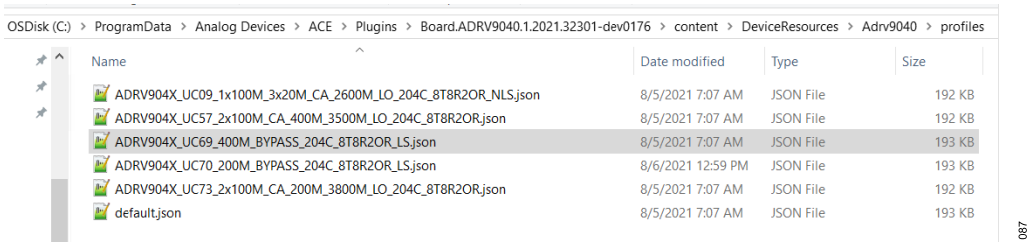


Figure 93.

3. Configure the GPIO already assigned for radio sequencer in the RS timing JSON as shown in Figure 94.

Figure 94.

STREAM PROCESSOR AND SYSTEM CONTROL

4. In the GUI, select the use case with changes done, and click on the Generate Profiles button as shown in Radio Sequencer Binary Generation Using GUI.

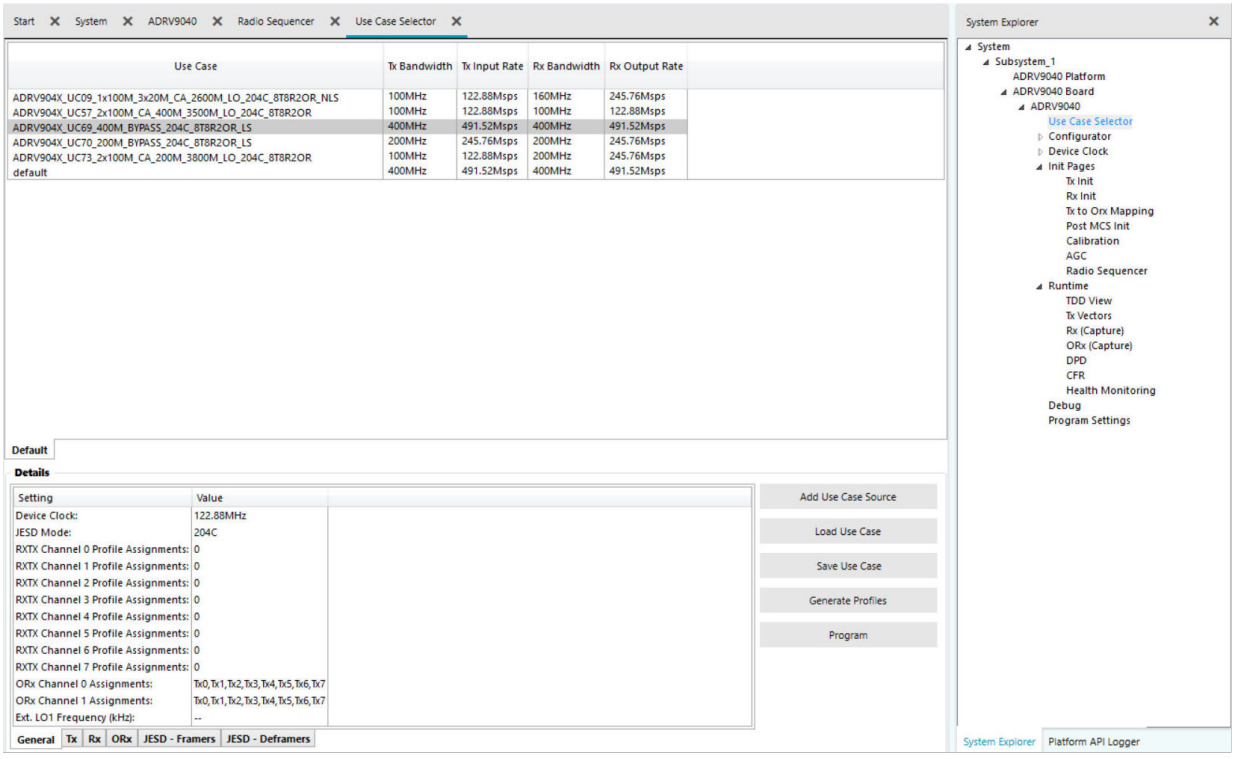


Figure 95.

5. Select the **Radio Sequencer** page under **Init Pages**. Click the **Enable** checkbox under the **Radio Sequencer** page and configure the parameters as shown in Figure 96.

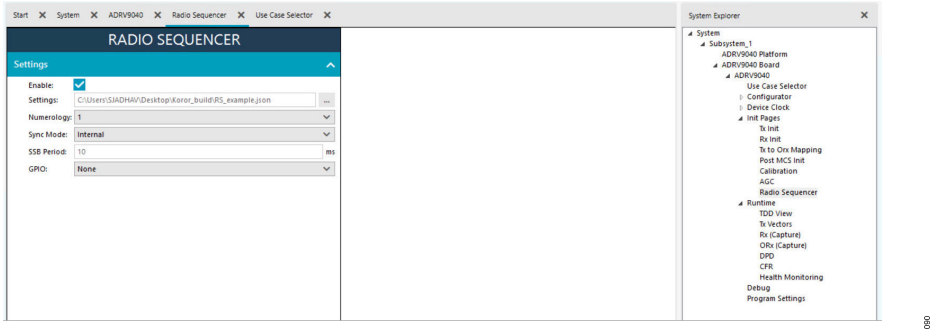


Figure 96.

6. Go to the ADRV9040 main page and click **Program Settings** as shown in Figure 97.

STREAM PROCESSOR AND SYSTEM CONTROL

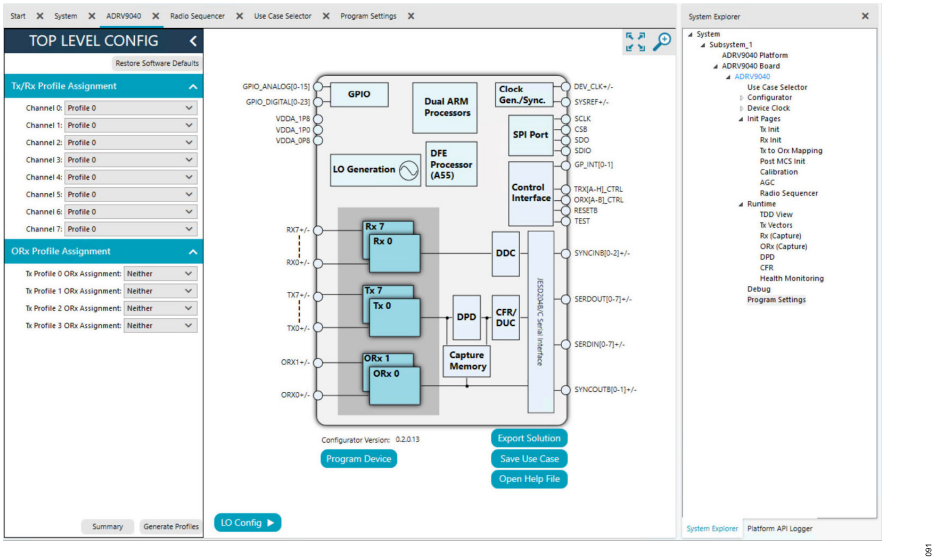


Figure 97.

7. After programming, all the resource files and binaries become available at **C:/Users/%USERNAME%/AppData/Local/Analog Devices /ACE/PluginFiles/Board.ADRV9040/Local/resources**.
8. Run the IronPython script to start the RS.

IronPython Script to Run the RS

```
import clr
import System
from System import *
import sys
import os
import glob # for finding plug-in's installation path
import math

ACE_DIRECTORY = 'C:\Program Files (x86)\Analog DevicesACE'

CLIENT_DIRECTORY = glob.glob(r'C:\ProgramData\Analog Devices\ACE\Plu▶
gins\Board.ADRV904*')[0] + 'lib'    # For ADRV904x

DEFAULT_IPADDRESS = '192.168.1.10'
DEFAULT_PORT = '5000'

# Add ACE and client to path
sys.path.append(ACE_DIRECTORY)
sys.path.append(CLIENT_DIRECTORY)

# Verify ADRV9040 plugin version used
print 'Loading client DLLs from', CLIENT_DIRECTORY
clr.AddReference("AnalogDevices.EvalClient")
clr.AddReference("AnalogDevices.EvalClient.Installers")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Ad9528.Device")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Board")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Device")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Fpga.Device")
clr.AddReference("AnalogDevices.EvalClient.Adrvgen6.Platform")
```

STREAM PROCESSOR AND SYSTEM CONTROL

```
clr.AddReference("Adrv904x.Configurator")
clr.AddReference("StreamGen")

from AnalogDevices.EvalClient import *
from AnalogDevices.EvalClient.Installers import *
from AnalogDevices.EvalClient.Adrvgen6.Board import *
from AnalogDevices.EvalClient.Adrvgen6.Board.BaseClasses import *
from AnalogDevices.EvalClient.AD9528 import *
from AnalogDevices.EvalClient.Adrvgen6.Ad9528.Device import *
from AnalogDevices.EvalClient.Adrvgen6.Fpga.Device import *
from AnalogDevices.EvalClient.FPGAGEN6 import *
from AnalogDevices.EvalClient.Device import *
from AnalogDevices.EvalClient.AdiCommon import *
from AnalogDevices.EvalClient.Adrvgen6 import *
from AnalogDevices.Adrvgen6.Platform import *
from AnalogDevices.Adrv904x.Configurator import *
from Adi.Design import *

def connect(ipAddress="", portNumber=""):
       """
       Connect IronPython to the command server.
       If TCP connection exceptions are seen, please verify that
       ACE is disconnected from the ADRV9040 command server (see header).
       """
       ipAddress = ipAddress or DEFAULT_IPADDRESS
       portNumber = portNumber or DEFAULT_PORT
       print 'Attempting to connect to {}:{}...'.format(ipAddress, portNumber)
       EvalClientManager.Instance.Initialize(CLIENT_DIRECTORY)
       transport = Transports.CreateDefaultTcpTransport(ipAddress + ':' + portNumber)
       context = ExecutionContext(transport)
       context.ErrorRetriever = ErrorRetriever()
       platform = EvalClientManager.Instance.PlatformBuilder.CreatePlatform('', context)
       platform.Timeout = 5000
       platform.OutputFilesBasePath = '.'
       configGen = ConfigGen()
       return platform

def spiWriteByte(address, data):
       data = Array[Byte]([data])
       numBytes = len(data)
       adrv904x.hal.RegistersByteWrite(None, address, data, numBytes)
       print 'spiWriteBytet0x{:04X}t0x{:02X}'.format(address, int(data[0]))

def spiReadByte(address, numBytes=1):
       data = Array[Byte]([0])
       adrv904x.hal.RegistersByteRead(None, address, data, None, numBytes)
       data = int(data[0]) # get first element of returned data array
       print 'spiReadBytet0x{:04X}t0x{:02X}'.format(address, data)
       return data

def adrv904x_RxTxEnableGet():
       c_en = adrv904x.radioctrl.RxTxEnableGet(0,0,0)
       print("Orx channel enabled =t",hex(c_en[1]))
       print("Rx channel enabled  =t",hex(c_en[2]))
       print("Tx channel enabled  =t",hex(c_en[3]))
       return c_en
```

STREAM PROCESSOR AND SYSTEM CONTROL

```
def adrv904x_RadioCtrlCfgGet(rxch,txch):
       radcfg = adi_adrvgen6_RadioCtrlModeCfg_t()
       z = adrv904x.radioctrl.RadioCtrlCfgGet(rxch,txch,radcfg)
       print "Tx mode     = ",radcfg.txRadioCtrlModeCfg.txEnableMode
       print "Tx channel  = ",tx_channel[int(math.log(radcfg.txRadioCtrlModeCfg.txChannelMask,2))]
       print "Rx mode     = ",radcfg.rxRadioCtrlModeCfg.rxEnableMode
       print "Rx channel  = ",rx_channel[int(math.log((radcfg.rxRadioCtrlModeCfg.rxChannel▶
Mask & 0xFF),2))]
       print "Orx mode    = ",radcfg.orxRadioCtrlModeCfg.orxEnableMode
       print "Orx channel = ",rx_channel[int(math.log((radcfg.orxRadioCtrlModeCfg.orxChannel▶
Mask & 0x300),2))]

if __name__ == '__main__':

       # Connect IronPython to command server
       platform = connect()

       # Get device objects
       board = platform.Boards[0]
       ad9528Dev = board.DeviceGet('ad9528', 0)
       fpgaDev = board.DeviceGet('fpga', 0)
       try:
            adrv904x = board.DeviceGet('adrvgen6', 0)
       except SystemError: # SW0.5 onwards
            adrv904x = board.Adrvgen6DeviceGet(0)

       tx_channel = [adi_adrvgen6_TxChannels_e.ADI_ADRVGEN6_TX0,adi_adrv▶
gen6_TxChannels_e.ADI_ADRVGEN6_TX1,adi_adrvgen6_TxChannels_e.ADI_ADRVGEN6_TX2,adi_adrvgen6_TxChan▶
nels_e.ADI_ADRVGEN6_TX3,adi_adrvgen6_TxChannels_e.ADI_ADRVGEN6_TX4,adi_adrvgen6_TxChannels_e.ADI_ADRV▶
GEN6_TX5,adi_adrvgen6_TxChannels_e.ADI_ADRVGEN6_TX6,adi_adrvgen6_TxChannels_e.ADI_ADRVGEN6_TX7]

       rx_channel = [adi_adrvgen6_RxChannels_e.ADI_ADRVGEN6_RX0,adi_adrv▶
gen6_RxChannels_e.ADI_ADRVGEN6_RX1,adi_adrvgen6_RxChannels_e.ADI_ADRVGEN6_RX2,adi_adrv▶
gen6_RxChannels_e.ADI_ADRVGEN6_RX3,adi_adrvgen6_RxChannels_e.ADI_ADRVGEN6_RX4,adi_adrvgen6_RxChan▶
nels_e.ADI_ADRVGEN6_RX5,adi_adrvgen6_RxChannels_e.ADI_ADRVGEN6_RX6,adi_adrvgen6_RxChannels_e.ADI_ADRV▶
GEN6_RX7,adi_adrvgen6_RxChannels_e.ADI_ADRVGEN6_ORX0,adi_adrvgen6_RxChannels_e.ADI_ADRVGEN6_ORX1]

       #Read the channels enabled and radio configuration
       adrv904x_RxTxEnableGet()
       adrv904x_RadioCtrlCfgGet(rx_channel[8],tx_channel[0])

       #Configure the radio to pin mode
       radcfg1 = adi_adrvgen6_RadioCtrlModeCfg_t()

       radcfg1.txRadioCtrlModeCfg.txEnableMode      = adi_adrv▶
gen6_TxEnableMode_e.ADI_ADRVGEN6_TX_EN_PIN_MODE
       radcfg1.txRadioCtrlModeCfg.txChannelMask     = 0xff
       radcfg1.rxRadioCtrlModeCfg.rxEnableMode      = adi_adrv▶
gen6_RxEnableMode_e.ADI_ADRVGEN6_RX_EN_PIN_MODE
       radcfg1.rxRadioCtrlModeCfg.rxChannelMask     = 0xff
       radcfg1.orxRadioCtrlModeCfg.orxEnableMode     = adi_adrvgen6_OR▶
xEnableMode_e.ADI_ADRVGEN6_ORX_EN_PIN_MODE
       radcfg1.orxRadioCtrlModeCfg.orxChannelMask  = 0x300
       adrv904x.radioctrl.RadioCtrlCfgSet(radcfg1)
```

STREAM PROCESSOR AND SYSTEM CONTROL

```
        # Start the radio Sequencer
        adrv904x.radioctrl.RadioSequencerRadioStart()
```

FRONT-END ANALOG SIGNAL PATH

TRANSMIT PATH

Figure 98 illustrates the transmit analog front end used in the ADRV904x family of devices. It is duplicated eight times for each transmitter. Each transmitter consists of a tuning baseband low-pass filter, upconvert quadrature mixers, and an RF variable gain amplifier. The baseband filter has a tunable bandwidth of 300 MHz to 840 MHz. The bandwidth is tuned at initialization via the loopback path and implemented in the Arm firmware. The upconverter has fixed gain that reduces quadrature errors, which are associated with attenuation changes in the mixer stages. A unique architecture of upconverters is chosen to reduce 3rd and 5th harmonics. Lowering these harmonics reduces the linearity requirements of the RF variable gain amplifier (VGA) as well as reduces the risk of aliasing these harmonics in the transmitter loopback path.

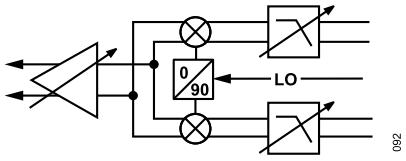


Figure 98. Transmit Analog Front-End Block Diagram

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

TRANSMITTER ATTENUATION CONTROL

The ADRV904x uses an accurate and efficient method of transmit power control (transmitter 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 Digital GPIO Input Modes for more details on GPIO attenuation controls.

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

TRANSMITTER ATTENUATION API FUNCTIONS

The following functions can be used after device initialization to reconfigure and control the transmitter attenuators settings. A short description of each API and any usage limitations are listed in Table 41. Details of the parameters, members, and enums for each command are presented in the doxygen help files included with each software build. Refer to those files when developing a software code. The structures and enumerators for these API functions are detailed in the doxygen documentation.

Table 41. List of Transmitter Attenuation Related API Functions

API Method Name	Comments
adi_adrv904x_TxAttenTableRead()	Reads Tx RF output attenuation table data.
adi_adrv904x_TxAttenSet()	This function allows the user to set the desired attenuation in units of mdB.
adi_adrv904x_TxAttenGet()	This function allows the user to read the desired attenuation in units of mdB.
adi_adrv904x_TxAttenCfgSet()	This function allows the user to configure the attenuation mechanism for one or more channels.
adi_adrv904x_TxAttenCfgGet()	This function allows the user to read the configuration of the attenuation mechanism for one or more channels.
adi_adrv904x_TxAttenPhaseSet()	Sets the Tx attenuation phase.
adi_adrv904x_TxAttenPhaseGet()	Reads the Tx attenuation phase.
adi_adrv904x_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.

FRONT-END ANALOG SIGNAL PATH

Table 41. List of Transmitter Attenuation Related API Functions (Continued)

API Method Name	Comments
adi_adrv904x_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_adrv904x_TxAttenUpdateCfgSet()	Sets the Tx attenuation update configuration for several channels: attenuation level source (S0, S1), update trigger (GPIO, SPI, None).
adi_adrv904x_TxAttenUpdateCfgGet()	Reads the Tx attenuation update configuration for several channels: attenuation level source (S0, S1), update trigger (GPIO, SPI, None).
adi_adrv904x_TxAttenUpdate()	Simultaneously updates 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 ADRV904x receiver path is instantiated eight times and shown in Figure 99. 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 receiver channel.

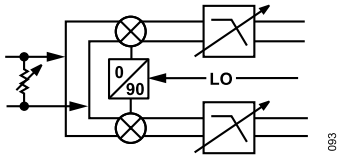


Figure 99. Receive Analog Front-End Block Diagram

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

RECEIVER MANUAL GAIN API FUNCTIONS

Table 42. Receiver Manual Gain API Functions

API Method Name	Comments
adi_adrv904x_RxGainSet()	Sets the Rx channel manual gain index. If the value passed in the gain index parameter is within the range of the gain table minimum and maximum indices, the Rx channel gain index is written to the transceiver.
adi_adrv904x_RxGainGet()	Reads the Rx AGC gain index for the requested Rx channel.
adi_adrv904x_RxMgcGainGet()	Reads the Rx MGC gain index for the requested Rx channel.
adi_adrv904x_RxTempGainCompSet()	Sets the temperature gain compensation parameter for the Rx channel only.
adi_adrv904x_RxTempGainCompGet()	Reads the temperature gain compensation parameter for the Rx channel only. Only one channel can be retrieved per call.
adi_adrv904x_RxGainTableLoad()	Loads the Rx gain table CSV file.
adi_adrv904x_RxGainTableChecksumRead()	Reads the Rx gain table file checksum value.
adi_adrv904x_RxGainTableChecksumCalculate()	Calculates the Rx gain table file checksum value.

OBSERVATION RECEIVER PATH

The ADRV904x observation receiver path is shown in Figure 100 and is instantiated two times for the ORx1 and ORx2 paths. As shown in Figure 100, the observation receiver path implements direct RF sampling. An RF ADC eliminates the need for an LO that eliminates 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 400 MHz. The attenuator also provides 50 Ω on-chip RF matching for the observation receiver input. The attenuator is designed to provide 0 dB, 1 dB, 6 dB, and 12 dB steps. The user must check the maximum operation input voltage level to the attenuator. An external 6 dB resistive pad is required at the input to provide additional return loss to the observation port.

FRONT-END ANALOG SIGNAL PATH

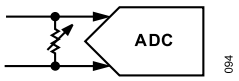


Figure 100. Observation Receive Analog Front-End and ADC Block Diagram

OBSERVATION RECEIVER ATTENUATION API FUNCTIONS

Table 43. Observation Receiver Attenuation API Functions

API Method Name	Comments
adi_adrv904x_OrxAttenSet()	Sets the desired attenuation in units of dB.
adi_adrv904x_OrxAttenGet()	Reads the attenuation in units of dB.

SYNTHESIZER CONFIGURATION

OVERVIEW

The ADRV904x has four PLL synthesizers: clock, RF (×2), 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 VCO cores VCO. The tuning range of the SERDES PLL VCO 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 LO generator (LOGEN) 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 DEVCLK. Figure 101 illustrates the PLL and associated distribution blocks used in the ADRV904x family of devices.

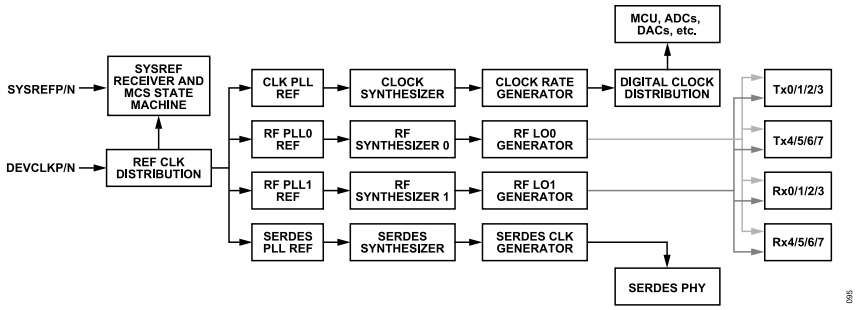


Figure 101. Synthesizer Interconnection and Clock/LO Distribution Block Diagram

DEVCLK

The external DEVCLK is used as a reference clock for the four synthesizers on chip, and, for optimum performance, the DEVCLK must 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 102. The device clock receiver is a noise sensitive differential RF receiver. The frequency range and amplitude specifications are in the data sheet. The equivalent AC circuit is given in Figure 103.

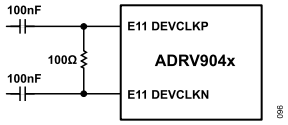


Figure 102. DEVCLK Input Connections

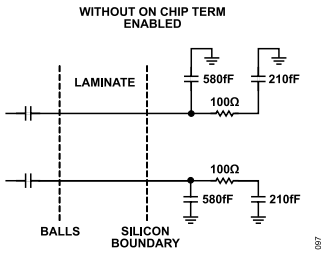


Figure 103. DEVCLK Equivalent AC Circuit

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:

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

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

SYNTHESIZER CONFIGURATION

Where *N* is the multiplier applied to the DEVCLK 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 20 × logN 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 data sheet are typically provided in the caption of the figures.

Figure 104 illustrates three DEVCLK noise gain responses with different loop bandwidths and phase margins. Each response is normalized to 0 dB by subtracting out the 20 × logN term. For example, for an F_{lo} of 2600 MHz and an F_{ref} of 245.76 MHz, the gain is 20.5 dB. 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.

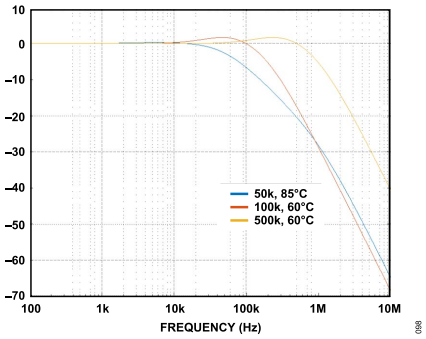


Figure 104. Normalized DEVCLK Noise Gain/PLL Closed-Loop Response

The plots provided in the ADRV904x data sheet are generated with a high quality, low noise reference clock. To meet the data sheet specifications for integrated phase noise, it is recommended to use a clock source with a phase noise at 1 kHz offset as shown in Table 44. This scales 6 dB per doubling of the 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 must evaluate the performance of the PLL with the clock source for their system. As shown in Table 44, the HMC7044 clock generation IC meets the requirements in the second column, while the AD9528 meets the requirements in the third column.

Table 44. 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 105 shows a simulated phase noise result showing the impact of reference clock phase noise on the RF LO phase noise at 3.6 GHz. Because the reference noise is only impacting the phase noise up to ~20 kHz, the integrated phase noise only changes from 0.16° to 0.17° when the DEVCLK source is changed from the HMC7044 to the AD9528.

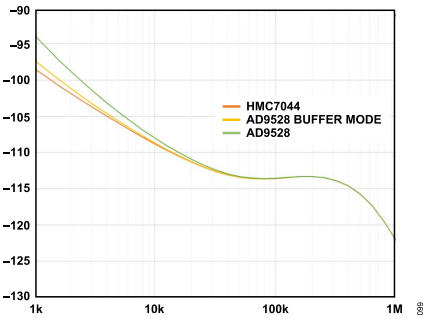


Figure 105. Simulated RF LO Phase Noise at LO = 3.6 GHz Comparing Impact of Different DEVCLK Noise Performance

SYNTHESIZER CONFIGURATION

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 Pin G11 and Pin G12 as shown in Figure 106. The SYSREF traces must 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.

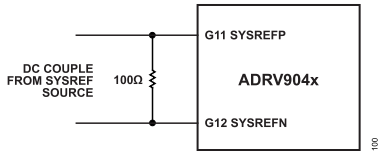


Figure 106. SYSREF Input Connections

SYSREF Setup and Hold Time Requirements

The ADRV904x SYSREF setup and hold time requirements with respect to DEVCLK are in the data sheet. To achieve maximum margin on setup/hold time, it is recommended to align SYSREF with the falling edge of DEVCLK. This would give ~1ns setup/hold time for the case of a 491.52 MHz DEVCLK.

If the SYSREF and device clock arrive at the same time, that is, 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 ADRV904x 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 107. Meeting the setup/hold times ensures SYSREF has no transitions within the keep out window.

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, for example, at the termination resistor.

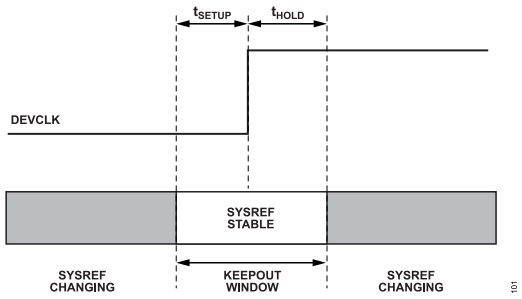


Figure 107. SYSREF Timing Keep Out Window

CLOCK SYNTHESIZER

The clock synthesizer is used to generate all the clocking signals necessary to run the device. Therefore, the clock synthesizer 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 ADRV904x ACE software configure the clock synthesizer appropriately based on the desired I/Q 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, receiver ADC sample and interface clocks, observation receiver and loopback ADC sample and interface clocks, and transmitter DAC sample and interface clocks.

The receiver ADC and transmitter 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 observation receiver ADC either runs at the same rate or twice this. To provide the converter clock rates listed, the CLK PLL operates at either 7372.8 MHz, 7864.2 MHz, or 11796.48 MHz.

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 RF PLL 1 is sourced from the reference generation block of the

SYNTHESIZER CONFIGURATION

device. The RF PLLs are fractional-N architectures. A default modulus value is programmed automatically by the 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 45. The observation receiver NCO also operates on an exact 1 kHz raster, providing 0 Hz error between the transmitter LO and the observation receiver 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 MHz to 7100 MHz. The LO divider boundaries are given in Table 45. It is recommended to rerun the init cals when crossing a divide by 2 boundary or when changing the LO frequency by ±100 MHz or more from the frequency at which the init cals are performed.

Table 45. 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**
RF PLL 0/RF PLL 1	221.875	443.75	443.75	887.75	887.5	1775	1775	3550	3550	7100
Divide by	32		16		8		4		2	

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 108. The network consists of a root divider located close to the RF VCO and separate leaf dividers located at each receiver and transmitter 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 LOs in each transmitter and receiver path and also non power of two LOs in either 4 × transmitters and 4 × receivers (4T4R) section. For example, in a multiband setup, where you want to generate a 3.5 GHz LO for a 4T4R section, 1.8 GHz for 2 × transmitters and 2 × receivers (2T2R) section, and 900 MHz for the remaining 2T2R section, the RF LOs can be configured as follows:

- To generate the 3.5 GHz for Tx0/Rx0 to Tx3/Rx3, RF LO 1 VCO is programmed to 14 GHz followed by divide by 4, where root divider = 2 and leaf divider = 2.
- To generate the 1.8 GHz for Tx4/Rx4 to Tx6/Rx6, RF LO 2 VCO is programmed to 7.2 GHz followed by divide by 4, where root divider = 2 and leaf divider = 2.
- To generate the 900 MHz for Tx6/Rx6 to Tx7/Rx7, RF LO 2 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 ADRV904x, a single RF LO in combination with the on-chip NCOs can be sufficient to support several multiband configurations. In this case, the unused RF LO can be powered down to save power.

Note that it is not recommended to set RF LO 1 = RF LO 2; this can cause unwanted coupling between the two PLLs. If a common RF LO is desired, then either RF PLL 1 or RF PLL 2 must be set to the desired frequency and muxed to both transmitter LO and receiver LO. That is, the configuration must be set to either TX LO = RX LO = RF LO 1 or TX LO = RX LO = RF LO 2 with the unused RF LO powered down.

SYNTHESIZER CONFIGURATION

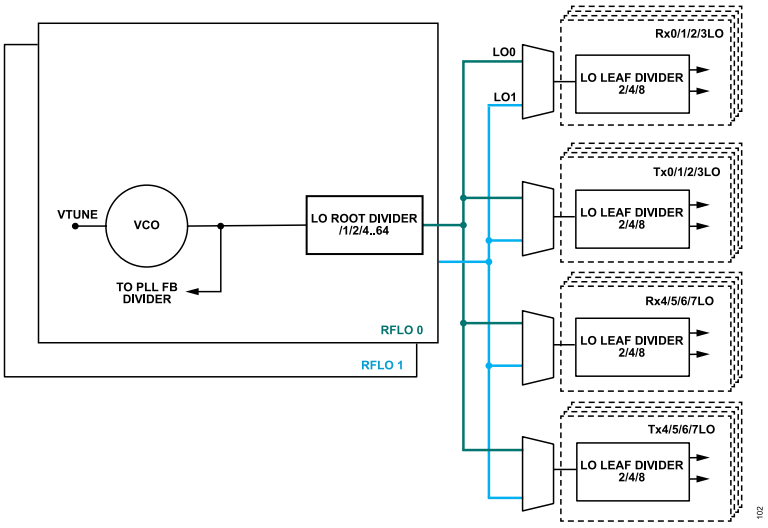


Figure 108. LO Switching Network

LO CONFIGURATION USING API FUNCTIONS

The basic LO configuration, that is whether using a single LO or dual LO and the required receiver and transmitter 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 reconfigure and control the internal LO settings. A short description of each API and any usage limitations are listed in Table 46. Details of the parameters, members, and enums for each command are presented in the doxygen help files included with each software build. Refer to those files when developing your software code.

The desired LO frequency can be set using the **adi_adrv904x_LoFrequencySet(...)** command as listed in the table. Note that the PLL unlock bits in the GP interrupt mask are masked off temporarily when setting a new LO frequency. This setup should prevent the GP interrupt from 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. Optionally, PLL loop bandwidth and phase margin can be set manually. The structures and enumerators for these API commands are detailed in the doxygen documentation. The lock status of the various on-chip PLLs can be verified via the API.

Table 46. List of LO Configuration Related API Functions

API Method Name	Comments
adi_adrv904x_LoFrequencySet(...)	Needs 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 that you must not change the frequency for either the CLK PLL or SERDES PLL from their configured defaults.
adi_adrv904x_LoFrequencyGet(...)	Needs to pass the loName of one of the four PLLs you want to read back the LO frequency in Hz from. Note that if you are using a different transmitter LO/receiver LO leaf divider values across the slices, then this method returns the highest frequency. For example, if LO 0 = Rx LO is set to 1.8 GHz for Rx0/Rx1/Rx2/Rx3 and a further divide by 2 or 0.9 GHz for Rx4/Rx5/Rx6/Rx7, then this method returns a frequency of 1.8 GHz. In this case, you must use the **adi_adrv904x_RxTxLOFreqGet(...)** method.
adi_adrv904x_RxTxLOFreqGet(...)	Similar to **adi_adrv904x_LoFrequencyGet(...)** , but this method allows you to read back the LO frequency individually for each Rx or Tx slice.
adi_adrv904x_RxLoSourceGet()	Reads the LO source for the selected Rx channel.
adi_adrv904x_TxLoSourceGet()	Reads the LO source for the selected Tx channel.
adi_adrv904x_LoopFilterSet(...)	This function allows the user to set the PLL loop filter bandwidth & phase margin.
adi_adrv904x_LoopFilterGet(...)	This function allows the user to get the PLL loop filter bandwidth & phase margin.
adi_adrv904x_PllStatusGet(...)	This function allows the user to get the PLL lock status for each of the four on-chip PLLs.

MULTICHIP SYNCHRONIZATION (MCS)

MCS is a mechanism on-chip to allow deterministic timing to be established and enable data alignment down to the sample level across multiple serial lane links. Therefore, MCS aligns multiple ADRV904x in the system.

SYNTHESIZER CONFIGURATION

As shown in Figure 109, the MCS state machine takes in both SYSREF and DEVCLK as its inputs. As part of device initialization, the MCS state machine resynchronizes the SYSREF signal to the DEVCLK domain and uses the SYSREF signal to reset critical timing blocks in the device in a deterministic fashion.

The MCS sequence is performed in four stages as shown in Figure 109, each one initiated with a SYSREF rising edge.

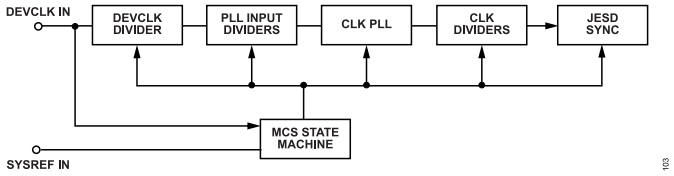


Figure 109. MCS State Machine

RF PLL PHASE SYNCHRONIZATION

The RF PLL phase synchronization description is included at this time for prototyping and evaluation purposes only. Consult Analog Devices 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 RF PLL to the same phase. Therefore, the phase between each device is aligned at startup in order that the phase between devices is repeatable and fixed. As part of device initialization, the MCS state machine resynchronizes the SYSREF signal to the DEVCLK domain and uses the SYSREF signal 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 in order that at every sample, the observed LO phase, is derotated 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 is measured, the adjustment can be applied into the first stage sigma delta modulator (SDM) of the PLL 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 110 is a block diagram of the phase synchronization system.

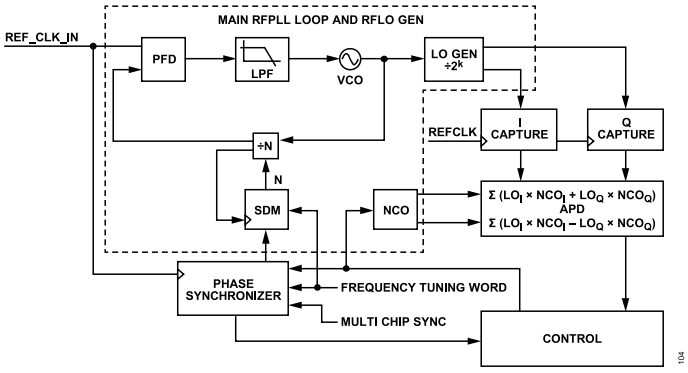


Figure 110. LO Phase Sync Functional Diagram

SYNTHESIZER CONFIGURATION

System Level Considerations

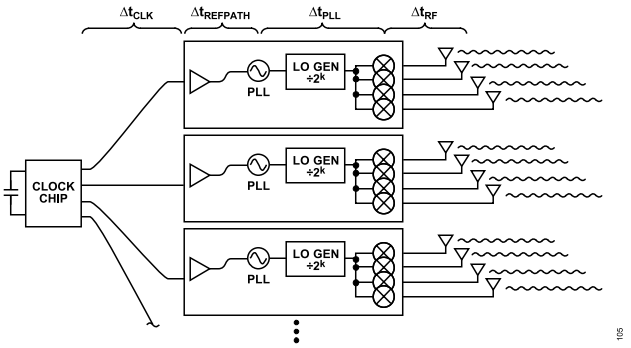


Figure 111. High Level Contributions to System Phase Per Antenna

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_{REPATH}), the PLL and LO divider path (t_{PLL}), and the RF and 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 setup are as follows:

- 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, for radio functionality. There is a separate A55 processor that has dedicated DFE features, like DPD, CLGC, and VSWR. This section outlines the Arm processor used for radio features. The firmware for these Arm processors is loaded during the initialization process. CPU0 is loaded first, followed by CPU1, then CPU0 is started, followed by CPU1. The firmware memory size is 641 kB. The Arm processors 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

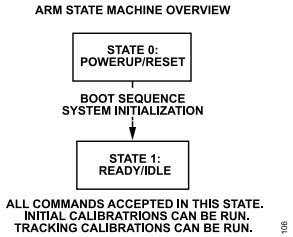


Figure 112. Arm State Machine

During initialization, the Arm processor undergoes two states as follows:

1. State 0: When the Arm core is powered up, the Arm processor 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 processor is enabled and begins its boot sequence.
2. State 1: After the Arm processor has been booted, it enters its ready/idle state. In this state, the processor 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. The following are three main sections of the initialization procedure:

- Pre MCS init
- MCS
- Post MCS init

Pre MCS init initializes the device up to the MCS procedure. The pre MCS init sequence is split into two commands that the application layer function calls. These are **adi_adrv904x_PreMcsInit()** and **adi_adrv904x_PreMcsInit_NonBroadcast()**. **adi_adrv904x_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_adrv904x_PreMcsInit_NonBroadcast()** command verifies that the Arm processor is programmed properly by verifying the Arm checksum and that the Arm processor is in the ready/idle state.

The MCS step uses SYSREF pulses to synchronize internal clocks within the transceiver and it is required for deterministic latency of JESD links so MCS is needed even if there is only one ADRV904x in the system. The function **adi_adrv904x_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_adrv904x_PostMcsInit()**. This command programs the PLLs, configures the radio control initialization structure, and instructs the Arm processors to perform initialization calibrations.

Pre MCS Initialization

The **adi_adrv904x_PreMcsInit()** function is in the **adi_adrv904x_utilities.c** file. It performs a sizeable part of the full chip initialization. The first step is to load the Arm image using **adi_adrv904x_CpuImageLoad()**. The Arm image, **ADRV9040_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 data memory using **adi_adrv904x_CpuProfileWrite()**. The Arm processor is then started and begins its boot sequence. This process is initiated by **adi_adrv904x_CpuStart()**.

ARM PROCESSOR AND DEVICE CALIBRATIONS

As part of the boot sequence, the Arm processor configures the device for the required profile (transmitter/receiver/observation receiver path configuration as determined by the use case), configures and enables the clock PLL, and configures the JESD framers and deframers. The Arm processor 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_adrv904x_CpuStartStatusCheck()** is called after the Arm processor 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 ADRV904x device clock. The function **adi_adrv904x_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 processor is ready to configure the radio, perform its initialization calibrations, and bring up the JESD link. This is achieved using the **adi_adrv904x_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 memory when an issue occurs. This can be achieved by using the **adi_adrv904x_CpuMemDump()** function. This function dumps the ADRV904x Arm program and data memory. The binary file that is generated needs to be sent to Analog Devices for analysis.

ARM API FUNCTIONS

Table 47. List of Arm API Functions

API Method Name	Comments
adi_adrv904x_CpuImageLoad()	Loads the ADRV904x CPU binary image.
adi_adrv904x_StreamImageLoad()	Loads the ADRV904x stream binary image.
adi_adrv904x_DeviceInfoExtract()	Extract the init info from the ADRV904x CPU profile binary image.
adi_adrv904x_CpuProfileImageLoad()	Loads the ADRV904x CPU profile binary image.
adi_adrv904x_PreMcsInit()	Executes ADRV904x pre MCS initialization sequence .
adi_adrv904x_PreMcsInit_NonBroadcast()	Executes the non broadcastable part of the pre MCS initialization sequence.
adi_adrv904x_MultichipSyncSet()	Prepares for incoming SYSREF pulses to synchronize the internal clock tree.
adi_adrv904x_MultichipSyncStatusGet()	Reads the multichip sync status.
adi_adrv904x_PostMcsInit()	Executes ADRV904x post MCS initialization sequence.
adi_adrv904x_CpuMemDump()	Captures the ADRV904x CPU program and data memory.

DEVICE CALIBRATIONS

The Arm processor 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 run time.

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_adrv904x_PostMcsInit( )**.

In some cases, it is required to run an initial calibration outside of **adi_adrv904x_PostMcsInit()**. This **adi_adrv904x_InitCalsRun()** command instructs the Arm processor to perform the requested calibrations.

ARM PROCESSOR AND DEVICE CALIBRATIONS

Table 48 shows the bit assignments of the calibration mask. Note that Table 48 provides a full list of initialization calibrations for the device. Some initial calibrations are not available for certain transceivers and applications.

The Arm processor sequences the initial calibrations as required, which are 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 48. adi_adrv904x_InitCalibrations Bit Assignments

Bit	Enum	Calibration	Description
D0	(Reserved)		
D1	ADI_ADRV904X_IC_RX_DC_OFFSET	Rx DC offset	Corrects for DC offset within the receiver chain.
D2	ADI_ADRV904X_IC_ADC_RX	ADC Rx	Calibrates the receiver ADC.
D3	ADI_ADRV904X_IC_ADC_ORX	ADC ORx	Calibrates the observation receiver ADC.
D4	ADI_ADRV904X_IC_ADC_TXLB	ADC Tx loopback	Calibrates the Tx loopback receiver ADC.
D5	ADI_ADRV904X_IC_TXDAC	Tx DAC	Calibrates the transmitter DAC.
D6	ADI_ADRV904X_IC_TXBBF	Tx BB filter	Tunes the corner frequency of the transmitter filter.
D7	ADI_ADRV904X_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_ADRV904X_IC_HRM	HRM	Performs HRM (harmonic reject mixer) harmonics rejection corrections.
D10	ADI_ADRV904X_IC_TXQEC	TxQEC	Performs an initial QEC calibration for the Tx path. It utilizes the Tx path and an internal loopback path.
D11	ADI_ADRV904X_IC_TXLOL	TxLOL	Performs an initial LO leakage calibration for the Tx path. It utilizes the Tx path and the internal loopback path.
D12	ADI_ADRV904X_IC_SERDES	SERDES	Performs an initialization calibration for the JESD204C data interface.
D13	(Reserved)		
D14	(Reserved)		
D15	(Reserved)		
D16	(Reserved)		
D17	ADI_ADRV904x_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 in order that 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 I/Q rates of the receiver, transmitter, and observation receiver paths. The Arm clock is derived from the clock PLL.

In Figure 113, 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 pseudorandom noise to calculate the required coefficients that are used to define the characteristics of the channel. Still others for example the transmitter 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 vary with use case and can also vary with the software version, for example, where improvements are made to reduce SERDES init cal times in later software. The init cal times are measured and stored by the internal Arm processor and can be queried using the **InitCalsWait_v2()** or **InitCalsDetailedStatusGet_v2()** APIs.

ARM PROCESSOR AND DEVICE CALIBRATIONS

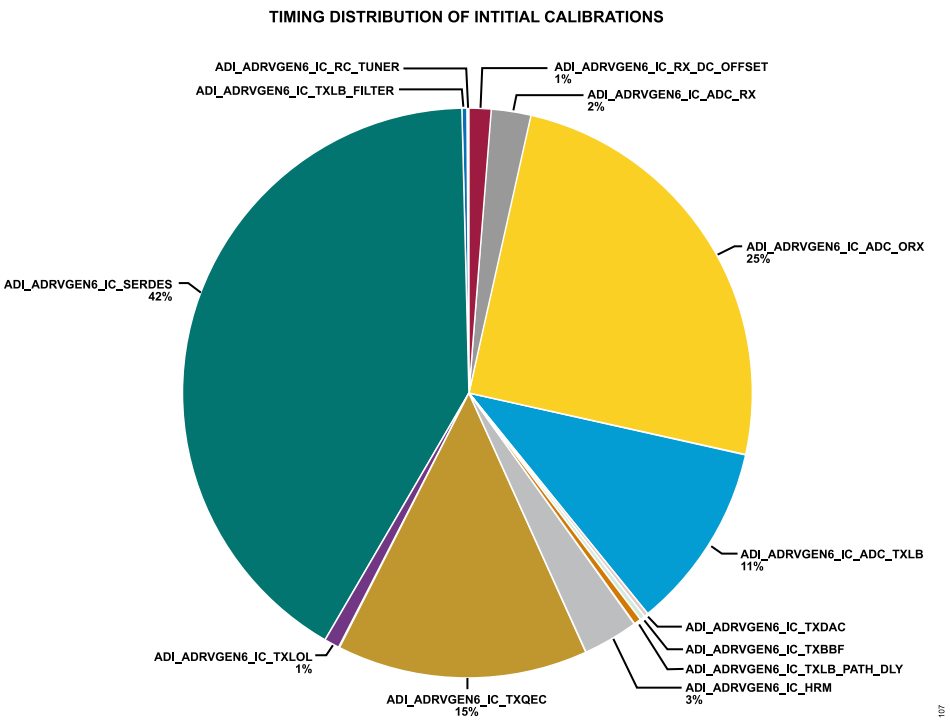


Figure 113. Initial Calibration Timings

Table 49. Inital Calibration Timings

Cal	Time (s)
ADI_ADRVGEN6_IC_RC_TUNER	0.01
ADI_ADRVGEN6_IC_RX_DC_OFFSET	0.26
ADI_ADRVGEN6_IC_ADC_RX	0.49
ADI_ADRVGEN6_IC_ADC_ORX	6.45
ADI_ADRVGEN6_IC_ADC_TXLB	2.55
ADI_ADRVGEN6_IC_TXDAC	0.01
ADI_ADRVGEN6_IC_TXBBF	0.03
ADI_ADRVGEN6_IC_TXLB_PATH_DLY	0.07
ADI_ADRVGEN6_IC_HRM	0.64
ADI_ADRVGEN6_IC_TXQEC	1.15
ADI_ADRVGEN6_IC_TXLOL	0.17
ADI_ADRVGEN6_IC_SERDES	5.46
ADI_ADRVGEN6_IC_TXLB_FILTER	0.06
Total	17.34

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_adrv904x_TrackingCalsEnableSet()** API function enables the tracking calibrations in the Arm processor. Table 50 shows the bit assignments of the enable mask.

Table 50. adi_adrv904x_TrackingCalibrationMask Bit Assignments

Bit	Enum
D0	ADI_ADRV904X_TC_RX_QEC
D1	ADI_ADRV904X_TC_TX_LOL

ARM PROCESSOR AND DEVICE CALIBRATIONS

Table 50. adi_adrv904x_TrackingCalibrationMask Bit Assignments (Continued)

Bit	Enum
D2	ADI_ADRV904X_TC_TX_QEC
D3	ADI_ADRV904X_TC_TX_SERDES
D4	ADI_ADRV904X_TC_RX_ADC
D5	ADI_ADRV904X_TC_TX_LB_ADC
D6	ADI_ADRV904X_TC_ORX_ADC

SYSTEM CONSIDERATIONS FOR CALIBRATIONS

Figure 114 shows 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 processor performs each of the initial calibrations, it is tasked with configuring the device as per Figure 115, with respect to enabling/disabling paths, for example. No user input is required in this regard.

TRANSMITTER LO LEAKAGE CALIBRATION

The ADRV904x uses a zero IF architecture which has energy transmitted at the LO frequency. To reduce this undesired emission, the transceiver has a transmitter LO leakage (LOL) correction algorithm. For optimal performance, the transmitter LOL calibration requires an initial calibration followed by a tracking calibration used during run time operation. The transmitter LOL algorithm applies complex valued correction to both I/Q datapaths at the transmitter QEC block in order to reduce the transmitter LOL level observed at the output of the transmitter.

TRANSMITTER LOL INITIAL CALIBRATION

The transmitter LOL initial calibration is used to make an initial correction to the LOL level. The initial calibration uses the internal loopback path only; therefore, it is not necessary to provide an external loopback connection. The signal path used during the initial calibration is in Figure 114 where unused components are grayed out.

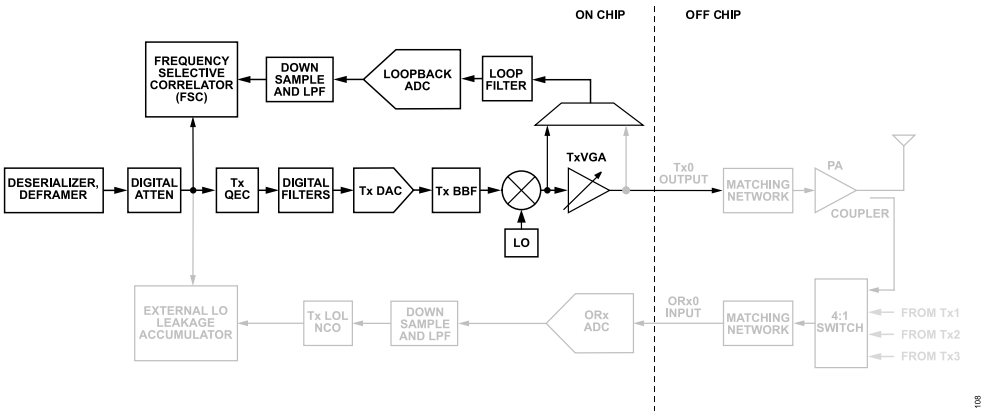


Figure 114. Transmitter LOLe Initial Calibration Correction Loop

Each transmitter channel has a corresponding internal loopback channel that is used for calibration purposes. The loopback channel has a correlator block (frequency selective corrector or FSC) that allows for correlation of the transmitted signal to the internally observed loopback signal. The correlation generates LOL correction coefficients. Because the transmitter LOL initial calibration uses the internal loopback path, the transmitter to observation receiver mapping state (that is, which transmitter channel is connected to a specific observation receiver channel) does not matter for the operation of this calibration.

During the transmitter LOL initial calibration, the user does not need to be transmitting traffic data. Instead, the device generate perturbation to estimate the initial correction for the channel. Additionally, the transmitter VGA is be powered down during this calibration to prevent large level signals entering the power amplifier stage. Note that you can send data or an idle pattern over JESD when the transmitter LOL init cal is running because the firmware temporarily disconnects the input data during the calibration.

ARM PROCESSOR AND DEVICE CALIBRATIONS

TRANSMITTER LOL TRACKING CALIBRATION

The transmitter LOL initial calibration is a prerequisite before enabling the transmitter LOL tracking calibration. The transmitter LOL tracking calibration uses an external loopback loop to track and correct changes over time. The external loop is shown in Figure 115.

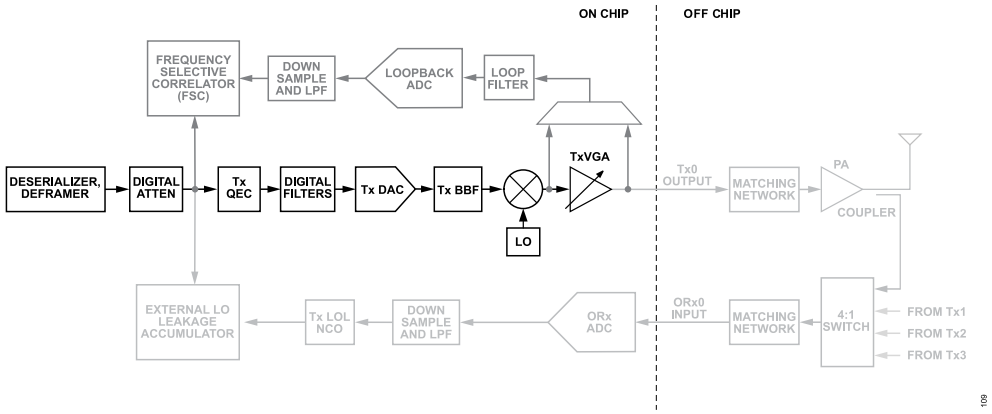


Figure 115. Transmitter LO Leakage and External Tracking Calibration Loop (Bottom Range Loop)—This Shows a Sample Configuration that Uses a 4:1 Mapping of Tx0/Tx1/Tx2/Tx3 into ORx0 Channel and Currently Tx0 is Available for External Path Observation

The necessity of 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 transmitter LOL include support for transmitter to observation receiver mapping in their software to utilize transmitter LOL tracking loop. When transmitter LOL tracking calibration is running, the user can transmit regular traffic data, and the algorithm can 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 transmitter attenuation.

Transmitter LOL Tracking Calibration (Internal Path)

The transmitter 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 transmitter QEC block.

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

Note that the transmitter LOL tracking calibration can operate while the observation receiver channel is enabled and sending data to the BBP.

Transmitter LOL Tracking Calibration (External Path)

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

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

Note that the external transmitter LOL tracking calibration can operate while the observation receiver channel is enabled and sending data to the BBP.

The external path system requirements for LOL tracking are as follows:

- Each transmitter must be observable at the observation receiver at least 100 ms every 6 seconds.
- The transmitter observation duration must be at least 1 symbol (17 µs). A 1 symbol every 1 ms configuration is supported.

ARM PROCESSOR AND DEVICE CALIBRATIONS

- The gain from transmitter input to observation receiver output must be at least −10 dB. That is the digital input level to the transmitter DAC to the digital output level from the observation receiver ADC. For example, if you are transmitting a signal with average power of −13 dbFS at the transmitter DAC output, the observation receiver signal at the observation receiver ADC output must be >−23 dbFS.
- A transmitter channel must only be observed on the same observation receiver channel. For example, this means that Tx0 must 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 that can lead to issues. An acceptable loopback scenario in this example is that Tx0 can only be observed at the ORx0 port.
- If the gain of the path between the transmitter output and the observation receiver 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 transmitter LOL init cal. Because the LOL corrections words are reset in this case, the LO leakage can temporarily degrade to uncorrected levels, but the convergence time must be quicker in this case as the calibration makes more aggressive updates. You do not need to use the **TxLolReset()** API for gradual gain 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 can cause instability algorithm performance.

Transmitter LOL Tracking Status

The ADRV9040 transmitter LOL tracking calibration continuously computes the new QEC filter DC offset and updates the actuator. Once the actuator hardware is updated, the update counter increases. Once LOL algorithm gets a converged result, the iteration counter increases. It is normal that iLOL algorithmtakes more than one update count to achieve a converged result. As a result, LOL algorithm is expected to observe a higher update count number than the iteration count number. The API function **adi_adrv904x_TrackingCalStatusGet()** is useful to monitor the tracking status, including the iteration count and update count numbers.

TX QEC CALIBRATION

The ADRV904x uses a zero IF transmitter architecture. Quadrature errors within the transmit path show up as unwanted energy on the image frequency. To reduce this undesired emission, the transceiver has a transmitter QEC algorithm. For optimal performance, the transmitter QEC calibration requires an initial calibration followed by a tracking calibration used during runtime operation. The transmitter QEC algorithm applies complex valued correction to I/Q datapaths at the transmitter QEC block in order to reduce the unwanted emissions on the image frequency observed at the output of the transmitter.

TRANSMITTER QEC INITIAL CALIBRATION

The transmitter 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 transmitter signal before transmitter VGA. The signal path used during the initial calibration is shown in Figure 116 where unused components are grayed out.

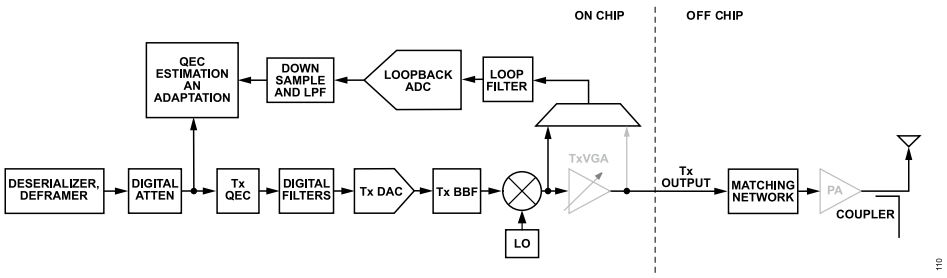


Figure 116. Transmitter QEC Initial Calibration Loop

Each transmitter channel has a corresponding internal loopback channel that is used for calibration purposes. The loopback channel has a QEC estimation and adaptation engine that allows for the correlation of the transmitted signal to the internally observed loopback signal and generates QEC coefficients. Because the transmitter QEC initial calibration uses the internal loopback path, the transmitter to observation receiver mapping state (that is, the transmitter channel connected to a specific observation receiver channel) does not matter for the operation of this calibration.

During the transmitter QEC initial calibration, the user does not transmit traffic data. Instead, the device transmits tones at various offsets from the LO in order to estimate the initial correction for the channel. Additionally, the transmitter VGA is powered down during this calibration to

ARM PROCESSOR AND DEVICE CALIBRATIONS

prevent large level signals entering the power amplifier stage. Powering down the transmitter VGA offers additional isolation for conditions where turning power amplifier off might cause an impedance mismatch.

TRANSMITTER QEC TRACKING CALIBRATION

The transmitter QEC initial calibration is required before enabling the transmitter QEC tracking calibration. There are two observation tap off points in the transmitter path, one before the VGA and one after, as shown in Figure 117. Both tap off points are connected to internal loopback path. This allows the tracking calibration to observe and track impairments that couple to the transmitter output, including but not limited to the transmitter 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. The transmitter QEC tracking calibration does not require an external loopback path, which utilizes the observation receiver path.

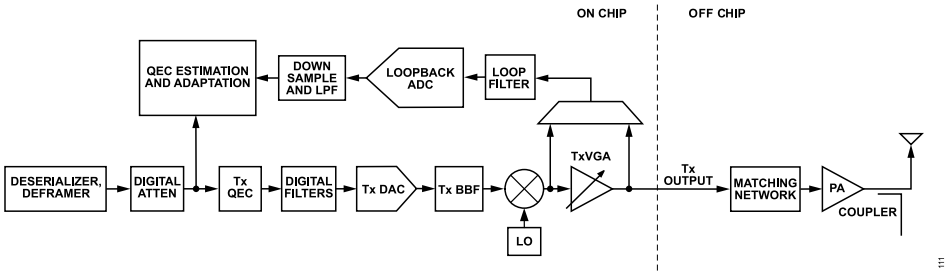


Figure 117. Transmitter QEC Tracking Calibration Loop with Two Tap Off Points

As the device operating conditions change, the updates to the QEC is achieved through tracking calibration. The transmitter 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 the QEC vs. the transmitter attenuation.

Transmitter QEC Tracking Calibration Frequency Planning

For the transmitter QEC tracking calibration to operate as intended, the user has to consider frequency planning.

Let f_s be the DAC sampling rate. The transmitter LO frequency f_{LO} must satisfy the formula

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

where:
i = all integers.
m = 2, 4, 6, 8, 10, 12. Table 51 gives a calculated example for the LO = 2.6 GHz.

Table 51. Transmitter QEC Tracking Calibrations Frequency Planning

m	Value of *i* that Minimizes $\left\| f_{LO} - i\frac{f_s}{m} \right\|$ $i = round\left(2m\frac{f_{LO}}{fs}\right)$	Nearest Blackout Frequency (in MHz) $\left(i\frac{f_s}{2m}\right)$	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 transmitter LO can be placed at exactly 2.6 GHz. If f_{LO} is within 1 MHz of the blackout frequencies, then the LO must be moved by at most ±1 MHz to avoid the nearest blackout frequency. The ADRV904x configurator/GUI has this LO frequency checking built-in and outputs an error if the LO frequency does not meet the requirements.

ARM PROCESSOR AND DEVICE CALIBRATIONS

RECEIVER QEC CALIBRATION

The ADRV904x 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 ADRV904x implements a QEC algorithm in firmware (FW), which is designed to perform blind, digital only, frequency dependent, and independent QEC.

The location of receiver QEC in the receiver path is shown in Figure 118. The goal of the algorithm is to estimate and track the quadrature error and apply the QEC using a filter. The algorithm is designed to keep tracking the receiver 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.

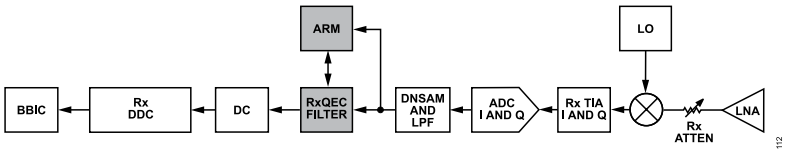


Figure 118. Receiver QEC in Receiver Path

QEC AND LOL CALIBRATION API FUNCTIONS

Table 52. List of QEC and LOL Calibration API Functions

API Method Name	Comments
adi_adrv904x_InitCalsRun()	Runs the QEC and LOL initial calibrations among other initial calibrations.
adi_adrv904x_InitCalsDetailedStatusGet()	Provides updates to represent the status of the QEC and LOL initial calibrations among other initial calibrations.
adi_adrv904x_InitCalsDetailedStatusGet_v2()	Provides updates to represent the status of the QEC and LOL initial calibrations among other initial calibrations.
adi_adrv904x_InitCalsWait()	Called after **adi_adrv904x_InitCalsRun()** to wait for initial calibrations to finish.
adi_adrv904x_InitCalsWait_v2()	Called after **adi_adrv904x_InitCalsRun()** to wait for initial calibrations to finish.
adi_adrv904x_InitCalsCheckCompleteGet()	Similar to **adi_adrv904x_InitCalsWait()**, except it does not block the thread waiting for the enabled initial calibrations to complete. This function returns the initial calibration status immediately, allowing the application layer to do other work while the calibrations are running.
adi_adrv904x_InitCalsCheckCompleteGet_v2()	Similar to **adi_adrv904x_InitCalsWait()**, except it does not block the thread waiting for the enabled initial calibrations to complete. This function returns the initial calibration status immediately, allowing the application layer to do other work while the calibrations are running.
adi_adrv904x_InitCalsAbort()	The initial calibrations can take several seconds. This function is called when the BBIC needs to intercede and stop the current initial calibration sequence.
adi_adrv904x_InitCalStatusGet()	Reads back the status of the calibration including metrics like error codes, the percentage of data collected for the current calibration, the performance of the calibration, and the number of times the calibration has run and updated the hardware.
adi_adrv904x_TrackingCalsEnableSet()	Enables or disables QEC or LOL tracking calibrations among other tracking calibrations.
adi_adrv904x_TrackingCalsEnableSet_v2()	Enables or disables QEC or LOL tracking calibrations among other tracking calibrations.
adi_adrv904x_TrackingCalsEnableGet()	Reads the set of tracking calibrations that are enabled.
adi_adrv904x_TrackingCalAllStateGet()	Reads the current state of all tracking calibrations.
adi_adrv904x_TrackingCalStatusGet()	Returns the status of the tracking calibration.
adi_adrv904x_TxLolReset()	Resets the Tx channel the LOL tracking calibration.
adi_adrv904x_CalPvtStatusGet()	Returns detailed status information specific to the private calibration.
adi_adrv904x_CalSpecificStatusGet()	Returns status specific information calibration.

TRANSMITTER ANALOG LPF CALIBRATION

The baseband LPF is designed as a second order Butterworth filter between the DAC output and the mixer input of the I/Q path of the transmitter channels as shown in Figure 119. Its bandwidth (3 dB corner) is adjustable for different user cases. The transmitter analog LPF calibration is implemented to do the following:

- Set the 3 dB corner of this filter precisely at the band edge associated with the user case.

ARM PROCESSOR AND DEVICE CALIBRATIONS

- Achieve the frequency response as a Butterworth filter with best in-band flatness.

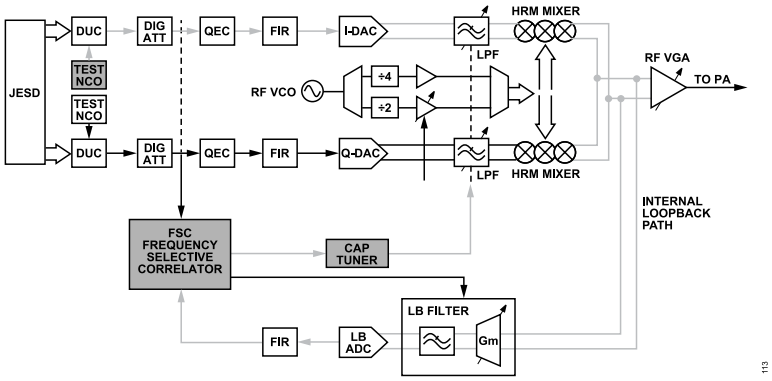


Figure 119. Transmitter Analog LPF Initial Calibration Loop

To calibrate the LPF bandwidth, the calibration tones are generated separately and sequentially from the test NCO that feeds into the transmitter signal path through the 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 FSCs. The capacitors at the analog LPF are tuned simultaneously based on the ratio of the power measurements done by the FSCs to achieve the accurate 3 dB corner frequency and in-band flatness.

LOOPBACK PATH DELAY INITIAL CALIBRATION

The loopback path delay calibration is used to measure the delay between the transmitter QEC actuator and the loopback receiver input to the FSC. The result is used to configure the FSC correlator for accurate estimation of the transmitter QEC and the transmitter LOL corrections. Therefore, the loopback path delay calibration needs to run before the transmitter QEC initial calibration and the transmitter LOL initial calibration. Note that if multiple calibrations are enabled in the calibration mask, the Arm firmware correctly sequences the calibrations to ensure the transmitter loopback calibration is run before transmitter LOL initial calibration or transmitter QEC init calibration.

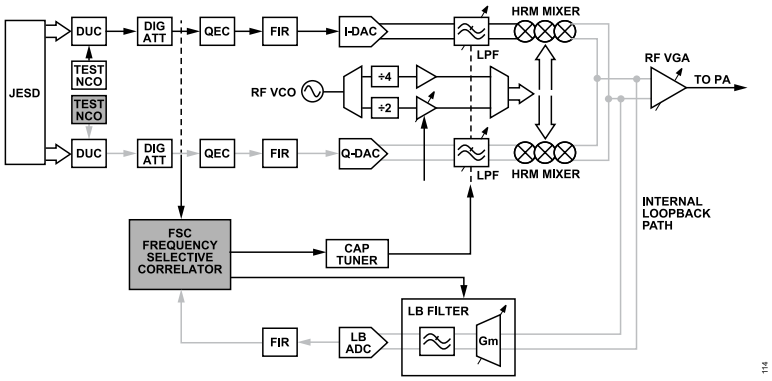


Figure 120. Loopback Path Delay Initial Calibration Loop

As shown in the block diagram in Figure 120, the calibration tones are generated by the transmitter test NCO, and feed through the whole transmitter channel and looped back internally at the output of the transmitter mixer to the loopback path. The FSC hardware measures the cross correlation between the transmitter data and the loopback receiver ADC data to estimate the path delay.

The loopback path delay initial calibration requires the following:

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

ARM PROCESSOR AND DEVICE CALIBRATIONS

RECEIVER DC OFFSET CALIBRATION

Overview

The receiver DC calibration comprises of two different corrections, the receiver radio frequency DC (RFDC) calibration, which attempts to correct the DC offset introduced by the analog front end, and the receiver baseband DC (BBDC) correction, which attempts to null any remaining DC power by using a narrow band filter to heavily reject DC in the digital datapath. All receiver DC offset calibrations are hardware-based and have very little interaction with the embedded Arm processors.

The sense point for the receiver RFDC calibration is after the programmable finite impulse response (PFIR) filter. The RFDC calibration attempts to correct all DCs 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 transimpedance amplifier (TIA) filter. The receiver BBDC offset block is directly after the sense point for the receiver RFDC (after the PFIR) and the BBDC attempts 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.

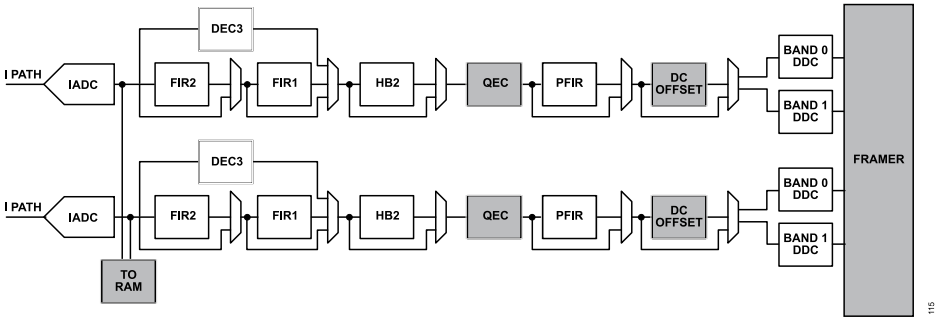


Figure 121. Receiver Channel Datapath

Receiver RFDC Offset Initialization Calibration

The receiver 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, and accumulators calculate and store the receiver DC correction in a table for each value of front-end gain. This initial calibration is the first step toward 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.

The maximum time for initial calibration using an I/Q rate of 245.76 MHz is approximately 1.5 ms per channel.

The maximum time for initial calibration using an I/Q rate of 307.2 MHz is approximately 1.2 ms per channel.

Receiver RFDC Offset Tracking Calibration

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

Receiver BBDC Tracking Calibration

The receiver BBDC tracking calibration uses normal receiver data to collect information regarding the receiver DC offset and apply a correction. The receiver 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 c is subtracted out and reduces the DC offset even further to meet the data sheet specifications. This implementation results in a very steep notch filter around DC, which removes heavily any DC . This calibration can be enabled or disabled during run time, 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 bandwidth. Table 53 shows the filter bandwidth and the convergence time in number of samples and for a sample rate of 307.2 MHz.

ARM PROCESSOR AND DEVICE CALIBRATIONS

Table 53. M Shift Configuration vs. Bandwidth for 307.2 MHz I/Q Rate

M Shift	3 dB Filter Bandwidth (Hz)	Number of Samples for Convergence	Convergence Time for Rx I/Q Rate of 307.2 MHz (ms)
9	24000	1625	0.01
10	12000	3250	0.01
11	6000	6500	0.02
12	3000	13000	0.04
13	1500	25000	0.08
14	750	50000	0.17
15	375	100000	0.33
16	188	200000	0.65
17	90	400000	1.3
18	50	600000	1.96
19	23	800000	2.61
20	12	1000000	3.26

The DC offset initialization/tracking calibrations are responsible for keeping the DC offset in the receiver datapath in line with data sheet specifications. Initialization calibrations can be run during the initialization of the device and can also be run during run time. This also applies to tracking calibrations such as the receiver DC tracking calibration. During run time, you may need to adjust the receiver BBDC tracking calibrations, and the function calls associated with these are described in Table 54.

RECEIVER DC OFFSET CONFIGURATION API FUNCTIONS

Table 54. List of Receiver DC Offset Configuration Related API Functions

API Method Name	Comments
adi_adrv904x_DigDcOffsetEnableSet()	Sets the enable/disable of the BBDC offset tracking calibration.
adi_adrv904x_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_adrv904x_RxChannels_e** mapping.
adi_adrv904x_DigDcOffsetCfgSet()	Sets the digital DC offset configuration for the selected channel/s. The user can select the 1 dB corner for the DC offset filter through **adi_adrv9040x_dcOffsetCfg_t** structure. FW calculates the required M shift and multiplier values based on this corner frequency and configures the transceiver device.
adi_adrv904x_DigDcOffsetCfgGet()	This function reads back the BBDC offset configuration for the selected channel.

ANTENNA CALIBRATION

The ADRV904x supports a transmitter antenna calibration (AC) mode and a receiver antenna calibration mode. An antenna calibration mode allows the following:

- Dedicated transmitter channel (for UL calibration) and receiver channel (for DL calibration) are activated with special programming calibrations disabled.
- User-programmable GPIOs are configured as outputs and toggled to configure appropriate AC mapping (exclusive to digital GPIO controlled AC).

AC Control Methods

The AC can be triggered by using the TRX[A-H]_CTRL pins or user selected digital GPIO pins. The pin mode to use is selected before run time in the **streamSettings** structure of the profile JSON file. Setting **EnableAntCal** to **true** enables the TRX[A-H]_CTRL pins to control the AC. Setting **EnableGpioAntCal** to **true** enables the GPIO pins to control the AC. **EnableAntCal** and **EnableGpioAntCal** cannot both be set to true.

ARM PROCESSOR AND DEVICE CALIBRATIONS

Configuring TRXA_CTRL Pin to TRXH_CTRL Pin for Transmitter and Receiver AC

First, set up the AC to use the TRXA_CTRL to TRXH_CTRL pins. In the 88th and 89th lines of the profile JSON file set:

```
"EnableGpioAntCal": false,
"EnableAntCal": true,
```

Second, assign the TRXA_CTRL to TRXH_CTRL pins to the AC. This can be done using the API function **adi_adrv904x_RadioCtrlTxRxEnCfgSet()** or it can be set up in the profile JSON file, which is described as follows:

This is how to assign the receiver channel/channels for the transmitter AC in the profile JSON file. In the profile JSON file, there is an array called **rxAltMapping**, and the next eight lines are its elements. Element 0 corresponds to the TRXA_CTRL pin, and this continues to Element 7, which corresponds to the TRXH_CTRL pin. The decimal value placed in the array rxAltMapping is the receiver channel/channels that is/are enabled in the transmitter AC. Because there are up to eight receiver channels that can be assigned to any of the pins in the TRXA_CTRL to TRXH_CTRL pin range, there is a bitwise mapping where Bit[0] corresponds to Rx0, and Bit[7] corresponds to Rx7. An example is shown in the following code block to set up Rx4 for the transmitter AC when the TRXD_CTRL pin is triggered. Rx4 is Bit[4]; therefore, it is 00010000b in binary and 16 in decimal. Therefore, a value of 16 is placed in Element 3.

```
"rxAltMapping": [
0,
0,
0,
16,
0,
0,
0,
0
],
```

This is how to assign the transmitter channel/channels for the receiver AC. In the profile JSON file, there is an array called **txAltMapping**, and the next eight lines are its elements. Element 0 corresponds to the TRXA_CTRL pin, and this continues to Element 7, which corresponds to the TRXH_CTRL pin. The decimal value placed in the array txAltMapping is the transmitter channel/channels that is/are enabled in the receiver AC. Because there are up to eight transmitter channels that can be assigned to any of the pins in the TRXA-_CTRL to TRXH_CTRL pin range, there is a bitwise mapping where Bit[0] corresponds to Tx0, and Bit[7] corresponds to Tx7. An example is shown in the following code block to set up Tx3 for the receiver AC when the TRXB_CTRL pin is triggered. Tx3 is Bit[3]; therefore, it is 00001000b in binary and 8 in decimal. Therefore, a value of 8 is placed in Element 1.

```
"txAltMapping": [
0,
8,
0,
0,
0,
0,
0,
0
],
```

Configuring GPIO Pins for Transmitter and Receiver AC

First, set up the AC to use the GPIO pins. In the profile JSON file, set:

```
"EnableGpioAntCal": true,
"EnableAntCal": false,
```

ARM PROCESSOR AND DEVICE CALIBRATIONS

Second, assign the GPIO pins to the receiver and transmitter ACs using the profile JSON file. The profile JSON file allows the definition of the GPIO_0 pin to GPIO_23 pin. Decimal 8 sets the GPIO to receiver AC and Decimal 9 sets the GPIO pin to the transmitter AC. An example where the GPIO_0 pin and GPIO_1 pin are used for the receiver and transmitter ACs, respectively, is shown in the following code block:

```
"Gpio00Selection": 8,
"Gpio01Selection": 9,
```

Finally, determine which transmitter/s turn on during the receiver AC, or which receiver/s turn on during the transmitter AC. Use the API function **adi_adrv904x_RadioCtrlAntCalGpioChannelSet()**.

Note that control of the GPIO pins for transmitter to receiver mapping is only available for the digital GPIO pin controlled AC.

Transmitter AC Mode

In the simplest terms, the transmitter AC mode works by taking a receiver channel/s, applying a fixed front-end RF gain, disabling the receiver AGC, disabling internal calibrations, and, if required, adjusting the receiver NCO. Once the transmitter AC is complete, all the changed receiver settings are returned to their values before the calibration started.

The transmitter AC can be set up to operate in two different ways as follows:

- Single pin control—a single pin both modifies and enables/disables the receiver channel/s that is monitoring the transmitters.
- Dual pin control—one pin, the pin assigned for the AC, modifies the receiver channel/s that is monitoring the transmitters. Another pin enables/disables the receiver channel that is monitoring the transmitters. This pin is assigned in the same way pins are assigned receiver enable/disable for TDD pin control. See the System Control section for further details.

Before enabling the transmitter AC, set up the following predefined parameters:

- 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 receiver NCO enable, which is determined by the **EnableNco** parameter in the **streamSettings** structure of the profile JSON file.
- The frequency the receiver NCO must change to, determined by the **adi_adrv904x_RadioCtrlAntCalConfigSet** or **adi_adrv904x_RadioCtrlAntCalConfigSet_v2** API function.
- The receiver gain, which is determined by the **adi_adrv904x_RadioCtrlAntCalConfigSet** or **adi_adrv904x_RadioCtrlAntCalConfigSet_v2** API function.

An example of single pin and dual pin control transmitter AC are as follows:

- Single pin control—the pin assigned to the transmitter AC goes high and this starts streams that make the required changes. Streams apply the fixed gain to the receiver, disable the AGC and calibrations, adjust the receiver NCO (if **EnableNco** is set to **true**) and enable the receiver channel. Once the BBIC has completed the transmitter AC, the same pin goes low to disable the receiver and restore all the receiver parameters back to their previous state.
- Dual pin control—the pin assigned to the transmitter AC goes high and this starts streams that make the required changes. Streams apply the fixed gain to the receiver, disable the AGC and calibrations, adjust the receiver NCO (if **EnableNco** is set to **true**). A second pin is responsible for enabling the receiver channel. Once the BBIC has completed the transmitter AC, the second pin triggers low to disable the receiver, and the first pin also triggers low and restores all the receiver parameters back to their previous state.

All gain/NCO settings restored at the falling edge of RX_ANT_CAL, see Figure 122 for more details. If the AGC was enabled prior to AC mode entry, it is disabled on the rising edge of AC mode and enabled again upon exit.

Receiver AC Mode

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

The receiver AC can be set up to operate in two different ways as follows:

- Single pin control—a single pin both modifies and enables/disables the transmitter channel that is monitoring the receivers.

ARM PROCESSOR AND DEVICE CALIBRATIONS

- Dual pin control—one pin, the pin assigned for the AC, modifies the transmitter channel that is monitoring the receivers. Another pin enables/disables the transmitter channel that is monitoring the receivers. This pin is assigned in the same way pins are assigned transmitter enable/disable for TDD pin control. See the System Control section for further details.

Before enabling the receiver AC, the following predefined parameters must be set up:

- 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 transmitter NCO enable that is determined by the **EnableNco** parameter in the **streamSettings** structure of the profile JSON file.
- The frequency of the the transmitter NCO must be changed to the one determined by the **adi_adrv904x_RadioCtrlAntCalConfigSet** or **adi_adrv904x_RadioCtrlAntCalConfigSet_v2** API function.
- The transmitter attenuation that is determined by the **adi_adrv904x_RadioCtrlAntCalConfigSet** or **adi_adrv904x_RadioCtrlAntCalConfigSet_v2** API function.
- The gain (rxGain) to set the receiver MGC to if the AGC freeze option is used (**enableFreeze** = 1). These are set in the **adi_adrv904x_RadioCtrlAntCalConfigSet_v2** API function. Note that **enableFreeze** only has an impact if the AGC is used. If using the MGC mode, then **enableFreeze** must be set to 0 to avoid any confusion.

The following is an example of a single pin and dual pin control transmitter AC setup:

1. Single pin control—the pin assigned to the receiver AC goes high, and this starts streams that make the required changes. Streams apply the fixed transmitter attenuation, disable the calibrations, adjust the transmitter NCO (if **EnableNco** is set to **true**) and cause the transmitter channel to enable. The receiver gain control, if **enableFreeze** = 1, changes from the AGC to MGC using the value of rxGain. Once the BBIC has completed the receiver AC, the same pin triggers low to disable the transmitter and restores all the transmitter parameters back to their previous state, and if **enableFreeze** = 1 when the receiver AC runs, the AGC is restored on the receivers.
2. Dual pin control—the pin assigned to the receiver AC goes high and this starts streams that make the required changes. Streams apply the fixed transmitter attenuation, disable the calibrations, adjust the transmitter NCO (if **EnableNco** is set to **true**). The receiver gain control, if **enableFreeze** = 1, changes from the AGC to MGC using the value of rxGain. A second pin is responsible for enabling the transmitter channel. Once the BBIC has completed the receiver AC, the second pin triggers low to disable the transmitter, and the first pin also triggers low and restores all the transmitter parameters back to their previous state and restores the receiver back to the AGC if it was selected before the receiver AC ran.

AC Path Mapping Using GPIOs

The ADRV904x AC feature allows flexibility to configure dedicated GPIOs as outputs and toggle these GPIOs during normal operation, downlink, and uplink calibration. AC mapping supports use of eight GPIOs (analog or digital), which can be configured for a given truth table.

Table 55. Example AC Mapping Using GPIOs

Device State	Bit 0—GPIO0	Bit 1—GPIO1	Bit 2—GPIO2	Bit 3—GPIO3	Bit 4—GPIO4	Bit 5—GPIO5	Bit 6—GPIO6	Bit 7—GPIO7
Normal Operation	1	0	0	1	0	0	0	0
Uplink Calibration	0	1	1	0	0	0	1	0
Downlink Calibration	0	0	0	0	1	1	0	1

The procedure to configure GPIOs for AC mapping is as follows:

1. Set the **GpioXXSelection** field in profile JSON file to indicate the GPIOs with their corresponding bit representation in AC mapping truth table. An example mapping is shown in the following code block. In this example, a setting of **80** indicates that GPIO0 represents the LSB bit in the AC mapping truth table. Meanwhile, a setting of **87** indicates GPIO7 is the MSB bit.

```
"Gpio00Selection": 80,
"Gpio01Selection": 81,
"Gpio02Selection": 82,
"Gpio03Selection": 83,
"Gpio04Selection": 84,
"Gpio05Selection": 85,
```

ARM PROCESSOR AND DEVICE CALIBRATIONS

```
"Gpio06Selection": 86,
"Gpio07Selection": 87,
```

2. In the profile JSON file, set **AntCalGpioOutputNormal**, **AntCalGpioOutputUlCal**, and **AntCalGpioOutputDlCal** according to the truth table. For the example given in Table 55, the settings in the following code block are required.

```
" AntCalGpioOutputNormal ": 9,
" AntCalGpioOutputUlCal ": 70,
" AntCalGpioOutputDlCal ": 176
```

After the configuration of the JSON, the ADRV904x toggles the dedicated GPIOs according to the truth table and the AC mode required.

AC Example

Consider the following TRXA_CTRL to TRXH_CTRL pin range example where:

- The TRXA_CTRL pin enables all transmitter channels.
- The TRXB_CTRL pin enables Tx3 for AC.
- The TRXC_CTRL pin enables all receiver channels.
- The TRXD_CTRL pin enables Rx4 for AC.
- **DisableTxRx** is set to **false**.

Figure 122 shows example the timing of the TRXx_CTRL pins for TDD radio control and AC modes.

- The term TX3_ANT_CAL means Tx3 enabled for the purposes of the receiver AC.
- The term RX4_ANT_CAL means Rx4 enabled for the purposes of the transmitter AC.

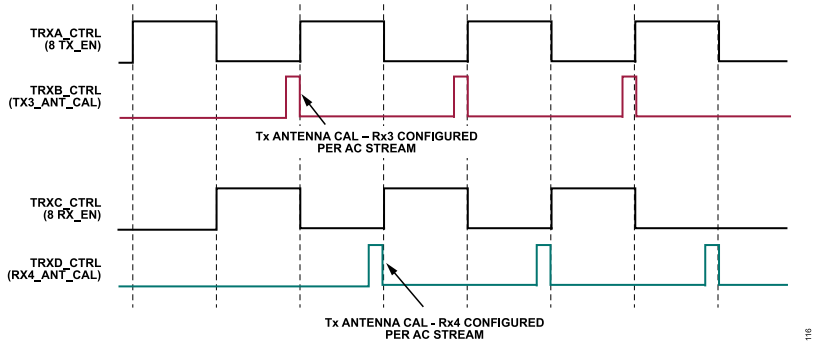


Figure 122. TDD Radio Control and AC

AC API FUNCTIONS

Table 56. List of AC Related API Functions

API Method Name	Comments
adi_adrv904x_RadioCtrlTxRxEnCfgSet()	Selects the Tx/Rx channels to be turned on during the TRXA_CTRL to TRXH_CTRL pin range AC.
adi_adrv904x_RadioCtrlTxRxEnCfgGet()	Reads the Tx/Rx channel selection for the TRXA_CTRL to TRXH_CTRL pin range AC.
adi_adrv904x_RadioCtrlAntCalGpioChannelSet()	Selects the Tx/Rx channels to be turned on during the GPIO AC. The GPIO pin can be selected via **GpioXXSelection** fields in the **streamSettings** structure in the profile JSON file.
adi_adrv904x_RadioCtrlAntCalGpioChannelGet()	Reads the Tx/Rx channel selection for the GPIO AC.
adi_adrv904x_RadioCtrlAntCalConfigSet()	Sets the gain and NCO for the Tx/Rx channels in the AC.
adi_adrv904x_RadioCtrlAntCalConfigGet()	Reads the gain and NCO for the Tx/Rx channels in the AC.
adi_adrv904x_RadioCtrlAntCalConfigSet_v2()	Sets the gain, NCO, and AGC for the Tx/Rx channels in the AC.
adi_adrv904x_RadioCtrlAntCalErrorClear()	Clears error bits for selected channels.
adi_adrv904x_RadioCtrlAntCalErrorGet()	Reads back the AC error status.

ARM PROCESSOR AND DEVICE CALIBRATIONS

CALIBRATION GUIDELINES AFTER RF LO FREQUENCY CHANGES

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

- Type 1—this RF LO frequency change is described by all of the following three 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 Table 57.
  - The LO frequency change does not step over the transmitter upconverter bias settings switching boundary (at 1600 MHz and 3600 MHz).
- Type 2—this RF LO frequency change 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 transmitter upconverter bias settings switching boundary (at 1200 MHz, 1600 MHz, and 3600 MHz).

Table 57. VCO Transition Boundary and LO Divider Boundary

Frequency (MHz)		VCO Boundary (MHz)				Leaf Divider	Root Divider
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
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 is as follows:

1. Disable all tracking calibrations
2. Disable all RF channels. If the TRX_CTRL/ORX_CTRL pins cannot stop toggling, put the device into API command control mode via **adi_adrv904x_RadioCtrlCfgSet()**, then **call adi_adrv904x_RxTxEnableSet()** to disable all channels.
3. Call **adi_adrv904x_LoFrequencySet()** to set the LO frequency. Note that the observation receiver NCO frequency needs to be adjusted accordingly to align with the new transmitter LO frequency. The configurator provides observation receiver NCO frequencies that correspond to specific transmitter LO frequencies. Call **adi_adrv904x_OrxNcoSet or adi_adrv904x_OrxNcoSet_v2** to set the observation receiver NCO frequency.
4. Call **adi_adrv904x_TxLolReset** with **resetType** set as ADI_ADRV904X_TX_LOL_SOFT_RESET to reset the channel estimation for the transmitter LOL calibration.
5. Call **adi_adrv904x_TxQecReset** with **resetType** set as ADI_ADRV904X_TX_QEC_TRACKING_CHANNEL_RESET to reset the channel estimation for the transmitter QEC calibration.
6. Enable relevant tracking calibrations.
7. Transition back to pin control mode, if necessary.

The LO frequency change procedure for Type 2 is as follows:

1. Disable all tracking calibrations
2. Disable all RF channels. If the TRX_CTRL/ORX_CTRL pins cannot stop toggling, put the device into API command control mode via **adi_adrv904x_RadioCtrlCfgSet()**, then call **adi_adrv904x_RxTxEnableSet()** to disable all channels.
3. Call **adi_adrv904x_LoFrequencySet()** to set the LO frequency. Note that the observation receiver NCO frequency needs to be adjusted accordingly to align with the new transmitter LO frequency. The configurator provides observation receiver NCO frequencies that correspond to specific transmitter LO frequencies. Call **adi_adrv904x_OrxNcoSet or adi_adrv904x_OrxNcoSet_v2** to set the observation receiver NCO frequency.
4. Rerun the following initial calibrations separately or with calMask set as 0x4E80:
   a. HRM initial calibration (ADI_ADRV904x_IC_HRM)

ARM PROCESSOR AND DEVICE CALIBRATIONS

   b. Loopback filter calibration (ADI_ADRV904x_IC_TXLB FILTER)
   c. Loopback path delay calibration (ADI_ADRV904x_IC_TXLB_PATH_DLY)
   d. Transmitter QEC initial calibration (ADI_ADRV904x_IC_TXQEC)
   e. Transmitter LOL initial calibration (ADI_ADRV904x_IC_TXLOL)
5. Enable relevant tracking calibrations.
6. Transition back to pin control mode, if necessary.

PA PROTECTION

The ADRV904x has two 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/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, nonoverlapping 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 nonsticky, it automatically clears 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 ADRV904x 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 Figure 123, Figure 124, Figure 125, and Figure 126.

Figure 123 shows the digital samples of a 50 MHz continuous wave (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 exceeds it.

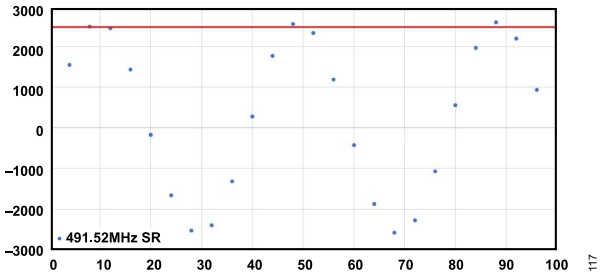


Figure 123. 50 MHz CW Sampled at 491.52 MSPS

Figure 124 shows the same 50 MHz CW interpolated by two; therefore, resampled at 983.04 MSPS. Two samples per positive cycle of the CW now exceed the threshold.

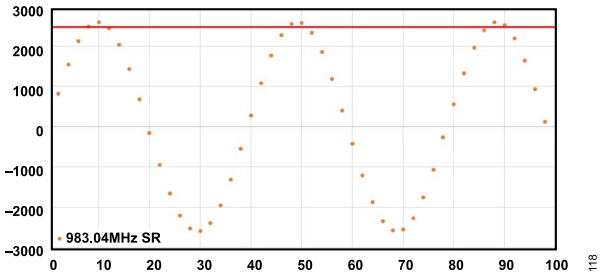


Figure 124. 50 MHz CW Sampled at 983.04 MSPS

Figure 125 shows a further interpolation by two; therefore, the 50 MHz CW is now resampled to 1966.08 MSPS. Four samples per positive cycle of the CW now exceed the threshold

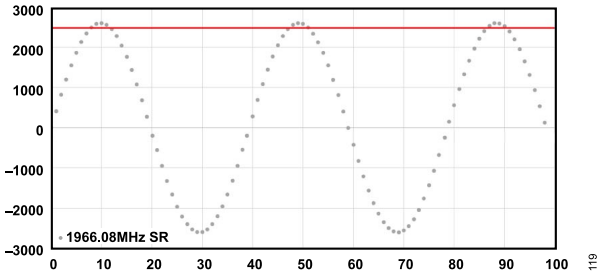


Figure 125. 50 MHz CW Sampled at 1966.08 MSPS

PA PROTECTION

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 126 describes the configuration options.

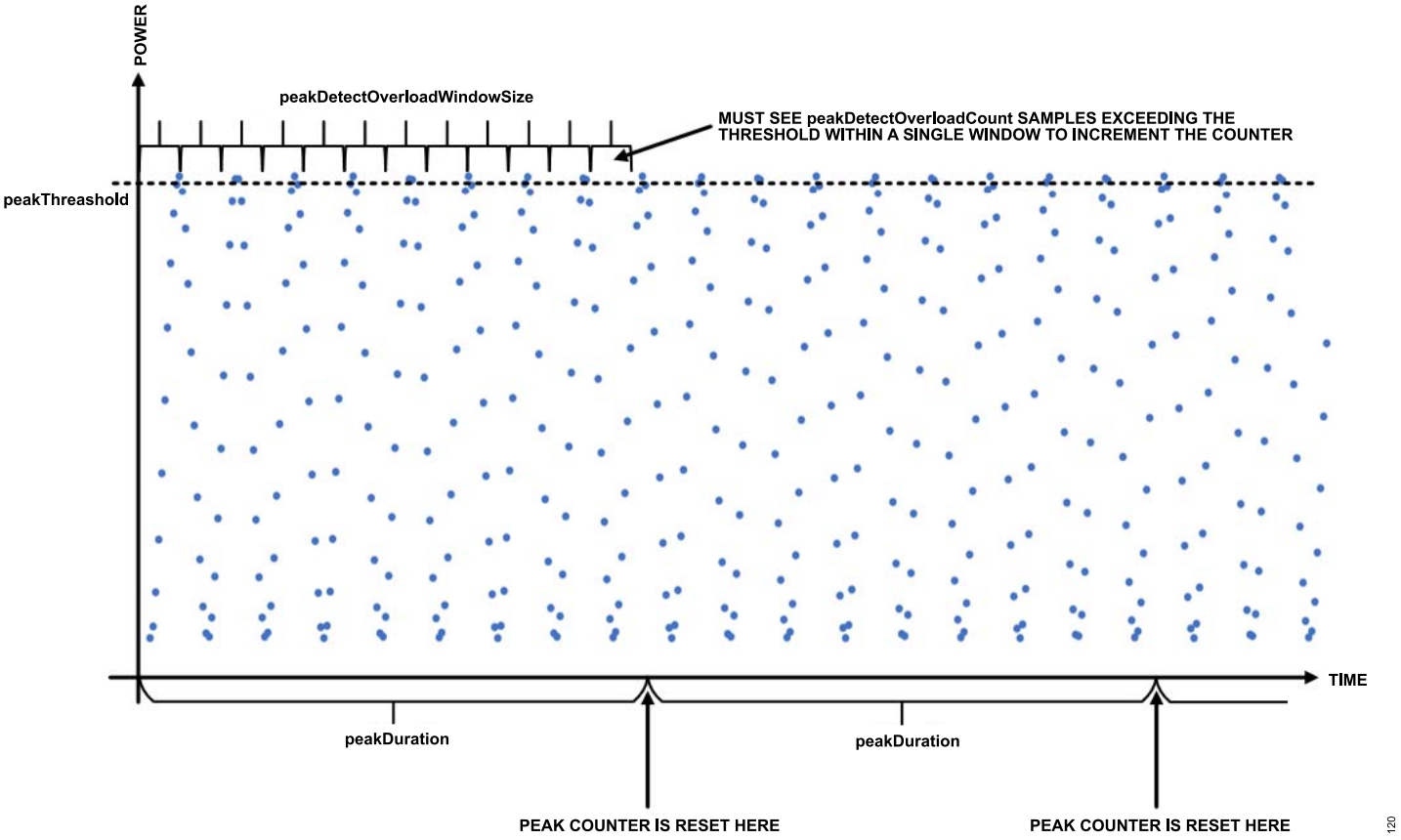


Figure 126. PA Peak Power Configuration Options

The descriptions of the parameters are as follows:

- **measDuration** sets the width of the nonoverlapping window, which triggers the reset of the peak counter. Values range from 0 to 15. The value is translated into an I/Q sample count using the formula $2^{(measDuration + 7)}$. If this parameter is set to three, the peak counter resets at every 1024 I/Q 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 I/Q samples sent across the JESD interface.
- **peakDetectOverloadWindowSize** sets the width of the nonoverlapping window, which looks for multiple samples exceeding the threshold to trigger the peak counter. Values correspond directly to the number of I/Q samples observed at the I/Q rate of the protection circuit. This value is automatically set to total *Input Interpolation Rate* × 2 to account for the interpolation effect.
- **peakDetectOverloadCount** sets the number of samples that must exceed the threshold within a window of **peakDetectOverloadWindow-Size** 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.
- **peakThreshold** sets the threshold that must be exceeded for a peak to be found. Values range from 0 to 65,535. When setting this threshold, the I/Q samples are considered to have a max value of 1,0. It is assumed that each (I/Q) sample must be within the unit circle; therefore, an example of a max sample is 1,0. Taking a signal with peaks at −3 dBFS, the peak (I/Q) amplitude value example is ~0.707,0, and the power ($I^2 + Q^2$) of this sample is ~0.5. Therefore, the corresponding threshold setting for this −3 dBFS peak sample is ~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/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.

PA PROTECTION

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 nonsticky, it automatically clears at the end of the subsequent measurement period if that measurement period does not contain sufficient average power to also trigger it.

The parameters used to configure the average power measurement are detailed as follows:

- **measDuration** sets the width of the nonoverlapping window during which the average power is calculated. Values range from 0 to 15. The value is translated into an I/Q sample count using the formula $2^{(measDuration + 7)}$. If this parameter is set to three, the power is averaged over 1024 I/Q 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 I/Q 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 65,535. When setting this threshold, the I/Q samples are considered to have a max value of 1,0. It is assumed that the (I/Q) sample pairs must be within the unit circle; therefore, a max sample pair is 1,0. Taking a signal with an average power of −3 dBFS, the average (I/Q) amplitude value example is ~0.707,0, and the power ($I^2 + Q^2$) of this sample is ~0.5. Therefore, the corresponding threshold setting for this −3 dBFS average power is ~0.5 × 2^{16} = 32,846.

SLEW RATE DETECTION (SRD) AND LIMITING

The SRD block can monitor the signal at the output of the QEC correction block, or at the input to the digital attenuation block. The slew rate between two adjacent samples is calculated as

$$\frac{2^{16}}{2^{2Np}}\left((I_n - I_{n-1})^2 + (Q_n - Q_{n-1})^2\right) \quad (11)$$

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 × $\log_{10}$(*srdOffset* ÷ 65,535). The alarm raised by the SRD block can be made sticky; therefore, requiring user intervention to clear it once it triggers, or it can be set to autoclear 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; therefore, blocking the offending samples from propagating through the datapath. This is enabled using the API function **adi_adrv904x_TxProtectionRampSampleHoldEnableSet()**.This results in a DC level persisting in the datapath. Therefore, the slew alarm is usually used to also trigger a ramp of the analog attenuator at the transmitter output to ramp down 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 transmitter 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 that can be programmed is from 0 to 65,535 and directly corresponds to the result of Equation 11. 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 reduce the slew rate of the original signal as can be seen in Figure 127.

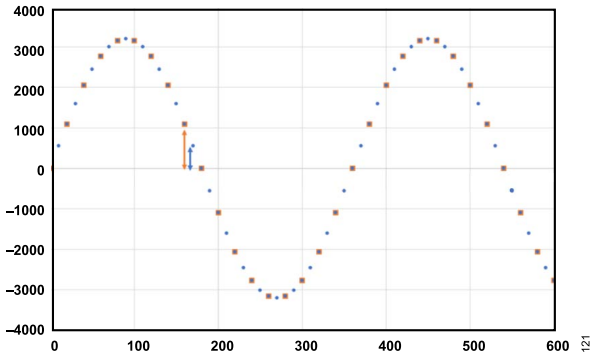


Figure 127. Slew Rate Comparison Between a Signal (Orange) and the Same Signal Interpolated by 2 (Blue)

PA PROTECTION

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

It is possible to enable the slew rate limiter interrupt in order 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 ramp-down has been enabled on occurrence of an SRD, the GPINT pin asserts from low to high, and the ramp-down starts typically within 100 ns. When the slew rate detector is cleared, the GPINT pin deasserts from high to low, and the transmitter analog front end ramps back up to its previous value. If the user wants the transmitter front end to fully ramp up before the GPINT pin deasserts, 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 that if you are splitting the interrupt sources separately to GPINT0 and GPINT1, you must ensure you map ARM0 Force GP Interrupt to GPINT0 and ARM1 Force GP Interrupt to GPINT1.

PA PROTECTION API FUNCTIONS

Table 58. List of PA Protection API Functions

API Method Name	Comments
adi_adrv904x_TxProtectionErrorGet()	Reads the status of events causing Tx power ramp-down.
adi_adrv904x_TxProtectionErrorClear()	Clears the error flags causing Tx power ramp-down.
adi_adrv904x_TxProtectionRampSampleHoldEnableSet()	Enables/disables the sample hold for Tx PA protection ramp.
adi_adrv904x_TxProtectionRampSampleHoldEnableGet()	Reads back sample hold ramp-down configuration for selected Tx channel.
adi_adrv904x_TxProtectionRampCfgSet()	Sets the Tx protection ramp configuration. This function configures Tx ramp-down in case of an average power error, peak power error, SRD error, PLL unlock, or deframer IRQ.
adi_adrv904x_TxProtectionRampCfgGet()	Reads the Tx protection ramp configuration.
adi_adrv904x_TxPowerMonitorCfgSet()	Sets the Tx power monitor configuration.
adi_adrv904x_TxPowerMonitorCfgGet()	Reads the Tx power monitor configuration.
adi_adrv904x_TxPowerMonitorStatusGet()	Reads average and peak power, average to peak ratio(if enabled), average power at the time when the last average power error occurred, and peak power at the time when the last peak power error occurred.
adi_adrv904x_TxSlewRateDetectorCfgSet()	Sets the Tx slew rate detector configuration.
adi_adrv904x_TxSlewRateDetectorCfgGet()	Reads the Tx slew rate detector configuration.
adi_adrv904x_TxSlewRateStatisticsRead()	Reads the slew rate detector statistics.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

The ADRV904x main receivers (Rx0 to Rx7) have a front end with a variable gain setting. Each receiver has an independent gain setting that is controlled externally in MGC or by an internal state machine that makes gain change decisions based on input signal levels in 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 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 must be sufficient because the received signal is typically a multicarrier signal. In this case, a gain change must be performed only under large over range or under range conditions. If a blocker does appear, a fast attack mode exists that must 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 sections and their functional descriptions:

- The Receiver Datapath section outlines the gain control and signal observation elements of the receiver chain. It also describes the concept of the receiver gain table.
- The Observation Receiver Gain Control section outlines the differences between the receiver and observation receiver gain control.
- The Gain Control Modes section explains how to select between the gain control modes.
- The MGC Mode section describes how to operate the device in MGC mode.
- The AGC Mode section describes the two principal modes of AGC operation, peak detect mode and peak/power detect mode.
- The AGC Clock and Gain Block Timing section describes the speed of the AGC clock and the various gain event and delay timers.
- The Peak And Power Detectors section outlines the operation and configuration of the gain control detectors in the device.
- The AGC API Functions section and the Rx and ORx Power Measurement API Functions section list out the API functions used for AGC and MGC.
- The AGC Sample Script section shows an example python code.
- The Gain Compensation, Floating Point Formatter, and Slicer section shows that 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

AGC—the gain control mode where each receiver's internal AGC state machine controls the receiver gain settings.

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 overrange condition.

Gain recovery—an increase in receiver gain index due to an underrange 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.

Overrange condition—an overrange 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.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

Underrange condition—an underrange 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 128 is a block diagram of the receiver 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).

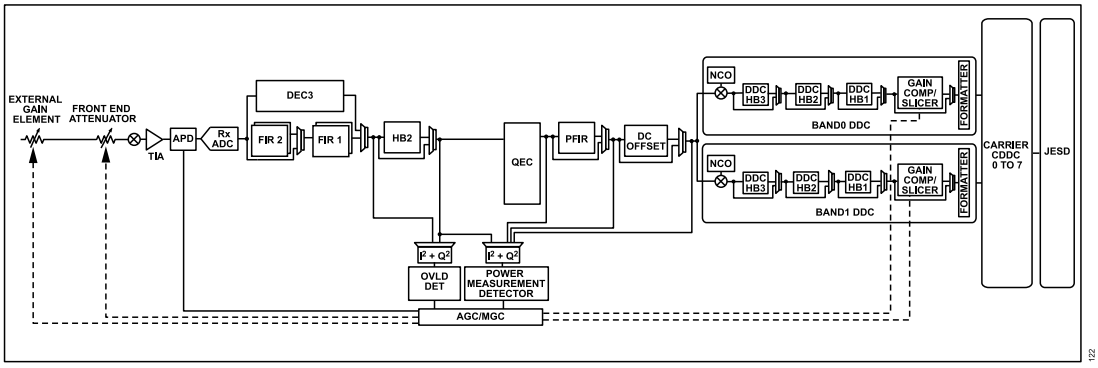


Figure 128. Receiver Datapath and Gain Control Blocks

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:

$$FE \quad Atten \quad Attenuation \quad (dB) = 20\log_{10}\left(\frac{256-N}{256}\right) \tag{12}$$

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 do the following:

- Provide small amounts of gain or attenuation to ensure the most consistent gain differences between different gain indices in the gain table.
- Provide digital gain compensation that 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 59.

Table 59. Signed Digital Gain/Attenuation Word Definition

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

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

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

The external 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 receiver to control an external element. Examples of the external element include LNA disable pins or input pins to a 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, Analog Devices 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 low-band and midband configurations where LO < 4.5 GHz. For high band applications with LO > 4.5 GHz, you must use the **RxGainTable_HB.csv** table. 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 dotted arrows in Figure 128.

Table 60 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. Refer to the documentation earlier in this section to translate values in the table to gain values.

Table 60. Sample Rows from the Default Receiver 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.

The gain index is programmed during initialization in the command **adi_adrv904x_RxGainTableLoad()**. This command is called during the **adi_adrv904x_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; therefore, any attenuation in the TIA affects the observed signal.

The second peak detector is called the HB2 overload detector, which 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.

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.

OBSERVATION RECEIVER GAIN CONTROL

The ORx0 and ORx1 channels are distinct from the receiver channels in the ADRV904x. The architecture of the observation receiver is based on an RF ADC design as opposed to the ZIF architecture used in the main receivers.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

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

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

Table 61. List of Receiver Gain API Functions

API Method Name	Comments
adi_adrv904x_RxGainTableWrite()	Programs the gain table settings for Rx channels.
adi_adrv904x_RxGainTableRead()	Reads the gain table entries for Rx channels requested.
adi_adrv904x_RxMinMaxGainIndexSet()	Updates the minimum and maximum gain indices for a requested Rx ,channel.
adi_adrv904x_RxGainTableExtCtrlPinsSet()	Enables or disables the routing of the external control word from an Rx channel gain table to its external analog GPIO pins.
adi_adrv904x_OrxAttenGet()	Reads the attenuation of an ORx channel.
adi_adrv904x_RxGainCtrlModeSet()	Sets the Rx gain control mode to MGC or AGC.
adi_adrv904x_RxGainCtrlModeGet()	Reads the Rx gain control mode with Rx channel index
adi_adrv904x_RxTempGainCompSet()	Sets the temperature gain compensation parameter for Rx channel only.
adi_adrv904x_RxTempGainCompGet()	Reads the temperature gain compensation parameter for Rx channel only. Only one channel can be retrieved per call.
adi_adrv904x_OrxAttenSet()	Sets the attenuation of ORx channels. Attenuation is 0 dB to 17 dB, in steps of 1 dB.

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.

MGC MODE

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: API functions and pin control. By default, if MGC is chosen the part is configured for API functions.

Table 62. List of MGC API Functions

API Method Name	Comments
adi_adrv904x_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 is written to the transceiver
adi_adrv904x_RxMgcGainGet()	Reads the Rx MGC gain index for the requested Rx channel.

AGC MODE

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_adrv904x_AgcCfg_t** is used to select the AGC mode of operation as shown in Table 63.

Table 63. 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.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

Peak Detect Mode

In peak detect 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 **hb2UnderRangeHighThresh**. The limit for the number of times a threshold needs to be crossed for an overrange or underrange 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 overrange 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 underrange. The AGC stable state (where it does not adjust gain) occurs when neither an underrange nor an overrange condition is occurring (the signal peaks are less than or equal to the high thresholds and greater than the lower threshold levels).

Each overrange/underrange condition has its own attack and recovery gain step as shown in Table 64.

Table 64. Peak Detector Gain Steps

Overrange/Underrange Condition	Gain Step
adcOvldHighThresh overrange	Reduce gain by **adcOvldGainStepAttack**.
adcOvldLowThresh underrange	Increase gain by **adcOvldGainStepRecovery**.
hb2HighThresh overrange	Reduce gain by **hb2GainStepAttack**.
hb2UnderRangeHighThresh underrange	Increase gain by **hb2GainStepHighRecovery**.

An overrange condition occurs when a high threshold has been exceeded a configurable number of times within a configurable period. An underrange 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 65 outlines these configurable counts for each threshold's overrange/underrange conditions.

Table 65. Peak Detector Threshold Exceeded Counts

Overrange/Underrange Condition	Threshold Exceeded Count
adcOvldHighTresh overrange	**adcOvldUpperThreshPeakExceededCnt**
adcOvldLowThresh underrange	**adcOvldLowerThreshPeakExceededCnt**
hb2HighThresh overrange	**hb2UpperThreshPeakExceededCnt**
hb2UnderRangeHighThresh underrange	**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** is set to 1, then **hb2HighThresh** must be crossed one or more times within the configurable period for the signal to be considered overranging the **hb2HighThresh**. Similarly, if **hb2UnderRangeHighThreshExceededCnt** is set to 1, then not crossing the **hb2UnderRangeHighThresh** even once during the configurable period indicates an underranging condition. However, if **hb2UnderRangeHighThresh** is crossed one or more times, then the signal does not become underranging (no gain recovery is required in this case).

The configurable period for determining underrange and overrange 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 receiver is powered on. The GUC value, and, therefore, the time spacing between possible gain changes, is user-programmable through the **agcGainUpdateCounter** parameter. Therefore, the GUC value sets a periodic interval, in AGC clock cycles, with which gain changes can be made. Typically, this might be set to frame or subframe boundary periods. The **agcSlowLoopSettlingDelay** (configured in AGC clock cycles) contributes to this periodic interval in addition to the **agcGainUpdateCounter**, as shown in Figure 136.

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 129 shows an example of the AGC response to a signal vs. 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 Figure 133 and described as follows:

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

1. During the first GUC period shown, the peaks of the signal are within the **adcOvldHighThresh** and **adcOvldLowThresh**; therefore, 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 has been detected to exceed the **adcOvldUpperThreshPeakExceededCnt**, the AGC decrements the gain index (reduces the gain) by **adcOvldGainStepAttack**.
4. Because one gain attack is not sufficient to read 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 underrange condition.
7. Although the signal is underranging, no gain changes are made at this GUC boundary assuming that enough peaks are detected to exceed the **adcOvldLowerThreshPeakExceededCnt**, while the interferer was present earlier in the same period.
8. As the signal continues to underrange, no peaks are detected above **adcOvldLowThresh**. Therefore, 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**.

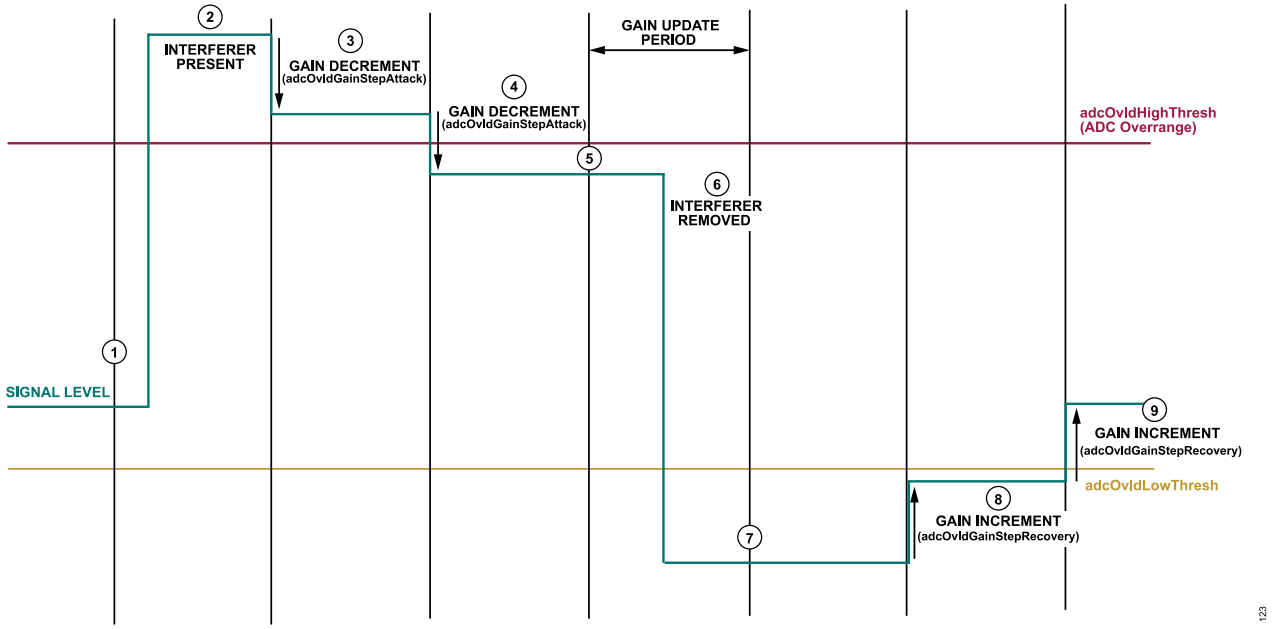


Figure 129. ADC Overload Thresholds and Gain Changes Associated with Underrange and Overrange Conditions

Figure 130 shows the same scenario but from the viewpoint of the HB2 detector considered in isolation.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

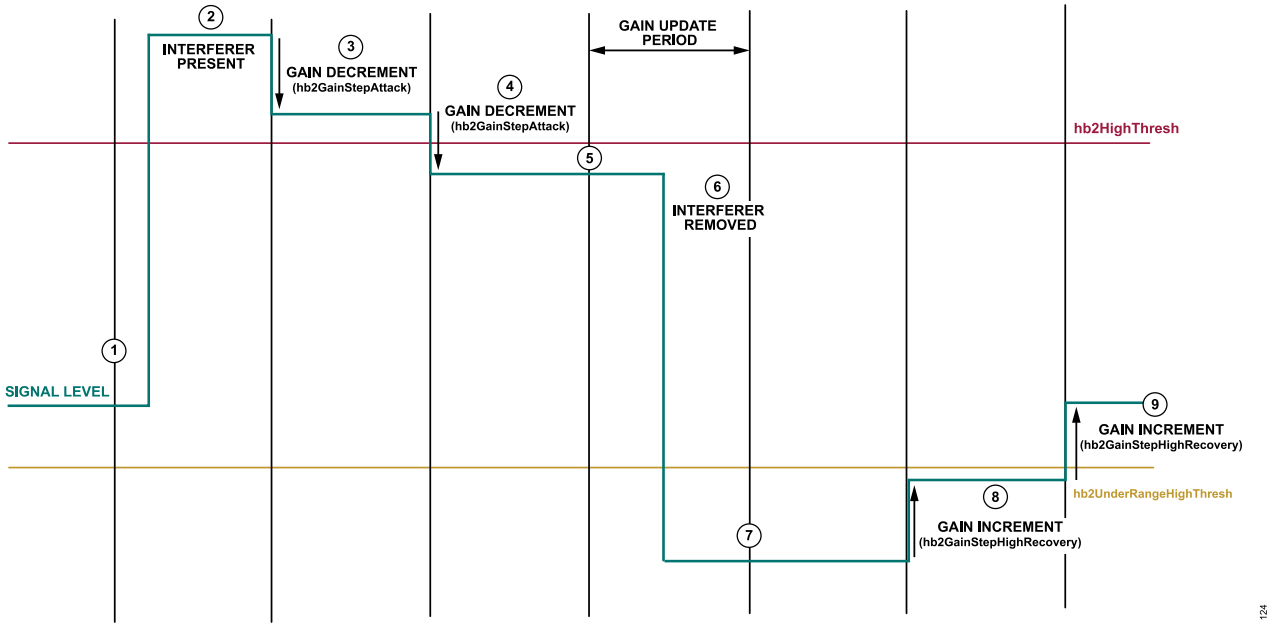


Figure 130. HB2 Thresholds and Gain Changes Associated with Underrange and Overrange Conditions

In both 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 overrange 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 to 3 are valid as shown in Table 66.

Table 66. agcGainChangeIfThreshHigh Settings

agcChangeGainIfThreshHigh[1:0]	Gain Change Following ADC Overload Overrange	Gain Change Following HB2 Overrange
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 131 shows how the AGC reacts when the agcChangeGainIfThreshHigh[1:0] 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 can 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 subframe 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.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

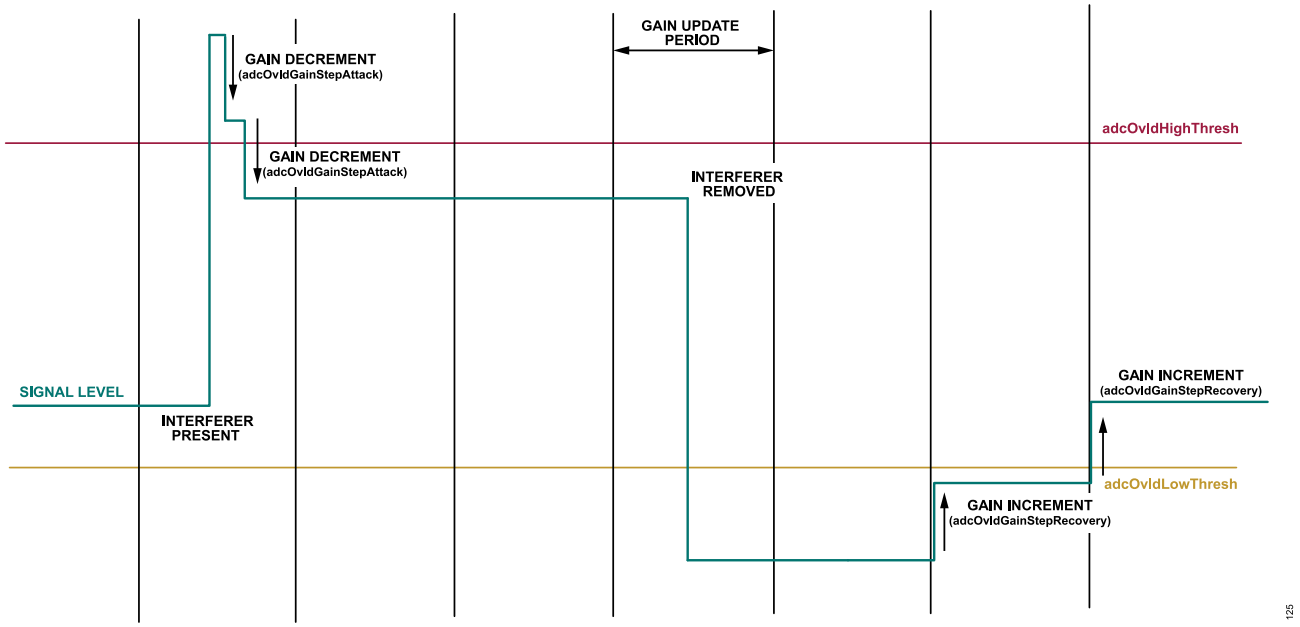


Figure 131. ADC Overload Gain Changes with Fast Attack Enabled

Figure 132 shows the same scenario but from the viewpoint of **agcChangeGainIfThreshHigh** being set for HB2.

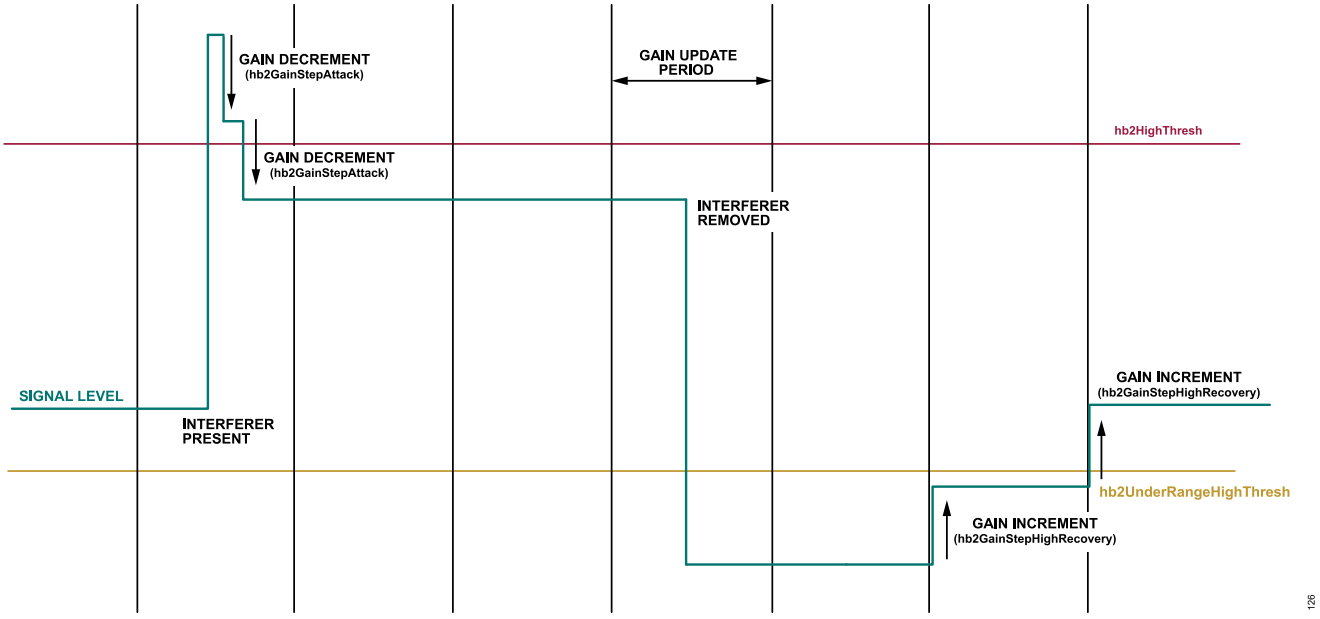


Figure 132. HB2 Gain Changes with Fast Attack Enabled

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 Receive Half Band 2 (HB2) section. Gain update intervals for the underrange thresholds are tabulated in Table 67.

Table 67. HB2 Peak Detector Recovery Steps and Intervals in Fast Recovery

Underranging Threshold	Gain Step	Gain Recovery Following HB2 Underrange
hb2UnderRangeLowThresh	**hb2GainStepLowRecovery**	After expiry of **hb2UnderRangeLowInterval** (replaces GUC)
hb2UnderRangeMidThresh	**hb2GainStepMidRecovery**	After expiry of **hb2UnderRangeMidInterval**

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

Table 67. HB2 Peak Detector Recovery Steps and Intervals in Fast Recovery (Continued)

Underranging Threshold	Gain Step	Gain Recovery Following HB2 Underrange
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 underrange 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 133 illustrates fast recovery with an example scenario described as follows:

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 underrange.
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**.

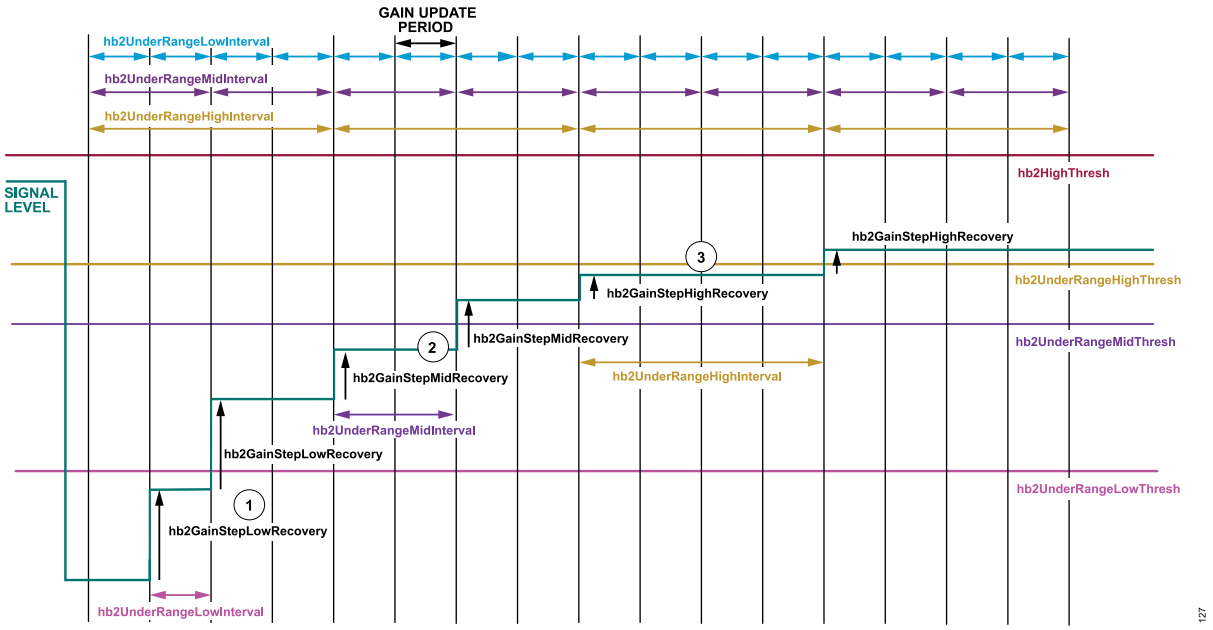


Figure 133. AGC Operation with HB2 Detector in Fast Recovery Mode

Priorities and Overall Operation

The **hb2HighThresh** threshold 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 adcOvldLowThresh and **hb2UnderRangeHighThresh** are set to equivalent values. Typical values are approximately −6 dBFS to −8 dBFS. This equivalence is approximate, as these thresholds have unique threshold settings and are not exactly equal. This section discusses the relevant priorities between the detectors, and how the AGC reacts when multiple threshold detectors have been exceeded. Table 68 shows the priorities between the detectors in default (slow attack) mode, when multiple overranges occur during the same GUC period.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

Table 68. Priorities of Attack Gain Steps

adcOvldHighThresh Overrange	hb2HighThresh Overrange	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 as follows:

1. **hb2UnderRangeLowThresh** underrange condition
2. **hb2UnderRangeMidThresh** underrange condition
3. **hb2UnderRangeHighThresh** underrange condition
4. **adcOvldLowThresh** underrange condition

Upon one underrange condition, the AGC changes the gain by the corresponding gain step size of this condition. However, if multiple conditions occur simultaneously, then the AGC prioritizes based on the priorities indicated; that is, if **hb2UnderRangeLowThresh** is reporting an underrange condition then the AGC adjusts the gain by **hb2GainStepLowRecovery** with two exceptions.

The **adcOvldLowThresh** has priority in terms of preventing recovery. If **adcOvldLowThresh** reports an overrange condition (at least the **adcOvldLowerThreshPeakExceededCnt** number of signal peaks has exceeded **adcOvldLowThresh** within a GUC period), then no further recovery is allowed. **adcOvldLowThresh** and **hb2UnderRangeHighThresh** must 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 no longer occurs. The reverse is not true, **hb2UnderRangeHighThresh** does not prevent the gain recovery toward **adcOvldLowThresh**. Given the strong recommendation that **adcOvldLowThresh** and **hb2UnderRangeHighThresh** being set equally, then a condition whereby **adcOvldLowThresh** is 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. In that case, the AGC makes the gain change of the highest priority detector with a nonzero recovery step. Figure 134 provides a flow diagram of the decisions of the AGC when recovering the gain in peak detect mode.

Power Detect Mode

In power detect mode, the power detector measurement is also used to control the gain of the receiver 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 overrange scenarios, and power measurement blocks inform gain increases in underrange scenarios. Power measurement detectors by themselves cannot allow for fast attack style operation. The power detector changes 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 overrange 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 128. 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.

Figure 135 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 does not modify the gain when the signal level is between **overRangeLowPowerThresh** and **underRangeHighPowerThresh**. This range is the target range for the power measurement.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

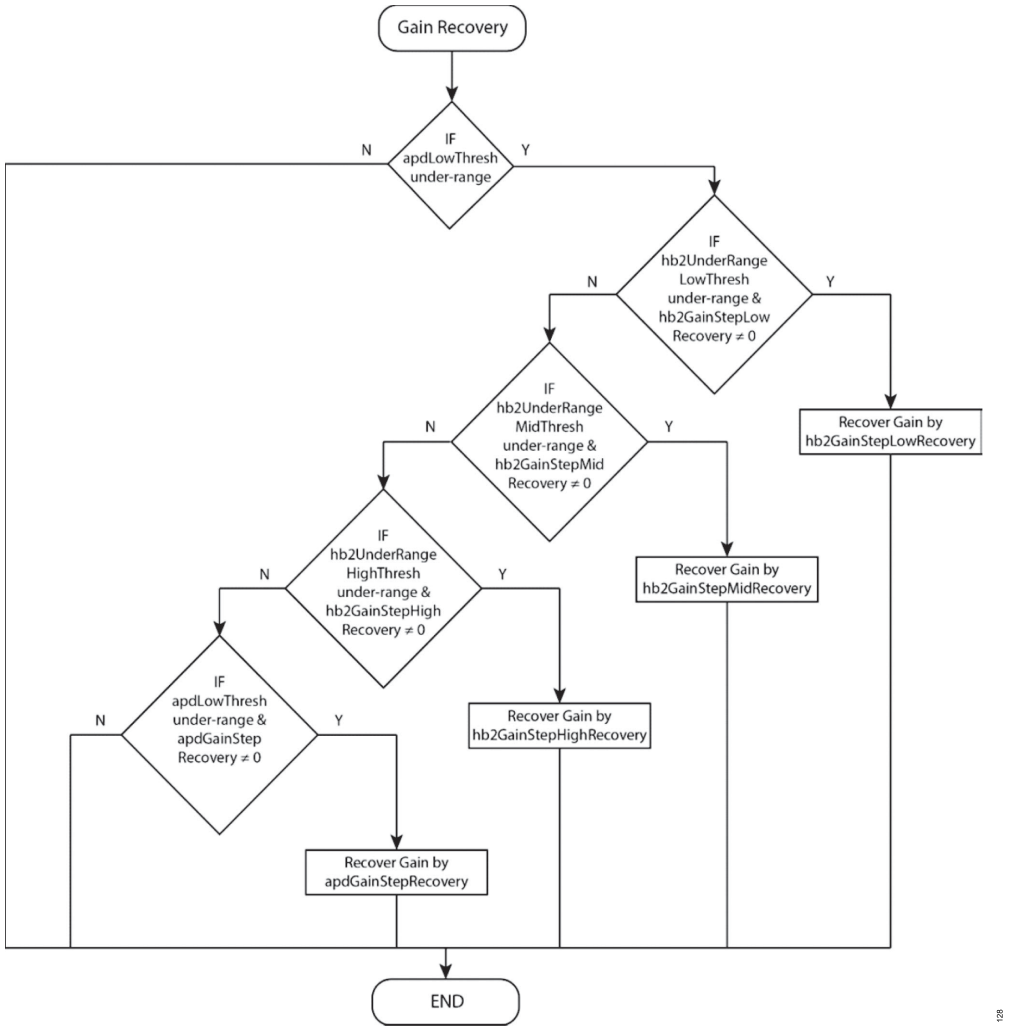


Figure 134. Flow Diagram for AGC Recovery in Peak Detect AGC Mode

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 increments 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 decreases the gain by **overRangeLowPowerGainStepAttack**.

It is possible for the AGC to get conflicting requests from the power and peak detectors. An example is 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:

1. ADC overload detector high threshold overrange
2. HB2 high threshold overrange
3. ADC overload detector low threshold overrange
4. HB2 low threshold overrange
5. Power measurement

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

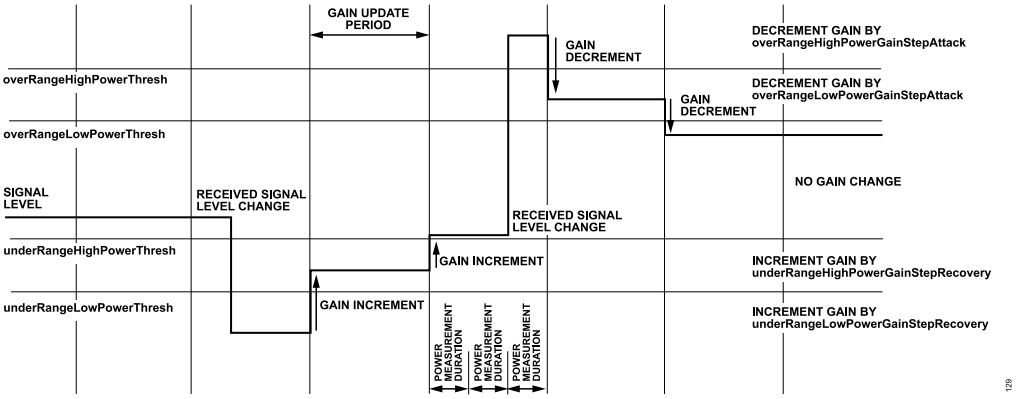


Figure 135. PMD Thresholds and Gain Changes for Underrange and Overrange Conditions

In this example, the gain decrements because the ADC overload high threshold overrange 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 are used to indicate an underrange condition that causes 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 can otherwise occur to a blocker visible to a peak detector but filtered before the power measurement block. In such a case, the peak detector can cause the AGC to decrease gain. It does this until the blocker is no longer exceeding the defined threshold. At this point, the power measurement block can request an increase in gain, and it does so until the detector's peak threshold is exceeded. This decreases 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 receiver 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 I/Q rate less than 500 MHz. For example, for a receiver profile with an I/Q output rate of 245.76 MSPS, the AGC clock is 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 × 100 µs = 49,152 (13)

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(*agcSlowLoopSettlingDelay*) + 5 (14)

Gain Update Period (AGC Clocks) = 491,139 + 2(4) + 5 = 49,152

Figure 136 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 as follows:

- Signal underranging/overranging 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 proceeds with any gain changes as deemed necessary at the GUC boundary (excluding fast attack and fast recovery modes).
- Signal underranging/overranging detected during the SLS delay— This is a special case, but rarely occurs in applications per the reasoning in explained above. As AGC's peak counters are reset during the SLS delay, any underranging or overranging is not flagged over the SLS

duration. The AGC is essentially blind to input signal conditions during the SLS delay. However, any signal underranging or overranging that persists onto the next GUC period is addressed as in the previous point.

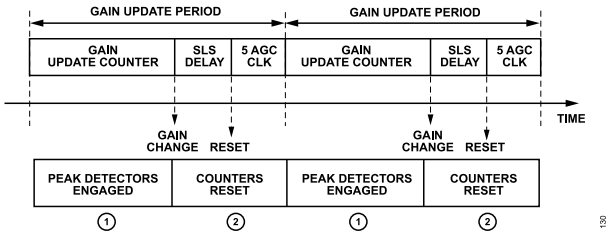


Figure 136. Gain Update Period

Note 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 the receiver 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 receiver 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 receiver is reenabled. Note that when agcResetOnRxon is 0 (default), the GUC pauses when the receiver is disabled, and resumes from its last value when receiver is reenabled, 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 overranging (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, underranging 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 in Figure 137.

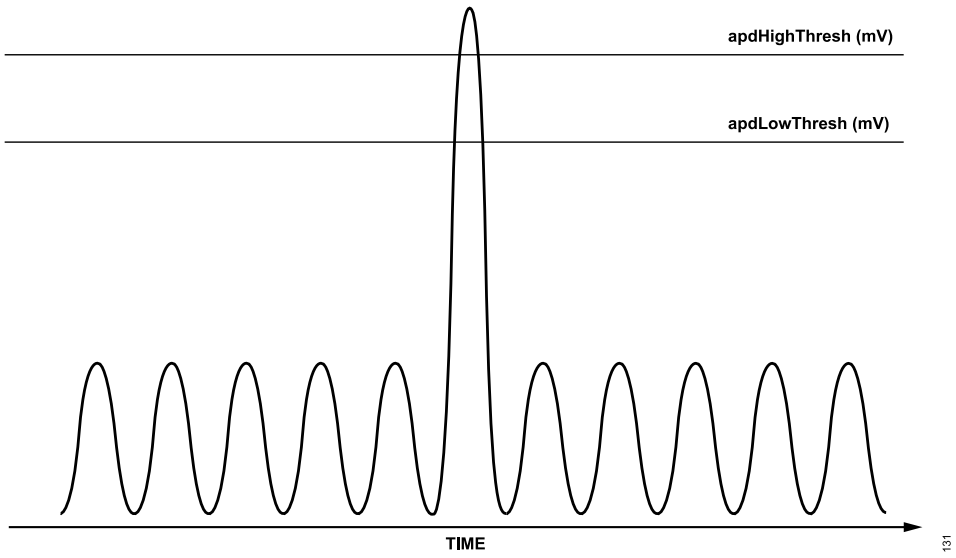


Figure 137. Analog Overload Detector Thresholds

The low threshold adcOvldLowThresh can be set to any value in the range of 0 to 8, where 8 represents the ADC's full scale. The API value can be scaled in terms of the ADC full scale (ADC dBFs) by using the following equation:

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

$$ADC\ \ Overload\ \ Low\ \ Threshold\ \ \left(in\ \ dBFS\right) = 20\log\left(\frac{adcOvldLowThresh}{8}\right) \quad (15)$$

Because 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 3 dB 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 is 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 are tabulated in Table 64 and Table 65.

Table 69. 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 overranging 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 underranging 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 overrange 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 underrange condition is observed.

Figure 138 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).

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

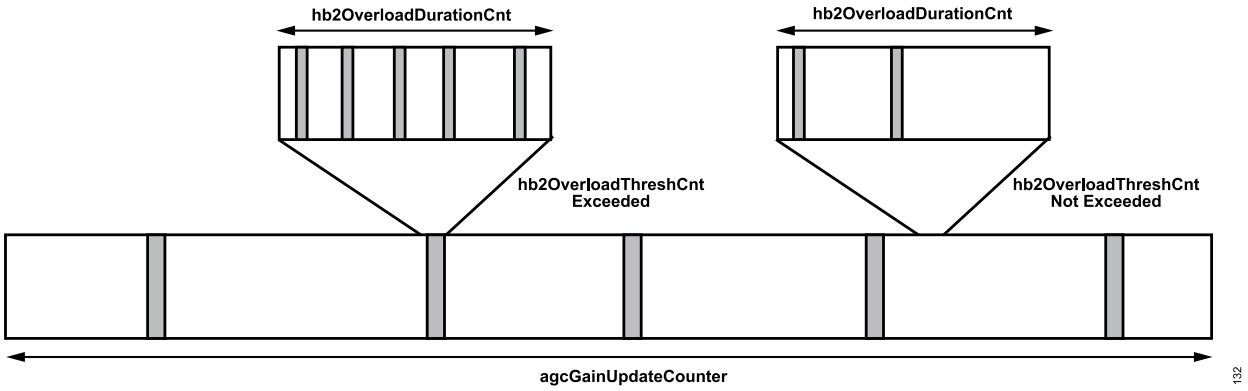


Figure 138. HB2 Detector's Two-Level Approach for an Overload Condition

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 70.

Table 70. 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 130, Figure 132, and Figure 133).

The thresholds are related to an ADC dBFS value using the following equations:

$$hb2HighThresh = 16384 \times 10\left(\frac{hb2High \quad dBFS}{20}\right) \tag{16}$$

$$hb2UnderRangeHighThresh = 16384 \times 10\left(\frac{hb2UnderRangeHigh \quad dBFS}{20}\right) \tag{17}$$

$$hb2UnderRangeMidThresh = 16384 \times 10\left(\frac{hb2UnderRangeMid \quad dBFS}{20}\right) \tag{18}$$

$$hb2UnderRangeLowThresh = 16384 \times 10\left(\frac{hb2UnderRangeLow \quad dBFS}{20}\right) \tag{19}$$

Note that these equations only apply if 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 overrange condition is not flagged until the threshold has been exceeded this amount of times in a gain update period.

Table 71. HB2 Overload Thresholds and Counters

HB2 Threshold	Counter
hb2HighThresh	hb2UpperThreshPeakExceededCnt
hb2UnderRangeHighThresh	hb2UnderRangeHighThreshExceededCnt
hb2UnderRangeMidThresh	hb2UnderRangeMidThreshExceededCnt
hb2UnderRangeLowThresh	hb2UnderRangeLowThreshExceededCnt

In AGC mode, the HB2 has programmable gain attack and gain recovery step sizes.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

Table 72. 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 73.

Table 73. 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:

$$Power\ Meas\ Duration\ (Rx\ Sample\ Clocks) = 8 \times 2^{powerMeasurementDuration} \quad (20)$$

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 74. List of AGC Configuration API Functions

API Method Name	Comments
adi_adrv904x_AgcCfgSet()	Configures all the AGC settings as described above.
adi_adrv904x_AgcCfgGet()	Reads back the AGC settings.
adi_adrv904x_AgcGainIndexRangeSet()	Configures min/max gain indices allowed for AGC operation.
adi_adrv904x_AgcGainIndexRangeGet()	Reads the AGC Gain range for selected channel.
adi_adrv904x_AgcReset()	Resets all AGC state machines and peak detector counters for selected channels.
adi_adrv904x_RxGainGet()	Reads the Rx AGC Gain Index for the requested Rx channel.
adi_adrv904x_RxTempGainCompSet()	Sets the Rx gain compensation.
adi_adrv904x_RxTempGainCompGet()	Reads the Rx gain compensation.

RX AND ORX POWER MEASUREMENT API FUNCTIONS

Table 75. List of Rx and ORx Power Measurement API Functions

API Method Name	Comments
adi_adrv904x_RxDecimatedPowerCfgSet()	Sets the Rx decimated power measurement block configuration.
adi_adrv904x_RxDecimatedPowerCfgGet()	Reads the Rx decimated power measurement block configuration.
adi_adrv904x_RxDecimatedPowerGet()	Reads the decimated power measurement for selected Rx/ORx channel.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

AGC SAMPLE SCRIPT

The following sample Python script is a function definition for settings the AGC values. The values below are Analog Devices 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
#Misc
AgcCfg.hb2OverloadDurationCnt = 32
AgcCfg.hb2OverloadPowerMode = 0
AgcCfg.hb2OverloadSignalSelection = 0
AgcCfg.enableHb2Overload = 1
AgcCfg1.agcPeak = AgcCfg
AgcCfg1.agcPower.powerEnableMeasurement = 0
```

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

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 in Figure 139. Figure 139 is a block diagram of the gain compensation portion of the Rx chain, showing the locations of the various blocks.

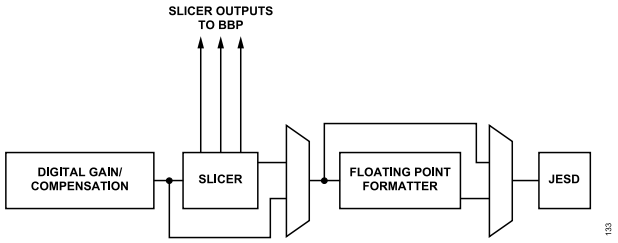


Figure 139. Gain Compensation, Floating Point Formatter, and Slicer Section of the Receiver Datapath

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 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. Therefore, 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 140 outlines this effect, with yellow boxes indicating the valid (used) bits in each case.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

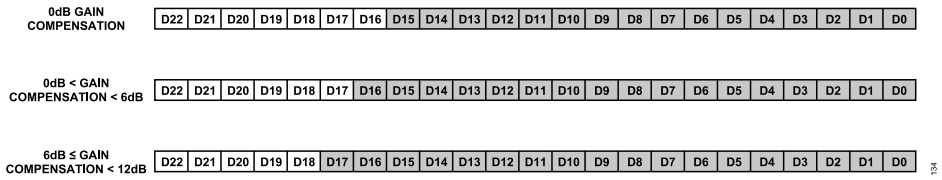


Figure 140. Bit Width of Received Signal for Increasing Gain Compensation

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, in order 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 of this document).

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 141 explains the operation. Initially the analog attenuator is applying no attenuation (0 dB), and, therefore, there is 0 dB digital gain applied to the signal. The slicer is in its default (0000) position. As the attenuation increases (0 dB 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. Therefore, the slicer 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).

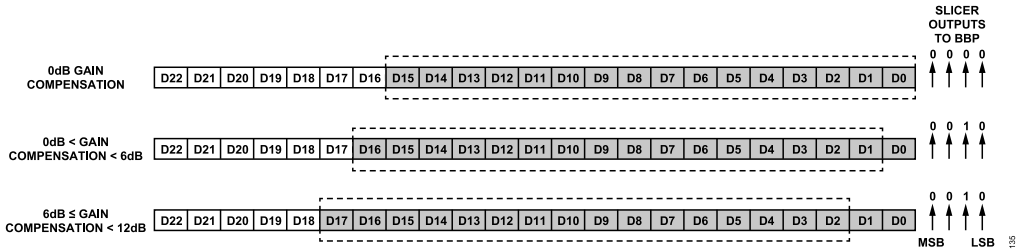


Figure 141. Slicer Bit Selection with Digital Gain

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 vs. digital gain for this 6 dB example is described in Table 76. Equivalent tables can be inferred for the other attenuation options.

Table 76. Slicer GPIO Output vs. 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

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

Table 76. Slicer GPIO Output vs. Digital Gain Compensation (Continued)

Digital Gain Compensation (dB)	Slicer Position (Value output on GPIOs)
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/Q, or 2 bits on both I/Q. These can be placed at the MSBs or LSBs. For the case where 2 bits are embedded onto both I/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.

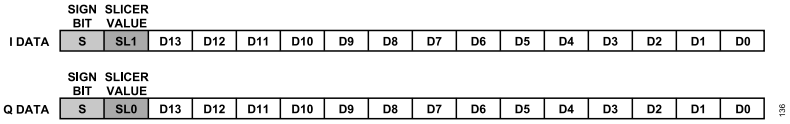


Figure 142. Encoding of Slicer Information as Control Bits (adi_adrv904x_RxSlicerEmbeddedBits = ADI_ADRV904X_EMBED_1_SLICERBIT_AT_MSB)

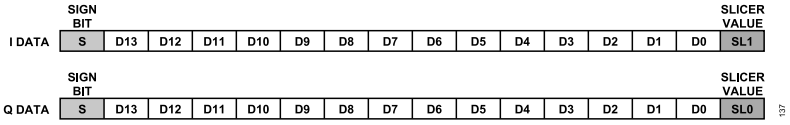


Figure 143. Encoding of Slicer Information as Control Bits (adi_adrv904x_RxSlicerEmbeddedBits = ADI_ ADRV904X_EMBED_1_SLICERBIT_AT_LSB)

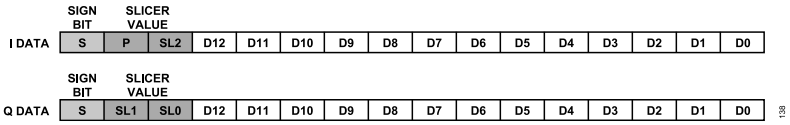


Figure 144. Encoding of Slicer Information as Control Bits (adi_adrv904x_RxSlicerEmbeddedBits = ADI_ ADRV904X_EMBED_2_SLICERBITS_AT_MSB _3_BIT_ SLICER)

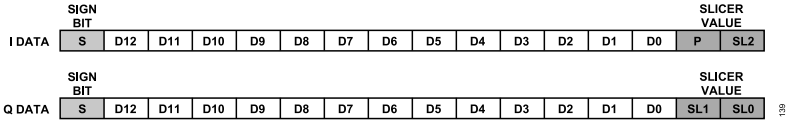


Figure 145. Encoding of Slicer Information as Control Bits (adi_adrv904x_RxSlicerEmbeddedBits = ADI_ ADRV904X_EMBED_2_SLICERBITS_AT_LSB _3_BIT_ SLICER)

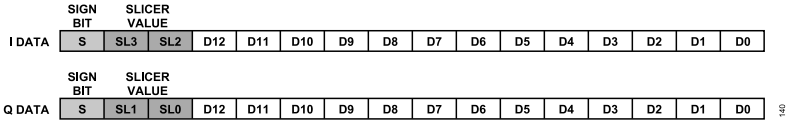


Figure 146. Encoding of Slicer Information as Control Bits (adi_adrv904x_RxSlicerEmbeddedBits = ADI_ ADRV904X_EMBED_2_SLICERBITS_AT_MSB _4_BIT_ SLICER)

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

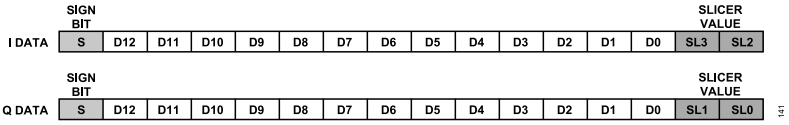


Figure 147. Encoding of Slicer Information as Control Bits (adi_adrv904x_RxSlicerEmbeddedBits = ADI_ ADRV904X_EMBED_2_SLICERBITS_AT_LSB _4_BIT_ SLICER)

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 IEEE 754 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.

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, therefore, can allow for more digital compensation to the represented, whereas more bits in the significand provide higher resolution. The available options for device's floating-point formatter are as follows:

- 5-bit exponent, 10-bit significand
- 4-bit exponent, 11-bit significand
- 3-bit exponent, 12-bit significand
- 2-bit exponent, 13-bit significand

It is also possible to pack the data in different formats (as shown in Figure 148) as follows:

- Sign, Exponent, Significand
- Sign, Significand, Exponent

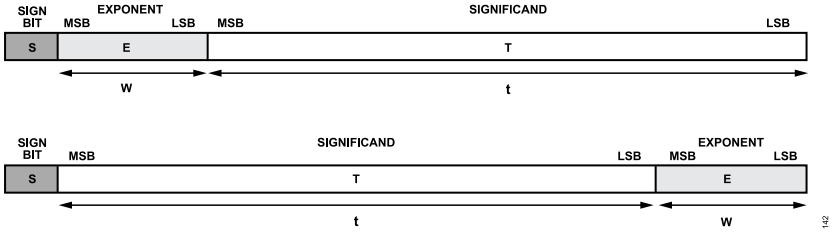


Figure 148. Floating-Point Number Representation

In Figure 148, 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, consider a 3-bit exponent. In IEEE 754, 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:

$$Value = 0 \quad (21)$$

If E = 0 and T ≠ 0:

$$Value = (-1)^S \times 2^{E - bias + 1} \times (0 + 2^{1-p} \times T) \quad (22)$$

If E ≠ 0:

$$Value = (-1)^S \times 2^{E - bias} \times (1 + 2^{1-p} \times T) \quad (23)$$

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

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 77 provides the values to use in these equations for the various IEEE 754 supported modes.

Table 77. 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

Figure 149 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.

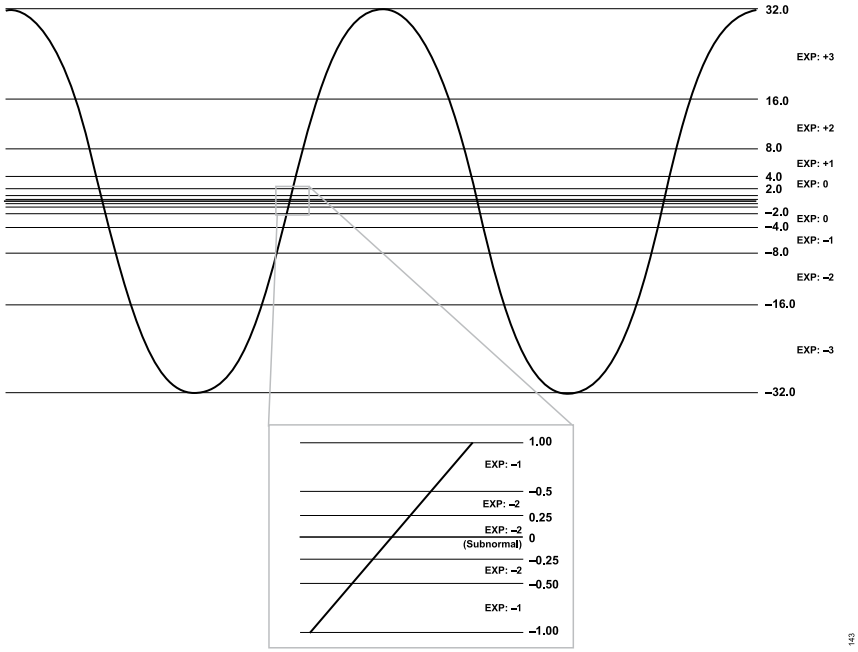


Figure 149. Visualization of the Floating-Point Formatter Values

The floating-point formatter also supports non-IEEE 754 modes, referred to as Analog Devices modes, where the largest exponent is not used to express NaN in accordance with IEEE 754. 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 78. Exponent Bit Widths of IEEE-754 and Analog Devices Modes

Exponent Bit Width (w)	IEEE-754 Mode Exponent Range (After Unbiasing)	Analog Devices Mode Exponent Range (After Unbiasing)
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.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

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 dB 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 DATA FORMAT DATA STRUCTURE

The Rx Data Formatter is not run time 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 79. List of Rx Formatter API Functions

API Method Name	Comments
adi_adrv904x_RxDataFormatGet()	Reads the Rx data path format configuration.
adi_adrv904x_RxSlicerPositionGet()	Reads the Rx gain slicer position.

DIGITAL FILTER CONFIGURATION

OVERVIEW

This section describes the digital filters within the ADRV904x. 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 overrange 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 to 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 ZIF 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. The band NCO is used to shift the carriers in case of Low IF configuration as shown in band DUC block diagram in Figure 150. API for configuring the NCOs are available in Table 80.

Figure 150 shows the signal path for the Rx0, Rx1, Rx2, Rx3, Rx4, Rx5, Rx6, and Rx7 signal chain. Grayed out blocks are not be described in this section but in their relevant sections of the user guide.

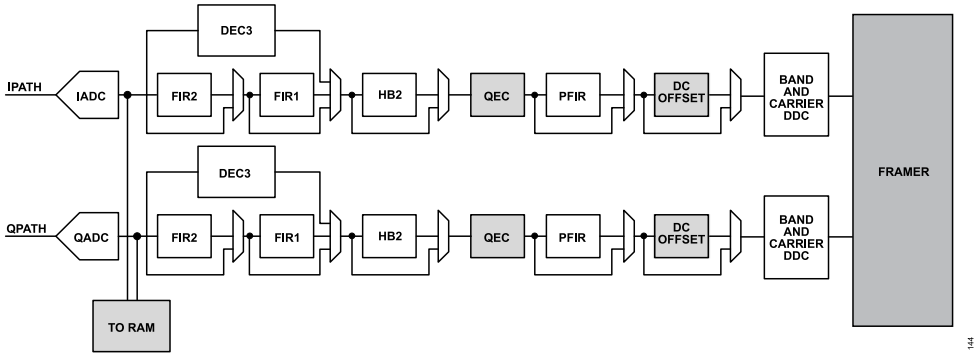


Figure 150. Rx Signal Path

Decimation Stages

The signal path can be configured in order 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, −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]

DIGITAL FILTER CONFIGURATION

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 that in this case, the Rx analog response will not be compensated for digitally in the ADRV904x; therefore, 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.

BAND DDC

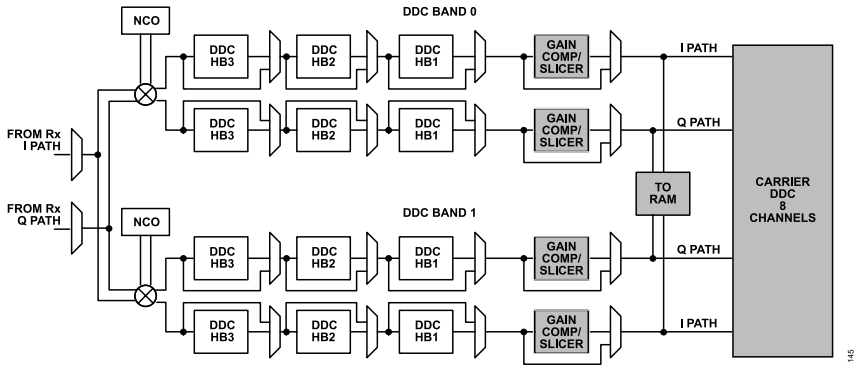


Figure 151. Band DDC High Level Block Diagram

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 151, the JESD rate is 122.88 MSPS and we have decimation of 2×, 2×, 1×. Therefore, the rate at which NCO can operate is ±245.76 MSPS. If HB1 interpolation is also 2×, then the NCO can operate with the max range of ±491.52 MSPS.

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]

RX DATAPATH API FUNCTIONS

Table 80. List of Rx Datapath API Functions

API Method Name	Comments
adi_adrv904x_RxOrxDataCaptureStart()	Captures the Rx data in an internal RAM. Use if JESD link is not brought up.
adi_adrv904x_RxNcoShifterSet()	Sets Rx NCO.
adi_adrv904x_RxNcoShifterGet()	Reads Rx NCO settings.
adi_adrv904x_RxTestDataSet()	Sets up a test signal to be sent out the framer instead of the ADC output.
adi_adrv904x_RxTestDataGet()	Reads the test signal being sent out the framer instead of the ADC output.

CARRIER DDC

Figure 152 depicts high-level view of Carrier DDC together with the Band DDC. Carrier DDC and filtering comes after the PFIR and calibrations blocks and before the JESD section.

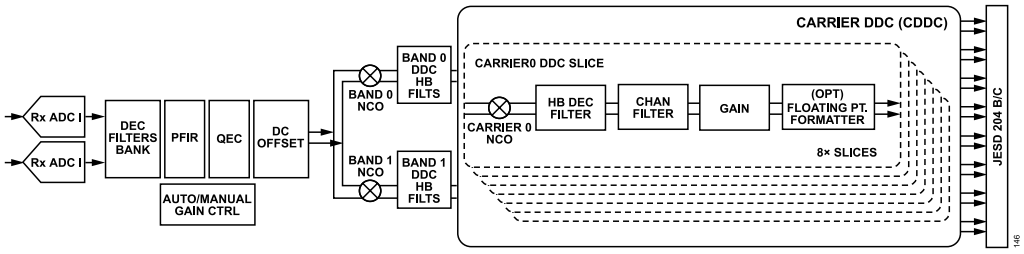


Figure 152. DDC High Level Block Diagram

There are 8 RX paths and each Rx path will have option for 8 carrier DDC as shown in Figure 152. Carrier DDC is fed by two Band DDCs preceding it. Design supports a maximum of 8 carriers to be separated out across two bands. The Mixers/NCOs does the carrier separation and down conversion. The multi-carrier data then flow through series of Half Band filters for decimation. Finally, the decimated carrier data is taken through channel filter that is used for carrier filtering to meet the ACLR requirements. It is followed by up samplers that may be required for rate matching across carriers.

Figure 153 captures the top-level architecture of Carrier DDC, depicting I/Q paths separately.

DIGITAL FILTER CONFIGURATION

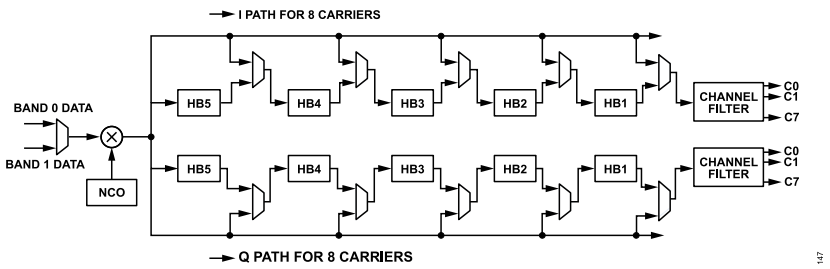


Figure 153. CDDC Top-Level Block Diagram

Salient Features of CDDC

- The maximum expected rate from individual Band DDC is 491.526 MSPS. Other sub-multiple rates like 245.76 MSPS, 122.88 MSPS, 61.44 MSPS, and up to a minimum of 7.68 MHz are supported.
  - A single band rate of 491.52 MSPS is also supported to cover FR2 cases.
  - Minimum sampling rate supported over JESD is 30.72 MHz.
  - Maximum aggregated rate over JESD is 491.52 MHz.
- The maximum number of carriers supported is 8 and it can be distributed across two bands in any kind of split. (Example 4 + 4 or 6 + 2 and so on)
- The maximum sample rate of all carriers combined in CDDC's should be less than 491.52 MHz.
- Output sample rate post HB decimation would be sub-multiples of 245.7 MSPS.
- Decimation of 1, 2, 4, 8, 16, 32 is supported as shown in Figure 152.
  - The decimation can be different for different carriers.
- Channel Filter performs only filtering operation and doesn't do any decimation.
  - The features of channel filter are detailed in later section (Configurable Channel Filter).
- All the blocks are optional/bypass-able in the path.
  - The unused carrier path can be turned off.

Overall CDDC Specification

Table 81 captures the top requirements of LTE and 5G carriers for filtering operation.

Table 81. System Requirements for Carrier Filtering (CDDC)

Parameter	LTE	5G NR	Note
Fpass	0.9 × BW/2	0.983 × BW/2	Passband edge
Fstop	BW/2	BW/2	Stopband edge
Apass	±0.1 dB	±0.1 dB	Passband ripple
Astop	70 dB	50 dB	Stopband rejection

Note that BW is the sample rate/bandwidth post decimation.

Table 82 shows the specifications used for filters based on the requirements shown in Table 81.

Table 82. ADRV904x Carrier DDC Specifications

Filters	Normalized BW with Respect to Input Sampling Rate/2	Stop Band Rejection	Passband Ripple
HB5	2.545%	90 dB	0.01 dB
HB4	5.09%	90 dB	0.01 dB
HB3	10.18%	90 dB	0.01 dB
HB2	20.35%	90 dB	0.01 dB
HB1	40.7%	90 dB	0.01 dB

DIGITAL FILTER CONFIGURATION

Table 82. ADRV904x Carrier DDC Specifications (Continued)

Filters	Normalized BW with Respect to Input Sampling Rate/2	Stop Band Rejection	Passband Ripple
Channel Filter	Programmable	50 dB/70 dB	0.05dB

40.7% is chosen for 50 MHz carrier running at 61.4 MSPS. This is the widest pass band possible through Half Bands.

Cascaded Frequency Response plot for LTE carrier (20 MHz) with Band DDC rate at 245.76 MSPS: Representation without decimation for a full view of the spectrum (the passband and stopband points are marked in Figure 154).

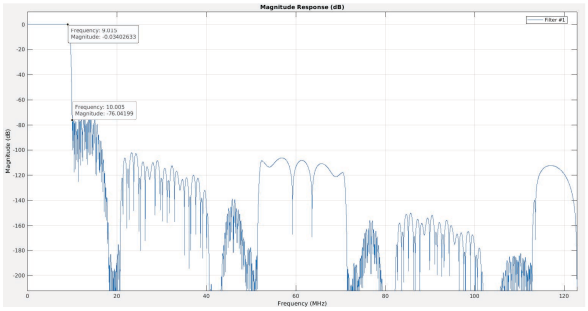


Figure 154. Combined Filter Response for LTE 20 Use Case BW

Cascaded Frequency Response plot for 5G NR (100 MHz) with Band DDC rate at 245.76 MSPS: Representation without decimation for a full view of the spectrum (the passband and stopband points are marked in Figure 155).

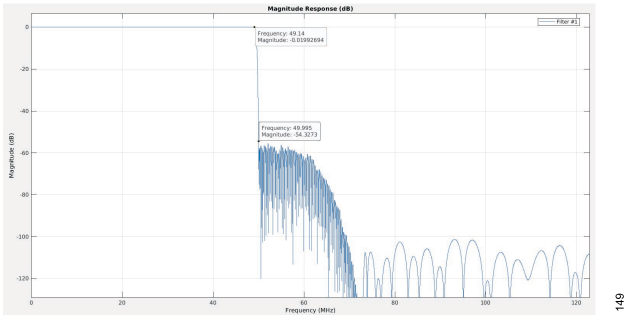


Figure 155. Combined Filter Response for 5G NR100 Use Case BW

NCO

A 48-bit Frequency Tuning Word is used in the NCO ensuring 100 dBFS SFDR, like Band DDC NCO. The Equation 24 is used for computation of FTW based on required downconversion.

$$\$signed(FTW[47:0]) \times nco\_clk\_freq/(2^{48}) \quad (24)$$

Two modes of FTW are supported, one a direct FTW feeding and the other through 1 kHz/1 ms clock where FTW is automatically calculated.

Further the 24 bit, 1 ms clock counter is used for coherent update of FTW after MCS ensuring that all channels maintain the same phase after the update (this is applicable for both the modes of FTW).

Simple Example:

A band at a rate of 245.76 MSPS, having two carriers:

- 100 MHz 5G NR whose final data rate is 122.88 MSPS.
- 20 MHz LTE whose final data rate is 30.72 MSPS.

The CDDC datapath for the two carriers is shown in Figure 156.

DIGITAL FILTER CONFIGURATION

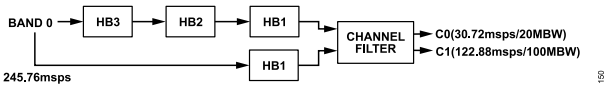


Figure 156. Two Carrier CDDC Configuration Example

Programming Requirements

- As per number of carriers, the GUI allows configuration of channel filter, bandwidth, and data rate for each subcarrier.
- Decimation required for each carrier (the GUI will automatically select the required HBs in datapath).
- Programming requirements of Channel filter is presented in next section.

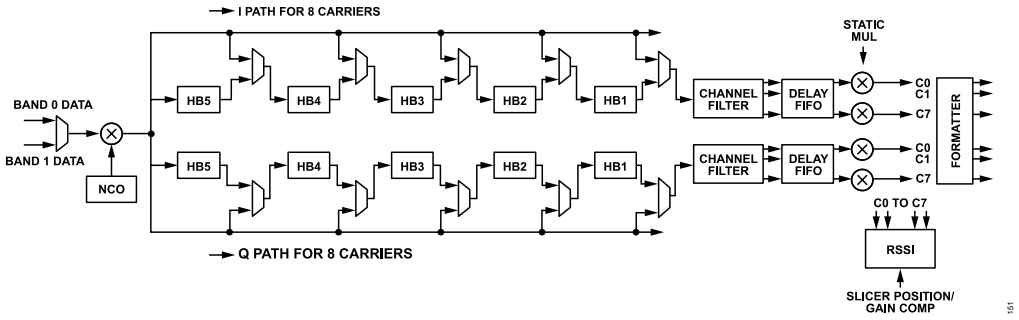


Figure 157. CDDC Block Diagram with Auxiliary Functions

Delay Buffer

Group delay of the carriers going through CDDC should be matched, though it could be of different rates. Due to rate difference and channel filter taps difference, the group delay will be different across carriers. Buffers are used to match the group delays across carriers and 8 instances of 256 sample deep memory are provided per channel, on both I path and Q path.

Each memory can be mapped to a carrier, making 1 to 1 mapping of 8 carriers and memories. Flexibility is provided to daisy chain up to four memories, which can act as a single delay element for one carrier. This is to provide higher delay for a high-speed carrier when it is mixed with slower carriers.

Gain/Slicer Propagation

In applications using gain compensation, digital gain is applied at Band DDC output to compensate for AFE attenuation. To avoid digital signal from going beyond 0 dB dynamic range, slicer operation is performed.

This slicer position/information is typically embedded in actual data and transferred over JESD to BBIC. This helps BBIC recover the data by applying reverse slicer operation and restore full dynamic range.

With CDDC stage coming after Band-DDC, the slicer information/position will be propagated through CDDC along with the data and finally to JESD. In this process when there is a gain change, slicer position also changes effectively resulting in difference of 1 dB/2 dB/4 dB/6 dB between successive samples and filters would operate across these samples having different gain.

Figure 158 shows the scheme used to propagate the slicer position.

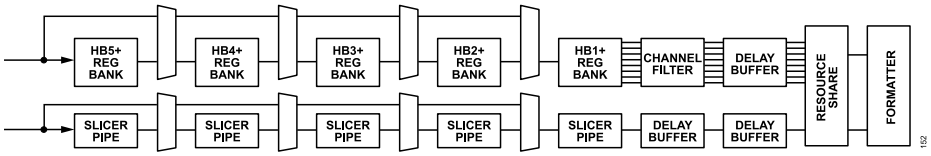


Figure 158. Slicer Info Propagation Through CDDC Filters

DIGITAL FILTER CONFIGURATION

Slicer position is propagated through the fastest carrier in CDDC, represented by slicer pipe in Figure 158. And finally, through delay buffers slicer position is matched with other carriers in the signal and sent out through formatter.

There is no special programming required for slicer propagation or slicer-delay matching, except selecting the required format in formatter.

RSSI

Receive Signal Strength Indicator (RSSI) is used to measure the signal strength for a programmable duration in both FDD and TDD modes. The RSSI measurement keeps states across RxON/RxOFF periods (particularly required in TDD mode). Band DDC prior to CDDC has this functionality per band, and the same feature is available per carrier at the channel filter output of the CDDC.

$$Rx\_RSSI = 10 \times \log_{10}\left(\frac{\Sigma_{i}^{i=N-1}\left(I_i^2 + Q_i^2\right)}{N}\right) - Gain \quad compensation \quad in \quad dB \tag{25}$$

The signal coming to CDDC is already gain compensated in Band DDC. Optionally it may carry slicer position along with the data which has to be applied in BBIC (Baseband IC) for getting full dynamic range. The CDDC RSSI by default provides the power seen at Band DDC input. Optionally it can also measure the power of the gain compensated signal seen by the BBIC. The option of enabling gain compensation is also available per carrier. The gain compensation calculation happens as follows:

- Default: Calculated value in dB − Slicer value in dB + Gain compensation in dB
- Gain compensation: Calculated value in dB − Slicer value in dB

Note:

- The Gain compensation in dB is directly used from Band RSSI.
- Since the input data to the CDDC RSSI already incorporates the gain compensation done after Band DDC, the calculation needs to be reset at gain change even when measuring raw power without compensation (Default mode).

Table 83 shows the slicer step size values used.

Table 83. Rx Slicer Step Size

Slicer Control	Slicer Step Size in dB
0	1
1	2
2	3
3	4
4	6
Default	8

Static Multiplier

A static multiplier is available in the path for the general fine control of the datapath gain/attenuation. The requirement was to provide 0.01 dB precision for a range of −96 dB to 36 dB in UL. A 23-bit multiplier is provided for each carrier path to cover the range, and this will be after the channel filter.

Eight set of registers are provided for 8 carriers. The default multiplication value for all the carrier is 1.

User must ensure that programmed value doesn't saturate the data. If there is a data saturation it will be indicated through GP-Interrupt to the host and as Arm M4 interrupt also.

Configurable Channel Filter

This section describes channel filter block that is used for filtering the individual carriers to meet the ACLR requirements. At the top level the block has a pool of multipliers and data banks that are allocated to different carriers as determined by their rates and tap requirements. There are two instances of this filter block one for I-channel of all the carriers and other for Q-channel.

The top features of the channel filter are listed as follows:

DIGITAL FILTER CONFIGURATION

- Support for maximum of 8 carriers.
- A total of 48 multipliers that is allocated to different carriers.
- A total of 1296 data pipe/sample registers that is allocated to different carriers.
- Support for symmetric and non-symmetric taps.
  - For non-symmetric taps effective number of data pipe registers are reduced by half to 648.
- Stop band attenuation spec is 50 dB for all the carriers (70 dB for LTE carries as per 3GPP spec/frequency offsets).
- The coefficient bit width is 16 for all the taps.
- Arriving at required of number of multipliers and data pipe registers are briefed in the following section.

Filter Coefficients

- Total number of filter taps available are 1296.
- 648 coefficients in total for all the carriers (symmetric case).
  - The coefficients are organized as 48 coefficient banks like data pipes.
- 48 Coefficient Banks are also distributed/allocated to carriers exactly like data samples.
  - Coefficients are programmed serially in the allotted banks.
- Coefficient bit width is 16-bit (1.15 fixed point format) to support stop band attenuation of 50 dB.
- Coefficients with smaller bit-widths would be normalized to 16 bits by scaling them up.
- The gain of the filter for each carrier is expected to be 0 dB.
- No hot swap of coefficients supported.
- Any over range occurring in the filter operation will be sent out as an interrupt to Arm and GPIO interrupt or in the JESD control channel.

CDDC Interleaver

The CDDC output comes out of channel filter as 8 separate carrier signals. Routing all these to JESD would be an unnecessary overhead since the maximum bandwidth of data to be sent to JESD will be less than or equal to hsdigclk frequency. Since the JESD interface to the CDDC supports a maximum frequency of hsdigclk, a single data bus is enough to transfer data from CDDC to JESD. The interleaver helps transfer the data from upto 16 separate channel filter outputs onto the single bus going to JESD.

Structure and Usage

The interleaver uses time-division multiplexing to transfer multiple sources on a single JESD line. To facilitate this, it supports up to 64 time slots. The list of bit fields is as follows:

- max_slot bitfield—determines the number of slots used for a given configuration
- slot_table registers—determines which source is assigned to which time slot
- slot_valid registers—determines whether a slot is valid or not
- init_slot bitfield—used to synchronize the jesd and cddc

The interleaver has a counter which down counts from max_slot to 0 repeatedly. At any given point of time the value of the counter determines the active slot number. This is used to determine the source to be transmitted using the slot table. The slot valid on the other hand uses the counter value to determine whether the current interleaver output is valid or not. This valid signal is sent from the cddc to the jesd to facilitate even better packing within the jesd where it can drop the invalid slots and send out only the valid data at a further reduced frequency (as compared to the cddc – jesd interface).

Table 84 shows an example slot table for explanation.

Table 84. Example JESD Slot Configuration with CDDC/CDUC Enabled

Carrier	Carrier Frequency (MHz)	Number of Samples Sent in Same Time as One Sample of Lowest Frequency Carrier
0	122.88	4
1	61.44	2
2	30.72	1

DIGITAL FILTER CONFIGURATION

Table 84. Example JESD Slot Configuration with CDDC/CDUC Enabled (Continued)

Carrier	Carrier Frequency (MHz)	Number of Samples Sent in Same Time as One Sample of Lowest Frequency Carrier
Sum	215.04	7

In Table 84, the jesd interface frequency needs to be more than or equal to 215.04 MHz to support sending out all the data. Hence the jesd_clk needs to be set up at 245.76 MHz which is the nearest possible clock frequency.

The max_slot value is determined such that only one sample of the base I/Q sample rate 30.72 MSPS is sent out. Hence max_slot = 245.76 MHz/30.72 MHz = 8

So the slot table can be created by using the following guiding factors:

- Total number of slots is 8
- Every alternate slot is for carrier 0
- Every 4th slot is for carrier 1
- Every 8th slot is for carrier 2

Note that the above criteria can be satisfied by multiple slot table configurations. Using any one of these is ok. One such slot table is Table 85.

Table 85. Slot Configuration

Carrier	Carrier 0	Carrier 1	Carrier 0	Carrier 2	Carrier 0	Carrier 1	Carrier 0	Invalid
Slot	7	6	5	4	3	2	1	0

The init_slot bitfield is used to determine the slot along with which to send a sync pulse to the jesd. This helps change the starting reference point from where the jesd should start looking at the data.

Upsampling/Downsampling

An optional rate upsampling is possible after channel filter for rate matching while interfacing with JESD. Example, a carrier at 30.72 MSPS rate can be up sampled to 61.44 MSPS. Upsampling can be controlled on individual carrier basis and supported ratios are in powers of 2 (up to 32).

Similarly, an optional rate downsampling block is provided before channel filter to skip the samples that are added in base band chip for carrier rate matching in JESD interface. Downsampling is available per carrier and can be controlled on individual carrier basis and supported ratios are in powers of 2 (up to 32).

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 I/Q 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. 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 159 shows the signal path for the Tx0, Tx1, Tx2, Tx3, Tx4, Tx5, Tx6, and Tx7 signal chain. Grayed-out blocks are not described in this section but in their relevant sections of the user guide.

DIGITAL FILTER CONFIGURATION

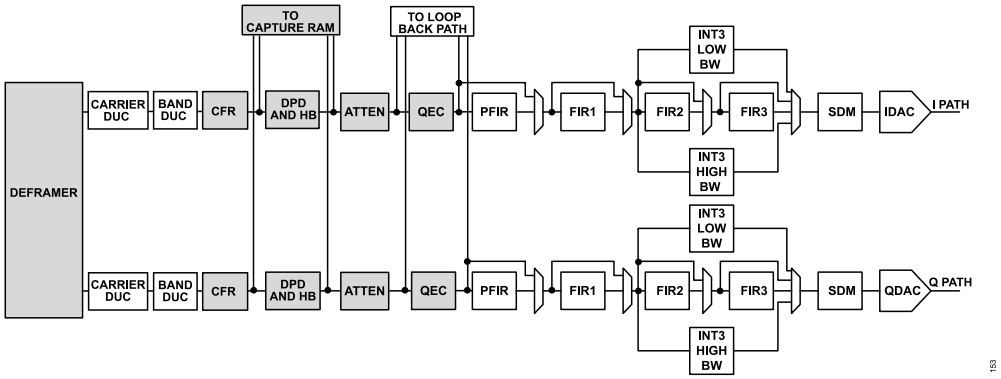


Figure 159. Tx Signal Path Diagram

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.

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,

DIGITAL FILTER CONFIGURATION

-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 (that is, 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 as follows:

- 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.

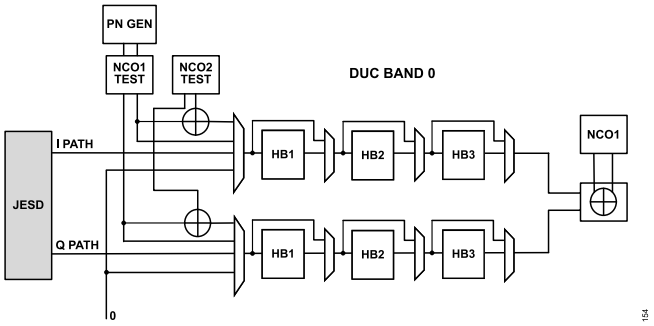


Figure 160. DUC Band 0

PN Generator (PN GEN)

The pseudo-random sequence generator can be muxed into the input of the Band0 DUC as shown in Figure 160 . This PN sequence generator can run up to 1 GHz.

Test NCO (NCO1 Test and NCO2 Test)

The two test NCOs in the digital upconvertor 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 FILTER CONFIGURATION

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.

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.

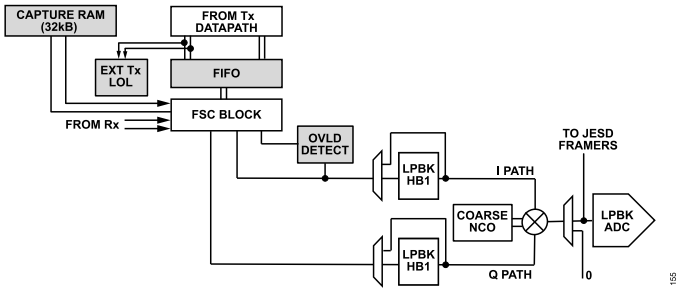


Figure 161. Tx Internal Loopback Path

Loopback Half-Band 1 Filter (LPBK HB1)

This filter is a decimating half-band filter with a stopband rejection of 80 dBc. 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.

DFE TSSI POWER MEASUREMENT

The Transmit Signal Strength Indicator (TSSI) power measurement is a function to measure the power going to the antenna. The signal power can be observed at different points on the Tx channel path. The power that is calculated by the TSSI block is given as:

$$TSSI\left(\text{dB}\right) = 10 \times \log_{10}\left(\frac{\sum_{i=0}^{i=N-1}\left(I_i^2 + Q_i^2\right)}{N}\right) \quad (26)$$

To configure the power measurement the API function **adi_adrv904x_DfePwrMtrTssiConfigSet()** is used. Tx DFE has two TSSI blocks for power measurement at various points along the CFR and DPD datapath blocks. The two meters are called TSSI_TX_1 and TSSI_TX_2, and the signals that can be observed are listed in Table 86.

Table 86. TSSI Measurement Tap-off Points

Meter	Source
TSSI_TX_1	Pre DPD HB1 Out

DIGITAL FILTER CONFIGURATION

Table 86. TSSI Measurement Tap-off Points (Continued)

Meter	Source
	Pre DPD HB2 Out after hard clipper
TSSI_TX_2	CFR In
	CFR Out
	Post DPD HB1 Out

The measurement duration for the power calculations is programmable and the two options are shown in Table 87.

Table 87. Measurement Duration for DFE TSSI Power Measurement Modes

Operation Mode	Power Measurement Duration	Description
1	$2^{duration0 + 4}$	In this mode of operation, the number of samples for measurement is a power of 2. The maximum value of duration that should be programmed is 27. The power calculations are accurate to 0.25 dB in this mode.
0	$2^{duration0} + 2^{duration1} + 2^{duration2} + 2^{duration3}$	In this mode of operation, the number of samples for measurement is not a power of 2 number. The total observation samples are a summation of up to 4 power of 2 numbers. The four durations should be programmed such that the total duration is less than 2^{28} samples and greater than 2^4. The accumulation happens for each power of 2 duration and are summed up to arrive at the final accumulated power. Each individual accumulation has to be weighted so as to get correct average power. This is done by setting the multiplication factor associated to the corresponding duration. The multiplying factor is a 0.10 format number. Not all four durations need to be used. The total number of active durations is determined by their corresponding multiplication factor. If a multiplication factor is set to 0, the corresponding duration is not used. However, it is required that the unused durations and multiplication factors that are 0 be at the higher index. For example: ▶ mulFactor0 = x, mulFactor1 = x, mulFactor2 = 0, mulFactor3 = 0 is valid. ▶ mulFactor0 = x, mulFactor1 = 0, mulFactor2 = x, mulFactor3 = x is not valid. The power calculations are accurate to 0.5 dB in this mode.

Table 88 shows the three different modes of operation which can be configured. The modes differ in when the first and subsequent measurements start

Table 88. DFE TSSI Power Measurement Modes

Measurement Mode	Name	Start of 1st Measurement	Start of Subsequent Measurements
0	FDD	When the measurement is enabled using **adi_adrv904x_DfePwrMtrTssiEnable()**.	At the end of the first measurement, a wait delay counter is started. The next measurement starts at the expiry of this counter.
1	TDD_MODE_1	At the rise of first TxOn after measurement is enabled and start delay counter expires. It is required that TSSI be enabled during Tx channel off period.	At the rise edge of next TxOn once the first measurement is completed and start delay counter expires.
2	TDD_MODE_2	At the rise of first TxOn after measurement is enabled and start delay counter expires. It is required that TSSI be enabled during Tx channel off period.	At the end of the first measurement, a wait delay counter is started. The next measurement starts at the expiry of this counter.

In TDD_MODE_1 and TDD_MODE_2, it is possible that the measurement does not complete during the TxOn period. In this case, after the measurement has started, when TxOn goes low, the measurement is halted. Once the TxOn goes high again, a counter with the period configured by **contDelay** is started. At the expiry of this counter, the power measurement resumes. Table 89 summarizes the delays used in the different modes.

Table 89. Delays for DFE TSSI Power Measurement Modes

Delay	Applicable Mode	Comments
startDelay	TDD_MODE_1 and TDD_MODE_2	This delay should be a non-zero value and should be set such that it is smaller than the TxOn period.

DIGITAL FILTER CONFIGURATION

Table 89. Delays for DFE TSSI Power Measurement Modes (Continued)

Delay	Applicable Mode	Comments
waitDelay	FDD and TDD_MODE_2	This delay should be a non-zero value. If TxOn goes low during wait delay, this counter is reset and wait delay is skipped.
contDelay	TDD_MODE_1 and TDD_MODE_2	This delay should be a non-zero value and should be set such that it is smaller than the TxOn period.

If during the measurement the Tx attenuation changes, the measurement is not normally affected. However, if it is required to reset the measurement and start a new measurement at gain changes, it can be done by setting **resetAtGainChange** to 1. If the gain change occurs within 4 cycles of the end of the measurement duration, the TSSI measurement is not reset.

Table 90 describes the behavior when a gain change occurs during different time periods when **resetAtGainChange** is set to 1.

Table 90. Behavior of DFE TSSI Power Measurement Modes Under Gain Change

Mode	Time When Gain Change Occurs	Behavior
FDD	Measurement period	Current measurement is abandoned, next measurement starts immediately
	Wait delay period	Next measurement starts at the end of wait delay
TDD_MODE_1	Measurement period	Current measurement is abandoned, next measurement starts at next TxOn
	Start delay period	Next measurement starts at the end of start delay
	Continue delay period	Current measurement is abandoned, next measurement starts at next TxOn
TDD_MODE_2	Measurement period	Current measurement is abandoned, next measurement starts after wait delay
	Start delay period	Next measurement starts at the end of start delay
	Continue delay period	Current measurement is abandoned, next measurement starts after gain change delay
	Wait delay period	Next measurement starts at the end of wait delay

In FDD mode, it is possible to start the measurement via a GPIO pin. To enable this mode, **fddPinMode** should be set to 1. When the selected GPIO pin is asserted, a start delay counter is started. At the expiry of this counter, TSSI measurement begins.

Figure 162, Figure 163, Figure 164, and Figure 165 detail the start, pause, and restart of measurement in the different modes.

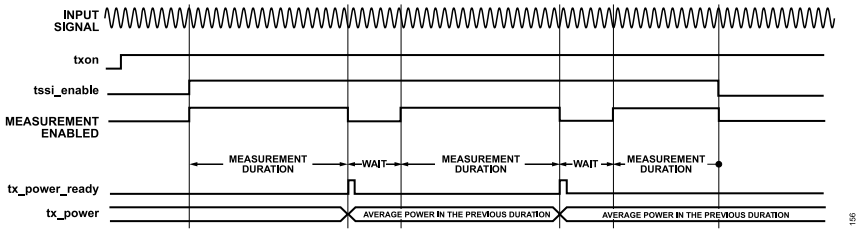


Figure 162. TSSI FDD Mode

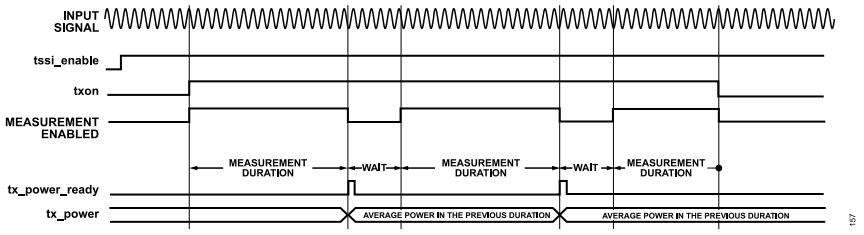


Figure 163. TSSI FDD Pin Mode

DIGITAL FILTER CONFIGURATION

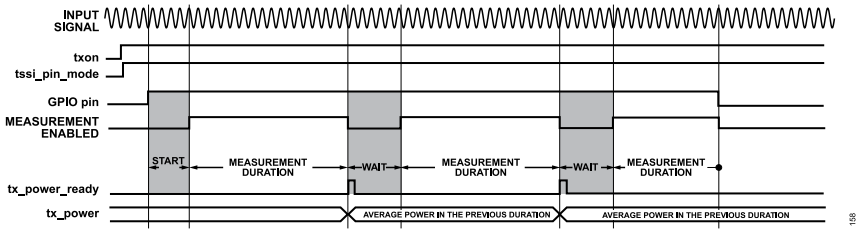


Figure 164. TSSI TDD Mode 1

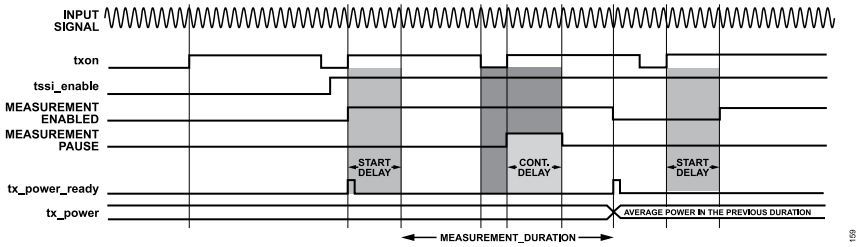


Figure 165. TSSI TDD Mode 2

The measurement is enabled by API **adi_adrv904x_DfePwrMtrTssiEnable()** and the results are retrieved using API **adi_adrv904x_DfePwrMtrTssiReadBack()**.

DPD Power Measurement

There are two power measurement blocks dedicated to DPD block: one at the input of DPD observing output of Pre-DPD HB1 and the other at the output observing Post-DPD HB1 output. The power measurements for those two can be configured using APIs **adi_adrv904x_DfePwrMtrDpdInConfigSet()** and **adi_adrv904x_DfePwrMtrDpdOutConfigSet()**, respectively.

The duration over which the measurements occurs is controlled by **dpdInOutPwrMeasDuration**. The number of samples over which measurement occurs is given by:

$$No.\ of\ Samples = 2^{dpdInOutPwrMeasDuration} \quad (27)$$

Table 91 shows two modes of operation.

Table 91. DPD Power Measurement modes

Mode	dpdInOutPeakToPowerMode	Description
Power measurement mode	0	In this mode, the average power is calculated for a given averaging duration.
Peak to power mode	1	In this mode, in addition to the average power for a given averaging duration, the largest ($I^2 + Q^2$) is also available for readback.

The measurement is enabled by API **adi_adrv904x_DfePwrMtrDpdInEnable()** and **adi_adrv904x_DfePwrMtrDpdOutEnable()**. The results of the power measurement are available for readback using the API **adi_adrv904x_DfePwrMtrDpdInOutReadBack()**.

TX DATAPATH API FUNCTIONS

Table 92. Tx Datapath API Functions

API Method Name	Comments
adi_adrv904x_TxTestToneSet()	Sets the specified Tx Test NCO to the given frequency, phase, attenuation, and enable state.
adi_adrv904x_TxTestToneGet()	Reads the specified Tx Test NCO's frequency, phase, attenuation, and enable state.
adi_adrv904x_TxNcoShifterSet()	Sets Tx NCO.
adi_adrv904x_TxNcoShifterGet()	Reads Tx NCO settings.

CARRIER DUC

Figure 166 displays a high-level flow on Tx, where Carrier DUC and Band DUC are displayed.

DIGITAL FILTER CONFIGURATION

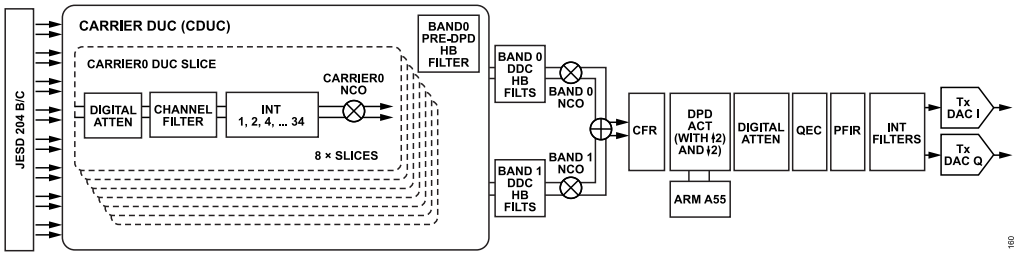


Figure 166. DUC High Level Block Diagram

Carrier DUC is fed by JESD/JRX taking in multiple carriers from ASIC/FPGA for transmission. It can take a maximum of 8 carriers that can be spread across two bands. Channel filtering is done first to meet the ACLR requirements, followed up series of Half Band filters for interpolation. The Mixers/NCOs then moves the carrier spectrum to appropriate regions, and they are summed up to form one or two bands.

Figure 167 captures the top-level architecture of Carrier DUC.

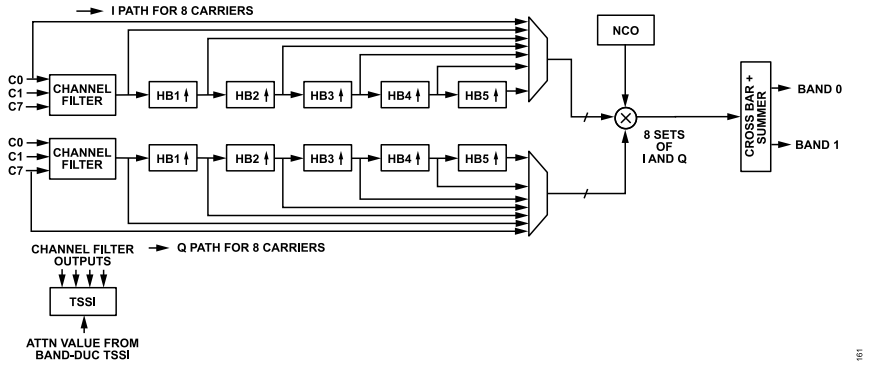


Figure 167. CDUC Top-Level Block Diagram

Salient Features of CDUC

- The maximum number of carriers supported is 8 and it can be distributed across two bands in any kind of split.
- Channel Filter performs only filtering operation and doesn't do any interpolation.
  - The features of channel filter are exactly same as in CDDC.
  - Any carrier combination (≤8) which can be filtered using 48 multipliers as explained in the Configurable Channel Filter section are supported.
- An optional down sampler precedes channel filter to ignore the samples that might have been added in the base band chip for rate matching.
- Interpolation of 1, 2, 4, 8, 16, 32 are supported as shown in Figure 167.
  - They can be different for different carriers.
- Output sample rate post HB interpolators would be sub-multiples of 245.76 MSPS.
  - The maximum expected rate at output of CDUC post-summation is 491.52 MSPS for each band. Other sub-multiple rates like 122.88 MSPS and 61.44 MSPS are supported.
  - A single band rate of 491.52 MSPS is also supported to cover FR2 cases (bypass CDUC CDDC).
- All the blocks are optional/bypass-able in the path.
  - The unused carriers can be turned off.

Overall CDUC Specification

Table 93 captures the top requirements of LTE and 5G carriers for filtering operation.

DIGITAL FILTER CONFIGURATION

Table 93. System Requirements for Carrier Filtering (CDUC)

Parameter	LTE	5G NR	Note
Fpass	0.9 × BW/2	0.983 × BW/2	Passband edge
Fstop	BW/2	BW/2	Stopband edge
Apass	±0.1 dB	±0.1 dB	Passband ripple
Astop	50 dB	50 dB/70 dB	Stopband rejection

Based on these requirements, the following specs (see Table 94) are used for filters. 40.7% is chosen for 50 MHz carrier running at 61.4 MSPS. This is the widest pass band possible through Half Bands.

Table 94. ADRV904x Carrier DDC Specifications

Filters	Normalized BW w.r.t. Input Sampling Rate/2	Stop Band Rejection	Passband Ripple
HB5	2.545%	90 dB	0.01 dB
HB4	5.09%	90 dB	0.01 dB
HB3	10.18%	90 dB	0.01 dB
HB2	20.35%	90 dB	0.01 dB
HB1	40.7%	90 dB	0.01 dB
Channel Filter	Programmable	50 dB/70 dB	0.05 dB

NCO

- A 48-bit Frequency Tuning Word along with 20-bit lookup table is used in the NCO ensuring 100 dBFS SFDR, similar to Band DDC NCO. The below formula is used for computation of FTW based on required upconversion:

$$\$signed(FTW[47:0]) \times nco\_clk\_freq/(2^{48}). \quad (28)$$

- Two modes of FTW are supported, one a direct FTW feeding and the other through 1 KHz/1 ms clock where FTW is automatically calculated.
  - Further the 24-bit, 1 ms clock counter is used for coherent update of FTW after MCS ensuring that all channels maintain the same phase after the update (this is applicable for both the modes of FTW).
- Couple of programmable phase offset registers are available.
  - There is no dynamic update of phase offset unlike Band DUC NCO which gets it offset from attenuator.

Simple Example

Two input carriers from base band to be transmitted: a 100 MHz 5G NR and 20 M LTE, at Band rate of 245.76 MSPS.

Figure 168 shows the CDUC datapath for the two carriers.

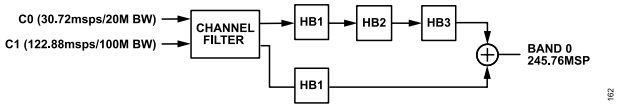


Figure 168. Two Carrier CDUC Configuration Example

Programming Requirements

- Number of carriers that are to be aggregated in band0 and band1.
- Programming requirements of Channel filter as covered in the NCO section.
- Down sampling if required for each carrier.
- Interpolation required for each carrier. (This will automatically select the required HBs in data path).
- The NCO frequency tuning word for each carrier to move them up from baseband.
- Baseband data rate that is required/expected post-aggregation.

DIGITAL FILTER CONFIGURATION

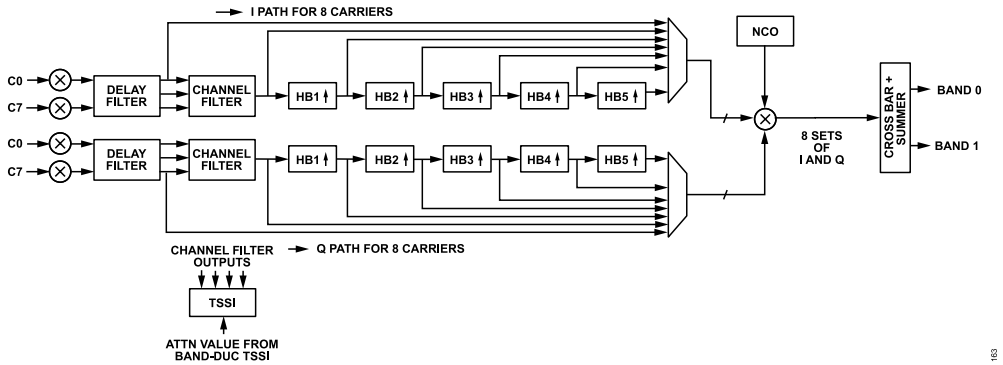


Figure 169. CDUC Top-Level Block Diagram with Auxiliary Functions

Static Multiplier

A static multiplier is available in the path for the general fine control of the data path gain/attenuation. The requirement was to provide 0.01 dB precision for a range of −90 dB to +36 dB in UL. A 23-bit multiplier is provided for each carrier path to cover the range, and this will be after the channel filter. Two registers are provided to enter the required gain in dB scale or in absolute fixed-point scale. Firmware can read the dB value and convert it back to fixed point format and write it, or directly user can write the absolute value.

The write to absolute scale register triggers the value written in the register to be used for gain multiplication.

Eight set of registers are provided for 8 carriers. The default multiplication value for all the carrier is 1.

Note that there is no slicing operation performed on this gain multiplication, hence user must ensure that programmed value doesn't saturate the data. If there is a data saturation it will be indicated through GP-Interrupt to the host and as Arm M4 interrupt also.

Static multiplier gain step size resolution is shown in Figure 170.

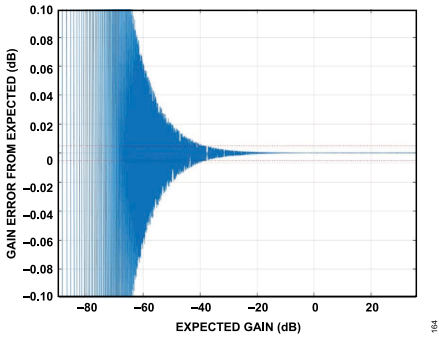


Figure 170. CDUC Static Multiplier Gain Step Resolution

CDUC Flush

In order to output a known data from the CDUC filters during the TXON phase, a mux is provided at the output of CDUC which sends out zero data until the filter pipeline is flushed (this depends on specific configuration of HB and channel filter which decides the largest carrier latency/pipeline delay).

A programmable 16-bit register (CDUC flush max delay) is provided for this purpose, which would be programmed by the configurator.

A configurable bit cduc_flush_enable is provided to enable this feature.

The output data from the mux is zero from the time cduc_flush_enable goes high until the counter expires after reaching the programmed CDUC flush max delay. After the counter expiry, the mux output contains the valid data coming out of the filters.

DIGITAL FILTER CONFIGURATION

CDUC Deinterleaver

Like the Interleaver present in CDDC, the CDUC requires a deinterleaver. This helps transfer the data from a single jesd channel to multiple channel filter inputs. It uses the same structure as the interleaver. The only difference is that the jesd data is broadcast to all the sources and the counter value is used to determine which carrier gets the data valid signal.

All the deinterleaver outputs carry the exact same data. The data_valid is used to determine which data corresponds to which carrier. This is shown in Figure 171. The green data goes to the carrier, and the yellow data is ignored.

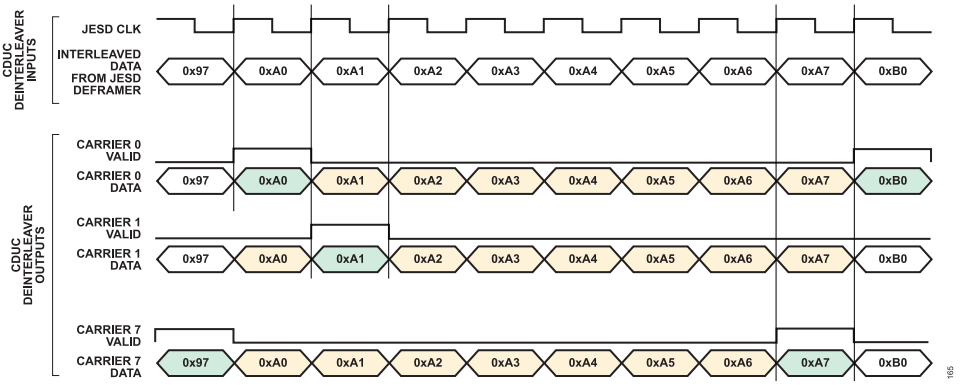


Figure 171. CDUC Deinterleaver Timing Diagram

The init_slot in the deinterleaver is used to determine which slot the jesd is sending the data for when it sends a sync pulse. This helps to synchronize the jesd and CDUC if required.

Upsampling

The following are the two ways to do upsampling:

- Upsample the complete frame
- Upsample individual carriers

To achieve upsampling of the complete frame, do the following:

1. Follow the normal guidelines given earlier to calculate the values of interleaver yoda registers.
2. Increase the jesd clock by the upsampling factor.

To upsample individual carrier, do the following:

1. Consider the upsampled frequency for the required carrier (for example, if a 7.68 MHz carrier is to be upsampled to 30.72 MHz then consider it to be a 30.72 MHz carrier instead of 7.68 MHz).
2. Follow the normal guidelines given earlier to calculate the values of interleaver yoda registers.
3. Run jesd clock at the frequency calculated using normal guidelines.

Downsampling

Downsampling is attained naturally through the normal programming guidelines for the deinterleaver. Since the slot table is programmed to capture only the required number of samples for each carrier, the only change required is to use the actual jesd clock frequency (instead of the supported frequency just higher than the sum of carrier frequencies) for calculating max_slot.

$$max\ \ slot = \frac{actual\ \ jesd\ \ frequency}{lowest\ \ carrier\ \ frequency} \quad (29)$$

Note that maximum slots supported by the design is 64; therefore, jesd clock to lowest carrier frequency ratio cannot be more than 64.

DIGITAL FILTER CONFIGURATION

TSSI

Transmit Signal Strength Indicator is used to measure the signal strength for a programmable duration in both FDD and TDD modes. The TSSI measurement keeps states across TxON/TxOFF periods (particularly required in TDD mode). Band DUC post to CDUC has this functionality per band, and the same feature is available per carrier in CDUC, just after channel filter outputs.

$$Tx\_TSSI = 10 \times \log_{10}\left(\frac{\Sigma_{i}^{i=N-1}\left(I_i^2 + Q_i^2\right)}{N}\right) \quad (30)$$

Signal attenuator is present after Band DUC for PA protection. Therefore, in the CDUC TSSI measurement, the down-stream attenuation is optionally compensated. Note that attenuation value to be compensated is directly taken from Band DUC TSSI and applied.

Upon attenuation change, the measurement gets reset (optionally) and new measurement resumes after 1 cycle delay (similar to Band TSSI) and as per above three modes. (This is because we ideally need a negative delay from attenuation change).

All delay counter widths are retained same as Band TSSI. The consolidated list is in Table 95.

Table 95. Delay Counter Width

Delay Counter	Counter Width
start_delay	13 (Same as TX DFE TSSI)
continue_delay	14 (Same as TX DFE TSSI)
wait_delay	24 (Same as TX DFE TSSI)

Note that since the Band TSSI now has filters before it, its usage may require some change.

Maximum measurement duration supported is 2^{28} cycles of carrier frequency (same as TX DFE TSSI).

GPIO indication of measurement ready is available per carrier for each channel. Typically, the expectation is all LTE carriers should have similar measurement duration and 5G carriers should have similar measurement duration. Therefore, it may be enough to get the ready indication for one carrier per LTE and 5G instead of all carriers. These GPIO indications are multiplexed along with TX control-out mux cross bar.

Similar to Band DUC TSSI, here too GPIO based measurement trigger is available which is particularly used for FDD mode. Here the input GPIO trigger will be common for all triggering measurement of all the carriers. All the control registers are available per carrier (except the attenuation compensation registers, as the value is directly used from Band DUC TSSI). From a user perspective, it can be considered there are 8 separate TSSI, one for each carrier. The power measurements are in dB scale, 7.2 format. (In steps of 0.25 dB) (same as Band TSSI). The power measurement is also available in linear scale in 0.36 format.

The gain compensation (same as the value used by Band TSSI) is also available for readback. The use of this is described in Equation 30. This should be used in the same manner even when the linear to db conversion for the calculated power number is done in the software.

CDUC Attenuation

To ensure that the aggregation of carriers doesn't cause overflow at the CDUC output, attenuators are provided to scale the carrier levels down after the filtering stages.

An 8-bit attenuator value is provided for this purpose, this is taken to be Q0.8 fixed point format value and multiplied with the summed carrier. This can provide 256 attenuation levels in the range of TBD dB (for an attenuator value of 255) to TBD dB (for an attenuator value of 0).

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.

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.

DIGITAL FILTER CONFIGURATION

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 172 shows the signal path for an ORx0 and ORx1 signal chain. Blocks that are not discussed in this section are grayed out.

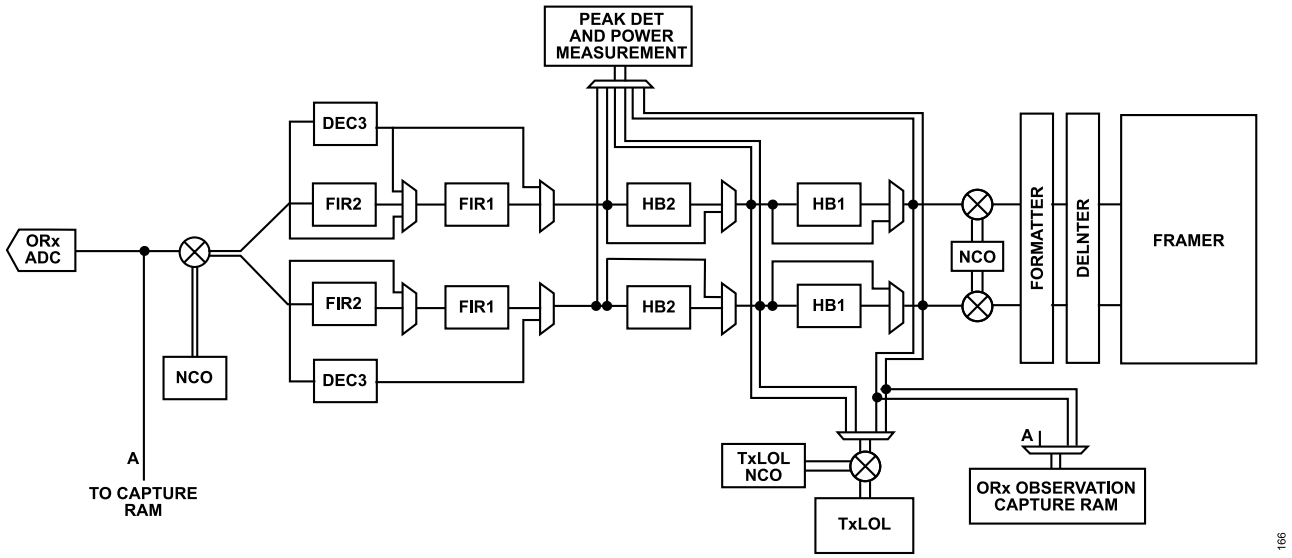


Figure 172. ORx Signal Path

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 173. 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 173, 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 173. Once the filtering is complete, (d) in Figure 173, the datapath uses the NCO at the end of the datapath to shift the wanted signal so that it is back centered around DC.

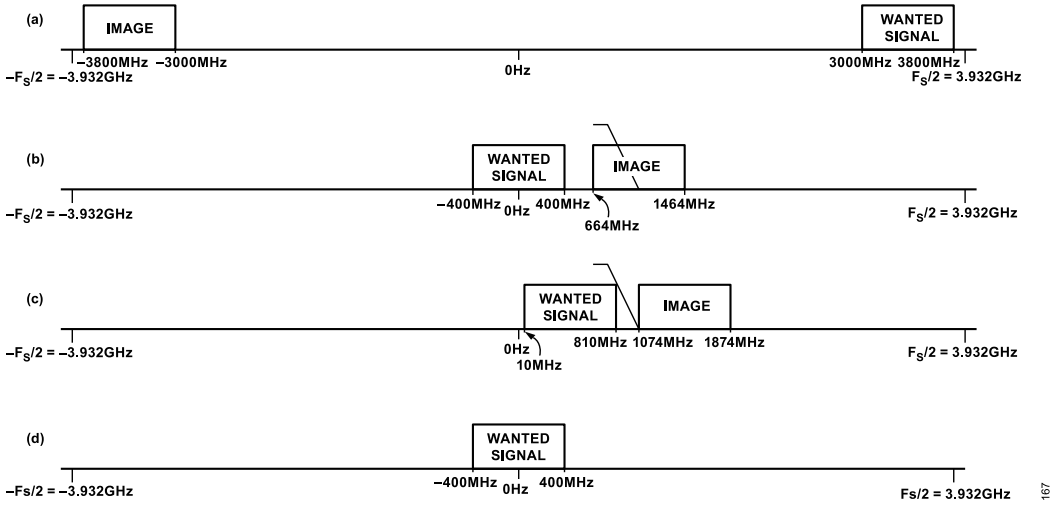


Figure 173. ORx Datapath NCO and Filtering Methodology

DIGITAL FILTER CONFIGURATION

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.

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 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]

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.

DIGITAL FILTER CONFIGURATION

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 96. List of ORx Datapath API Functions

API Method Name	Comments
adi_adrv904x_RxOrxDataCaptureStart()	Captures the ORx data in an internal RAM. Use if JESD link is not brought up.
adi_adrv904x_OrxNcoSet()	Sets ORx NCO.
adi_adrv904x_OrxNcoGet()	Reads ORx NCO settings.
adi_adrv904x_OrxNcoFreqCalculate	This function calculates the ORx NCO frequency shifts, given the Tx Synthesis Bandwidth limits.

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. The ADRV904x 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 166, NCO1 is the Tx DUC NCO that can be used to convert the desired bands at the appropriate frequency, the user can call adi_adrv904x_TxNcoShifterSet() to change Tx NCO frequency at any time.

RX DDC NCO

As shown in Figure 152, Rx DDC NCO can be used to convert the desired bands at the appropriate frequency, the user can take the following procedure to change its frequency:

1. Disable Rx QEC tracking if it is running.
2. Call adi_adrv904x_RxNcoShifterSet() to change Rx DDC NCO frequency.
3. Enable Rx QEC tracking.

ORx NCO

On ORx channel as shown in Figure 172, 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_adrv904x_OrxNcoSet().

DIGITAL PRE-DISTORTION (DPD)

This section provides an overview of the digital pre-distortion (DPD) function provided in the transceiver, including a hardware architecture overview, an overview of the DPD algorithm with references to features that enhance DPD robustness and performance in dynamic signaling conditions, and a summary of API software interface functions to configure these DPD functions. DPD enables users to achieve higher power amplifier (PA) efficiency by extending the linear operating region of the power amplifier, while still meeting adjacent channel leakage ratio (ACLR) requirements in the Tx signal chain for compliance with 3GPP and European Telecommunications Standards Institute (ETSI) standards for 5GNR, LTE and other technologies. The DPD function in the transceiver supports a carrier bandwidth of up to 400 MHz.

DIGITAL FRONT-END SYSTEM LEVEL OVERVIEW

The ADRV904x DFE signal chain context diagram is shown in Figure 174. The ADRV904x DFE Subsystem includes the following:

- DUC. The JESD Tx data from the baseband is fed into the Carrier Digital Up-Converter (CDUC) which can be used to shift and combine up to 8 individual carriers across 2 bands through a combination of mixers/NCOs and half band filters. A Band-Digital Up-Converter (BDUC) at the output of the Carrier Digital Up-Converter can be used to up-convert the combined band output of the CDUC.
- CFR. A high performance CFR is placed at the output of Band Digital Up-Converter. The CFR can be used to reduce the Peak-To-Average Power Ratio of the Tx signal, enabling the Power Amplifier to operate at a higher efficiency.
- DPD. The Digital Pre-Distortion engine includes a user configurable actuator for pre-distorting the Tx baseband data along with capture buffers for observing the Power Amplifier output in order to generate the pre-distortion coefficients. Each transmit channel has its own DPD actuator that can be independently configured. The DPD engine also has two half band filters at the input which can be used to interpolate the transmit data input to the DPD actuator.
- DFE Processor. The DFE processor is an Arm A55 quad-core processor for implementing DFE application software such as DPD, CLGC, and VSWR algorithms. The DFE processor which wraps the hardware drivers and utility functions such as data capture and DPD actuator update are contained in the DFE API software.

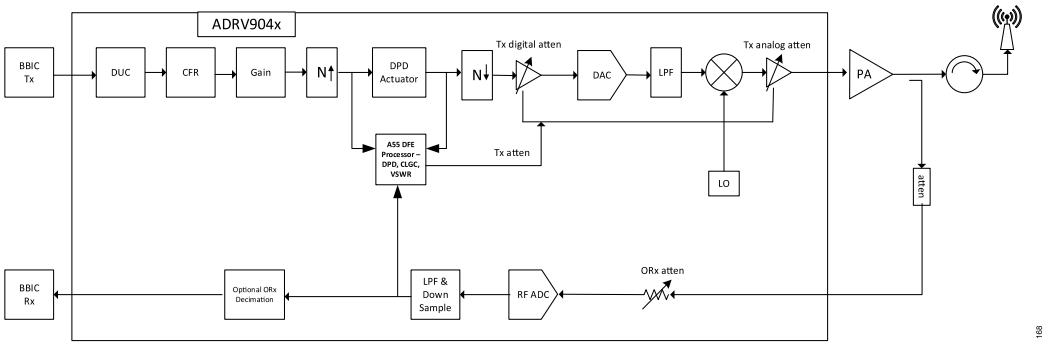


Figure 174. ADRV904x Signal Chain with DFE Processing Blocks Highlighted

DPD INTRODUCTION AND PRINCIPLE OF OPERATION

DPD is a technique for improving the linearity of a non-linear system such as a power amplifier by introducing precise anti-distortion into the input waveform that compensates for the PA's in-band nonlinear products. DPD works on the principle of pre-distorting the transmit data in the digital domain to cancel the distortion caused by PA compression in the analog domain. In this way, DPD can improve PA power-added efficiency by double or more, allowing the PA to be pushed further into saturation while maintaining linearity requirements.

A baseband model of the power amplifier is created and trained on the input and output digital-baseband samples that pass through the PA as shown in Figure 175. The pre-distorter then applies an inverse of the PA model function to input samples before passing them to the transmitter output. The cascade of the pre-distorter response and the PA response becomes a nearly linear system.

DIGITAL PRE-DISTORTION (DPD)

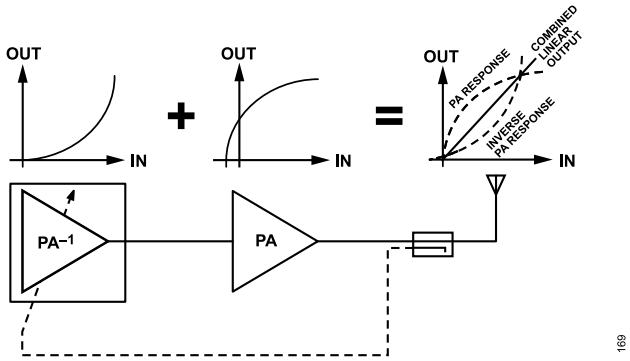


Figure 175. Concept of Digital Pre-Distortion for Linearizing the PA Response

The intermodulation distortion products between various subcarriers due to PA non-linearities in a wideband transmission protocol such as LTE/NR manifest as power leaked into adjacent channels. Adjacent Channel Leakage Ratio (ACLR) is defined as the ratio of the transmitted power on the assigned channel to the power leaked in the adjacent radio channel. The ACLR performance improvement following the application of DPD to the baseband data is captured in Figure 176. These plots illustrate how the out of band non-linearities due to intermodulation products of an LTE 20 MHz signal are reduced by 15-20 dB after the application of DPD.

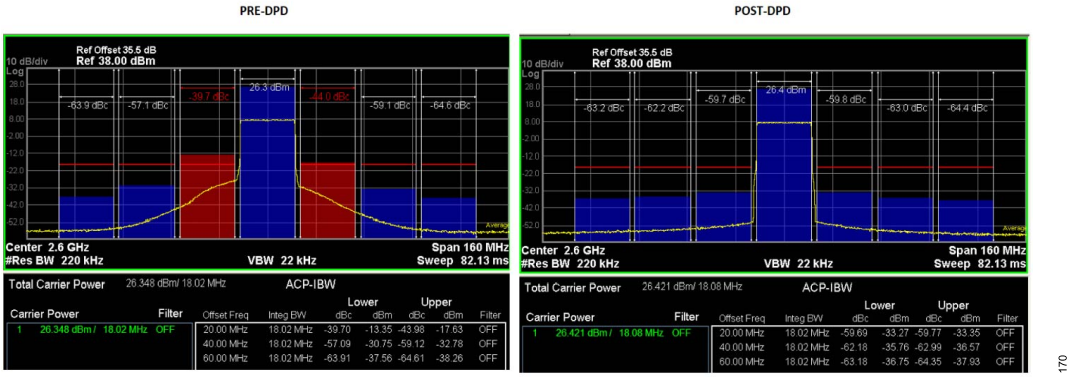


Figure 176. Power Spectral Density Showing Improvement in ACLR After Application of DPD for a 20 MHz LTE Signal

TRANSCEIVER DPD OVERVIEW

The DPD feature on this transceiver enables users to offload power amplifier linearization tasks from the baseband processor to the transceiver. With DPD implemented on the transceiver, the user does not need to allocate JESD serializer/de-serializer resources for observing power amplifier feedback data through the ORx channels, resulting in significant system power savings. Interpolators leading to the DPD actuator allow the baseband processor to transmit data at a lower rate on the JESD204B/JESD204C link than is needed for the full DPD correction bandwidth. The lower data rate at JESD translates directly into power savings and lower no. of lanes. Integration of the DPD into the transceiver chip results in significant system level cost, space, and power savings when compared to conventional FPGA/ASIC based implementations.

A simplified block diagram of the transceiver DPD system is shown in Figure 177. A brief description of the individual blocks is as follows:

- Transmit Datapath. The digital baseband signal from the JESD de-framer output goes through an optional Crest Factor Reduction (CFR) block for reduction of the overall peak to average power ratio (PAPR) of the signal followed by a digital interpolation filter which interpolates the baseband signal by a factor of 1x, 2x or 4x for analyzing the baseband signal over the DPD analysis bandwidth. The inverse PA model is applied by the DPD engine, followed by decimating half band filter to bring the data rate to <1GHz for the rest of the transmit signal chain including digital to analog conversion, up conversion by a mixer before the signal is fed into the actual amplifier.
- Observation Datapath. The DPD algorithm relies on observing the non-linearities via a feedback path. The feedback path is realized using an integrated observation receiver (ORx). The PA output data is sampled through the observation receiver, down converted and digitized for further analysis by the firmware.
- DPD Processing . The DPD engine is based on an abbreviated implementation of generalized memory polynomial (GMP) and Dynamic Deviation Reduction (DDR) that are generalized subsets of the well-known Volterra series. The simplified polynomials model many PA

DIGITAL PRE-DISTORTION (DPD)

characteristics such as weak nonlinearities, temperature variation, and memory effects. The inverse PA model is applied on the interpolated digital baseband samples through DPD actuator hardware. A dedicated embedded Arm processor (Arm-A55) is used for computation of the GMP coefficients.

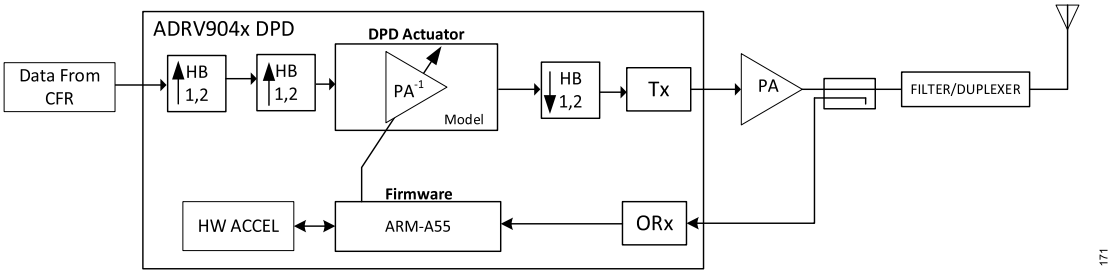


Figure 177. Simplified Block Diagram of Transceiver DPD

DPD Actuator Overview

The DPD actuator is a programmable polynomial calculator. Supported polynomials are as follows:

- Generalized Memory Polynomial (GMP)
- GMP with Dynamic Deviation Reduction (DDR)

GMP Mode

The DPD actuator implements a programmable GMP calculator captured in Equation 31.

$$y_{GMP}(n) = \sum_{i=0}^{j<16} \sum_{i=0}^{i<16} \sum_{k} c_{i,j,k} \quad P_k(|x(n-i)|)x(n-j) \tag{31}$$

To compensate for memory effects in a large bandwidth signal, a higher order polynomial is required. The DPD actuator can be programmed to support up to 255 coefficients for wide bandwidth signals. The structure of the DPD actuator is shown in Figure 178. For every pre-distorted output, the GMP model calculates the Sum of Product expression. The product terms consists of a modeling coefficient $c_{i,j,k}$ magnitude power basis function P_k (|x(n-i)|), and cross memory term x(n-j). Each DPD model consisting of GMP terms can utilize up to 50 look-up-tables (LUT) on one specified bank. Refer to the GMP Model and Look Up Table section for more information on the mapping GMP terms to LUTs.

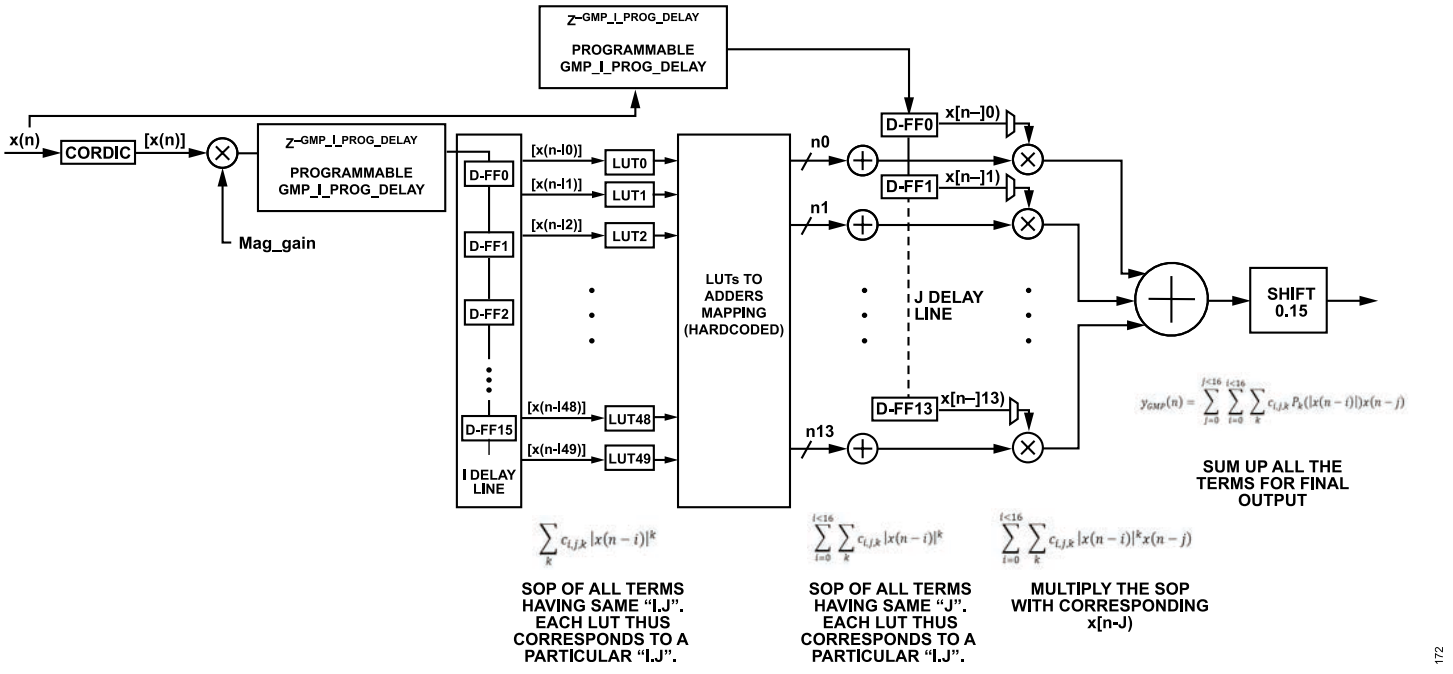


Figure 178. ADRV904x DPD Actuator Functional Diagram

DIGITAL PRE-DISTORTION (DPD)

GMP + DDR Mode

The DPD actuator implements also implements a programmable GMP + DDR calculator captured in Equation 32.

$$y_{GMP}\left(n\right) = \sum_{i=0}^{i<16}\sum_{j=0}^{j<14}\sum_{k} c_{i,j,k}\ \ P_k(|x(n-i)|)x(n-j) + \sum_{i=0}^{i<3}\sum_{j=0}^{j<3}\sum_{k} c_{i,j,k}\ \ P_k(|x(n-i)|)x(n-j) + \sum_{j=1}^{6}\sum_{k} d_{j,k}\ \ P_k(|x(n)|)x^2(n)x\times(n-j) + \sum_{j=1}^{6}\sum_{k} e_{j,k}\ \ P_k(|x(n)|)x(n)|x(n-j)|^2 + \sum_{j=1}^{6}\sum_{k} f_{j,k}\ \ P_k(|x(n)|)x\times(n)x^2(n-j) \quad (32)$$

The first term in this equation is the from Equation 31. The second term in this equation is an additional optional GMP polynomial with fewer number of terms. The remaining 3 terms are the DDR2 terms, referred to as DDR7, DDR8, and DDR9 terms henceforth. The DDR terms consists of a modeling coefficient $d_{j,k}$, $e_{j,k}$, $f_{j,k}$.

DPD Half Band Filters

There are two Half Band filters that could be enabled based on the input data rate and the desired DPD actuator rate. The half band filters have the passband-edge and stopband-edge frequencies are equidistant from the half band frequency Fs/4.

Each DPD half-band filter provides either 1× or 2× interpolation rate. A maximum of 4× interpolation can be achieved by cascading two DPD HB filters. DPD half-band 1 supports a bandwidth of approximately 82% with respect to input data rate. For example, DPD-HB1 supports a 400 MHz Bandwidth signal at a 491.52 MSPS input rate. DPD half-band 2 supports a bandwidth of approximately 41% with respect to input data rate. For example, DPD-HB2 supports a 400 MHz bandwidth signal at a 983.04 MSPS input rate. These two characteristics of half band filters can be seen in Figure 179 (2× interpolation) and Figure 180 (4× interpolation).

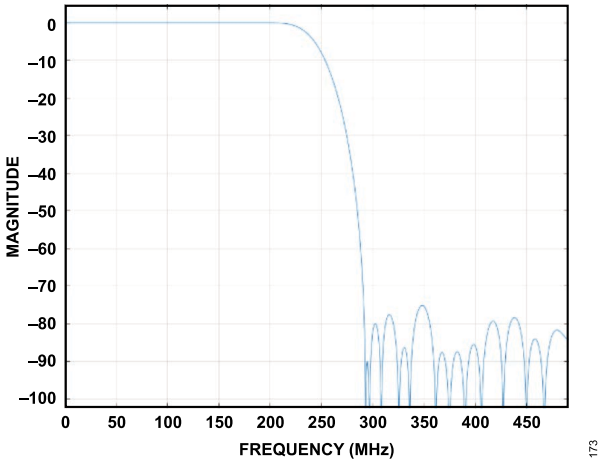


Figure 179. Response for Half Band 1 Enabled (x2): 82% of Fs

DIGITAL PRE-DISTORTION (DPD)

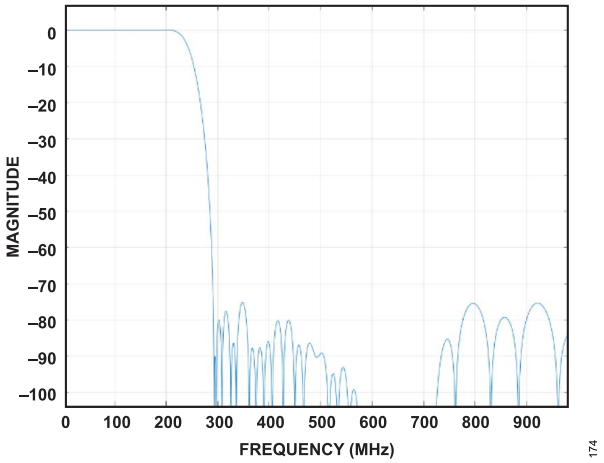


Figure 180. Response for Both Half Band 1 and 2 Enabled (x4)

Post DPD Half Band Decimation Filters

The Half Band decimation filter that can be enabled based on the output data rate of the DPD actuator. The DPD half-band filter provides either 1× or 2× decimation rate. The filter supports a bandwidth of approximately 86% with respect to input data rate. These characteristics of half band decimation filter can be seen in Figure 181.

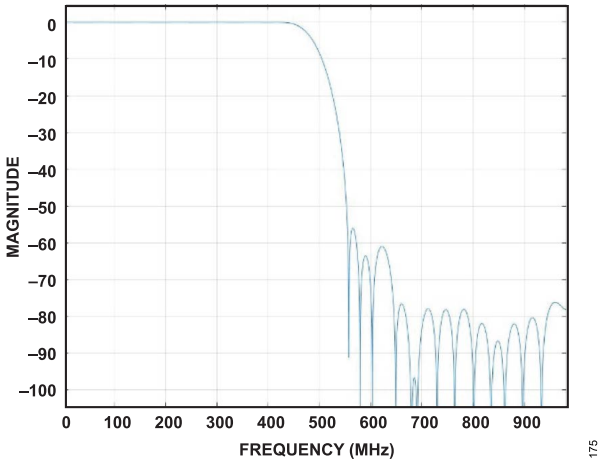


Figure 181. Response for Both Half Band 1 and 2 Enabled (x4)

DPD ALGORITHM OVERVIEW

The ADRV904x DPD algorithm supports both indirect learning and direct learning DPD mechanisms for extracting DPD model coefficients. The details of direct and indirect DPD learning mechanisms are provided in the following sections.

The user can configure the transceiver DPD learning algorithm through the adi_adrv904x_DpdTrackingConfigSet() API using the settings listed in Table 97.

Table 97. DPD Direct Learning Setting

adi_adrv904x_DfeAppCalDpdTrackCfg.forceDirect	DPD Learning Mechanism Selected
0	Indirect Learning
1	Direct Learning

DIGITAL PRE-DISTORTION (DPD)

Indirect Learning

Indirect learning involves using the observation receiver data (PA output data) as a reference for predicting the input samples corresponding to the reference. The function used for predicting the input samples is known as the inverse PA model. Once the prediction of input samples corresponding to the observed data is good, the estimated inverse PA model is used to pre-distort the transmit data. In Figure 182, Y represents the observed samples at the output of the PA and X represents the input samples to the PA. The estimation engine computes the inverse PA model that is applied to transmit data (represented as U) in the DPD actuator.

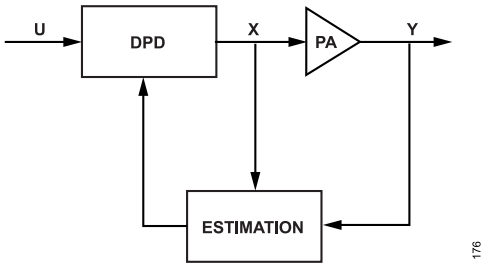


Figure 182. DPD Indirect Learning Architecture

The mathematical representation of the DPD coefficient estimation is shown in Figure 183. The DPD engine observes N samples of PA input samples (X) and PA output samples (Y), and computes M-coefficients (c) corresponding to the inverse PA function F(x).

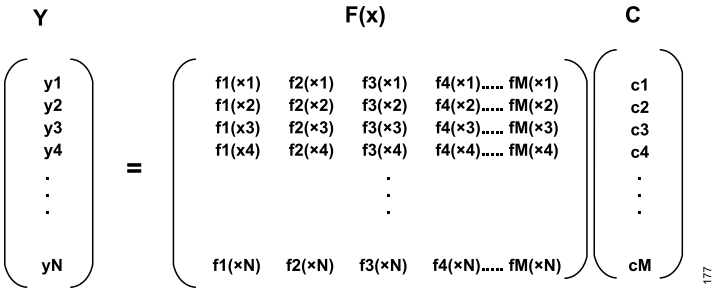


Figure 183. DPD Indirect Learning Coefficient Computation

The coefficient set (c) is estimated through a least squares approximation as described in matrix multiplication equations (1) to (3).

$$Y = F \times C \quad (1)$$

$$F^H Y = (F^H F)C \quad (2)$$

MULTIPLY BY COMPLEX CONJUGATE OF F ON BOTH SIDES.

$$\underbrace{(F^H F)^{-1}}_{\text{AUTO-CORRELATION}} \underbrace{(F^H Y)}_{\text{CROSS-CORRELATION}} = C \quad (3)$$

TAKE THE INVERSE OF THE AUTO-CORRELATION FUNCTION TO OBTAIN C.

178

Figure 184.

DPD Direct Learning

DPD direct learning involves using the pre-DPD actuator Tx signal (U) as reference to minimize the error between the observed (Y) and reference data (U).

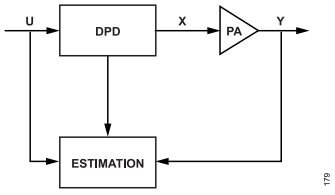


Figure 185. DPD Direct Learning Architecture

DIGITAL PRE-DISTORTION (DPD)

The mathematical representation of the DPD coefficient estimation via direct learning is described as follows. An error E is defined as the difference between observed (Y) and pre-DPD actuator data (U).

$$E = Y - U \tag{33}$$

The PA is modeled as the function Fx multiplied by adaptive coefficients C through the error matrix E as shown in Figure 186.

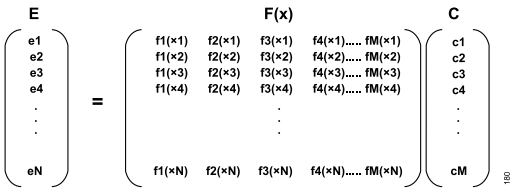


Figure 186. DPD Direct Learning Coefficient Computation

The direct learning outcome is an iterative solution, where the current coefficients are based on memory of previously computed coefficients and currently estimated coefficients (C).

The Direct Learning approach uses a parameter, μ, which is the convergence factor that defines the step size for learning coefficients. The convergence factor μ lies in the range [0% to 100%] and is user configurable in the transceiver using the API adi_adrv904x_DpdTrackingConfigSet() through the parameter adi_adrv904x_DfeAppCalDpdTrackCfg.mu. A higher value of the convergence factor μ would result in faster convergence. However, a really high value of μ might result in an unstable system. The user can start with a convergence factor of 50% and tune the value based on characterization of the system for convergence time and stability.

Comparison Between DPD Indirect Learning and Direct Learning

- The DPD indirect learning algorithm is time efficient, since it estimates coefficients through inversion in a single update. The DPD indirect learning algorithm is preferred when a quicker adaptive response is required by the system.
- The DPD direct learning algorithm is more accurate but iterative in nature as highlighted in the previous section and requires a longer time to converge compared to indirect learning.

DPD Coefficient Estimation

The maximum number of coefficients is limited to 255 (M = 255), and the number of samples used to calculate the coefficients is typically 32768 samples (N = 32768). Although the number of samples N is user configurable using the API adi_adrv904x_DpdTrackingConfigSet(), ADI recommends setting the number of samples to 32768, which provides a good balance between estimation time and sample size.

The DPD algorithm runs on a dedicated quad core ARM processor (ARM-A55) and calculates the coefficients corresponding to GMP terms of the inverse PA model. This model pre-distorts the digital baseband signal before digital-to-analog conversion and transmission of samples to the transmitter upconverter (this output becomes the RF input to the PA). The PA output is sampled using an external loopback to an observation receiver channel input.

The DPD engine then correlates the observation receiver and transmitter samples through a dedicated correlator hardware on the ADRV904x transceiver to calculate the latest set of coefficients. The DPD engine performs a brief check on model error before updating the LUTs that feed the correction coefficients into the DPD actuator hardware. Details regarding the GMP model, the actuator and the LUTs are provided in sections to follow.

GMP Model and Look Up Table

For wideband signals such as LTE and 5GNR, PAs begin to exhibit short-term memory effects. These are effectively non-uniform frequency responses in certain components like the biasing network, decoupling capacitors, or supply circuitry. A given output of the power amplifier depends not only on the current input, but also on past input values (similar to a FIR filter model). These memory effects can usually be captured with memory taps in the model if the PA frequency response is locally smooth over the band. As bandwidth increases, more memory taps are required to accurately model the PA.

GMP Only Mode LUT

In GMP only DPD implementation, the model is represented as

DIGITAL PRE-DISTORTION (DPD)

$$y_{GMP}\left(n\right) = \sum_{i=0}^{j<16} \sum_{i=0}^{i<16} \sum_{k} c_{i,j,k} \quad P_k(|x(n-i)|)x(n-j) \tag{34}$$

where
x(n) is a complex baseband input signal to the DPD actuator.
y(n) is a complex valued output signal of the DPD actuator.
$c_{i,j,k}$ is the complex valued coefficient of the GMP terms.

The DPD supports a sparse GMP model consisting of a maximum of 256 GMP terms and coefficients in which the memory term (i), the cross term (j), and the power term (k) are each restricted to a value between 0 to 15. A more complete equation for the GMP model with the limits on GMP terms is shown in the Table 98.

The GMP model is user programmable through the API adi_adrv904x_DpdModelConfigDpdSet(). Details pertaining to programming the DPD model are captured in the ADRV904x DPD Model section.

The GMP model for a particular operating point of the PA is determined by the user and programmed into the transceiver during initialization. The GMP model is mapped to LUTs in the DPD actuator. Each feature is specified with a unique combination of i, j, and k indices. In general, low index values are more significant in sparse GMP models. Therefore, restrictions are placed on actuator data path accordingly. The LUT restrictions are illustrated Table 98. Each row shares the same multiplier, therefore, the same j value.

Table 98. GMP Mapping to DPD Actuator LUTs

		LUT Amplitude (i)															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**16**
Signal (j)	0	0	1	2													
	1		3	4	5												
	2			6	7	8											
	3				9	10	11	12									
	4					13	14	15									
	5					16	17	18	19	20	21	22					
	6							23	24	25							
	7							26	27	28	29						
	8							30		31	32	33					
	9							34			35	36	37				
	10											38	39	40			
	11												41	42	43		
	12													44	45	46	
	13														47	48	49

GMP + DDR Mode LUT

In GMP + DDR DPD implementation the GMP + DDR model is represented as

$$\begin{aligned} y_{GMP}\left(n\right) = & \sum_{i=0}^{i<16} \sum_{j=0}^{j<14} \sum_{k} c_{i,j,k} \quad P_k(|x(n-i)|)x(n-j) + \\ & \sum_{i=0}^{i<3} \sum_{j=0}^{j<3} \sum_{k} c_{i,j,k} \quad P_k(|x(n-i)|)x(n-j) + \\ & \sum_{j=1}^{6} \sum_{k} d_{j,k} \quad P_k(|x(n)|)x^2(n)x \times (n-j) + \\ & \sum_{j=1}^{6} \sum_{k} e_{j,k} \quad P_k(|x(n)|)x(n)|x(n-j)|^2 + \\ & \sum_{j=1}^{6} \sum_{k} f_{j,k} \quad P_k(|x(n)|)x \times (n)x^2(n-j) \end{aligned} \tag{35}$$

where
x(n) is a complex baseband input signal to the DPD actuator.

DIGITAL PRE-DISTORTION (DPD)

y(n) is a complex valued output signal of the DPD actuator.
$c_{i,j,k}$ is the complex valued coefficient of the GMP terms.

The first term in this equation is the GMP term from the GMP only mode which uses a subset of the LUT's from figure 13 depending on the DPD Actuator Mode (adi_adrv904x_DfeAppCalDpdModelDesc_t.mode) been used. DPD Actuator Mode 0 is GMP only mode, see Table 101 for LUT reserved for DDR terms.

The second term in this equation is an additional optional GMP polynomial with fewer number of terms. See Table 99.

The remaining 3 terms are the DDR7, DDR8, and DDR9 terms use for modeling memory effects through DDR polynomials, the LUTs reserved for DDR7, DDR8, and DDR9 terms depend on the DPD Actuator Mode been used, see Table 101 for LUT reserved for DDR terms.

Table 99. LUT map (i, j) for GMP1 Terms

		LUT Amplitude (i)		
		0	**1**	**2**
Signal (j)	0	0	1	2
	1	3	4	5
	2	6	7	8

Table 100. GMP + DDR Mapping to DPD Actuator LUTs

		LUT Amplitude (i)															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**16**
Signal (j)	0	9_1 7_4	9_27_ 5	9_3 7_6													
	1		9_4	9_5 8_1	9_6 8_2												
	2			8_4	8_5	8_6											
	3				9	10	11	12									
	4					13	14	15									
	5					16	17	18	19	20	21	22					
	6							23	24	25							
	7							26	27	28	29						
	8							30		31	32	33					
	9							34			35	36	37				
	10											38	39	40			
	11												8_3	8_2	8_1		
	12													7_68_ 4 9_3	7_5 8_3 9_2	7_4 9_1	
	13														7_3 9_6	7_2 9_5	7_1 9_4

Table 101. DPD Actuator Modes (adi_adrv904x_DfeAppCalDpdModelDesc_t.mode)

DPD Actuator Mode	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
No. LUT for DDR	0	18	12	12	6	6	6	9
LUTs Reserved for DDR	NA	0 to 8, 41 to 49	0 to 5, 44 to 49	0 to 5, 44 to 49	0 to 2, 47 to 49	0 to 2, 47 to 49	0 to 2, 47 to 49	41 to 49
LUT 0	GMP	DDR9	DDR9	DDR9	DDR7	DDR9	DDR9	GMP
LUT 1	GMP	DDR9	DDR9	DDR9	DDR7	DDR9	DDR9	GMP
LUT 2	GMP	DDR9	DDR9	DDR9	DDR7	DDR9	DDR9	GMP
LUT 4	GMP	DDR9	DDR9	DDR8	GMP	GMP	GMP	GMP
LUT 3	GMP	DDR9	DDR9	DDR9	GMP	GMP	GMP	GMP
LUT 5	GMP	DDR9	DDR9	DDR8	GMP	GMP	GMP	GMP
LUT 6	GMP	DDR8	GMP	GMP	GMP	GMP	GMP	GMP

DIGITAL PRE-DISTORTION (DPD)

Table 101. DPD Actuator Modes (adi_adrv904x_DfeAppCalDpdModelDesc_t.mode) (Continued)

DPD Actuator Mode	0	1	2	3	4	5	6	7
LUT 7	GMP	DDR8	GMP	GMP	GMP	GMP	GMP	GMP
LUT 8	GMP	DDR8	GMP	GMP	GMP	GMP	GMP	GMP
LUT 41	GMP	DDR8	GMP	GMP	GMP	GMP	GMP	DDR8
LUT 42	GMP	DDR8	GMP	GMP	GMP	GMP	GMP	DDR8
LUT 43	GMP	DDR8	GMP	GMP	GMP	GMP	GMP	DDR8
LUT 44	GMP	DDR7	DDR7	DDR8	GMP	GMP	GMP	DDR9
LUT 45	GMP	DDR7	DDR7	DDR8	GMP	GMP	GMP	DDR9
LUT 46	GMP	DDR7	DDR7	DDR7	GMP	GMP	GMP	DDR9
LUT 47	GMP	DDR7	DDR7	DDR7	DDR7	DDR9	DDR7	DDR7
LUT 48	GMP	DDR7	DDR7	DDR7	DDR7	DDR9	DDR7	DDR7
LUT 49	GMP	DDR7	DDR7	DDR7	DDR7	DDR9	DDR7	DDR7

ADRV904x DPD Model

An example ADRV904x DPD model is shown in Table 102. The terms (i = 2, j = 1), (i = 3, j = 2), (i = 4, j = 3) are mapped to LUT4, LUT7, and LUT10 in theADRV904x LUT layout described in Table 98. The poly term maps the polynomial to either a GMP or a DDR term. The value 0 represents GMP in this example.

Table 102. Example of User Programmed GMP Model

i	j	k	Poly	a.real	a.imag
2	1	2	0	0	0
2	1	3	0	0	0
3	2	1	0	0	0
3	2	5	0	0	0
3	2	7	0	0	0
4	3	1	0	0	0
4	3	3	0	0	0
4	3	8	0	0	0

Equation 36, Equation 37, and Equation 38 represent the GMP terms that are mapped to LUT4, LUT7, and LUT10 described in Table 102.

$$Y_{lut4} = |x(n-2)|^2.x(n-1) + |x(n-2)|^3.x(n-1) \quad (36)$$

$$Y_{lut7} = |x(n-3)|.x(n-2) + |x(n-3)|^5.x(n-2) + |x(n-3)|^7.x(n-2) \quad (37)$$

$$Y_{lut10} = |x(n-4)|.x(n-3) + |x(n-4)|^3.x(n-3) + |x(n-4)|^8.x(n-3) \quad (38)$$

The user can program the complex coefficients to 0. The DPD tracking calibration determines the coefficients and applies them to the GMP terms on each update.

The model can be programmed using the API adi_adrv904x_DpdModelConfigDpdSet() through the data structure adi_adrv904x_DfeAppCalDpdModelDesc_t. The total number of features in the model is conveyed through the variable adi_adrv904x_DfeAppCalDpdModelDesc_t.features. Each row of the table or the feature is programmed through the array adi_adrv904x_DfeAppCalDpdModelDesc_t.feature. Each element of the array corresponds to one row in the table shown in Table 102.

Initializing Pre-calibrated Coefficients During Start-up

The DPD functionality on the transceiver provides a mechanism for loading pre-calibrated coefficients into the DPD model to prevent emissions during start-up and also guide DPD updates at the beginning. This is done by loading non-zero real and imaginary coefficients in the feature, adi_adrv904x_DfeAppCalDpdModelDesc_t.feature.a.real and adi_adrv904x_DfeAppCalDpdModelDesc_t.feature.a.imag. Each of the Tx channels can be loaded with different DPD models depending on use case.

DIGITAL PRE-DISTORTION (DPD)

DPD SAMPLE CAPTURE

The DPD algorithm relies on observing the samples distorted by the PA through an observation channel to estimate the DPD coefficients. The DPD algorithm captures the observation samples after they have been processed by the observation receiver channel, and before and after the DPD actuator in batches of 4096 samples. The total number of samples that the DPD algorithm needs to capture is configured using the API **adi_adrv904x_DpdTrackingConfigSet()** through the parameter **adi_adrv904x_DfeAppCalDpdTrackCfg_t.adapt.numDpdSamples**. The number of samples must be a multiple of 4096. Increasing the number of samples increases the processing time and computation load. Conversely, fewer samples could impact the accuracy of coefficient estimation. Analog Devices recommends the number of samples be set to 32768, which provides a good balance between accuracy of estimation of coefficients and processing time.

For successful captures, the transmitter to observation channel external signal routing needs to be conveyed to the firmware through proper Tx to ORx mapping. See Transmitter To Observation Receiver Mapping section for further details.

DPD Sample Capture Process

The DPD algorithm implements a peak detection-based capture strategy since the high-power signal levels contain more useful information for deriving DPD coefficients. The device's calibration scheduler can initiate DPD capture at any available point in time when the transmit signal chain is enabled. The sequence of events involved in DPD sample capture process is shown in Figure 187.

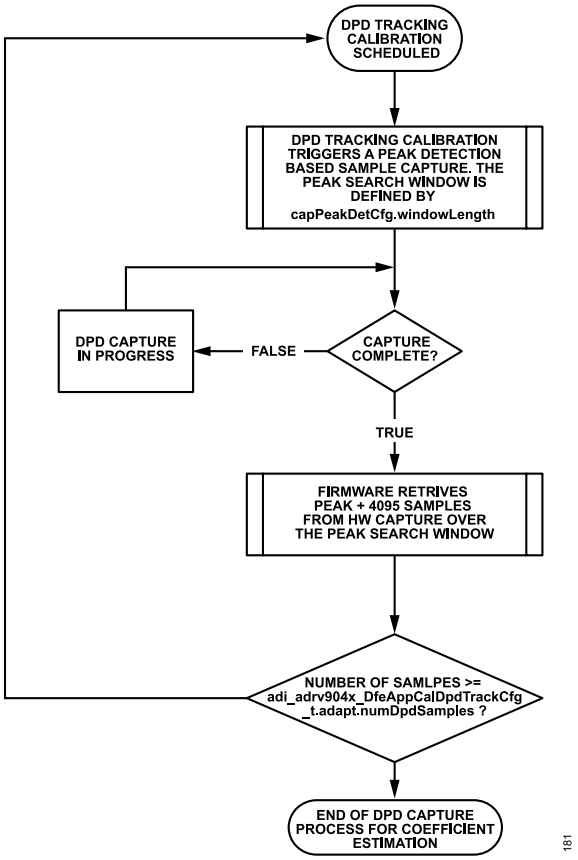


Figure 187. DPD Tracking Calibration Sample Capture Sequence

Peak Search Window and Peak Detection Based Capture

The DPD capture engine contains a peak detector that triggers captures on the largest peak seen in a specified time window as shown in Figure 188. Peak detection occurs at the DPD input and after CFR correction is applied.

The peak detection time window is specified in number of samples at the DPD actuator rate using the API adi_adrv904x_DpdCaptureConfig-Set() through the parameter adi_adrv904x_DfeAppCalDpdCaptureCfg_t.capPeakDetCfg.windowLength. While longer peak search windows result in more accurate high-power DPD modeling, the capture process also takes longer to complete. The goal of peak search window

DIGITAL PRE-DISTORTION (DPD)

optimization is to increase the probability of capturing high-power data without penalizing the rest of the system operation. In order to do so, the system designer must study the worst-case signal statistics. This DPD contains only one adaptation engine, so the capture mechanism is shared between all active channels.

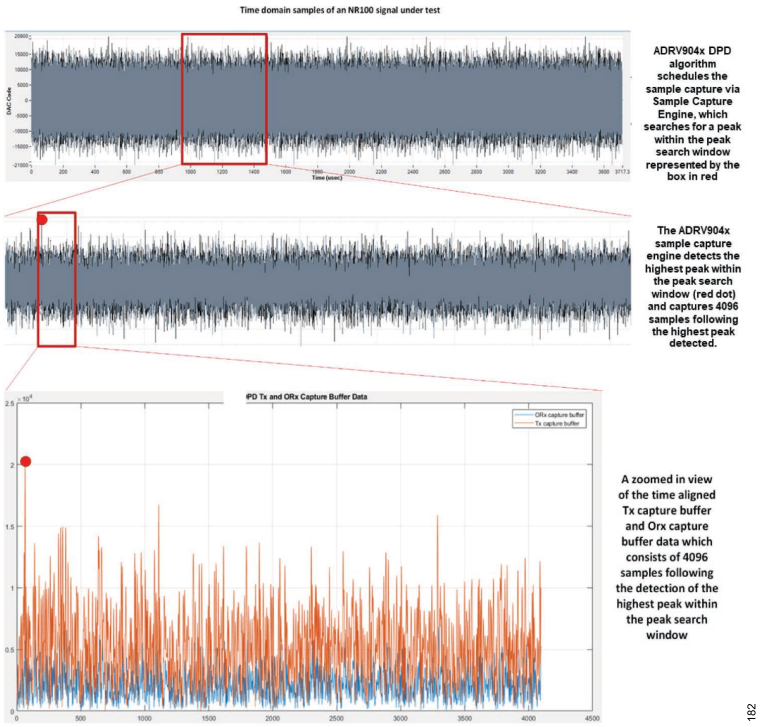


Figure 188. DPD Peak Detection Based Sample Capture Process for DPD Adaptation

DPD Sample Capture in TDD Mode

There are additional considerations that a system developer needs to consider for configuring the DPD in TDD mode. In TDD mode, the DPD sample capture process spans multiple TDD downlink slot periods (Tx ON periods), with each batch of 4096 samples captured during one TDD downlink slot period (Tx ON period) through the peak detection process highlighted in the previous section. The peak search window size is specified by adi_adrv904x_DfeAppCalDpdCaptureCfg_t.capPeakDetCfg.windowLength and configured using the API adi_adrv904x_DpdCaptureConfigSet(). The search window size is restricted to a maximum of one TDD downlink slot period (one Tx ON period) minus 4096 samples.

An example of a typical sample capture process in TDD mode is illustrated in Figure 189. In this example, the total number of DPD samples specified by adi_adrv904x_DfeAppCalDpdTrackCfg_t.adapt.numDpdSamples is set to 16384 samples or four batches of 4096 samples. Each batch of 4096 samples corresponding to peaks P1, P2, P3, and P4, respectively, are captured over four TDD downlink slot periods (four Tx ON periods). If each Tx ON period consists of M samples at the DPD actuator rate, then the maximum peak search window size that can be configured using adi_adrv904x_DfeAppCalDpdCaptureCfg_t.capPeakDetCfg.windowLength is restricted to (M-4096) samples. At the end of four TDD downlink slot periods, a composite capture data consisting of 4 batches of 4096 samples corresponding to peaks P1, P2, P3 and P4 respectively are used to estimate the DPD coefficients as described in the Indirect Learning section.

DIGITAL PRE-DISTORTION (DPD)

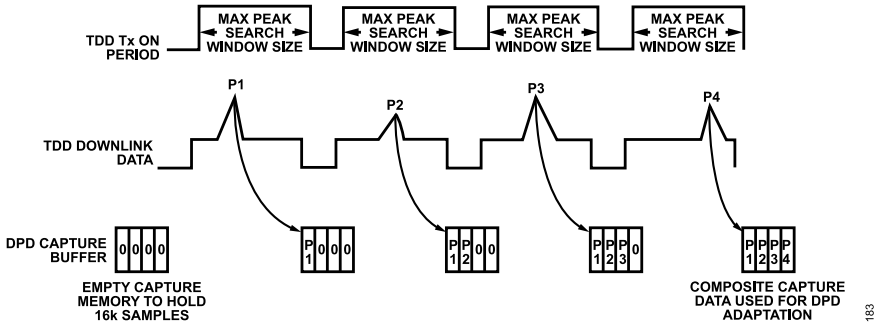


Figure 189. DPD Sample Capture Process in TDD Mode

DPD DYNAMICS

The transceiver DPD is designed to react to dynamic signaling conditions. The algorithm defines four models that can be implemented depending on the power levels to achieve the best dynamic performance. The model type can be programmed using the API adi_adrv904x_DpdModelConfigDpdSet() through the data structure adi_adrv904x_DfeAppCalDpdModelType_e. The four DPD models are defined Table 103.

Table 103. DPD Models Explained

DPD Model	Description
M-Table (Max Power Table) (DPD_MODEL_TYPE_DPD_0)	Default model in all 3 DPD modes. The conditions for updating this table in different DPD modes are listed as follows: ▶ DPD MODE0. The model defined by M-Table is updated on every DPD iteration if update criteria is met as described in DPD_MODE0 section in Table 104. The update criteria is described in the DPD Modes of Operation section. ▶ DPD MODE1. The model defined by M-Table is updated when the rms power of DPD capture samples exceeds previously recorded maximum rms power as described in DPD_MODE1 section in the Table 104 section. ▶ DPD MODE2. The model defined by M-Table is updated only when the rms power of DPD capture samples exceeds M-Threshold specified by the configuration adi_adrv904x_DfeAppCalDpdTrackCfg_t.mThresholdDB, and the rms power of DPD capture samples exceeds the previously recorded maximum rms power. Refer to Table 104 for more details on DPD Mode 2 operation.
C-Table (Current-Table) (DPD_MODEL_TYPE_DPD_1)	The model defined by C-Table is a low power model only applicable only in DPD MODE2 when the rms power of DPD capture samples is below the M-Threshold value specified by adi_adrv904x_DfeAppCalDpdTrackCfg_t.mThresholdDB as described by DPD_MODE2 section in the Table 104 section.
R-Table (Recovery Table) (DPD_MODEL_TYPE_DPD_2)	The R-Table or recovery table is a recovery model that is a user defined model for the recovery/safe operation of the PA.
U-Table (Unity Gain Table)	Unity gain model in which output is equal to input. This model is usually activated in low power conditions where pre-distortion is not necessary.

DPD Modes of Operation

The DPD functionality supports three modes of operation that are listed in Table 104. The user can select one of the three modes based on the application. The DPD engine needs to react to changing signal conditions, so Analog Devices has developed proprietary algorithms that address this requirement. Within a cost-bounded implementation there is no solution that will achieve absolute performance on any time scale of measurement. The Analog Devices solution is an optimized compromise between performance and complexity.

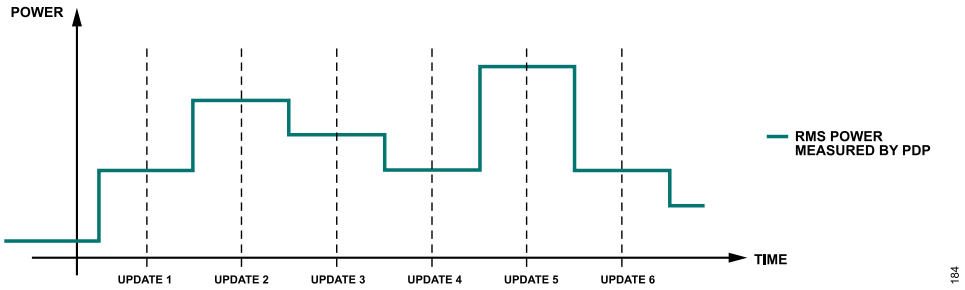
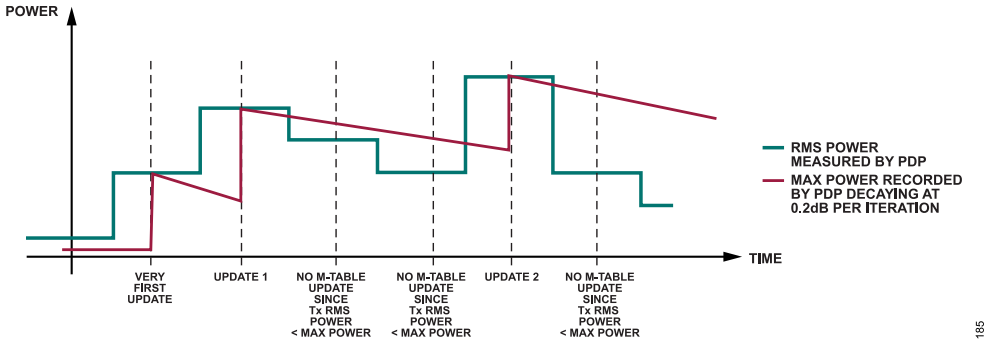
The DPD mode of operation can be configured through the API adi_adrv904x_DpdTrackingConfigSet() using the parameter adi_adrv904x_DfeAppCalDpdTrackCfg_t.updateMode parameter.

Table 104. DPD Modes of Operation

DPD Mode of Operation	Description
DPD_MODE0	DPD coefficients corresponding to the GMP model are updated once every second. This mode offers the best sustained performance for any signal at the expense of transient emissions when the signal changes rapidly.

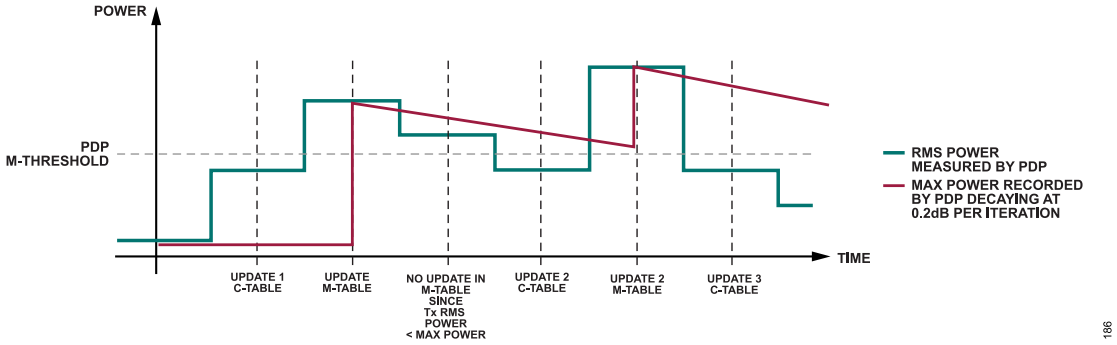
DIGITAL PRE-DISTORTION (DPD)

Table 104. DPD Modes of Operation (Continued)

DPD Mode of Operation	Description
	Shown in Figure 190 is the DPD updates in Mode 0. The DPD updates coefficients once every update period independent of the RMS power measured by the DPD captures for coefficient computation.  *Figure 190. DPD Mode 0 Update Example*
DPD_MODE1	DPD coefficients corresponding to the GMP model are updated only if the RMS power measured by the DPD exceeds the previously recorded maximum RMS power by the DPD algorithm. The RMS power is calculated on the samples captured by the DPD for coefficient computation. The number of samples to capture for a DPD update is specified by adi_adrv904x_DfeAppCalDpdTrackCfg_t.adapt.numDpdSamples. Typically this is set to 16384 samples. The recorded maximum power decays at a fixed rate of 0.2 dB per update. Shown in Figure 191 is an illustration of DPD updates in Mode 1. The DPD algorithm updates coefficients only when the RMS power measured by the DPD during the update exceeds the previously recorded maximum power. In this example, there is no update between Update 1 and Update 2 because the RMS power is below the max power recorded.  *Figure 191. DPD Mode 1 Update Example* DPD Mode 1 offers the best mitigation of transient emissions when the signal changes rapidly, at the expense of sustained performance in certain low-power signal conditions.
DPD_MODE2	In this mode, the DPD algorithm maintains 2 separate look up tables, one for low power region and the other for the high power region. Depending on the RMS power measured by the DPD on the samples captured for coefficient computation, the DPD algorithm either switches to the high power look up table (M-Table) or the low power look up table (C-Table), and the same look up table is active until the next DPD update. The RMS power threshold separating the low power and high power region is user configurable through the parameter adi_adrv904x_DfeAppCalDpdTrackCfg_t.mThreshold. The RMS power is calculated on the samples captured by the DPD for coefficient computation. The number of samples to capture for a DPD update is specified by adi_adrv904x_DfeAppCalDpdTrackCfg_t.adapt.numDpdSamples. Typically this is set to 16384 samples. Figure 192 is an illustration of DPD updates in Mode 2 operation. When multiple models are used adi_adrv904x_DfeAppCalDpdModelDesc_t.actDepth must equal 2

DIGITAL PRE-DISTORTION (DPD)

Table 104. DPD Modes of Operation (Continued)

DPD Mode of Operation	Description
	 *Figure 192. Illustration of DPD Updates in Mode 2* This offers a compromise between modes 0 and 1. There is some mitigation of transient emissions when the signal changes rapidly and sustained performance in many signalling conditions.

Transmitter Low Power Threshold

The DPD continuously integrates the baseband power level at the input of the DPD actuator so that it can switch between the different models described in Table 104. The power is measured as part of the DPD Robustness Framework (see DPD Robustness) that runs continuously in the background when enabled. If the mean rms power of the DPD input samples is below the mean threshold specified by dpdStabilityConfig.bit[0].threshold0, the unity gain table is activated, and if the mean rms power of the DPD input samples is higher than the Tx low power threshold specified by dpdStabilityConfig.bit[0].threshold0, the DPD model defined by M-Table is activated in DPD Mode 0 and Mode 1. In DPD mode 2, there is an additional threshold specified by adi_adrv904x_DfeAppCalDpdTrackCfg_t.mThresholdDB described in the Transmitter M-Threshold section. The dynamics of the DPD based on the transmit baseband input level is shown in Figure 193.

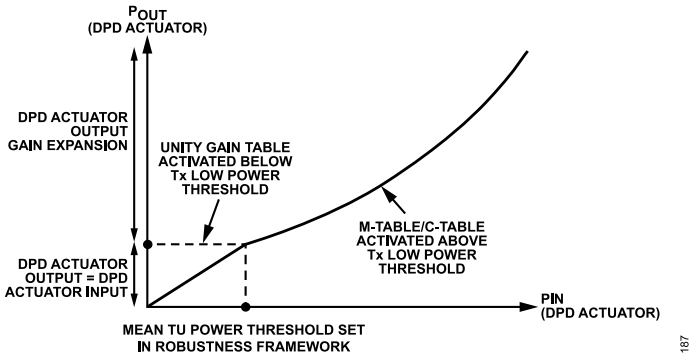


Figure 193. DPD Dynamics and Tx Low Power Threshold

Transmitter M-Threshold

The M-Threshold is a max power threshold specified by adi_adrv904x_DfeAppCalDpdTrackCfg_t.mThresholdDB that is valid only in DPD mode 2 operation (adi_adrv904x_DfeAppCalDpdTrackCfg_t.updateMode = DPD_MODE2). There are two DPD models (M-Table, C-Table) that the DPD tracking calibration maintains and updates. The DPD model update mechanism in DPD mode 2 operation is described in Table 104. Note that the switching mechanism between the models M-Table (high power), C-Table (low power) and U-Table (unity gain) is based on the 10 ms integrated rms power of the DPD actuator input samples, similar to the mechanism described in the DPD Characterization for Optimizing M-Threshold section.

The dynamics of the DPD based on the Tx baseband input level with the M-Threshold taken into consideration is shown in Figure 194.

DIGITAL PRE-DISTORTION (DPD)

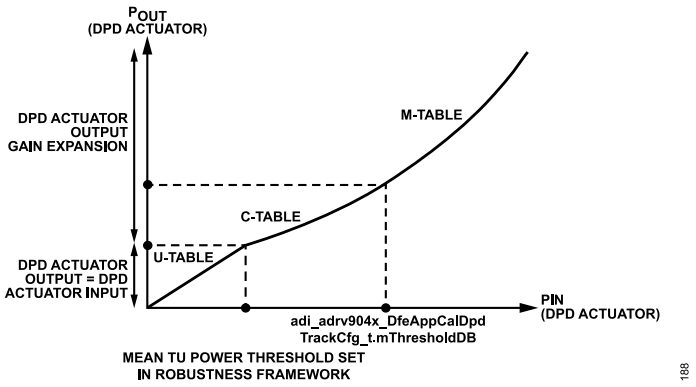


Figure 194. ADRV904x DPD Dynamics in DPD Mode 2

DPD Power Meter

The power threshold measurements for both DPD Mode 1 and DPD Mode 2, low power and mThreshold are measured using the DPD power meter , this is configured with API adi_adrv904x_DfePwrMtrDpdInConfigSet() and must be enabled for these mode.

DPD REGULARIZATION

DPD regularization is used to make the DPD coefficient estimation less sensitive to missing data and prevent over-fitting. The DPD is essentially a curve fitting process, and Figure 195 outlines the optimum fitting to achieve in a system. A higher regularization prevents over-fitting, in turn improving stability but limiting the ACLR improvement. On the other hand, a low regularization allows for better ACLR improvement, but stability of the DPD needs to be kept in check.

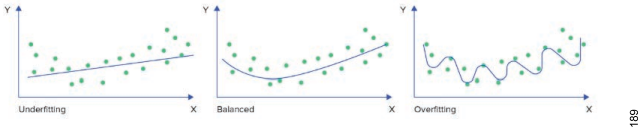


Figure 195. Comparison of Underfitting, Overfitting, and Balanced Fitting

The AM-AM characteristics of a PA for a case where there is sparse data in the high power region is shown in Figure 196. In this case, a low regularization value would result in overfitting and causing instability.

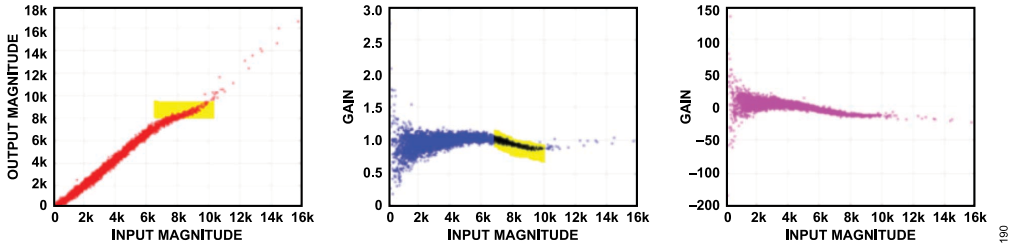


Figure 196. DPD Regularization for Decreasing Sensitivity to Sparse Data

For the AM-AM characteristics shown above, the effect of low regularization and optimum regularization on DPD is shown in Figure 197. With low regularization, the DPD algorithm has a tendency to overfit resulting in high power scattering. With the optimum regularization, the sensitivity of DPD algorithm to sparse data in high power is minimized.

DIGITAL PRE-DISTORTION (DPD)

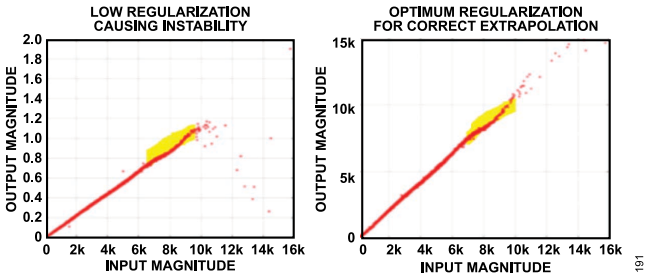


Figure 197. Effect of DPD Regularization on DPD Stability

The DPD provides user configuration for regularization via adi_adrv904x_DfeAppCalDpdTrackCfg_t.indirectRegValue and adi_ adrv904x _DfeAppCalDpdTrackCfg_t.directRegValue for indirect learning and direct learning mechanisms configured via the adi_adrv904x_ DpdTrackingConfigSet() API.

DPD Regularization in DPD Mode 2

For DPD Mode 2 operation where separate pre-distortion coefficients are maintained for low power (C-Table) and high power (M-Table) data. This is typically intended to be used with PA applications where the PA non-linearity characteristics could vary for low power and high power signals.

Table 105. DPD Regularization Parameters in DPD Mode 2

Regularization Parameter	Target DPD Actuator Model
adi_adrv904x_DfeAppCalDpdTrackCfg_t.indirectRegValue[0]	M-Table
adi_adrv904x_DfeAppCalDpdTrackCfg_t.indirectRegValue[1]	C-Table

DPD ROBUSTNESS

This section provides an overview of the features that enhance DPD robustness. These features include DPD stability metrics and flexible architecture for setting up recovery actions based on these pre-defined DPD metrics. The user can optionally turn on the DPD robustness feature to protect DPD against erroneous adaptations in abnormal conditions.

In the transceiver DPD, abnormal conditions are detected by monitoring the following metrics:

- Pre-DPD capture RMS power
- Pre-DPD capture peak power
- Post-DPD capture RMS power
- Post-DPD capture peak power
- Observed capture RMS power
- Observed capture peak power
- Pre-DPD capture Tx to ORx EVM (Direct EVM) which is a measure of DPD linearization performance
- Post-DPD Capture Tx to ORx EVM (Indirect EVM) which is a measure of non-linearity in the gain line up
- Indirect error that indicates if the pre-distorted samples match the expected result after experiencing PA distortion

From a user's point of view, the following are the two sets of actions that are required to be taken:

- Define user-configurable thresholds with adi_adrv904x_DfeAppCalDpdStabilityBitCfg_t.bit[] through API DpdStabilityCfgSet().
- Define the fault conditions and actions for fault conditions with adi_adrv904x_DfeAppCalDpdStabilityActCfg_t.error[] through API DpdStabilityCfgSet().

Example: measure transmit capture power, if power < threshold, skip DPD update.

Calculation of Metrics

After estimation of the DPD coefficients, the error between the predicted and measured predistortion is computed to determine the expected DPD performance. Detection of a large error prevents application of bad coefficients can be calculated using Equation 39.

DIGITAL PRE-DISTORTION (DPD)

$$IndirectError = \frac{||x_{tx} - F_y c||}{||x_{tx}||} \quad (39)$$

where
x_{tx} is a vector of Tx samples after DPD actuator (post-DPD data).
F_y is a matrix of features (such as items in GMP) formulated by ORx samples.
c is the DPD coefficients vector; and the operator $||.||$ is the (Euclidean) norm of vectors.

Similarly, indirect EVM and direct EVM are calculated using Equation 40 and Equation 41.

$$IndirectEVM = \frac{||x_{tx} - y||}{||x_{tx}||} \quad (40)$$

$$DirectEVM = \frac{||x_{tu} - y||}{||x_{tu}||} \quad (41)$$

where
x_{tu} is a vector of Tx samples before the DPD actuator (pre-DPD data).
y is a vector of ORx samples. All samples are time aligned and gain and phase equalized. The size of the vectors is the number of samples used in each update of the DPD coefficients.

Transmit signal mean and peak power are calculated from the pre-DPD samples in x_{tu} and post-DPD samples in x_{tx}, respectively. Observation receiver mean and peak power are calculated by the samples in y.

Defining Fault Conditions

The user can define fault conditions through the data structure adi_adr904x_DfeAppCalDpdStabilityCfg_t() described in Table 106 and Table 107. The fault conditions are programmed through the API DpdStabilityCfgSet().

Table 106. DPD Fault Condition Metric adi_adrv904x_DfeAppCalDpdStabilityCfg_t.bit

Bit	Fault Condition
0	Define Fault Conditions for Mean Tu Power
1	Define Fault Conditions for Peak Tu Power
2	Define Fault Conditions for Mean Tx Power
3	Define Fault Conditions for Peak Tx Power
4	Define Fault Conditions for Mean Orx Power
5	Define Fault Conditions for Peak Orx Power
6	Define Fault Conditions for Direct EVM
7	Define Fault Conditions for Indirect EVM
8	Define Fault Conditions for Indirect error

Table 107. DPD Fault Condition Definition

Member	Data Type	Description
LTGT	UInt16	The user can select a greater than (GT) or less than (LT)comparator for defining a fault condition. For example, the user can define a fault condition where Tx RMS power is greater than a certain threshold and ORx RMS power is lesser than a certain threshold.
threshold0	Int16	Defines a threshold for a lower severity level.
threshold1	Int16	Defines a threshold for a higher severity level.
persistentCnt	UInt16	Persistence count is defined with the expectation that more aggressive recovery actions will be required in case the fault conditions persist.

The fault conditions defined in Table 107 can be associated with a recovery action programmed through the API DpdStabilityCfg-Set() through the parameter adi_adrv904x_DfeAppCalDpdStabilityCfg_t.error[].mask for selecting the fault condition, see Table 108, and adi_adrv904x_DfeAppCalDpdStabilityCfg_t.error[].actionWord for selecting the recovery action, see Table 109 Table 109

DIGITAL PRE-DISTORTION (DPD)

Table 108. ADRV904x Error Mask Definition

mask	Fault Condition
0x001	Define Fault Conditions for Mean Tu Power
0x002	Define Fault Conditions for Peak Tu Power
0x004	Define Fault Conditions for Mean Tx Power
0x008	Define Fault Conditions for Peak Tx Power
0x010	Define Fault Conditions for Mean Orx Power
0x020	Define Fault Conditions for Peak Orx Power
0x040	Define Fault Conditions for Direct EVM
0x080	Define Fault Conditions for Indirect EVM
0x100	Define Fault Conditions for Indirect error

Table 109. ADRV904x Error Action Definition

Action	actionWord	Description
DPD_SKIP_UPDATE	0x01	Skips DPD update.
DPD_RESET_ALL_LUTS	0x02	Resets all LUTs.
DPD_RESET_DYNAMIC_STATS	0x04	Resets DPD max power recorded (used in DPD Mode 1 and DPD Mode 2).
DPD_SW_TO_M_TABLE	0x08	Switches to M table.
DPD_SW_TO_R_TABLE	0x10	Switches to R table.
DPD_RESET_FIRST_DPD	0x20	Soft resets to run 3 back-to-back indirect learning updates like the very first DPD update.
DPD_SW_TO_MIN_DIRECT_EVM	0x40	Switches to coefficients corresponding to minimum direct EVM metric from the time of last DPD reset.

The DPD fault conditions configured in the firmware on startup are listed in Table 110.

Table 110. Recommended Default Definition of Fault Condition Matrix

Metric	Comparator	Low Threshold	High Threshold	Persistent Count
Mean TU Power (Pre-DPD)	Less Than	−36 dBFS	−46 dBFS	10
Peak TU Power (Pre-DPD)	Less Than	−26 dBFS	−35 dBFS	10
Mean Tx Power (Post-DPD)	Less Than	−36 dBFS	−46 dBFS	10
Peak Tx Power (Post-DPD)	Less Than	−26 dBFS	−35 dBFS	10
Mean ORx power (Post-DPD)	Less Than	−36 dBFS	−46 dBFS	10
Peak ORx power (Post-DPD)	Less Than	−23 dBFS	−32 dBFS	10
Pre-DPD capture Tx to ORx EVM (Direct EVM)	Greater Than	3%	12%	10
Post-DPD capture Tx to ORx EVM (Indirect EVM)	Greater Than	12%	15%	10
Indirect Error	Greater Than	5%	12%	10

Table 111. Example DPD Fault Condition and Recovery Action Config

Error Condition	Error 0 - Fault Condition threshold0 Violated Once	Error 0 - Persistent Fault Condition Threshold0 Violated Persistently	Error 1 - Fault Condition threshold0 Violated Once	Error 1 - Persistent Fault Condition Threshold1 Violated Persistently
FAULT CONDITION	-	Mean Tu Power OR Peak Tu Power	-	Indirect EVM Only
RECOVERY ACTION	Do Nothing	Skip DPD LUT Updates	Do Nothing	Reset All LUTs

The following is an example code snippet to setup recovery actions shown in Table 110:

```
# Configure Fault Conditions
# Define Fault Conditions for Mean Tu Power. Flag warning if mean Tu power is less than -36dBFS and
error if mean Tu power is lower than -46dBFS
dpdStabilityConfig.bit[0].LTGT = 0 # Select Less Than Switch
dpdStabilityConfig.bit[0].threshold0 = -3600
dpdStabilityConfig.bit[0].threshold1 = -4600
dpdStabilityConfig.bit[0].persistentCnt = 10
```

DIGITAL PRE-DISTORTION (DPD)

```
# Define Fault Conditions for Peak Tu Power. Flag warning if peak Tu power is lesser than -26dBFS and
error if peak Tu power is lower than -36dBFS
dpdStabilityConfig.bit[1].LTGT = 0 # Select Less Than Switch
dpdStabilityConfig.bit[1].threshold0 = -2600
dpdStabilityConfig.bit[1].threshold1 = -3600
dpdStabilityConfig.bit[1].persistentCnt = 10

# Define Fault Conditions for Indirect EVM. Flag warning if Indirect EVM is greater than 12% and error
if Indirect EVM is greater than 22%
dpdStabilityConfig.bit[7].LTGT = 1 # Select Greater Than Switch
dpdStabilityConfig.bit[7].threshold0 = 12
dpdStabilityConfig.bit[7].threshold1 = 22
dpdStabilityConfig.bit[7].persistentCnt = 10
# Configure Recovery Actions
# No action for threshold0 violated once
dpdStabilityConfig.error[0].mask = 0x000
dpdStabilityConfig.error[0].actionWord = 0x00
# Skip DPD update if Mean Tu power or Peak Tu power is below threshold0
dpdStabilityConfig.error[1].mask = 0x001 | 0x002
dpdStabilityConfig.error[1].actionWord = 0x01
# No action for threshold0 violated once
dpdStabilityConfig.error[2].mask = 0x000
dpdStabilityConfig.error[2].actionWord = 0x000
# Reset all LUTs if indirect EVM exceeds threshold0
dpdStabilityConfig.error[3].mask = 0x080
dpdStabilityConfig.error[3].actionWord = 0x02
```

DPD ACTUATOR GAIN MONITORING FOR ROBUSTNESS

Principle of Operation

The DPD gain monitoring mechanism uses power meters at the input and output of the DPD actuator to determine when to switch between DPD models if the actuator gain violates a programmable threshold. The gain monitoring mechanism can be used to monitor gain over range as well as gain under range. Gain over range can occur when DPD is trying to expand gain to compensate for gain compression. A gain under range condition can occur due to bad coefficients or gain compression.

Gain can be monitored either sample by sample or averaged over a number of samples. The maximum number of samples that can be averaged is 128k samples (~266 µs worth of samples at a 491.52 MSPS rate). The average gain over several thousands of samples should not typically exceed 1 dB. The gain monitoring mechanism can be setup to switch to a unity gain or any other model if the gain/attenuation across the actuator is in the range of several dBs. Figure 198 represents a high level overview of the DPD actuator hardware in the signal chain.

Table 103 explains the DPD models implemented in the transceiver. The user has an option to switch to either one of these 4 DPD models in case the actuator output experiences high gain or high attenuation as explained in the previous section.

DIGITAL PRE-DISTORTION (DPD)

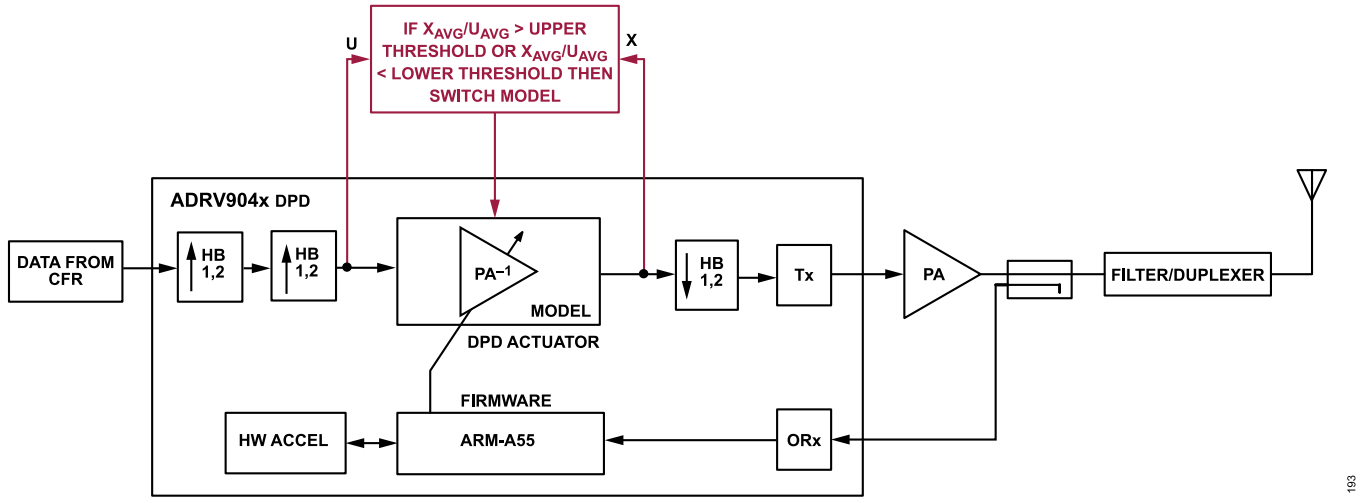


Figure 198. DPD Actuator Gain Monitoring Functional Diagram

DPD Actuator Gain Monitoring Configurations

The adjustments listed in Table 112 can be used to configure the DPD actuator gain monitoring feature.

Table 112. DPD Gain Monitoring Configurations

Gain Monitor Configuration	Description
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutGainMonitorEn	Enables/disables gain monitoring.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutQualLim	Minimum signal level above which gain monitoring will be exercised.
adi_adrv904x_DpdActGainMonitorCtrl_t. highGainModelAutoLoadEnable	Configuration to explicitly enable/disable high gain over range detection.
adi_adrv904x_DpdActGainMonitorCtrl_t.	Configuration to explicitly enable/disable low gain under range detection.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutMaxGainLim	High gain threshold above which a model switch can be initiated by the gain monitoring hardware. Only applicable if gain overrange detection is enabled.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutMinGainLim	Low gain threshold (attenuation) below which a model switch can be initiated by the gain monitoring hardware. Only applicable if gain under range detection is enabled.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. modelTableForMaxGainLim	Model to switch to on violation of high gain threshold.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. modelTableForMinGainLim	Model to switch to on violation of low gain threshold.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutPwrMeasDuration	Power measurement duration.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutPwrMeasContDlyCntr	Waits before continuing with power measurement after TXON.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutPwrMeasEn	Enables Tx power measurements.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutPwrMeasPause	Pauses Tx power measurements.
adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t. dpdInOutPwrMeasTDDModeEn	Runs power measurements in TDD mode.

DPD Actuator Gain Monitoring API

The API functions used to control gain monitoring are listed in Table 113.

Table 113. DPD Actuator Gain Monitoring API

API Function	Description
adi_adrv904x_DpdActuatorGainMonitorConfigSet()	This function sets the DPD gain monitor configuration described in Table 112. Gain monitoring can be used to automatically switch to selected models when actuator gain overrange or under range is seen. Gain violation thresholds are user configurable. This function takes in as argument, a pointer to the device data structure, and a pointer to a structure of type adi_adrv904x_DfeAppCalDpdPowerMeterCfg_t which contains configurations shown in Table 112.
adi_adrv904x_DpdActuatorGainMonitorConfigGet ()	This function returns the DPD gain monitor configuration applied in the device for the requested Tx channel.

DIGITAL PRE-DISTORTION (DPD)

An example Python script to set up the gain monitor with the configuration described in Table 114 is shown in Table 114.

Table 114. Gain Monitor Configuration Parameters

Configuration	Value
Gain Monitor Enable	Enabled
Gain Detection Qualifying Threshold	−42 dBFS
Gain Overrange Detection Enable	Enabled
Gain Under range Detection Enable	Enabled
Gain overrange threshold	+6 dB
Gain under range threshold	−6 dB
Gain over range model select	Unity Gain (Model 3)
Gain under range model select	Recovery Table (Model 2)
Decay Rate	1

DPD Actuator Gain Monitoring + Model Switching State Machine Representation

The flow chart in Figure 199 describes the function of the gain monitoring state machine. The DPD gain monitoring, once enabled, runs independently from the DPD actuator. It monitors the gain of the signal across the actuator until it is turned off, as shown in Figure 199.

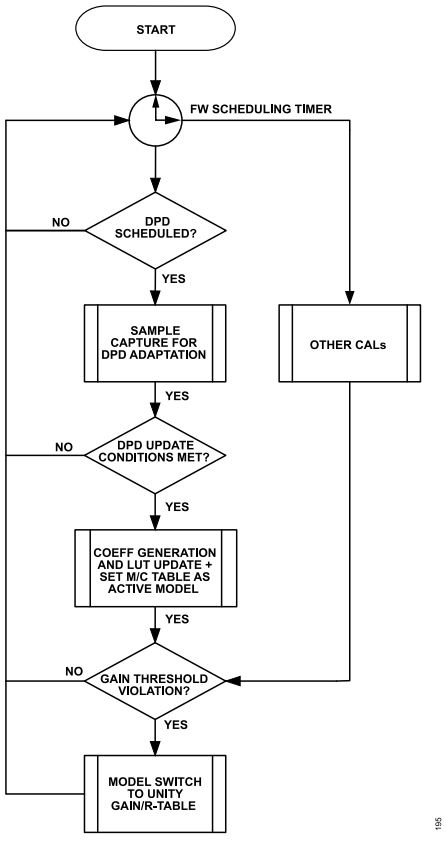


Figure 199. DPD Gain Monitoring State Machine

TDD LUT SWITCHING

For certain applications/power amplifiers which require a different DPD model to be applied to deal with transients when Tx is turned ON, a timing offset relative to Tx ON rising edge can be specified for switching models as shown in Figure 200. In this mode, two models can be programmed. This time-based model switching configuration is programmed with adi_adrv904x DpdTddLutSwitchCfgSet() where adi_adrv904x_DfeAppDpdActTddLutSwitch_t. txOnMdlSwitchDlyCnt set the time from TX_ON to model switching.

DIGITAL PRE-DISTORTION (DPD)

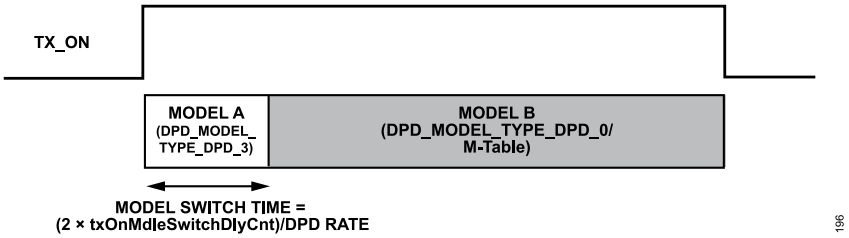


Figure 200. TDD LUT Switching

Table 115. TDD LUT Switching Modes

txOnMdl0SwitchEn	txOnMdl1SwitchEn	Description
0	0	TDD LUT switching disabled
0	1	Model B/M-Table only
1	0	Model A only
1	1	TDD LUT switching enabled

DPD ACTUATOR BYPASS

The ADRV904x DPD provides a mechanism to bypass pre-distortion through TRXx_CTRL pins. The TRXx_CTRL-based DPD actuator bypass is typically used for antenna calibrations in M-MIMO applications, where the pre-distortion must be disabled for the duration of antenna calibration to prevent pre-distortion from affecting antenna calibration accuracy. Figure 201 describes how this process is implemented.

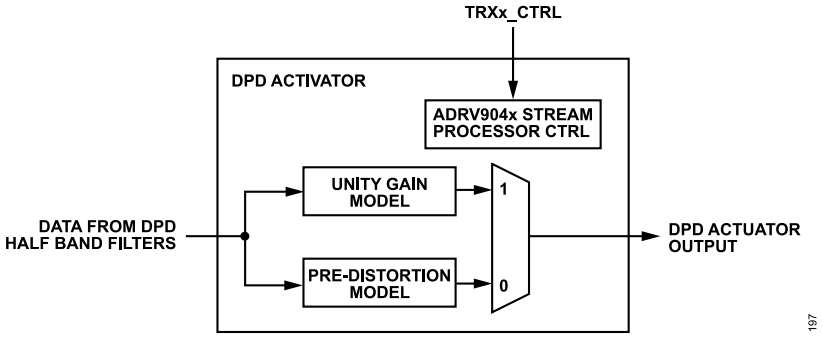


Figure 201. DPD Actuator Bypass through TRXx_CTRL

The TRXx_CTRL control for DPD actuator bypass is managed by the stream processor in the transceiver.

DPD STATUS

The user can obtain the status of the DPD tracking calibration during run time through the API command adi_adrv904x_DfeDpdCalSpecificStatusGet() which updates a data structure of type adi_adrv904x_DfeAppCalDpdStatus_t supplied by the user. The DPD status data structure returns the following information captured in Table 116.

Table 116. DPD Recovery Action Bit-mask Definition

Member	Data Type	Description
hdr.errorCode	UInt32	DPD error status. Refer to the file adi_adrv904x_dfe_app_err_codes_t.h for a full list of DPD error codes returned by the DPD status. If the DPD is functioning correctly, this parameter should return ADI_ADRV904X_DFE_APP_ERR_CODE_NO_ERROR.
hdr.percentComplete	UInt32	Percentage of DPD update completed.
hdr.iterCount	UInt32	No. of DPD updates scheduled.
hdr.updateCount	UInt32	No. of successful DPD updates.
activeModel	UInt32	Currently active DPD model (M-Table/C-Table/U-Table) described in Table 103.
updatedModel	UInt32	Latest updated DPD model (M-Table/C-Table/U-Table) described in Table 103.
stability	float()	Current values of DPD stability metrics described in Table 110.

DIGITAL PRE-DISTORTION (DPD)

RECOMMENDED SEQUENCE FOR ENABLING THE DPD TRACKING CALIBRATION

The sequence for running DPD tracking calibrations is shown in Table 117.

Table 117. DPD Tracking Calibration Bringup Sequence

Step	Action	APIs Used
1	Program the device and run initial calibrations (including TxQEC initial calibration) with the PA turned off.	(Utility function adi_daughterboard_Program() can be used to program the device)
2	Set up external Tx to ORx mapping	adi_adrv904x_TxToOrxMappingSet()
3	Adjust ORx attenuation to an appropriate value to avoid saturation. The default attenuation setting in the ADRV904x is 0 (0 dB attenuation).	adi_adrv904x_OrxAttenSet()
4	Run the Tx external LO Leakage initial calibration.	adi_adrv904x_InitCalsRun()
5	If using the ADRV904x CFR, configure the CFR settings.	adi_adrv904x_CfrConfigSet()adi_adrv904x_CfrCorrectionPulseWrite()
6	Load the DPD model.	adi_adrv904x_DpdModelConfigDpdSet()
7a	Assert DPD Reset.	adi_adrv904x_DpdReset()
7b	If a unique DPD model is required to be applied to each Model.	
8	Set up DPD mode of operation, DPD peak search window size and low power threshold.	adi_adrv904x_DpdTrackingConfigSet()
9	Set up the DPD capture configuration.	adi_adrv904x_DpdCaptureConfigSet()
10	Set up DPD fault conditions and recovery actions (optional).	adi_adrv904x_DpdStabilityCfgSet()
11	Enable Tx QEC and Tx LO Leakage tracking Calibrations.	adi_adrv904x_TrackingCalsEnableSet()
12	Enable DPD Tracking Calibration.	adi_adrv904x_DfeTrackingCalsEnableSet()

DPD STABILITY METRICS CHARACTERIZATION

The following stability metrics are at the disposal of the user for defining stability:

- Transmit power thresholds.
- Observed power thresholds.
- Direct EVM—Difference between measured pre DPD transmitted and observed samples.
- Indirect EVM—Difference between measured post DPD transmitted and observed samples.
- Indirect Error—Difference between measured post DPD transmitted and predicted samples. Predicted samples are obtained by applying the DPD model to the observed samples.

All the above metrics are user configurable. The Measuring DPD Adaptation Performance Through Direct EVM and Indirect Error section provides characterization data which might provide some guidance regarding factors that could influence the configuration of the stability metrics defined.

Measuring DPD Adaptation Performance Through Direct EVM and Indirect Error

Figure 202, Figure 203, and Figure 204 show three cases where the stability metrics direct EVM and indirect Error parameters are compared for different ACLR performance levels of an NR100 signal TM3.1 signal.

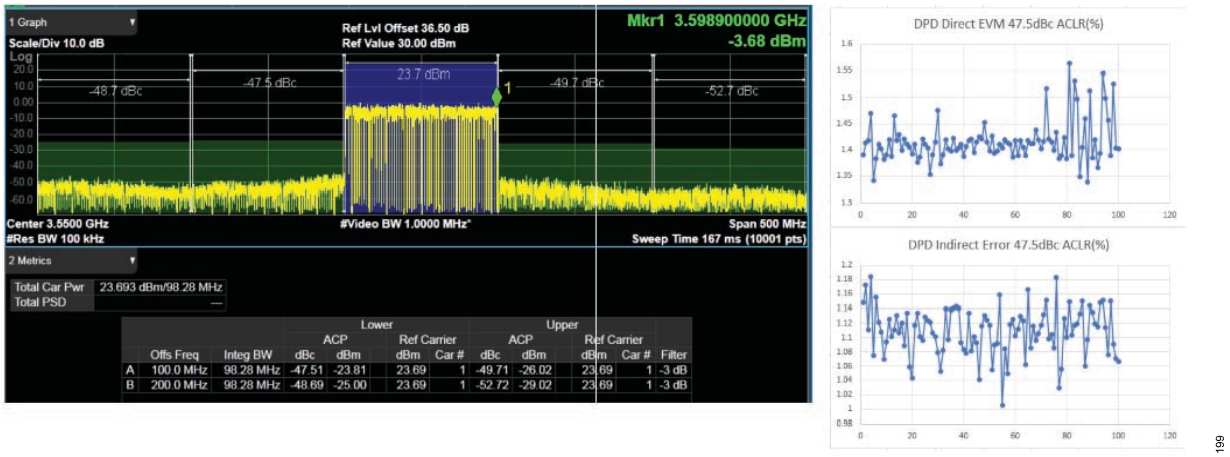
Case 1: The ACLR performance is ~46.6 dBc in L3 channel and corresponds to a direct EVM of 2%.

DIGITAL PRE-DISTORTION (DPD)

198

Figure 202. Bad ACLR Performance Correlation with a High Direct EVM Value

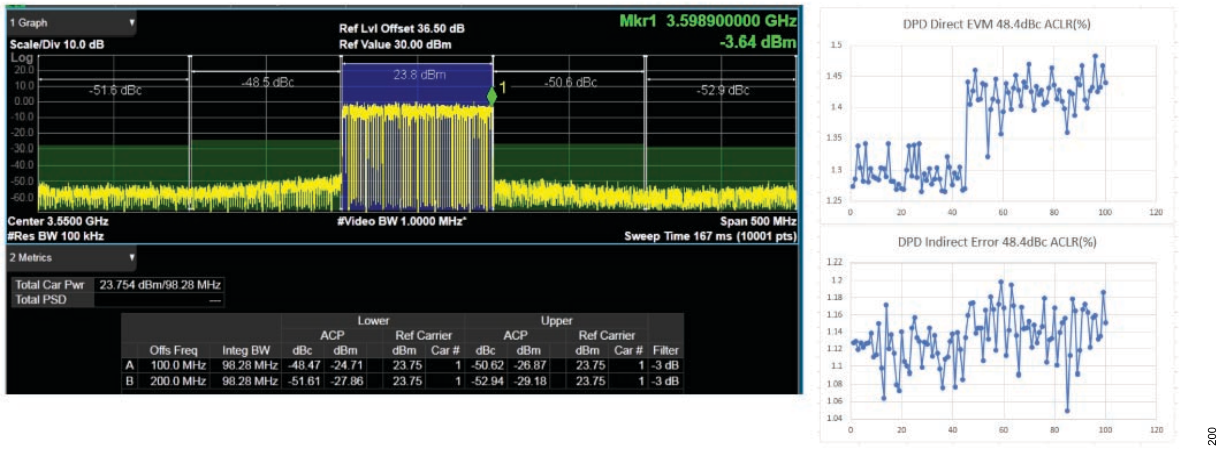
Case 2: The ACLR performance is better (~47.5 dBc) than Case 1, which also corresponds to reduced direct EVM.



199

Figure 203. Improved ACLR Performance Corresponds to Improved Direct EVM

Case 3: The ACLR performance is best amongst the three cases (~48.5 dBc) which is reflected in the reduced direct EVM numbers.



200

Figure 204. Direct EVM for the Best ACLR Performance

DIGITAL PRE-DISTORTION (DPD)

Observation Receiver Attenuation vs. Stability Metrics

Figure 205 shows the trend for the EVM and error stability metrics for different observation receiver channel attenuation values. It can be observed that as observation receiver attenuation is increased, the error percentage also increases. The same might be true for a low SNR transmitter to observation receiver channel. It is advised to increase the threshold for stability metrics as the observation receiver attenuation increases or the channel SNR decreases.

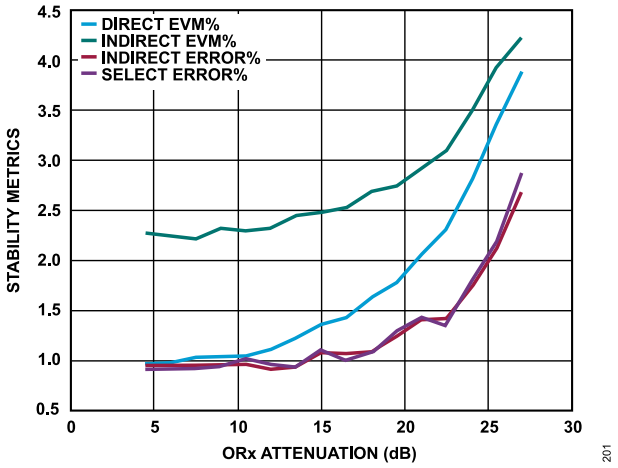


Figure 205. ORx Attenuation vs. Stability Metrics

Observation Receiver Interference

Although unlikely, any interference in the observation receiver channel can cause DPD instability and can cause the firmware to derive wrong DPD coefficients resulting in poor performance as shown in Figure 206, which includes the bench characterization of observation receiver interference levels affecting ACLR and stability metrics. Depending on the application and the ACLR performance, the user can budget for the threshold of stability metrics.

The data in Figure 206 is captured with a 100 MHz interference signal fed into the ORX Path. The actual wanted ORX signal is −4 dBm and the 100 MHz BW interference signal level is varied from −50 dBm to 0 dBm. Over the Interferer power sweep range, ACLR, Direct EVM, and Indirect EVM are measured and plotted. ORX SINR is the ratio of the wanted signal level to the unwanted interferer signal level, for example, ORX SINR is −39 dB for an unwanted interferer signal level of −43 dBm and a wanted signal level of −4 dBm.

At ORX interference of SINR of −39 dB, the DPD metrics and ACLR start degrading. It is up to the user to decide on the thresholds till what interference level, the ACLR is meeting the specifications and make sure that the Interference is limited to the target limit in the system and make under any condition, the interference would not be beyond the SINR limit.

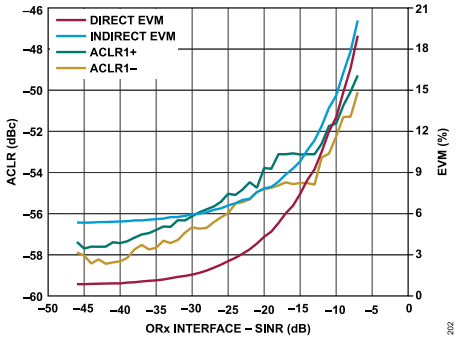


Figure 206. ORx Interference vs. Stability Metrics

DIGITAL PRE-DISTORTION (DPD)

Transmit Signal vs. Stability Metrics

Shown in Figure 207 is the degradation of stability metrics with decreasing transmitter signal power. When the signal level is close to –36 dBFS, it can be observed that the EVM percentages are close to 5%. At a transmitter signal level close to –46 dBFS, the EVM percentages are close to 15%, causing further ACLR degradation.

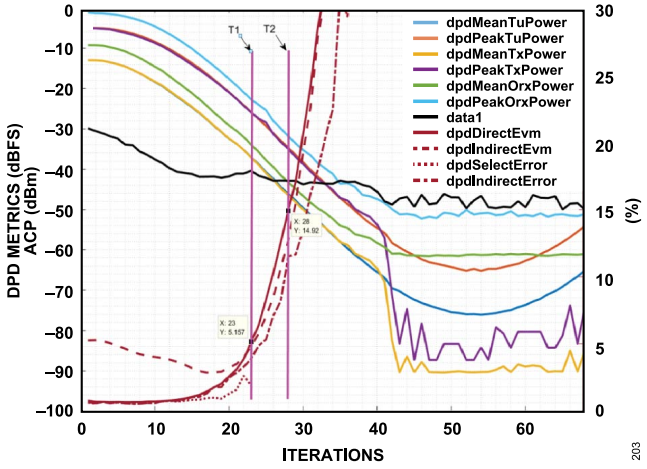


Figure 207. Tx Signal Level vs. ACLR Degradation

DPD Stability Metrics Summary

The following are some of the scenarios that could cause degradation of DPD performance; however, it is advised that the user characterize the system under test for EVM corruption that is specific to the system or conditions prevalent before configuring the thresholds:

- DPD performance can be measured by direct EVM. The direct EVM numbers are lower when the performance on the DPD adaptation is good.
- As ORx attenuation increases, an increase in EVM percentages can be observed.
- Interference/high noise levels in Tx to ORx channels can cause the EVM and error percentages to increase. Fault conditions and corresponding recovery actions can be defined for EVM numbers to avoid bad DPD updates.
- As Tx signal level decreases, the EVM percentages increase. However, an argument can be made that DPD might not be required at lower signal levels for certain PAs.
- Another factor to note is the DPD model that the user configures the part with. An incompatible prior DPD model configured by the user can cause the EVM and error percentages to increase leading to poor DPD performance.
- Catastrophic conditions such as loss of signals can also lead to high EVM and error percentages which can be monitored by the user.

DPD CHARACTERIZATION FOR OPTIMIZING M-THRESHOLD

A DPD characterization test for optimizing M-Table threshold in DPD Mode 2 includes the following steps:

1. Bring up DPD with a full power, full bandwidth signal (for example, TM3.1a at −14 dBFS and 100 MHz BW)
2. Once DPD converges, turn off DPD.
3. Sweep power in 1 dB steps from full power to ~15 dB backed off level.
4. Record ACPR, ACP and EVM of demodulated data at each level. An example tabulation of characterization data is shown in Figure 208.

DIGITAL PRE-DISTORTION (DPD)

Channel Power		Adjacent Channel Power Relative(dBc)				Adjacent Channel Power Absolute(dBm)				
Test Signal RMS(dBFS)	Carrier Power(dBm)	Lower B	Lower A	Upper A	Upper B	Lower B	Lower A	Upper A	Upper B	EVM(%rms)
-14	37	-50.4	-49.7	-49.3	-49.8	-13.1	-11.2	-11.6	-12.05	1.52
-15	36.1	-49.3	-47.9	-48.3	-49.3	-13.6	-10.57	-10.43	-12.79	1.48
-16	35.1	-48.7	-46.3	-46.4	-48.9	-14.18	-9.6	-10.6	-13.7	1.513
-17	34.5	-48.4	-44.8	-44.4	-48.5	-15.54	-10.5	-10.21	-14.4	1.633
-18	33.6	-47.8	-43.2	-43.8	-48.2	-15.8	-10.1	-10.7	-15.84	1.69
-19	32.3	-47.7	-42.9	-43.6	-47.8	-14.8	-10.74	-10.5	-14.8	1.76
-20	31.5	-46.3	-42.8	-43	-47	-14.74	-10.33	-10.63	-14.45	1.87
-21	30.7	-45.8	-42.1	-42.2	-46.2	-15.01	-10.41	-10.64	-15.13	1.94
-22	29.7	-44.9	-41.2	-41.2	-45.4	-15.7	-10.6	-10.6	-15.4	2.08
-23	28.8	-44.1	-40.3	-40.6	-44.7	-15.5	-10.5	-10.5	-15.5	2.33
-24	27.9	-43	-39.4	-38.6	-43.6	-15.15	-10.49	-10.93	-15.94	2.49
-25	27	-42.6	-38.7	-38.8	-42.9	-15.4	-10.58	-10.91	-16.23	2.69

204

Figure 208. DPD Characterization for Optimizing M-Threshold in DPD Mode 2

For the example shown in Figure 208, the characterization data relative to a 45 dBc ACP specification and 2.5% EVM is plotted in Figure 209. The figure shows that, although DPD performance is sustained across power levels, the EVM violates the 2.5% specification at approximately −24 dBFS. Based on the characterization data, it can be determined that that optimum M-Table threshold in DPD mode 2 is −24 dBFS.

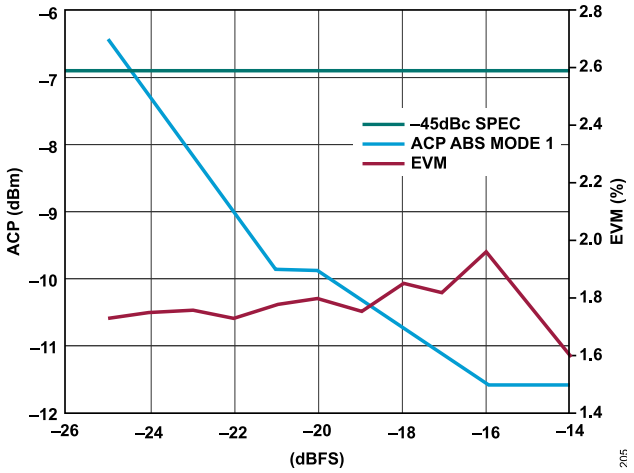


Figure 209. DPD Characterization Data Plot for Characterizing M-Threshold

DPD API FUNCTIONS

Table 118. DPD API Functions

API Method Name	Comments
adi_adrv904x_DpdModelConfigDpdSet()	Configures the base DPD model for the requested Tx channel(s).
adi_adrv904x_DpdModelConfigDpdGet()	Retrieves the DPD model feature set for the requested Tx channel from the device.
adi_adrv904x_DpdReset()	DPD reset for a requested Tx channel.
adi_adrv904x_DpdCaptureConfigSet()	Sets the DPD capture configuration.
adi_adrv904x_DpdCaptureConfigGet()	Returns the DPD capture configuration.
adi_adrv904x_DpdTrackingConfigSet()	Sets the DPD tracking calibration configuration.
adi_adrv904x_DpdTrackingConfigGet()	Returns the DPD tracking calibration configuration.
adi_adrv904x_DfeDpdCalSpecificStatusGet()	Returns the status of the DPD calibration specific status.
adi_adrv904x_DpdPowerMeterConfigSet()	Sets the DPD power meter configuration.
adi_adrv904x_DpdPowerMeterConfigGet()	Returns the DPD power meter configuration.
adi_adrv904x_DpdStabilityCfgSet()	Sets the DPD Stability/Robustness configuration.
adi_adrv904x_DpdStabilityCfgGet()	Returns the DPD Stability/Robustness configuration.
adi_adrv904x_DpdTddLutSwitchCfgSet()	Sets the TDD LUT switching configuration.
adi_adrv904x_DpdTddLutSwitchCfgGet()	Returns the DPD TDD LUT switching configuration.

DIGITAL PRE-DISTORTION (DPD)

DPD CTC MODE 1

Charge Trapping Effects of GaN PA

Gallium Nitride (GaN) power amplifier architectures are popular due to their high efficiency, higher power density, and high-frequency wide bandwidth operation capabilities. However, GaN PA architectures also suffer from charge-trapping effects. The charge trapping effects can be understood as input-dependent gain modulation of the PA. The manifestation of charge-trapping effects is captured in Figure 210. The Gallium Nitride (GaN) Inter-modulation Distortion (IMD) dynamics are multi-dimensional. They can last several milliseconds compared to LDMOS where the charge trapping effects are uni-dimensional and last over a few nanoseconds.

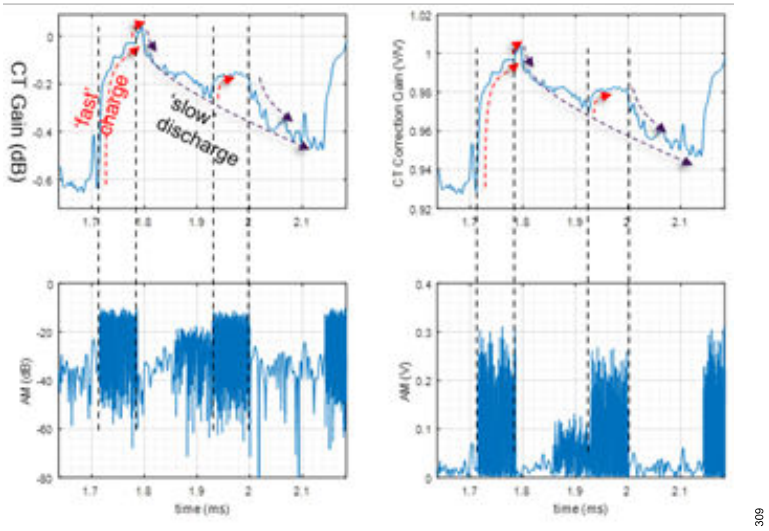


Figure 210. Charge Trapping Effects in a Gallium Nitride Power Amplifier

CTC Mode 1 Overview

GaN (Gallium Nitride) power amplifiers exhibit charge trapping effects resulting in poor EVM with TM2-like bursty waveforms. The EVM, measured with TM2 waveform, can be further improved through the CTC Mode 1 in comparison to regular DPD. CTC Mode 1 enhances the capture selection by using multiple captures targeted to specific peak powers. This helps to make DPD more relevant on average to data that is bursty at the symbol level.

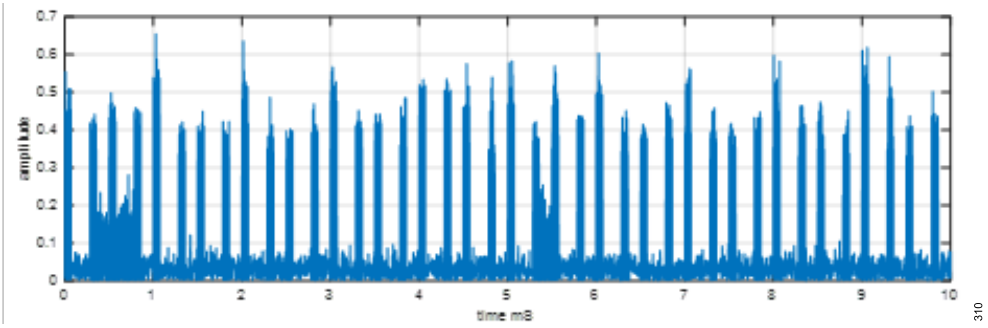


Figure 211. ETM2 Amplitude Frame Plot (10 ms)

DIGITAL PRE-DISTORTION (DPD)

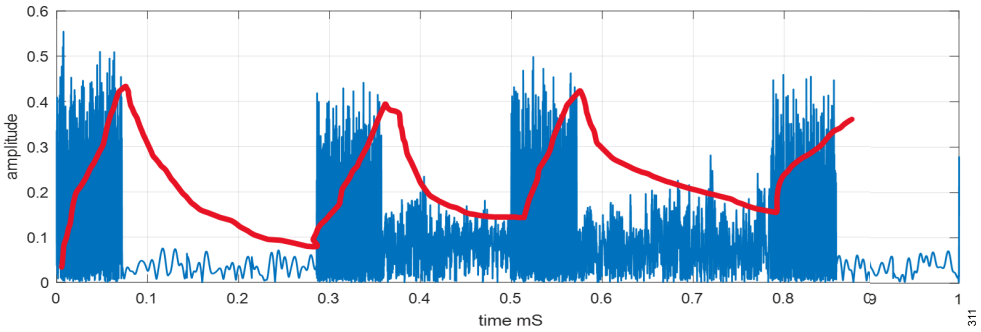


Figure 212. Zoomed in Data of 1 ms

Regular DPD data captures are peak search captures, that capture the highest peak data at the center ±2k samples as shown in Figure 213.

This capture does not present a composite view of charge trapping effects (red trace overlaid above) for DPD to correct.

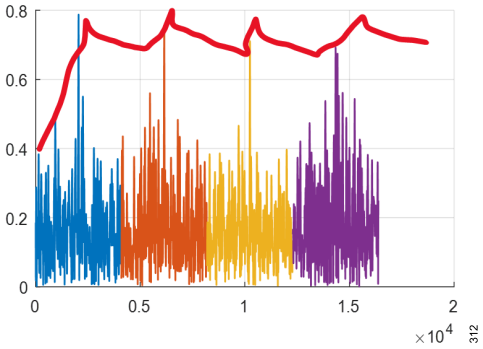


Figure 213. Regular DPD Capture

CTC Captures are special captures that capture data within a specified peak amplitude range, that will present a more composite view of charge trapping effects (red trace overlaid on capture in Figure 213).

Given the diversity of capture data, the DPD has a better chance of correcting for the trapping/de-trapping effect shown in the red trace as shown in Figure 214.

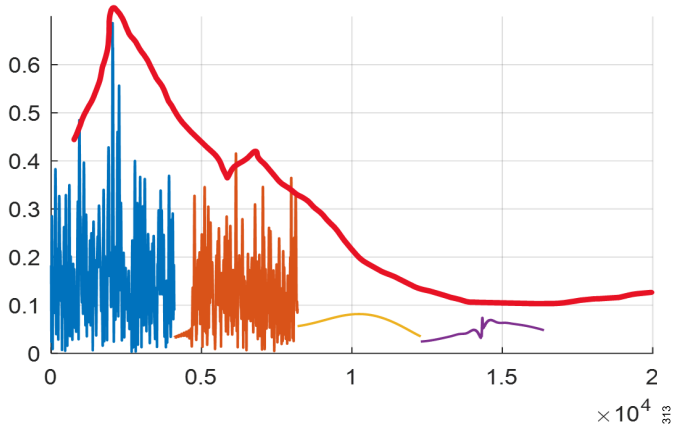


Figure 214. CTC Capture

CTC Mode 1 Principle of Operation

The ADRV9040 DPD capture engine has a special mode in which the user can specify the amplitude constraints for a specific capture. Furthermore, the user can specify the weight at each amplitude level. When the amplitude range condition is met, the peak ±2k samples are performed by the DPD engine.

DIGITAL PRE-DISTORTION (DPD)

A power profile can be created and the data corresponding to certain power levels are selected to create a composite capture data for DPD correction.

In the example shown in Figure 215, targeted captures are performed at specific amplitudes marked with (*) on the waveform. The DPD is presented with composite data with different amplitude ranges so that the correction can take charge of trapping effects at various amplitudes and generate a more accurate average response to cancel these effects.

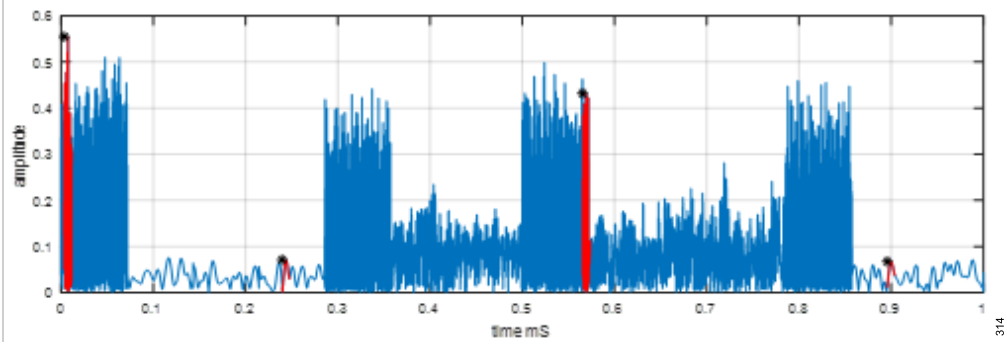


Figure 215. Targeted Amplitude Peaks

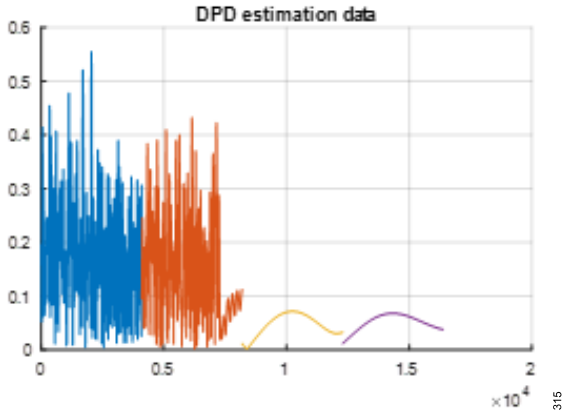


Figure 216. Aggregated Capture Data

CTC Mode 1 API Functions

The ADRV9040 CTC Mode 1 is enabled by specifying CTC Mode 1 model via adi_adrv904x_DpdModelConfigDpdSet( ) API.

The structure, DfeAppCalDpdModelDesc_tàdpdPartial must be set to a value of CTC_1. See the SW package for more details.

CTC Mode 1 configuration parameters are specified in Table 119 through adi_adrv904x_DfeAppCalDpdAdpCfg_t structure.

Table 119. CTC Mode 1 API Functions

Parameter	Description
ctc1StatsSampleNum	Number of batches of back-to-back captures monitored by the DPD capture engine to generate capture statistics.
ctc1PeakRankRatio	Represents 3 user specified peak rank percentile values as described in the previous slide.
ctc1CaptureWeighting	Weighting assigned to each peak percentile as shown in the example from previous slide. Maximum no. of batches supported is 8. The user can assign a weighted value between 0 to 8 batches for a specific peak percentile.
ctc1DeltaPercentage	Error tolerance value for each peak percentile. Typical value is ±5% around a peak percentile value.

DIGITAL PRE-DISTORTION (DPD)

Table 119. CTC Mode 1 API Functions (Continued)

Parameter	Description
ctc1ManualEnabled	If this bit is set to 0, a special automatic mode is enabled. In the automatic mode, the peak rank ratio and the capture weighting are automatically determined by the firmware. User supplied peak rank ratio and capture weighting are ignored.

CTC Mode 1 Example Capture Profile

```
· NumofDPDSamples = 3x4096
· PeakRankRatio = {0.95 0.75 0.3}
· Delta percentage = 0.05
· capture weighting = {1 1 1}, th
§ 1 capture batch of 4096 samples performed with peak between 95th percentile peak +/- 5% = 100% - %
§ 1 capture batch of 4096 samples performed with peak between 75th percentile peak /- 5% = 80% -0%
§ 1 capture batch of 4096 samples performed with peak between 30th percentile peak +/- 5% = 35% - 25%
```

CREST FACTOR REDUCTION (CFR)

This section of the user guide describes the CRF functionality of the ADRV904x and hardware blocks inside CFR.

The ADRV904x provides Crest Factor Reduction (CFR) to assist in keeping power amplifiers (PA) linear. A typical communications RF sub-system consists of an antenna, Power Amplifier (PA), and RF transceiver that translates digital baseband signals to RF, as shown in Figure 217.

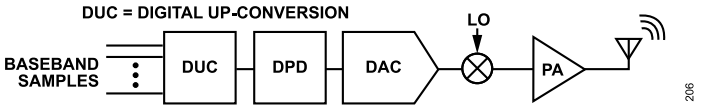


Figure 217. Typical RF Transmitter

It is highly desirable to drive the PA at the highest input power possible without having the PA saturate. Most modern communications protocols such as Long Term Evolution (LTE) are Orthogonal Frequency Division Multiplexing (OFDM) based in which the final waveform is an orthogonal summation of subcarriers that carry information, and where each subcarrier has its own center frequency and modulation scheme. In the time domain, sometimes the peaks of these subcarriers can align to produce an aggregate large OFDM waveform peak (shown in Figure 218).

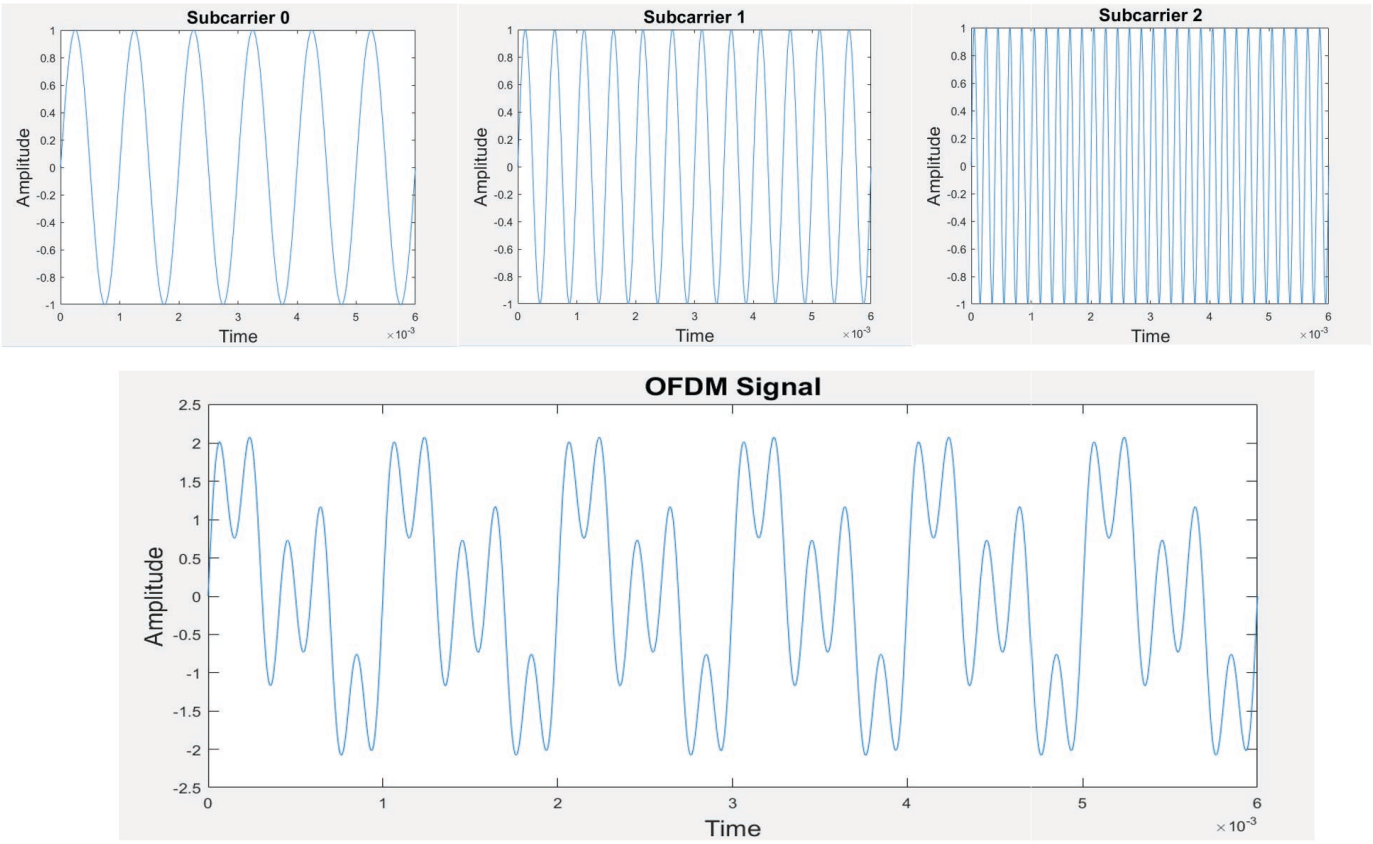


Figure 218. Example Illustration of Orthogonal Summation of Subcarriers Causing Large Peaks in an OFDM Waveform

These peaks increase the overall dynamic range needed for the OFDM signal through a signal chain, leading to an increase in the Peak to Average Ratio (PAR) of the signal.

Modern communication PAs used to amplify such OFDM waveforms are only linear for a certain power range. Most of the input signal (average power) will be within this linear range. However, the signal may have peaks that exceed the PA's linear operation range. To avoid saturation of the output signal due to these peaks, we could potentially attenuate the desired signal. This method ensures that the range required by the signal is within the PA's linear range. However, this is undesirable as it would reduce the average power at the expense of maintaining a given PAR, making the system less efficient. An alternative to attenuation is to use CFR – where instead of attenuating the whole signal, we attenuate portions of the signal that are above the PA's linear range. This results in a constant output power while reducing the PAR and thus ensuring that the signal remains within the PA's linear range. This is summarized in Figure 219.

CREST FACTOR REDUCTION (CFR)

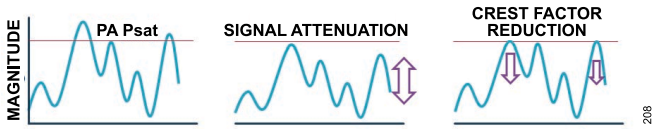


Figure 219. Effect of Signal Attenuation vs. CFR

It is important to note that CFR leads to higher in-band and out-of-band noise levels. This effect results in Error Vector Magnitude (EVM) degradation while also increasing the noise power spectral density, resulting in increase in Adjacent Channel Leakage Ratio (ACLR). It is important to optimize the CFR algorithm to make sure that the CFR block's impact is within the customer's system level specifications (derived from 3GPP specifications or other regulatory standards).

CFR ALGORITHM OVERVIEW

Several CFR algorithms and implementations exist in the literature—only a few have been commercially viable. Clipping and filtering and pulse cancellation are two of the more popular techniques. Even though optimal cancellation of peaks is technically achievable (target PAR requested is met exactly), the latency requirements imposed by modern communication standards makes the design of a real-time CFR engine quite challenging. Spectral regrowth of corrected peaks by interpolating stages in the datapath, such as interpolating filters and Digital to Analog Converters (DACs), is also a concern. An ideal CFR block has very low latency and zero missed peaks.

The ADRV904x implements CFR using a variation of the pulse cancellation technique by subtracting a pre-computed pulse from the detected peaks to bring the signal within the PA's linear range. The CFR block consists of three copies of CFR engines, each of which uses a detection threshold to detect the peaks and a correction threshold to which the detected peaks are attenuated. The pre-computed pulses may be stored within the device at the start-up or be updated during run-time. These spectrally shaped correction pulses are subtracted from the data stream to bring the signal within the PA's linear range. The correction pulse needs to be spectrally shaped to manage the noise leakage into adjacent bands. Figure 220 shows the architecture implemented in each of the CFR engines.

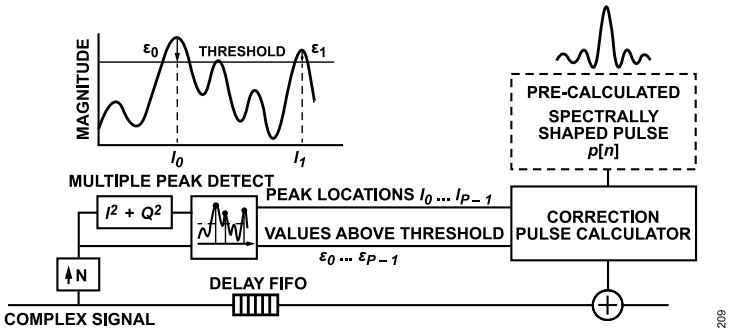


Figure 220. CFR Engine Architecture in ADRV904x

The complex I/Q signal (transmitter data) goes into a variable delay FIFO and correction is applied at its output. The input data also goes into an interpolator, which can interpolate by 1×, 2×, or 4× times the input sample rate. This interpolated data is then fed into a peak detector. The peak detector determines the location of all the peaks in the signal and the delta by which they exceed the programmable threshold. These peaks can be corrected by using a pre-computed spectrally shaped pulse (also called correction pulse) which is stored in a pulse RAM. Multiple peaks can be simultaneously cancelled by time-shifting and combining these spectrally shaped pulses.

This corrected output signal is then passed to two more similar CFR engines to correct missed peaks, as well as peaks that need further correction after passing through the first engine.

The CFR block within the transceiver consists of three cascaded CFR engines followed by a hard clipper (located before DPD actuator) to clip the few peaks that are skipped by all three CFR engines. At the output of each engine, there exists a mux that can be programmed to bypass CFR or apply correction (shown in Figure 221).

CREST FACTOR REDUCTION (CFR)

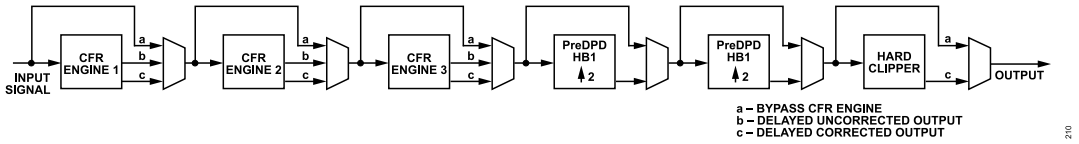


Figure 221. ADRV904x Cascaded CFR Engine Blocks

The Hard Clipper unit at the very end of the CFR block clips any signal above a programmable threshold. This threshold generally should not be lower than the thresholds used in the CFR engines as it will lead to increase in noise (both in-band and out of band). Each of the individual units above can be bypassed depending on the use case and desired performance.

The transceiver incorporates the CFR block at the very beginning of the transmitter datapath right after the JESD block. Note that the CFR engine can handle a maximum input sampling rate of 983.04 MHz with an internal interpolation up to 2×. Therefore, the maximum sampling rate for the base and correction pulses is 1966.08 MHz.

OVERVIEW OF BLOCKS USED IN CFR

This section provides a summary of the major blocks used in each of the CFR engines.

Delay RAM FIFO

The CFR Delay RAM FIFO is used to delay the input signal so that there is enough time to detect, process, and correct peaks within the input signal. The required latency per engine is roughly equal to 1.5 × half_pulse_length ÷ interpolation. Adding extra latency is acceptable, but generally not preferred due to low latency requirements on the full system.

Interpolator

The interpolator can be programmed to interpolate the data by 4, 2, or 1. The main purpose of the interpolator is to produce finer timing resolution so that the peak detector can find the location of the peaks more accurately. The interpolator takes in the transmitter data and outputs the interpolated data to the peak detector.

Peak Detector

The purpose of the peak detector block is to find groups of peaks that are above a certain threshold and then output the peak I and Q values along with peak locations relative to the first peak. The peak detector looks for peaks that are above a certain programmable threshold. Since peaks don't occur very often, a coarse peak detection on every sample is performed using Equation 42.

$$I \quad or \quad Q > \frac{Threshold}{\sqrt{2}} (Course \quad peak \quad detection) \tag{42}$$

If this condition is false, then there is no need to compute the $I^2 + Q^2$ complex calculation. This saves some power by avoiding frequent multiplications. In the case when the sample is larger, the magnitude of the incoming complex signal is checked to determine if it is above the threshold, using the inequality shown in Equation 43 .

$$I^2 + Q^2 > Threshold^2 \ (Fine\ peak\ detection) \tag{43}$$

If the condition in Equation 43 is true, multiple consecutive samples that exceed the threshold can be found. These peaks can be corrected using a single pulse which is superimposed on to the maximum valued peak of the group of samples.

Pulse RAM

The pulse RAM holds the correction pulse for the carrier. It is possible to load two correction pulses corresponding to two desired carrier configurations. The user can pre-load these two correction pulses and be able to switch between two carrier configurations on-the-fly. This mode is known as the hot-swap mode. When CFR correction pulses are loaded, only half of CFR correction pulse without center tap is required to transport it to Pulse RAM. Correction Pulse maximum size can be multiple of 4 and up to 512.

CFR PARAMETERS AND CORRECTION PULSE LOADING DURING INITIALIZATION

At initialization, CFR parameters (delay, over-sampling rate, thresholds) and CFR correction pulse will be included in Profile. bin which is transported to Radio CPU and Pulse RAM respectively from the Configurator. Assume that there are two carrier configurations, binary profile

CREST FACTOR REDUCTION (CFR)

includes both CFR parameters and CFR correction pulse 1, 2. When it is transported to BBIC, it will be extracted to two parts, CFR parameters and CFR correction pulse 1, 2 as shown in Figure 222.

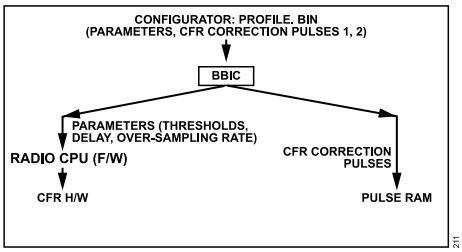


Figure 222. CFR Parameters and Correction Pulse Loading During Initialization

DYNAMIC CONFIGURATION CHANGES

Carrier Configuration Pulse Change

Pulse Configuration changes happen dynamically when the base station adds or drops carriers. CFR disabling CFR allows correction to queued correction to flush out before configuration changes. After flushing and carrier change, CFR settings will be changed, and CFR be reenabled. Device will have two sets of registers to support two carrier configurations and allow easy switching.

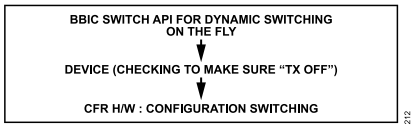


Figure 223. Stream Correction Pulse Dynamic Switching I on the Fly

For dynamic carrier configuration, there are two way to change CFR configuration. One is an automatic process where before CFR configuration change, device will check the current status (TX On, Off). When the status is TX OFF, device changes the CFR Configuration as shown in Figure 142.

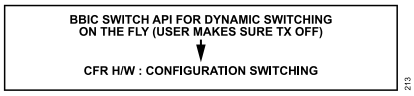


Figure 224. Immediate Correction Pulse Dynamic Switching II on the Fly

The second is an immediate process which BBIC API will generate parameters to CFR H/W directly using API. For the immediate process, BBIC need to check whether TX is ON or OFF before applying new CFR configuration. After CFR H/W receives the command, it flushes the unnecessary FIFO data and switches to a new configuration.

Threshold Changes On the Fly

CFR threshold can be set during or after initialization. Additionally, CFR threshold can be adjusted during runtime without having to turn respective Tx channel OFF.

CFR Model Results

This section provides test results (ACLR and EVM) of the CFR with different sample rates and correction pulses.

Table 120. Simulated CFR Performance

Sample Rate (MHz)	PAR In (dB)	PAR Out (dB)	ACLR In (dB)	ACLR Out (dB)	Avg EVM Out (%)	Pulse Length	Configuration (MHz)
245.76	11.45	7.99	80.05	75.22	3.0	1025	20, 80, 100
491.52	11.67	8.18	80.90	76.52	2.91	2049	20, 50 (Empty), 80, 50 (Empty), 100
491.52	11.66	8.18	80.96	76.57	2.98	2049	20,100 (Empty), 80, 100 (Empty), 100

CREST FACTOR REDUCTION (CFR)

Procedure for Updating Correction Pulses and CFR Thresholds

The ADRV904x also provides the flexibility to change correction pulses on the fly without needing to run CFR initialization calibration. The recommended procedure assumes user already successfully followed the initial procedure for setting up CFR given in Procedure for setting up CFR section. The recommended sequence to update correction pulses on-the-fly is as follows:

1. Set Configuration Inactive to update Configuration using adrv904x.dfe_cfr.CfrActiveCfgSet.
2. Set Pulse Inactive to update Pulse using adrv904x.dfe_cfr.CfrActivePulseSet.
3. Load the CFR correction pulse using adrv904x.dfe_cfr.CfrCorrectionPulseWrite.
   a. ramSelectMask needs to be ADI_ADRV904X_CFR_PULSE_SINGLESHOT = 0x1.
4. Program the CFR control configuration via adrv904x.dfe_cfr.CfrThresholdSet.
5. Verify the CFR control configuration via adrv904x.dfe_cfr.CfrThresholdGet.
6. Enable the CFR engine via adrv904x.dfe_cfr.CfrEnableSet.
7. Set Configuration Active to update Configuration using adrv904x.dfe_cfr.CfrActivePulseSet.
8. Set Configuration Active to update Configuration using adrv904x.dfe_cfr.CfrActiveCfgSet.
9. Program the Hard Clipper via adrv904x.tx.HardClipperConfigSet.

CFR API Functions

Table 121. CFR API Functions

API Method Name	Comments
adi_adrv904x.dfe_cfr.CfrThresholdSet()	Configure the CFR Threshold for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrThresholdGet()	Retrieves the CFR Threshold for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrCtrlConfigSet()	Configure the CFR Control for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrCtrlConfigGet()	Retrieves the CFR Control for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrCorrectionPulseWrite()	Writes the CFR pulse for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrCorrectionPulseRead()	Retrieve the CFR pulse for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrEnableSet()	Configure the CFR Enable for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrEnableGet()	Retrieves the CFR Enable for the requested Tx channel(s).
adi_adrv904x.tx.HardClipperConfigSet()	Configure the CFR HardClipper for the requested Tx channel(s).
adi_adrv904x.tx.HardClipperConfigGet()	Retrieves the CFR HardClipper for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrStatisticsCfgGet()	Retrieves the CFR Stastics for the requested Tx channel(s).
adi_adrv904x.dfe_cfr.CfrActivePulseSet()	Configure the CFR Active pulse for the requested Tx channel(s).
adi_adrv904x_CfrControlPeakDurationSet ()	Configure the CFR Peak Duration for the requested Tx channel(s).
adi_adrv904x_CfrControlPeakDurationGet ()	Retrieves the CFR Peak Duration for the requested Tx channel(s).
adi_adrv904x_CfrControlDelaySet ()	Configure the CFR Control Delay for the requested Tx channel(s).
adi_adrv904x_CfrControlDelayGet ()	Retrieves the CFR Control Delay for the requested Tx channel(s).

CLOSED LOOP GAIN CONTROL (CLGC)

CLGC OVERVIEW

Closed Loop Gain Control (CLGC) is a closed-loop tracking calibration that adjusts front-end Tx attenuations to maintain a constant desired gain in the Tx path from baseband input to PA output such that the PA output power is constant with respect to changes in temperature or other variations. The desired Tx gain is set by adjusting the Tx front-end attenuation such that the PA output reaches the desired power with the maximum nominal level of the baseband signal. The digital swing of the baseband signal is not affected, since the attenuation is applied on the analog front end of the Tx channel.

The PA output power may fluctuate around the target level over time due to temperature changes, and other factors, given a constant Baseband signal level. The CLGC can help reduce the fluctuation and meet the standards and regulatory requirements (for example, ±2 dB fluctuation is required by the LTE/5GNR standard). The Baseband signal level may change significantly due to the dynamics of power control and traffic load in a network (for example, dynamic range can be up to 20 dB in LTE). The CLGC can help keep the output power linearly proportional to the baseband signal level.

ELEMENTS OF CLGC

Shown below in Figure 225 is a simplified block diagram of the ADRV9040 CLGC system. In a nutshell, the CLGC algorithm observes the transmit data x(n), and the post-PA observation data y(n) and adjusts the transmit front-end attenuation such that the overall loop gain (ORx data / Tx data) remains constant.

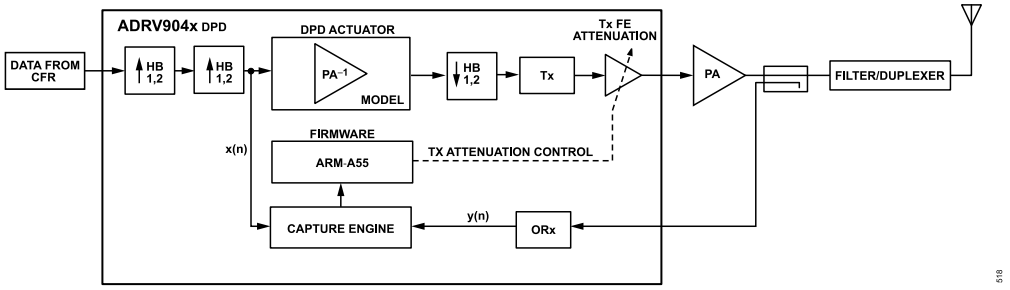


Figure 225. CLGC Simplified Block Diagram

The elements of the CLGC system are described as follows:

- Transmit Datapath. The digital baseband signal from the ADRV9040 de-framer output goes through a Crest Factor Reduction (CFR) block for reduction of the overall peak-to-average ratio of the signal followed by a digital up-converter which interpolates the baseband signal by a factor of 1×, 2×, or 4× for analyzing the baseband signal over the DPD analysis bandwidth. The upconverted data before the DPD actuator is then used as the reference Tx data for the closed loop gain control algorithm.
- Observation Datapath. The CLGC algorithm relies on observing any fluctuations in the PA output power through a feedback path. The feedback path is realized through the ADRV9040 Observation Receiver (ORx). The PA output data is coupled into the observation receiver, down-converted, and digitized for loop gain estimation and correction by the ADRV9040 firmware.
- Capture Engine. The Tx and ORx samples for CLGC measurement are captured and aligned in the capture engine, pre-processing is performed and the pre-processed samples are passed onto the embedded Arm processor for CLGC measurement and loop gain control.
- CLGC Processing. The CLGC algorithm is implemented in the transceiver's firmware running on an embedded Arm processor. The CLGC is a tracking calibration that tracks the observed and baseband data and uses a loop gain estimator to track changes in the overall loop gain. Refer to the CLGC Algorithm Overview section for an overview of the CLGC algorithm.

CLGC ALGORITHM OVERVIEW

The CLGC algorithm is designed to maintain a constant loop gain and overcome any minor fluctuations in the PA output power due to variations in temperature and other operating conditions. Loop gain is defined as the ratio of the power level of observed data to the power level of the baseband transmit data.

$$Loop\ \ Gain\ \ = \ \ \frac{ORx\ \ RMS\ \ Power}{Tx\ \ RMS\ Power} \tag{44}$$

The CLGC algorithm relies on the post-PA feedback data to estimate the loop gain and adjust the front-end Tx attenuation on the transceiver. Shown in Figure 226 is the observation points of the CLGC algorithm.

CLOSED LOOP GAIN CONTROL (CLGC)

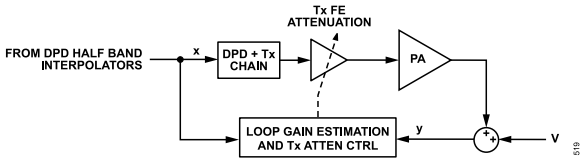


Figure 226. CLGC Algorithm Observation Points

The signal path from the reference baseband Tx input to the observed data for loop gain estimation can be divided into 4 sections as listed in Table 122. The total loop gain observed at the observation receiver (ORx) includes the front-end attenuation out of the transceiver, gain of the power amplifier, coupling attenuation for feedback, and the observation receiver (ORx) front-end attenuation.

Table 122. Observed Data for Loop Gain Estimation

Section	Gain Equation	Comments
Tx Section	*Xtx*(*n*) = *gTX* + (*x*(*n*) + *vtx_DAC_Quant*(*n*))	*gTX* = Total Tx Attenuation *x(n)* = Tx baseband data *vtx_DAC_Quant* = Tx DAC quantization noise
PA Section	*XPA*(*n*) = *gPA*. *Xtx*(*n*) + *vPA*(*n*)	*gPA* = PA gain at the PA operating point *vPA* = Additive noise determined by ACLR
Coupling Section	*Yorx*(*n*) = *gCPL*.*XPA*(*n*) + *vorx*(*n*)	*gCPL* = Coupling attenuation in the ORx path *vorx* = In band thermal noise and additive noise determined by noise figure in ORx path
ORx Section	*y*(*n*) = *gorx*. *Yorx*(*n*) + *vorx_ADC_Quant*(*n*)	*gorx* = Front end ORx attenuation *vorx_ADC_Quant* = ORx ADC quant noise

Total gain seen at y(n),

g = gorx. gCPL. gPA. gTX

Loop Gain Estimation and Convergence

A simplified equation of the loop gain estimation is described in Equation 45. The RMS power is calculated on captured Tx samples *x(n)* and ORx samples*y(n)*, where *n* is the total no. of samples captured per CLGC update. The value *n* is configured by the user. Refer to the CLGC Measurement section for details on CLGC sample capture.

$$Estimated\ \ Loop\ \ Gain\ \ = \ \ \frac{ORx\ \ RMS\ \ Power}{Tx\ \ RMS\ \ Power} = \frac{\sum_{n=0}^{N} |y(n)|^2}{\sum_{n=0}^{N} |x(n)|^2} \quad (45)$$

With the loop gain estimated, a loop gain error can be defined as

$$Loop\ \ gain\ \ error\ \ = \ \ \frac{Estimated\ \ Loop\ \ Gain}{Expected\ \ Loop\ \ Gain} \quad (46)$$

The Estimated Loop Gain is determined by the CLGC algorithm, and the expected loop gain is configured by the user. The objective of the CLGC is to reduce the Loop Gain Error ratio to 1 (linear) or 0 dB. The CLGC converges to the expected loop gain by tuning the transceiver front-end Tx attenuation as shown in Figure 226.

Observation

The ORx samples can be related to the Tx samples in the loop gain estimation engine through Equation 47.

$$y(n) = g \times x(n) + v(n) \quad (47)$$

where
x(n) is the input Tx samples from a user's BBIC.
y(n) is the output samples from ORx.
v(n) is the equivalent additive noise from the entire loop, which may include thermal noise, quantization noise, phase noise, and nonlinear distortions.

CLOSED LOOP GAIN CONTROL (CLGC)

g is the desired loop gain set by the user based on a nominal level of *x(n)* and the expected output power of the PA.

ENABLING THE CLGC TRACKING CALIBRATION

The CLGC tracking calibration can be enabled using the API, **adi_adrv904x_ClgcTrackingConfigSet()**.

The following are two pre-requisites for enabling the CLGC tracking calibration:

- The correct Tx to ORx mapping is required to be configured during initialization. The API adi_adrv904x_TxToOrxMappingSet () can be used for this purpose. The correct mapping of the transmit path to the observation path needs to be communicated to the firmware to capture data from the correct observation receiver.
- The DPD model needs to be loaded and the DPD reset has to be asserted. This step is required to initialize the DPD actuator.

Refer to the Recommended Sequence for Enabling CLGC Tracking Calibration section in this document for the list of commands to be used for bringing up CLGC.

Note that when the CLGC tracking calibration is enabled, the CLGC doesn't actively control the loop gain. The user has to enable CLGC tracking within adi_adrv904x_DpdTrackingConfigSet() API to enable active control of the CLGC loop gain when DFE tracking calibrations are enabled with adi_adrv904x_DfeTrackingCalsEnableSet(). Details regarding the CLGC modes of operation are captured in the CLGC Modes of Operation section.

Note that the CLGC tracking calibration works in synchronization with the DPD algorithm, and the CLGC is scheduled once per second by the firmware.

CLGC MODES OF OPERATION

The CLGC functionality can operate in two modes as follows:

- Passive loop gain measurement. This mode of operation is typically activated to determine the initial operating point of the PA. When a transmitter is activated, the user determines the PA operating point by sending traffic and measuring the overall loop gain from ORx to Tx. During this stage, the user can take advantage of the passive loop gain measurement mode, in which the CLGC algorithm simply measures the loop gain without actively adjusting the Tx front-end attenuation. Once the ideal operating point is determined, the user can then enable active loop gain control mode.
- Active loop gain control. In this mode, the CLGC measures the loop gain from ORx to Tx baseband and adjusts the Tx front-end attenuation to maintain the loop gain. This mode of operation is typically activated during runtime once the initial operating points are determined, and the initial ORx attenuation, and Tx attenuation settings are configured. For the active loop gain control mode, the user is required to configure the expected loop gain using the API **adi_adrv904x_ClgcTrackingConfigSet()** through the parameter **expLoopPowGain** in the data structure **adi_adrv904x_DfeAppCalClgcTrackCfg_t**.

The user can select the CLGC mode of operation through the **adi_adrv904x_ClgcTrackingConfigSet()** API, using the parameter **enClgc** in **adi_adrv904x_DfeAppCalClgcTrackCfg_t**data structure as shown in Table 123

Table 123. CLGC Mode of Operation Configuration

enClgc	CLGC Mode Activated
0	Passive loop gain measurement
1	Active loop gain control

Figure 227 captures a typical CLGC bring-up sequence during which the passive and active loop gain control modes are active at various stages.

In the passive loop gain measurement mode, the user can retrieve the ORx RMS power and Tx RMS power as well as the loop gain estimated by the CLGC algorithm using the API **adi_adrv904x_DfeClgcStatusGet()** . The structure **adi_adrv904x_DfeAppCalClgcStatus_t** has members **txPwr**, **orxPwr**, and **loopPowGain, txAttenAdjdB** which can be monitored to adjust the initial operating point of the PA and determine the desired loop gain.

The passive loop gain measurement mode is mostly used in a factory calibration setting. The optimal operating point for the PA is determined, the loop gain, Tx attenuation , and ORx attenuation values are noted, and the values are used in the field.

CLOSED LOOP GAIN CONTROL (CLGC)

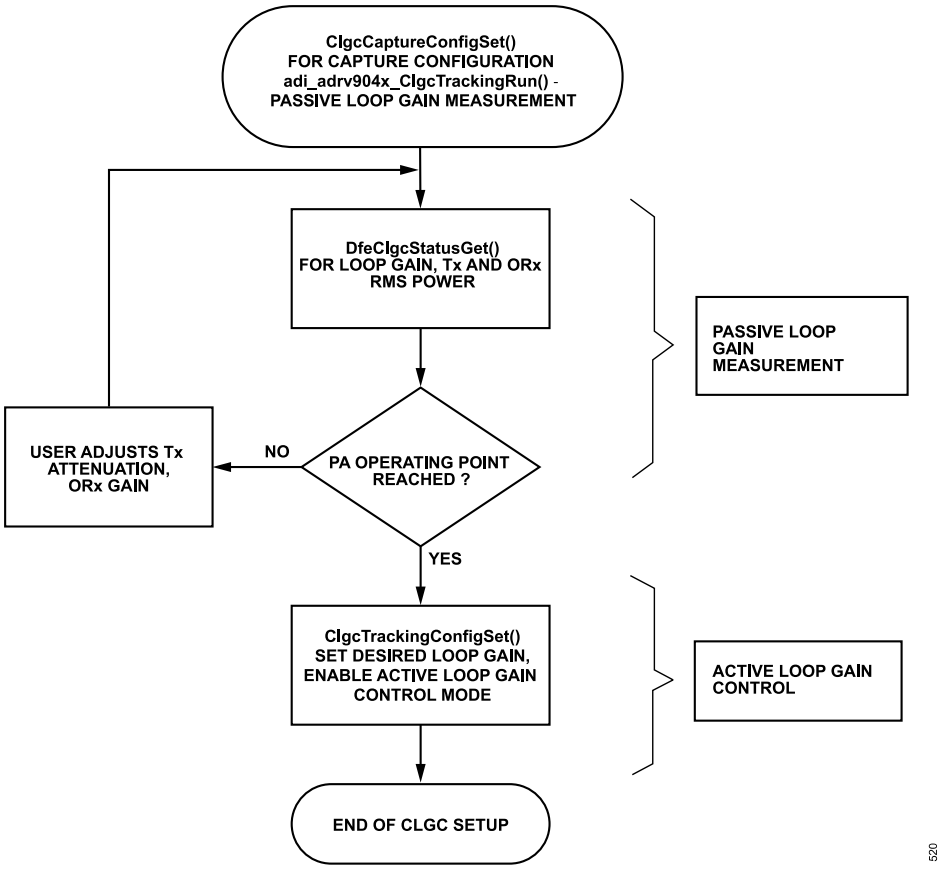


Figure 227. CLGC Modes of Operation During Bring-up

CLGC MEASUREMENT

The CLGC measurement cycle is common to both passive loop gain measurement and active loop gain control modes. In active loop gain control mode, the Tx attenuation is also adjusted by the CLGC algorithm. The Tx attenuation adjustment is covered in the TSSI Mode section.

The following are two capture modes of CLGC operation supported:

- TSSI Mode. Supports power measurement through a TSSI power meter on the Tx and ORx path. This mode is generally preferred in the case of normal traffic such as TM3.1a.
- Peak Mode. Supports power measurement through the DPD peak search capture hardware. This mode is generally preferred in case of sparse traffic such as TM2a/SSB sync signals transmitted during carrier setup.

TSSI Mode

There are two TSSI power meters placed in the Tx (see Figure 228) and ORx paths (see Figure 229) for measuring power to estimate loop gain.

CLOSED LOOP GAIN CONTROL (CLGC)

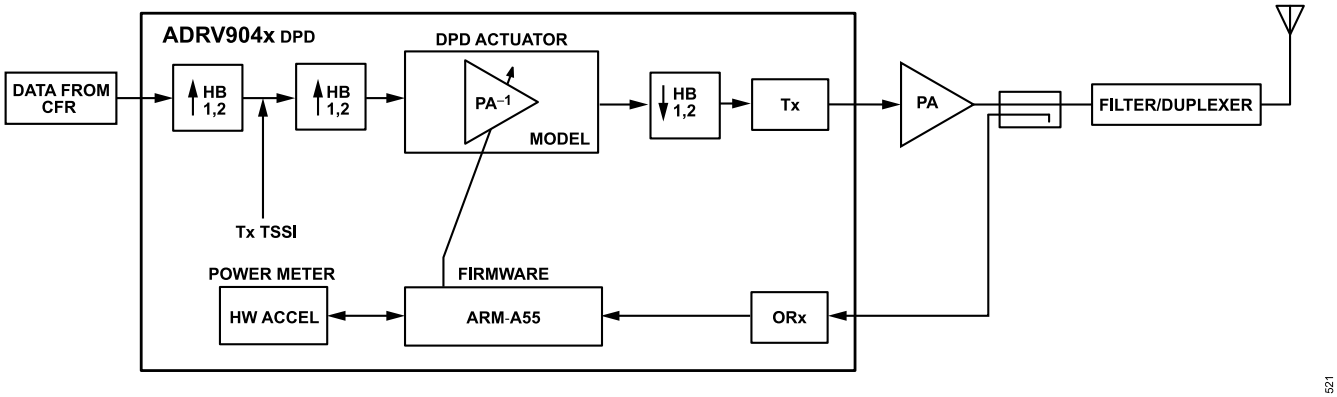


Figure 228. Tx TSSI Power Meter Tap Off Point

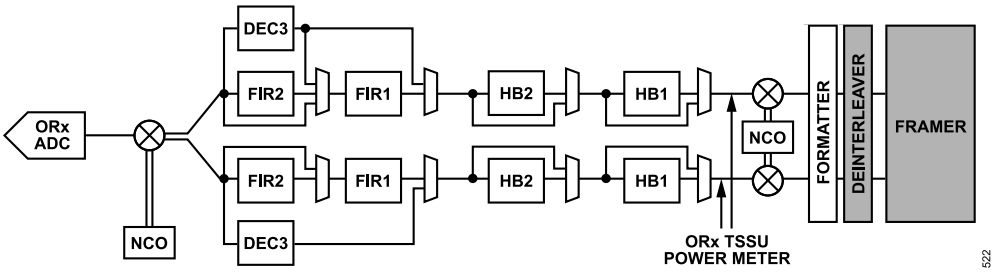


Figure 229. ORx TSSI Power Meter Tap Off Point

The loop gain is defined in Equation 48.

$$Loop\quad Gain = \frac{RMS\quad power\quad measured\quad by\quad ORx\quad TSSI}{RMS\quad power\quad measured\quad by\quad Tx\quad RSSI} \tag{48}$$

The rms power measured by the TSSI meter is given by

$$RMS\quad Power\quad = \frac{\Sigma_0^{N-1} I^2 + Q^2}{N} \tag{49}$$

where *N* = number of complex I/Q samples over which power is accumulated.

The TSSI power meter has the following specifications:

- The TSSI power meter can function at a maximum rate of 1 GHz.
- The TSSI power meter can accumulate power over a maximum duration of 234 ms.
- The Tx and ORx TSSI power meters are independently controlled through the Tx and ORx EN signals respectively.

Note that for TSSI operation in TDD mode, Analog Devices recommends Tx and ORx radio control signals to follow similar timing.

CLGC capture mode is specified by the parameter capMode defined in the data structure adi_adrv904x_DfeAppCalClgcCaptureCfg_t of adi_adrv904x_ClgcCaptureConfigSet. The list of parameters used to determine a successful capture is listed in Table 124.

Table 124. CLGC Capture Configuration

Parameter	Description	User Configuration Access
Capture Mode	CLGC captures can be either TSSI power detectors or peak detection.	adi_adrv904x_DfeAppCalClgcCaptureCfg_t. capMode
Capture Duration	When in TSSI capture mode this is the TSSI measurement time (µs).	adi_adrv904x_DfeAppCalClgcCaptureCfg_t. capDurationUs
Capture Period	When in TSSI capture mode this is the TSSI timeout timer (µs) and when in peak detection mode this is the window length.	adi_adrv904x_DfeAppCalClgcCaptureCfg_t. capPeriodUs

CLOSED LOOP GAIN CONTROL (CLGC)

Table 124. CLGC Capture Configuration (Continued)

Parameter	Description	User Configuration Access
Minimum Tx Power Threshold	If the Tx power is below this user configured threshold, then CLGC will not run.	adi_adrv904x_DfeAppCalClgcCaptureCfg_t. minTxPowThres
Minimum ORx Power Threshold	If the ORx power is below this user configured threshold, then CLGC will not run.	adi_adrv904x_DfeAppCalClgcCaptureCfg_t. minOrxPowThres

Peak Mode

The Peak Mode that supports power measurements through DPD peak search capture hardware. Refer to the Peak Search Window and Peak Detection Based Capture section for more details. This mode is generally preferred in case of sparse traffic such as TM2a/SSB sync signals transmitted during carrier setup.

Tx and ORx Qualifying Threshold

As described in , the CLGC power is required to meet the threshold criteria to be considered for loop gain estimation. The thresholds are configured by the user through the API adi_adrv904x_ClgcCaptureConfigSet() using the parameters minTxPowThres and minOrxPowThresthat, which are part of adi_adrv904x_DfeAppCalClgcCaptureCfg_tdata structure. If either one of the Tx or ORx powers does whereas not meet the criteria, then CLGC is not run. Figure 230 shows an example scenario in which the Tx/ORx signals meet the threshold criteria in capture 1, where as the RMS power in capture 2 fails to meet the threshold criteria. Therefore, data in capture 2 is discarded and not considered for measurement.

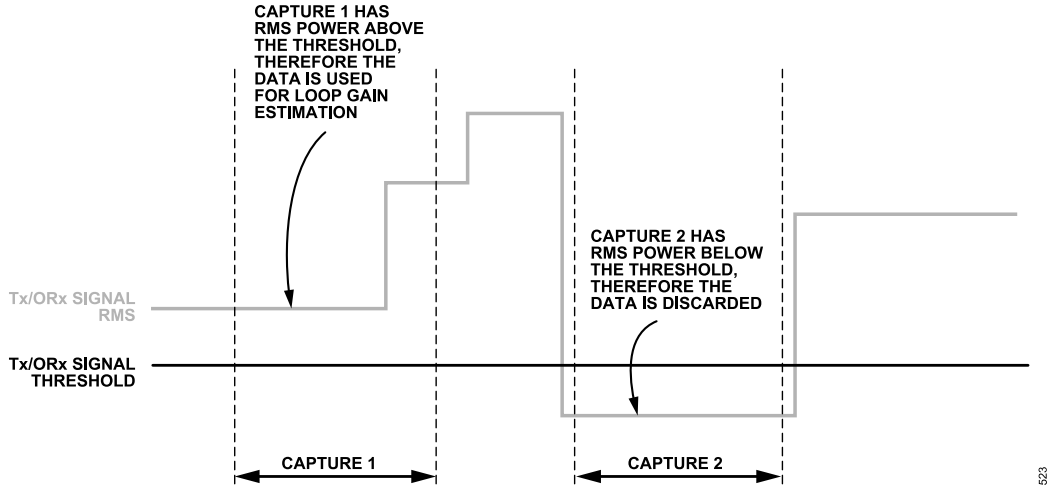


Figure 230. Tx and ORx Qualifying Threshold for CLGC Measurement

The complete CLGC measurement cycle for a single update period is captured in Figure 230. The flowchart explains the measurement and its interactions with the measurement parameters described in Table 125. The important thing to note is that the CLGC only captures samples while the Tx/ORx threshold is met. Once the threshold criteria are met, the CLGC proceeds to measure the loop gain and doesn't capture any further samples.

Table 125. CLGC Tracking Configuration

Parameter	Description	User Configuration Access
Enable CLGC	Enable Close Loop Gain Control	adi_adrv904x_DfeAppCalClgcTrackCfg_t. enClgc
Enable PA protection	Enable PA protection	adi_adrv904x_DfeAppCalClgcTrackCfg_t. enPaProtect
Expected Tx to ORx Loop Gain	Expected Tx to ORx loop power gain for CLGC to track on	adi_adrv904x_DfeAppCalClgcTrackCfg_t. expLoopPowGain
Maximum Loop Gain Adjustment	Maximum loop gain adjustment in dB for Tx attenuation	adi_adrv904x_DfeAppCalClgcTrackCfg_t. maxLoopGainAdjustdB
Minimum Tx Attenuation Limit	Minimum Tx attenuation limit in dB	adi_adrv904x_DfeAppCalClgcTrackCfg_t. minTxAttenLimitdB

CLOSED LOOP GAIN CONTROL (CLGC)

Table 125. CLGC Tracking Configuration (Continued)

Parameter	Description	User Configuration Access
Maximum Tx Attenuation Limit	Maximum Tx attenuation limit in dB	adi_adrv904x_DfeAppCalClgcTrackCfg_t. maxTxAttenLimitdB

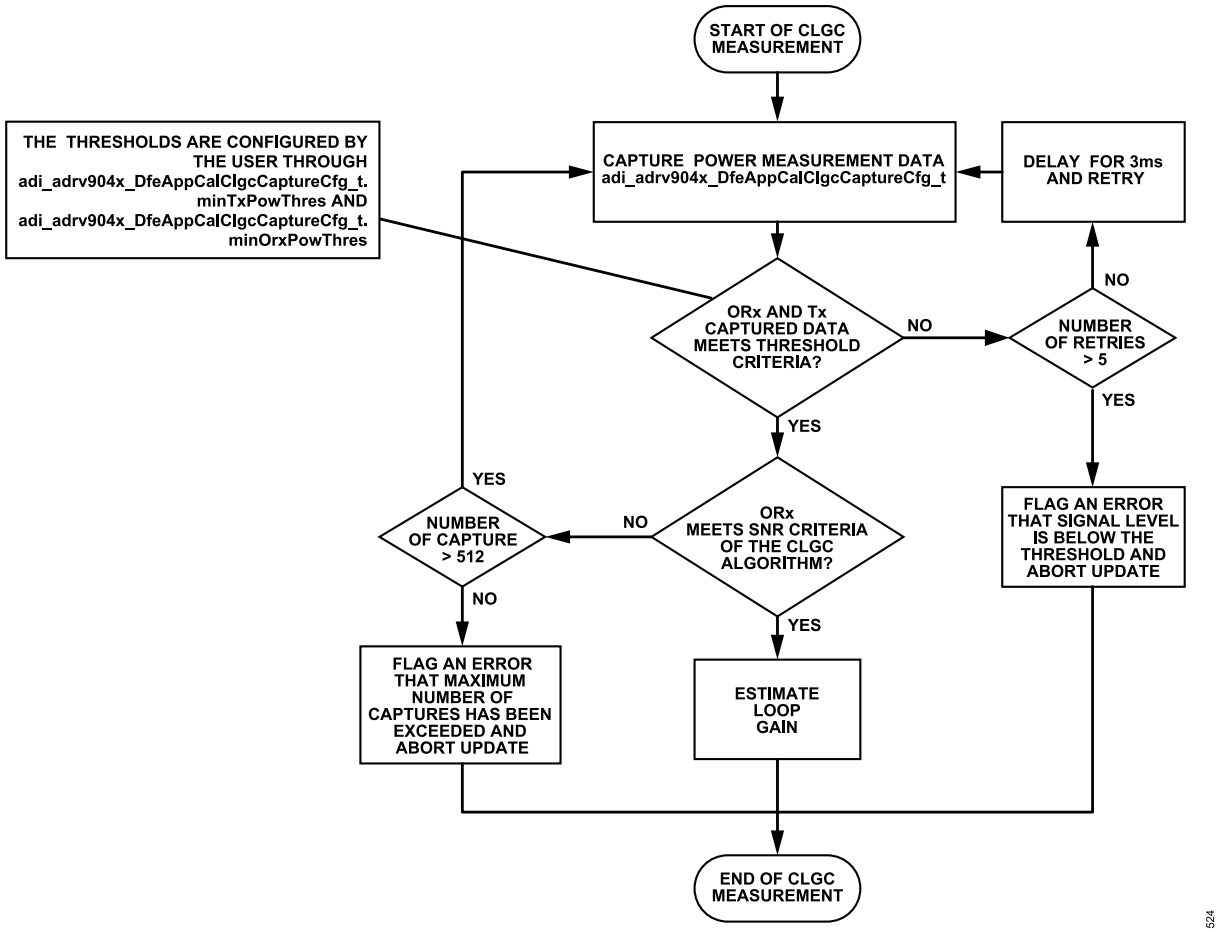


Figure 231. CLGC Measurement Cycle

CLGC TX ATTENUATION CONTROL

As shown in Figure 1, the CLGC algorithm tunes the Tx front-end attenuation in the transceiver to converge to the requested loop gain. The user is required to configure the Attenuation limits and the step size parameters through the API adi_adrv904x_ClgcTrackingConfigSet(). The Tx attenuation control parameters that a user needs to configure are described in Table 126.

Table 126. TX Attenuation Control User Configuration

Parameter	Description	User Configure Access
Minimum Tx Attenuation	The absolute value of the lower limit of Tx attenuation is configured in dB. For example, if the lower Tx attenuation limit is configured as 3 dB, then the CLGC algorithm cannot adjust the ADRV904x Tx front-end attenuation beyond 3 dB. This parameter can be used to mitigate CLGC over-compensating during any catastrophic situations when ORx data is corrupted, and the loop gain estimated is bad. Setting the minimum Tx attenuation limit ensures that the PA is not overdriven.	adi_adrv904x_DfeAppCalClgcTrackCfg_t. minTxAttenLimitdB

CLOSED LOOP GAIN CONTROL (CLGC)

Table 126. TX Attenuation Control User Configuration (Continued)

Parameter	Description	User Configure Access
Maximum Tx Attenuation	The absolute value of the upper limit of Tx attenuation is configured in dB. For example, if the upper Tx attenuation limit is configured as 30 dB, then the CLGC algorithm cannot adjust the ADRV904x Tx front-end attenuation beyond 30 dB. This parameter can be used to mitigate CLGC over-compensating during any catastrophic situations when ORx data is corrupted, and the loop gain estimated is bad. Setting the maximum Tx attenuation limit ensures that the Tx front-end attenuation does not go under range.	adi_adrv904x_DfeAppCalClgcTrackCfg_t. maxTxAttenLimitdB
Tx Gain Adjustment Step	Maximum step size for adjusting the Tx attenuation per CLGC update with a resolution of 0.05 dB	adi_adrv904x_DfeAppCalClgcTrackCfg_t. maxLoopGainAdjustdB

The Tx gain adjustment step size is a trade-off between the time required to converge versus transient spectral emissions due. A smaller Tx gain adjustment step results in smaller transient emissions, but takes a longer time to converge. A large step size could result in transient emissions due to a large change in power, but the convergence time is lower compared to a smaller step size.

An example is shown in Figure 232, where the desired loop gain is 2.8 × Tx gain adjustment step size relative to the initial loop gain. The CLGC algorithm adjusts the Tx attenuation over 3 update periods to reach the desired loop gain as shown in Figure 232. Note that the CLGC update period is 1 second.

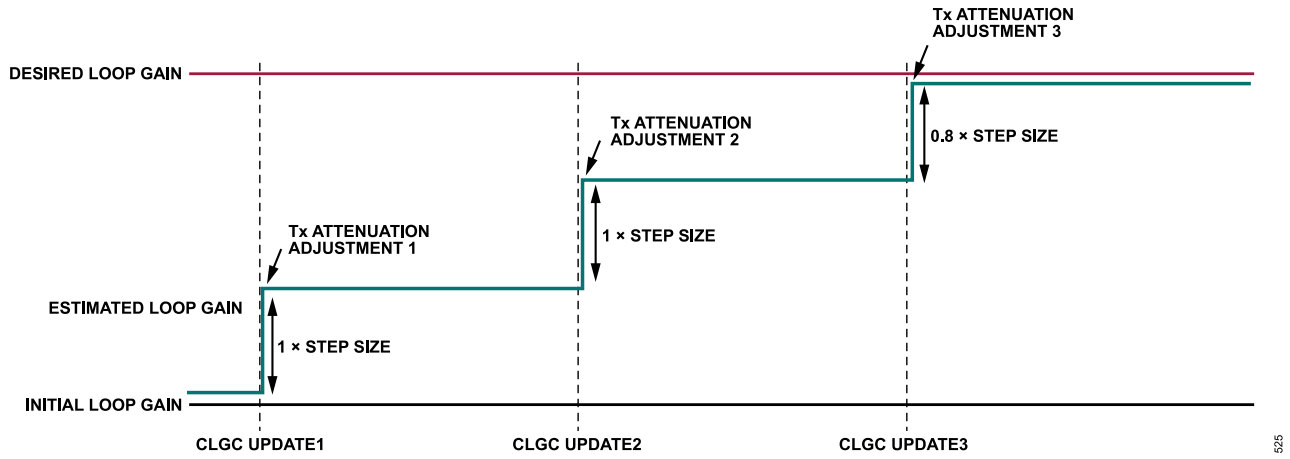


Figure 232. CLGC Loop Gain Convergence for 2.8× Step Size

Figure 232 represents the complete CLGC update cycle including the measurement and Tx attenuation control for a single CLGC update. Figure 232 represents the measurement and update for a single CLGC update cycle.

CLOSED LOOP GAIN CONTROL (CLGC)

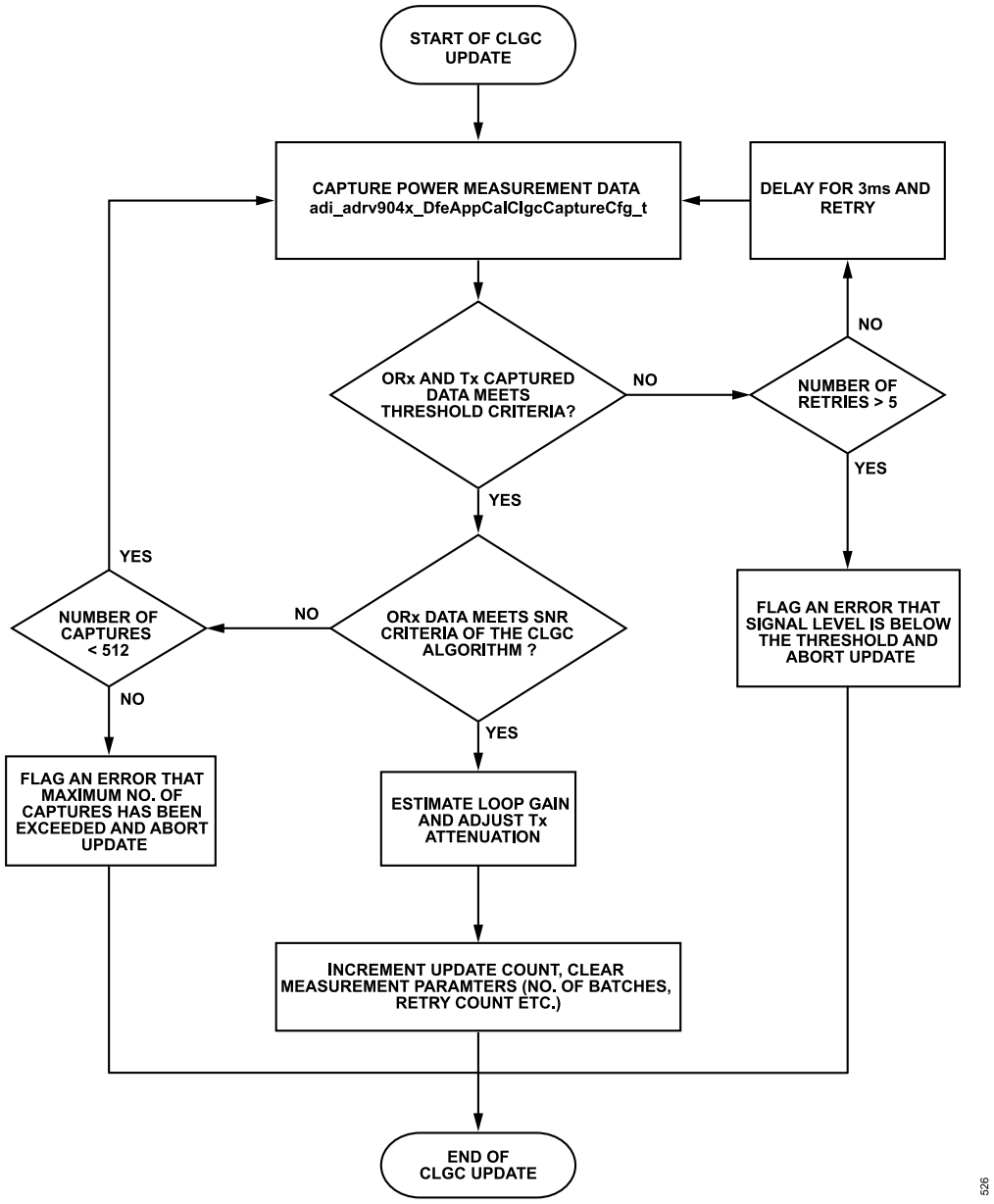


Figure 233. CLGC Active Loop Gain Control Update Cycle

RECOMMENDED SEQUENCE FOR ENABLING CLGC TRACKING CALIBRATION

Table 127. DPD Tracking Cal Bring up Sequence

Step	Action	ADRV904x APIs used
1	Program the device and run initial calibrations (including TxQEC initial calibration) with the PA turned off.	(Utility function adi_daughterboard_Program() can be used to program the device)
2	Set up external Tx to ORx mapping.	adi_adrv904x_TxToOrxMappingSet()
3	Adjust ORx attenuation to an appropriate value to avoid saturation. The default attenuation setting in ADRV904x is 0 (0 dB attenuation).	adi_adrv904x_OrxAttenSet()
4	Run the Tx external LO Leakage initial calibration.	adi_adrv904x_InitCalsRun()
5	If using ADRV904x CFR, configure the CFR settings.	adi_adrv904x_CfrConfigSet()adi_adrv904x_CfrCorrectionPulseWrite()
6	Load the DPD model.	adi_adrv904x_DpdModelConfigDpdSet()
7	Assert DPD Reset.	adi_adrv904x_DpdReset()

CLOSED LOOP GAIN CONTROL (CLGC)

Table 127. DPD Tracking Cal Bring up Sequence (Continued)

Step	Action	ADRV904x APIs used
8	Set up DPD mode of operation, DPD peak search window size and low power threshold.	adi_adrv904x_DpdTrackingConfigSet()
9	Set up the DPD capture configuration.	adi_adrv904x_DpdCaptureConfigSet()
10	Set up DPD fault conditions and recovery actions (optional).	adi_adrv904x_DpdStabilityCfgSet()
11	Set up CLGC configurations and target loop gain.	adi_adrv904x_ClgcCaptureConfigSet() adi_adrv904x_ClgcTrackingConfigSet()
12	Enable Tx QEC and Tx LO Leakage tracking Calibrations	adi_adrv904x_TrackingCalsEnableSet()
13	Enable DPD and CLGC Tracking Calibration.	adi_adrv904x_DfeTrackingCalsEnableSet()
14	Monitor DPD tracking calibration status.	adi_adrv904x_DfeDpdCalSpecificStatusGet()
15	Monitor CLGC tracking calibration status.	adi_adrv904x_DfeClgcStatusGet()

CLGC API FUNCTIONS

Table 128. CLGC API Functions

API Method Name	Comments
adi_adrv904x_ClgcCaptureConfigSet()	Configures the CLGC Capture Config for the requested Tx channel(s).
adi_adrv904x_ClgcCaptureConfigGet()	Retrieves the CLGC Capture Config for the requested Tx channel from the device.
adi_adrv904x_ClgcTrackingConfigSet()	Configures the CLGC Tracking Config for the requested Tx channel(s).
adi_adrv904x_ClgcTrackingConfigGet()	Retrieves the CLGC Tracking Config for the requested Tx channel.
adi_adrv904x_ClgcTrackingRun()	Starts the CLGC Tracking Run for the requested Tx channel(s).
adi_adrv904x_ClgcTrackingReset()	Resets the CLGC Tracking for the requested Tx channel(s).
adi_adrv904x_DfeClgcStatusGet()	Retrieves the CLGC Status for the requested Tx channel.

VSWR

VSWR INTRODUCTION

VSWR (Voltage Standing Wave Reflection) refers to a feature used to monitor the transmission line termination quality at the antenna (to detect disconnections or bad cables). Poor termination of a transmission line (that is, mismatched impedance) will cause a reflected signal to travel in the reverse direction. This results in reduced efficiency as less power will be delivered to the antenna. The reflected signal can also damage components such as power amplifiers.

VSWR function utilizes forward and reverse (reflected power from antenna) power measurement of test Ram data. Forward power (G_f) is fed back to ORx from the forward coupler and reverse power (G_r) via the reverse coupler.

With the forward and reflected power, we can compute the forward-reflected ratio as

$$\Gamma = G_r(t)/G_f(t)) \quad (50)$$

The reflected power can be from multiple sources like the cavity filter, Antenna cable, or Antenna, the VSWR feature measures the reflected power with Max amplitude and use that for Return loss.

And then the VSWR can be computed by the user as

$$VSWR = 1 - \Gamma/1 + \Gamma \quad (51)$$

Note that VSWR is not reported directly by the API, only Return loss is reported.

VSWR HARDWARE REQUIREMENTS

For utilizing the VSWR feature in ADRV904x, there should be forward and reverse signals sampled back to ORx for power measurement. The same ORx port is used for other DFE cals like DPD and CLGC. When VSWR is running DFE cals will not run.

Typically, two couplers and an SP2T switch will be required to feed-forward and reverse power back to the ORx port for power measurement.

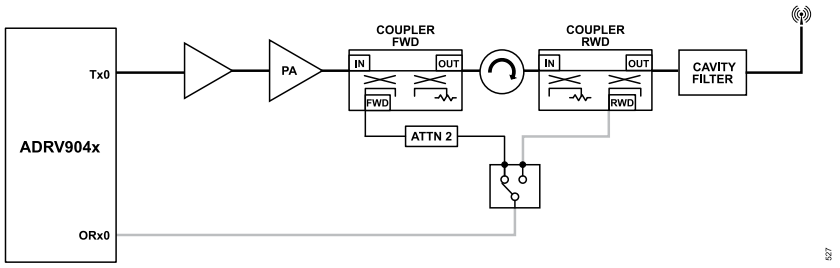


Figure 234. EVB VSWR Hardware Requirement, Separate Forward and Reverse Couplers Used

Below is one example circuit for reference where two directional couplers are used for measuring forward and reverse power.

Note: Simplified diagram, the 1:4 ORx control switch is not shown in Figure 234.

Customers can use a dual directional coupler instead of the 'Coupler RWD' as well. This is not recommended if using DPD as any interference from the antenna port can couple back to the ORx path while running DPD and can impact DPD performance.

Another example will be to use the receive path from the circulator for the VSWR reverse measurement.

Note: Instead of a second switch, a directional coupler as well can be used.

In this configuration, there is some additional loss between the forward coupled output to ORx switch and the reverse path to the ORx switch. This difference can be added as an ORx gain offset parameter in the VSWR configuration.

VSWR

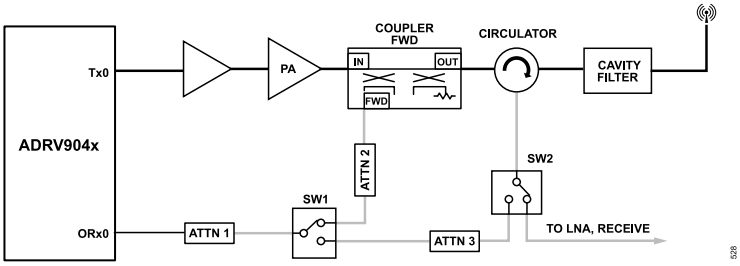


Figure 235. EVB VSWR Hardware Requirement, the Reverse Path Taken from the Isolator Port

VSWR IMPLEMENTATION

The VSWR function measures forward and reverse power and reports it back to the Antenna return loss. This is with the assumption that the highest amplitude reflection comes from the antenna or antenna cable. VSWR function detects the max reflected power and reports the return loss for that signal.

VSWR Test RAM Signal

VSWR power measurements are correlated power measurements based on a test signal Injected from a test RAM. This signal is a low-power signal injected in a band as per the CDUC/CDDC configuration. The test RAM injection point is in between DPD HB1 and DPD HB2.

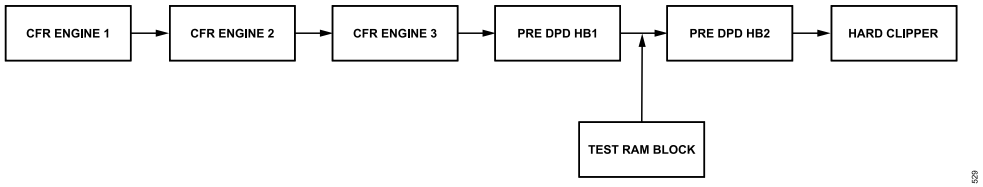


Figure 236. Test RAM Data Injection Data Point

VSWR test RAM is generated as per the carriers configured to avoid unwanted emissions. Test RAM data is kept at −65 dBFs by default. This level is user-configurable.

For Good EVM, typically keep the test ram data at −50 dBc from the actual transmitted signal.

The typical recommended input signal level from BBP is −14 dBFs (8 dB PAR and 6 dB for DPD gain expansion).

VSWR Test RAM Signal Generation

VSWR test RAM generation will be part of ACE GUI in future versions of software releases. As of now, test RAM waveforms are generated using Matlab code.

VSWR Accumulator

VSWR accumulator is an input field, the default recommended is 10,000. The more the accumulator the more will be the power measurement accuracy, but the time taken to run VSWR will increase.

The power measurement reported by VSWR is accumulated power and there will be a difference in power reported as per the accumulation number entered. Accumulated power read back (Forward power and Reverse power) cannot be used for absolute power measurement of the signal. Since power measurements are used for return loss, the scaling is equally applied for forward and reverse power measurements and will not impact return loss value.

Return Loss Qualifying Threshold

Return loss qualifying threshold is an input field. The VSWR engine will not report VSWR if the calculated return loss value is below this value. Default is kept at 40 dB which means it will report Return loss for all practical cases.

VSWR

TX Attenuation Change Handling

If a Tx attenuation change occurs (whether manually by the user or due to CLGC) during data capturing, it will affect VSWR accuracy. For this reason, VSWR will check the Tx attenuation values before and after capturing. If it has changed by more than 0.5 dB, the VSWR calculations will not proceed, and that iteration will be exited. In this case, it reports a 0xE048u error code while trying to capture the VSWR statistics.

This is not expected to be a common occurrence as typically CLGC is configured to make updates of 0.1 dB or 0.2 dB step size.

API FUNCTIONS

Table 129. List of VSWR API Functions

API Method Name	Comments
adi_adrv904x_DfeTrackingCalsEnableSet( )	Enables or disables the VSWR tracking cal. Channel mask has to be set as TX_VSWR_MASK for enabling/disabling VSWR.
adi_adrv904x_VswrPlaybackDataWrite()	Writes the VSWR pulse RAM waveform. The playback waveform data would have real and imaginary data in Tabbed format and it contains 1023 samples of maximum. This data set is added to the signal on data path after PreDPD HB1 block.
adi_adrv904x_VswrPlaybackDataRead()	Reads back the VSWR pulse RAM.
ADI_API adi_adrv904x_ErrAction_e adi_adrv904x_TxToOrxMappingAndVswrDirSet()	Sets the Tx to ORx mapping and VSWR direction set.
ADI_API adi_adrv904x_ErrAction_e adi_adrv904x_TxToOrxMappingAndVswrDirGet()	Reads the Tx to ORx mapping and VSWR direction set.
adi_adrv904x_VswrTrackingConfigSet()	Sets the VSWR tracking config set.
adi_adrv904x_VswrReset()	Resets the VSWR for specified channels.
adi_adrv904x_VswrStatusGet	Reads the VSWR status data of selected channel.

VSWR ALARMS AND GPINT

Alarms are set using VswrTrackingConfigSet API.

Based on the return loss measured, there is an option to generate 2 alarms, minor alarm, and major alarm based on pre-defined return loss thresholds.

For example, if minor alarm threshold is kept at 10 db, and major alarm threshold is kept at 6 dB, if the return loss calculated is less than 10 dB for x count in y iterations a Minor alarm will be reported. X and y are pre-programmer iteration counts.

Same way if the reported return loss is less than 6 dB a major alarm is reported.

When alarms are generated the software reports this via corresponding error code as well.

The error code for minor Alarms is 0xE04B and for major alarms is 0xE04C.

GP_INT can be raised using bit D45 of the upper GP_INT word. When that occurs the BBIC will have to use API functions to determine the cause (as there can be multiple sources for raising that bit), and to discover which channel (and potentially which reflection) the alarm refers to.

Errors and Error Codes

The VSWR Errors and Error codes can be found in adrv904x_dfe_app_error_tables.h from the source file package.

DTX

Traditional BTS (Base Transceiver Station) consumes considerable power even if there is no user in the cell. However, a hardware feature called discontinuous transmission (DTX), which could deactivate some components of the BTS during the empty transmission time intervals (TTIs) significantly lowers the idle power consumption. DTX could detect the consecutive amount of zero samples with the hardware circuitry to identify the empty TTIs and then put some components in deactivation mode to save power.

The DTX mode in ADRV9040 uses the TX Slice processor to gate off the clocks from the TX datapath and ramp down the TX VGA block to save power, in the meanwhile, TX trackings like TX QEC and TX LOL would be paused.

Upon detecting DTX, a stream processor is triggered which is used to power down certain blocks in the transmit chain. Another stream processor is triggered to power up the channel once valid data is to be transmitted again.

- There is an important trade-off between response time and power savings in this functionality. Powering down more blocks will further decrease power consumption but will also result in a longer response time for the channel to be powered up again.
- As mentioned above, TX Attenuation can be ramped down to save power, and the digital clocks can be gated off the datapath which can be configured through JSON profile.
- TX tracking calibrations (QEC and LO Leakage) are paused when DTX is activated and resumed when DTX is de-activated by default.

GPIO for external components are components external to the transceiver may also be powered down during a DTX period. To achieve this functionality, the DTX streams generate a GPIO signal to indicate DTX power-down and power-up events. This feature is supported by ADRV9040 (details are provided in the PA_EN Output on ADRV904x section).

This DTX function supports three modes.

DTX MODES

DTx supports three modes as follows:

- Automatic Mode. Triggers the power down by observing a specific number of zero samples on Tx data, power up stream is triggered as soon as a non-zero sample is detected.
- SPI Controlled Mode. In this mode, power down/up is triggered by the API command.
- PIN Controlled Mode. In this mode, a GPIO pin is specified to control power up/down, a rising edge on the pin triggers the power down and a falling edge triggers a power-up.

DTX MEASUREMENT (RESPONSE AND RECOVERY TIME)

Pin Controlled Mode

In Figure 237, DTX mode is controlled by GPIO (DTX_GPIO in blue color). When DTX_GPIO goes high, DTX mode is activated, and TX attenuation is ramped down so the TX signal is shut off. When DTX_GPIO goes low, DTX mode is deactivated, and TX attenuation is ramped up and the signal comes out of the TX channel.

DTX

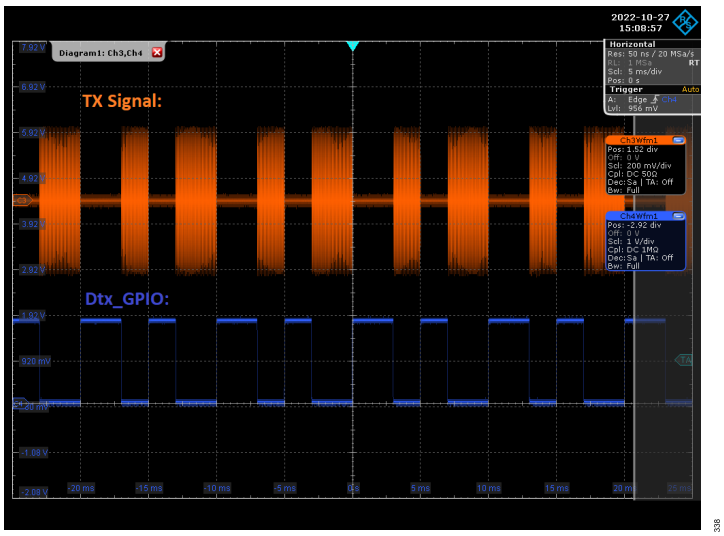


Figure 237. TX Signal in DTX Pin Control Mode (DTX_GPIO is the Control Pin)

Figure 238 shows that DTX response time from the rising edge of DTX_GPIO to signal off completely is about 570 ns. Figure 239 shows DTX recovery time, from the falling edge of DTX_GPIO to TX signal ramping up to full scale, takes about 730- ns. This measurement was done on UC69, and the response time and recovery time may slightly vary over different use cases and configurations.

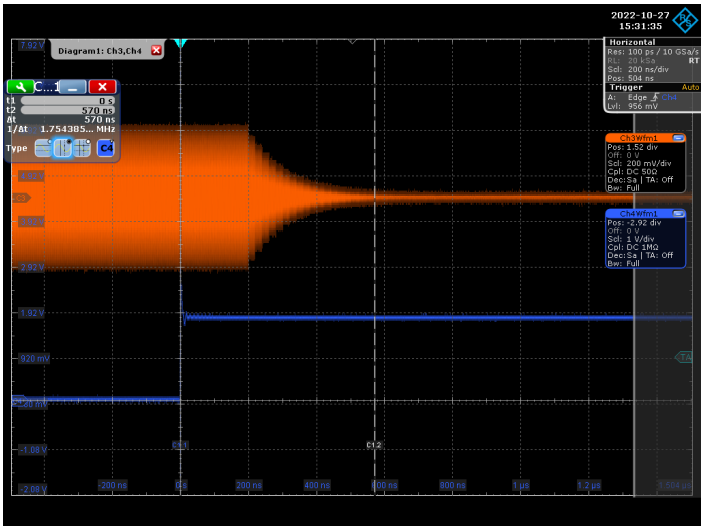


Figure 238. DTX Response Time (Blue Curve is DTX_GPIO Control)

DTX

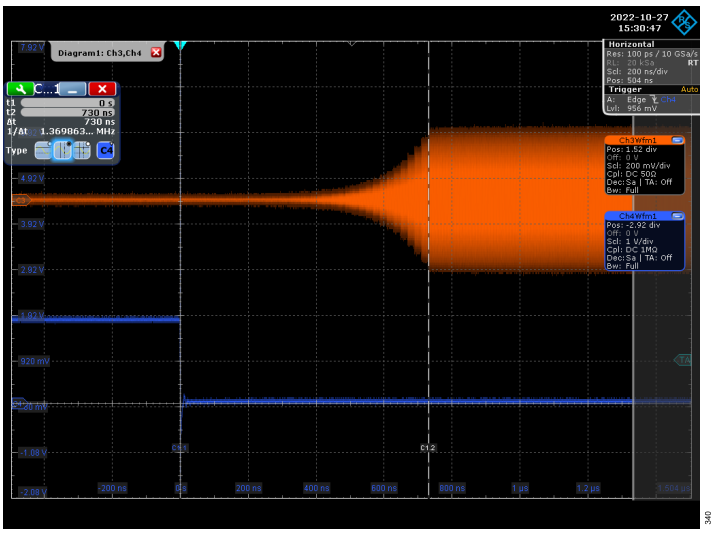


Figure 239. DTX Recovery Time (Blue Curve is DTX_GPIO Control)

Automatic Mode

DTx in Automatic mode trigger the power down by observing a specified number of zero samples on Tx data and the power up stream is triggered as soon as a non-zero sample is observed.

The test setup is as follows:

- 10 MHz CW tone from the baseband.
- 140 samples with 491.52 MHz sample rate are added to the 10 MHz waveform.
  - This is just for the test purpose, however, the real discontinuous transmission is usually symbol basis which includes a large amount of zeros.
- DTX is configured as Automatic Mode.
- DTX is enabled on TX7 (the waveform in Orange color) and DTX is disabled on TX6 (the waveform in Green color) for comparison purposes in Automatic Mode and Figure 241.
- DtxZeroCounter is set to 10 samples which are recommended to be greater than 10 samples.

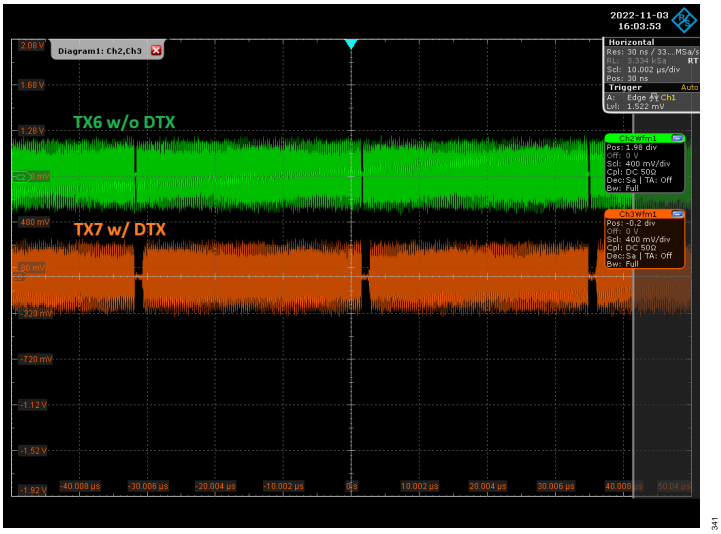


Figure 240. DTx in Automatic Mode

DTX

Zoom into a single DTX event in Figure 241, as shown on the waveform in green (with DTX disabled). The zeros period is about 284 ns, which is shorter than the response time that was measured in Figure 238. In this condition, the DTX response stream and recovery stream happens in a sequence which means, once 10 zeros are detected, the DTX stream starts to pause the clock to datapath and ramps down TX attenuation. However, the non-zero data comes before the completion of ramp down, then the recovery stream waits in the tasks execution queue to let the ramp-down finish then starts to ramp up TX attenuation and resume clock to the datapath. In Figure 241, the total response and recovery time is about 1.37 µs which is aligned with the results in Figure 238 and Figure 239 .

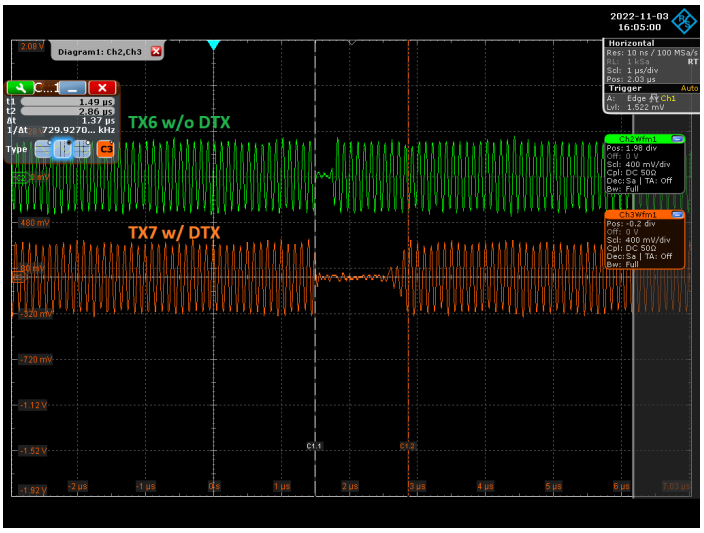


Figure 241. Response and Recovery Time in Automatic Mode

DTx Preserve Clocks Mode

The ADRV9040 JSON file provides one parameter DtxPreserveClocks which allows the users to configure whether clock is ON or not when DTX mode is activated.

Setting DtxPreserveClocks to be false means clock is turned off when DTX is activated, true means clock is still ON, and only TX attenuation is ramp down or up when DTX event happens. In Figure 242, the response and recovery time is measured which takes 1.39 µs where clock is preserved in DTX mode, but TX VGA was still ramped up and down. Comparing this result with Figure 241, they are close because the attenuation ramp up/down dominates the response and recovery time.

```
"streamSettings": {
"TestMacro": true,
…
"EnableNco": false,
"DisableTxRx": false,
"DtxPreserveClocks": true,
"PaProtExtendedGpInt": false,
"TxToOrxMappingAutoSwitchNcoEnable": true,
"DeviceType": 9040
},
```

DTX

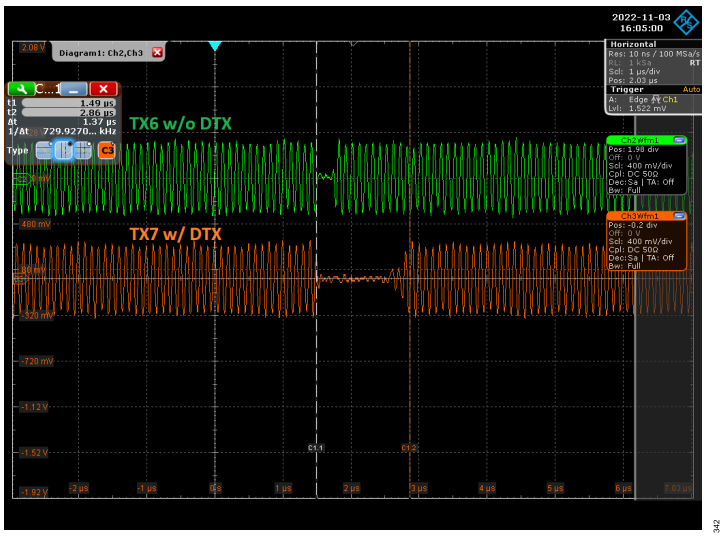


Figure 242. Koror DTX Response and Recovery Time when DtxPreserveClocks = True

The discontinuous transmission is generally symbol basis which duration is typically from several micro-sec to tens of micro-sec depending on the sub-carrier spacing. The response and recovery time discussed above is much shorter than the symbol duration which means DTX fits well on the symbol basis power saving. However, the recovery time may need to be paid more attention because, the channel recovery starts with first non-zero data, and it takes hundreds of nano-sec to power up the transmitter which may ruin some traffic data. The following EVM measurement is intended to evaluate the influence of the recovery duration (investigation was done on ADRV9040):

- UC69 Profile is used
- LTE20 modulated waveform for EVM measurement (Brown color in Figure 243)
- PA_EN which indicates DTX event was used to trigger spectrum analyzer and EVM measurement (Blue curve in Figure 243)
- DTX is configured as Automatic Mode
- DtxZeroCounter is set to 10 samples (which is recommended to be greater than 10 samples)

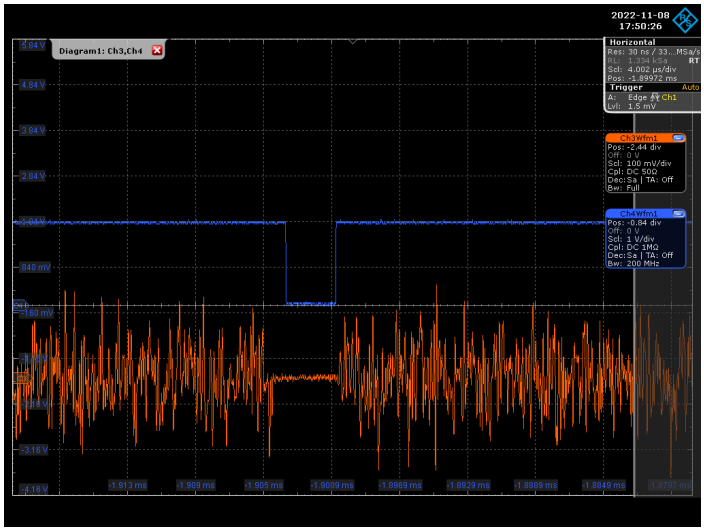


Figure 243. LTE20 Waveform and PA_EN Output

DTX

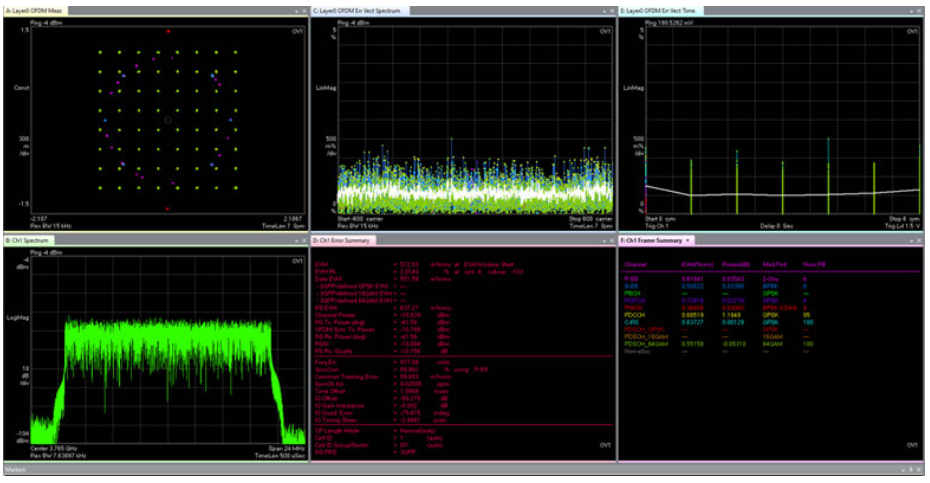


Figure 244. EVM Result Just After DTx Recovery

According to the results in Figure 244, we don't see any obvious degradation on the first symbol EVM. Actually this is expected because the channel latency is longer than the recovery duration that we measured above.

Figure 245 shows the transmitter block diagram of UC69, DTX detection block locates at the input of DUC0, and the DTX power up, specifically for the TX attenuation ramps up actually happens at the end of the signal chain on the analog front end, between these two blocks the channel latency is about 2510 ns which means the channel ramps up before the traffic data arrives, which explains the first symbol EVM wasn't affected by the DTX power up.

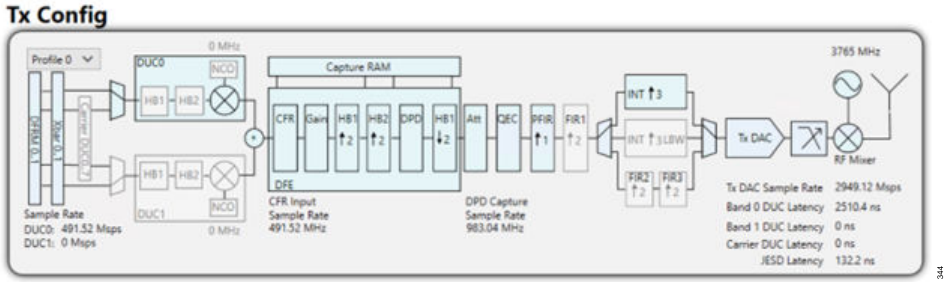


Figure 245. TX Block Diagram and Latency on UC69

POWER SAVING EXAMPLE IN DTX

Table 130 shows the power saving example in DTX for the ADRV9040.

Table 130. Power Saving in DTx

	Mode	0.8 V/mA	1.0 V/mA	1.8 V/mA	Power/mW
Use Case 69	Tx Normal Mode	9535	2794	1922	13882
	Clock Preserved Mode	9523	2786	1179	12527
	Entire DTX Mode	2705	2786	1177	7049

PA_EN OUTPUT ON ADRV904X

As discussed above DTX can be used to turn off some components in the system to save power consumption, inside transceiver, TX attenuation can be ramped down and clocks can be gated off from data path to save power, additionally, on ADRV9040 the DTX activate and deactivate status can be mapped to GPIO to control external components like PA and works as PA_EN. As shown in Figure 243, the curve in blue color shows the DTX status, low level of the blue curve means DTX was activated and it can be used to turn off PA, and high level means non-zero samples were detected, and transmitter comes out of DTX mode, which can be used to turn on PA.

- In TDD system, PA's ON/OFF is typically controlled by TDD timing from base band. If the user system wants DTX function to be involved to control the PA, a combined logic between DTX and TDD is necessary in the base band processor.

DTX

- The PA_EN output logic polarity on GPIO is configurable through ACE GUI as shown in Figure 246. Or, it can be configured in JSON file with parameterPaEnPinsMode, value 1 means active high mode, and value 0 means active low mode.

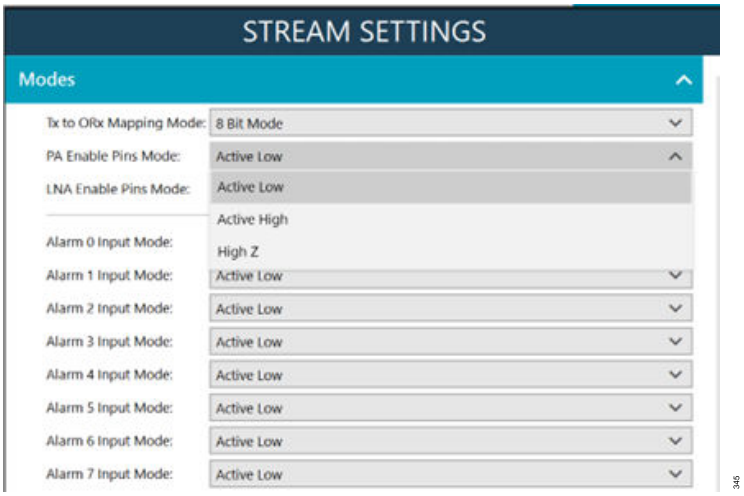


Figure 246. PA_EN in Pin Mode

The following is an example to map PA_EN controls to GPIO_9 to GPIO_16 on ADRV9040:

1. The configuration in JSON profile is shown in the following code block:

```
"streamSettings": {
"TestMacro": true,
"CaptureRamDelayUs": 0,
"Gpio00Selection": 255,
"Gpio01Selection": 255,
"Gpio02Selection": 255,
"Gpio03Selection": 255,
"Gpio04Selection": 255,
"Gpio05Selection": 255,
"Gpio06Selection": 255,
"Gpio07Selection": 255,
"Gpio08Selection": 255,
"Gpio09Selection": 30,
"Gpio10Selection": 31,
"Gpio11Selection": 32,
"Gpio12Selection": 33,
"Gpio13Selection": 34,
"Gpio14Selection": 35,
"Gpio15Selection": 36,
"Gpio16Selection": 37,
"Gpio17Selection": 255,
"Gpio18Selection": 255,
"Gpio19Selection": 255,
"Gpio20Selection": 255,
"Gpio21Selection": 255,
"Gpio22Selection": 255,
"Gpio23Selection": 255,
…
"AnalogGpio15Selection": 255,
"PaEnPinsMode": 0,
"LnaEnPinsMode": 0,
```

```
"Alarm0InputPinMode": 0,
}
…
"radioCtrlPreInit": {
"radioSequencerSsbSyncGpioCtrl": 24,
"radioSequencerGpioDigOut": [
24,
24,
24,
24,
24,
24,
24,
24,
24,
9,
10,
11,
12,
13,
14,
15,
16,
24,
24,
24,
24,
24,
24,
24
],
…
```

2. The API adi_adrv904x_RadioCtrlPaSelectInputSet() is used to configure the input of PA_EN to be both RS and DTX by setting adi_adrv904x_PaEnInput-Sel_e.
3. Call API functions as shown in Table 131 to configure DTX mode.
4. Turn ON then OFF TX channel by calling API adi_adrv904x_RxTxEnableSet(…).

DTX MODE API FUNCTIONS

Table 131 shows the ADRV9040 API functions for DTx.

Table 131. API Functions for DTx

API Functions Name	Comments
adi_adrv904x_DtxCfgSet(…)	Configures DTX data structure for each channel.
adi_adrv904x_DtxCfgGet(…)	Reads DTX configuration for each channel.
adi_adrv904x_DtxForceSet(…)	Forces to DTX mode, used in SPI mode.
adi_adrv904x_DtxGpioCfgSet(…)	Configures GPIOs to enable/recover DTX.
adi_adrv904x_DtxGpioCfgGet(…)	Reads GPIO assignment for each channel.
adi_adrv904x_DtxStatusGet(…)	Reads DTX status for each channel.

The ADRV9040 provides one parameter to configure DTX mode in JSON file.

```
"streamSettings": {
“TestMacro”: true,
```

DTX

```
…
"EnableNco": false,
"DisableTxRx": false,
"DtxPreserveClocks": false,
"PaProtExtendedGpInt": false,
"TxToOrxMappingAutoSwitchNcoEnable": true,
"DeviceType": 9040
},
```

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 ADRV904x than what is possible over the SPI interface. The distinction of analog GPIO does not imply analog behavior of the pin.

Note that all GPIOs comes up as Inputs during reset. The pad inputs would be floating without any drive unless BBIC is already driving it.

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 processor 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, request the functionality to an Analog Devices 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 132 summarizes the input digital GPIO features.

Table 132. 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. Work with Analog Devices Applications Engineering if specific signals are required for observation in an application.

DIGITAL GPIO API FUNCTIONS

Table 133. Digital GPIO API Functions

API Method Name	Comments
adi_adrv904x_GpioStatusRead()	Returns the full status of all digital and analog GPIOs.
adi_adrv904x_GpioConfigAllGet()	Returns the currently routed signal and channel Mask for all 24 digital GPIO pins.
adi_adrv904x_GpioConfigGet()	Returns the currently routed signal and channel Mask for a selected digital GPIO pin.

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

Table 133. Digital GPIO API Functions (Continued)

API Method Name	Comments
adi_adrv904x_GpioManualInputDirSet()	Allocates and configures selected digital GPIO pins for Manual Input Mode.
adi_adrv904x_GpioManualInputPinLevelGet()	Returns input read levels for all digital GPIO pins configured as Manual Inputs.
adi_adrv904x_GpioManualOutputDirSet()	Allocates and configures selected digital GPIO pins for Manual Output Mode.
adi_adrv904x_GpioManualOutputPinLevelGet()	Returns output drive levels for all digital GPIO pins configured as Manual Outputs.
adi_adrv904x_GpioManualOutputPinLevelSet()	Sets output drive levels for all digital GPIO pins configured as Manual Outputs.
adi_adrv904x_GpioMonitorOutRelease()	This API function releases a GPIO if it is currently routing a monitor output function.
adi_adrv904x_GpioMonitorOutSet()	This API function configures a monitor output function for a GPIO.
adi_adrv904x_StreamGpioConfigSet()	Sets the GPIO pin assignments that trigger streams.
adi_adrv904x_StreamGpioConfigGet()	Reads 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 in Table 134.

Table 134. 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. Details on the API are to follow in further SDUG releases.

ANALOG GPIO API FUNCTIONS

Table 135. List of Analog GPIO API Functions

API Method Name	Comments
adi_adrv904x_GpioStatusRead()	Returns the full status of all digital and analog GPIOs
adi_adrv904x_GpioAnalogConfigAllGet()	Returns the currently routed signal and channel Mask for all 16 Analog GPIO pins
adi_adrv904x_GpioAnalogConfigGet()	Returns the currently routed signal and channel Mask for a selected Analog GPIO pin
adi_adrv904x_GpioAnalogManualInputDirSet()	Allocates and configures selected analog GPIO pins for Manual Input Mode
adi_adrv904x_GpioAnalogManualInputPinLevelGet()	Returns input read levels for all analog GPIO pins configured as Manual Inputs
adi_adrv904x_GpioAnalogManualOutputDirSet()	Allocates and configures selected analog GPIO pins for Manual Output Mode
adi_adrv904x_GpioAnalogManualOutputPinLevelGet()	Returns output drive levels for all analog GPIO pins configured as Manual Outputs
adi_adrv904x_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 136. Suggested use of the GP_INT pins is GP_INT1

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

pin can act as the high priority interrupt pin and GP_INT0 can act as the low priority interrupt pin. 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.

A description of the interrupt sources and their bit positions within the 48-bit Lower Word general purpose interrupt mask is provided in Table 136. The description of the 48-bit Upper Word of the general purpose interrupt mask is provided in Table 137.

Table 136. GP_INTERRUPT Bitmask Description Lower Word

Bit Position	Brief Description	Component
D47	Currently Unused	Open
D46	Currently Unused	Open
D45	Currently Unused	Open
D44	SPI1 Paging Error	SPI
D43	SPI0 Paging Error	SPI
D42	SPI1 Core Error	SPI
D41	Core Stream Processor Error	Stream Proc
D40	ORx1 Stream Processor Error	Stream Proc
D39	ORX0 Stream Processor Error	Stream Proc
D38	Tx7 Stream Processor Error	Stream Proc
D37	Tx6 Stream Processor Error	Stream Proc
D36	Tx5 Stream Processor Error	Stream Proc
D35	Tx4 Stream Processor Error	Stream Proc
D34	Tx3 Stream Processor Error	Stream Proc
D33	Tx2 Stream Processor Error	Stream Proc
D32	Tx1 Stream Processor Error	Stream Proc
D31	Tx0 Stream Processor Error	Stream Proc
D30	Rx7 Stream Processor Error	Stream Proc
D29	Rx6 Stream Processor Error	Stream Proc
D28	Rx5 Stream Processor Error	Stream Proc
D27	Rx4 Stream Processor Error	Stream Proc
D26	Rx3 Stream Processor Error	Stream Proc
D25	Rx2 Stream Processor Error	Stream Proc
D24	Rx1 Stream Processor Error	Stream Proc
D23	Rx0 Stream Processor Error	Stream Proc
D22	Deframer IRQ [13] - Deframer 1 - 204C command word parity error	Deframer
D21	Deframer IRQ [12] - Deframer 1 - Asserted with SYNC meaning the link dropped	Deframer
D20	Deframer IRQ [11] - Deframer 1 - Sysref phase error, unexpected Sysref phase received	Deframer
D19	Deframer IRQ [10] - Deframer 1 - PCLK is running too fast in relation to the sample/convertor clock	Deframer
D18	Deframer IRQ [9] - Deframer 1 - PCLK is running too slow in relation to the sample/convertor clock	Deframer
D17	Deframer IRQ [8] - Deframer 1 - 204C link layer error	Deframer
D16	Deframer IRQ [7] - Deframer 1 - 204C link layer error. CRC error count has exceeded threshold	Deframer
D15	Deframer IRQ [6] - Deframer 0 - 204C command word parity error	Deframer
D14	Deframer IRQ [5] - Deframer 0 - Asserted with SYNC meaning the link dropped	Deframer
D13	Deframer IRQ [4] - Deframer 0 - Sysref phase error, unexpected Sysref phase received	Deframer
D12	Deframer IRQ [3] - Deframer 0 - PCLK is running too fast in relation to the sample/convertor clock	Deframer
D11	Deframer IRQ [2] - Deframer 0 - PCLK is running too slow in relation to the sample/convertor clock	Deframer
D10	Deframer IRQ [1] - Deframer 0 - 204B link layer error	Deframer
D9	Deframer IRQ [0] - Deframer 0 - 204C link layer error. CRC error count has exceeded threshold	Deframer
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

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

Table 136. GP_INTERRUPT Bitmask Description Lower Word (Continued)

Bit Position	Brief Description	Component
D6	Framer IRQ [6] - Framer 2 - PCLK is running too slow in relation to the sample/convertor clock	Framer
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 137. GP_INTERRUPT Bitmask Description Upper Word

Bit Position	Brief Description	Component
D47	Radio Sequencer Error	Radio Sequencer
D46	Currently Unused	Open
D45	DFE SW States	Arm
D44	DFE SW Errors	Arm
D43	Currently Unused	Open
D42	Currently Unused	Open
D41	Currently Unused	Open
D40	Currently Unused	Open
D39	Currently Unused	Open
D38	Currently Unused	Open
D37	SPI0 Abort	SPI
D36	SPI1 Abort	SPI
D35	SPI Clock Read abort	SPI
D34	Source Reducer Error Indication [SPI1]	SPI
D33	Source Reducer Error Indication [SPI0]	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

GENERAL PURPOSE INPUT/OUTPUT CONFIGURATION

Table 137. GP_INTERRUPT Bitmask Description Upper Word (Continued)

Bit Position	Brief Description	Component
D10	Tx0 PA Protection – Average OR Peak Power	Transmitter
D9	ARM0 Force GP Interrupt	Arm
D8	ARM0 Watchdog Timer Timeout	Arm
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 138. GP Interrupt API

API Method Name	Comments
adi_adrv904x_GpIntPinMaskCfgGet()	Retrieves the General Purpose (GP) Interrupt Pin Mask Config for GP Int pins: GP_INT0, GP_INT1, or both
adi_adrv904x_GpIntPinMaskCfgSet()	Sets the General Purpose (GP) Interrupt Pin Mask Config for GP Int pins: GP_INT0, GP_INT1, or both
adi_adrv904x_GpIntStatusClear()	Clears the General Purpose (GP) Interrupt status bits selected in the gpIntClear word.
adi_adrv904x_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_adrv904x_GpIntStickyBitMaskGet()	Gets the General Purpose (GP) Interrupt Type (pulse/edge (sticky) vs level triggered) for all interrupt sources
adi_adrv904x_GpIntStickyBitMaskSet()	Sets the General Purpose (GP) Interrupt Type (pulse/edge (sticky) vs level triggered) for all interrupt sources
adi_adrv904x_DfeCpuPintSwStatusGet()	Returns current status of DFE CPU Interrupt Sources D44 and D45 in upper mask.
adi_adrv904x_DfeCpuPintSw0DetailedInfoGet()	Retrieves detailed information once D44 in upper word is flagged.
adi_adrv904x_DfeCpuPintSw1DetailedInfoGet()	Retrieves detailed information once D45 in upper word is flagged.

How to Use GP_INT

The setup and usage for the GP_INT command is as follows:

1. Initialize device.
2. Set up 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. Operate device. The BBIC should monitor the GP_INT1 and/or GP_INT0 for rising edges indicating an interrupt has occurred.
4. If the GP_INT1 or GP_INT0 pins assert, call their associated interrupt handler API command. The interrupt handler will return information related to the interrupt source to the user. Calling this command may be sufficient to clearing the error. Either handler function returns a recovery action which suggests further action if necessary.
5. Perform recovery action.

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 ADRV904x has run into a problem that may require user intervention to resolve. The GP_INT handler functions tries to resolve the error by reading back the status. The bits in the status register are sticky, but the pin is not. The pin represents whether the interrupt source is active or not. The register indicates what interrupts have occurred since the last status read.

When a handler command is called, the first step in the procedure is to temporarily modify the GP_INT1/GP_INT0 bitmask such that no other interrupts can assert GP_INT1/GP_INT0 while the handler is invoked. This masking is followed by retrieval of the GP_INT status. The final step in the handler is to restore initial bitmask for GP_INT1/GP_INT0. In some cases, reading the error is sufficient to clearing the error – this is

the case for short-term, intermittent errors. If the error persists, then the status will continue to indicate the interrupt and further intervention is necessary.

Alternative to GP_INT Handler

An alternative to the handler is to read back the GP_INT status directly. While the GP_INT pins have independent bitmasks, there is only one set of status words to read back. The status word is not maskable and will indicate all errors since the previous read of the status word. There will be a clear status API command which clears the status register. If the interrupt source is still active, the appropriate status bit will re-assert after being cleared.

JTAG BOUNDARY SCAN

The ADRV904x 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 139 for the available modes.

Table 139. 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 ADRV904x is in JTAG, the following pins are used for JTAG access (see Table 140).

Table 140. JTAG Pin Interface

JTAG Pin	ADRV904x Pin	Description
TCK	GOIO_7	Test Clock.
TMS	GOIO_6	Test Mode State. Sampled at the rising edge of TCK to determine the next state.
TDI	GOIO_5	Test Data In. Sampled at the rising edge of TCK.
TDO	GOIO_4	Test Data Out. Valid on the galling edge of TCK.
TRST	GOIO_3	Test Reset. An option pin which can rest the TAP controller's state machine.

THERMAL CONSIDERATIONS

In the ADRV904x data sheet, thermal resistance values are provided using Table 141.

Table 141. Thermal Resistance Values from Data Sheet

Package Type	θ_{JA} (°C/W)	θ_{JCtop} (°C/W)	θ_{JB} (°C/W)	Ψ_{JT} (°C/W)	Ψ_{JB} (°C/W)
BP-736-1	9.46	0.38	2.13	0.33	2.16

The following list describes how each item in Table 141 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 numbers
- 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 1 ft cube JEDEC environment and 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.

$$\theta_{JA} = \frac{T_J - T_{Amb}}{P_{Diss}} \tag{52}$$

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.

$$\theta_{JCtop} = \frac{T_J - T_{Top}}{P_{Flow}} \tag{53}$$

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 1 mm of package body). θ_{JB} is simulated based on JEDEC defined environment and PCB.

$$\theta_{JB} = \frac{T_J - T_{Board}}{P_{Flow}} \tag{54}$$

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.

$$\psi_{JT} = \frac{T_J - T_{Top}}{P_{Diss}} \tag{55}$$

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.

$$\psi_{JB} = \frac{T_J - T_{Board}}{P_{Diss}} \tag{56}$$

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 as same as the definition in the JEDEC. However, the thermal environment on the board-level or system-level is more complicated in general, and the heat fluxes over a wide spectrum of environments, such as PCB, heatsink, or some convection cooling. DELPHI compact model is created from the detailed model using a step-by-step simulation and statistical optimization process over a wide spectrum of environment. It has high degree of boundary condition independence (BCI) and is also computational 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.

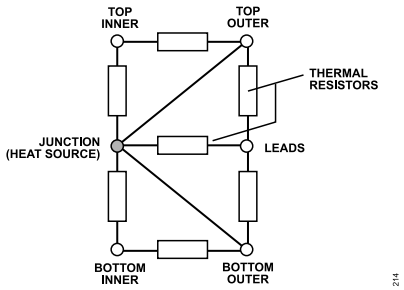


Figure 247. Network Compact Model

The DELPHI thermal resistance network is comprised of a limited number of nodes connected to each other by thermal resistor links (see Figure 247). 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 Analog Devices for the ADRV904x 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 Analog Devices before changing the power values in the DELPHI model.

THERMAL API FUNCTIONS

Table 142. List of Thermal API Functions

API Method Name	Comments
adi_adrv904x_TemperatureGet()	Reads the temperature from up to 12 sensors on die.
adi_adrv904x_TemperatureEnableGet()	Reads the temperature sensors that are enabled.
adi_adrv904x_TemperatureEnableSet()	Sets the temperature sensors that are to be enabled.

Note that the ADRV904x temperature sensor readback value has an accuracy of ±6.5°C.

POWER MANAGEMENT CONSIDERATIONS

The ADRV904x family of devices requires four different power supply domains as follows:

- 0.8 V Digital—this supply is connected to the device through the twenty (20) VDIG_0P8 pins. This is the supply that feeds all digital processing and clock generation, so it will draw much higher current than the other supply inputs. 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 20 inputs be connected to a common power plane to minimize mismatch and I × R drop. There is a dedicated sense pin (VDIG_SENS_0P8) that provides a direct sensing point on the chip. The power regulator used to supply 0.8 V should use this connection point for a remote sense function so that the regulator can modify its output to maintain a 0.8 V supply at the VDIG_0P8 pins.
- 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 143.
- 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 143.
- Interface Supply—the VIF_1P8 supply is a separate power domain shared with the BBP 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 143 lists the pin number, the pin name, and a brief description of the block it powers in the ADRV904x. 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 143. Power Supply Pins and Functions

Pin Name	Pins	Type	Domain[1]	Voltage	Recommended Routing/Notes	Description
VDIG_0P8	H12, H15, H18, H21, J12, J15, J18, J21, K12, K15, K18, K21, L12, L21, M12, M21, N12, N21, P12, P21	Digital	N/A	0.8 V	Route as a single power plane or multiple parallel sub-planes to minimize resistance. Use a power supply with remote sense, connect directly to the VDIG_SENS_0P8 pin (R16).	Digital core supply
VCLKSYN_1P0	U22	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
VCLKGEN0_1P0	U18	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
VCLKGEN1_1P0	W7	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

POWER MANAGEMENT CONSIDERATIONS

Table 143. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain[1]	Voltage	Recommended Routing/Notes	Description
VCLKGEN2_1P0	W26	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	A27	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 with a ferrite bead if necessary. Very sensitive to aggressors.	LO supply
VTXLO1_1P0	A25	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	A23	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	E11	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	E22	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	U11	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	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.	Device clock supply

POWER MANAGEMENT CONSIDERATIONS

Table 143. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain[1]	Voltage	Recommended Routing/Notes	Description
VSER_1P0	AB16, AC16	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	AB17, AC17	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	H9	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	T9	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	H24	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
VCONV3_1P0	T24	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	P8	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	P25	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	L9, M9	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 pins L9/M9 and L10/M10.	ORX0 channel ADC sample clock supply
VSCLK1_1P0	L24, M24	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 pins L24/M24 and L23/M23.	ORX1 channel ADC sample clock supply

POWER MANAGEMENT CONSIDERATIONS

Table 143. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain[1]	Voltage	Recommended Routing/Notes	Description
VBB0_1P0	F9	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	U7	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
VBB2_1P0	F24	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
VBB3_1P0	U26	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	F8	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	U6	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
VANA2_1P8	F25	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
VANA3_1P8	U27	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	U20	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	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.	SYSREF clock supply
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

POWER MANAGEMENT CONSIDERATIONS

Table 143. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain[1]	Voltage	Recommended Routing/Notes	Description
VVCO1_1P8	A30	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	U14	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	J9	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	R9	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	J24	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	R24	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	L8	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	N8	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
VORX2_1P8	L25	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
VORX3_1P8	N25	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	E1	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

POWER MANAGEMENT CONSIDERATIONS

Table 143. Power Supply Pins and Functions (Continued)

Pin Name	Pins	Type	Domain[1]	Voltage	Recommended Routing/Notes	Description
VTX1_1P8	V1	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
VTX2_1P8	E32	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
VTX3_1P8	V32	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	R15	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	U19	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	U13	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
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
VVCO1_1P0	A29	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

[1] N/A means not applicable.

Note that 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. Analog Devices recommends adding the following variation guidelines to supply maximums to ensure proper operation under all conditions.

Table 144. Variation Guidance for Supply Domains

Power Supply Domain	Variation
1.8 V	10%
1.0 V	20%
0.8 V	X × 15% + 3.125 A (assumes X is the nominal current consumption for a given use case)

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 RESET signal after power has stabilized before initializing the device.

The power-down sequence recommendation is similar to the 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 backfeeding to circuits that have been powered down.

Note that the 1.8 V supply needs to be below 1.1 V before the 0.8 V supply drops below 0.5 V to avoid any chance for damage to occur.

POWER MANAGEMENT CONSIDERATIONS

POWER SUPPLY ARCHITECTURE

The diagram in Figure 248 outlines the power supply inputs on the ADRV904x. This configuration follows the recommendations outlined in Table 143 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 0 Ω resistor).

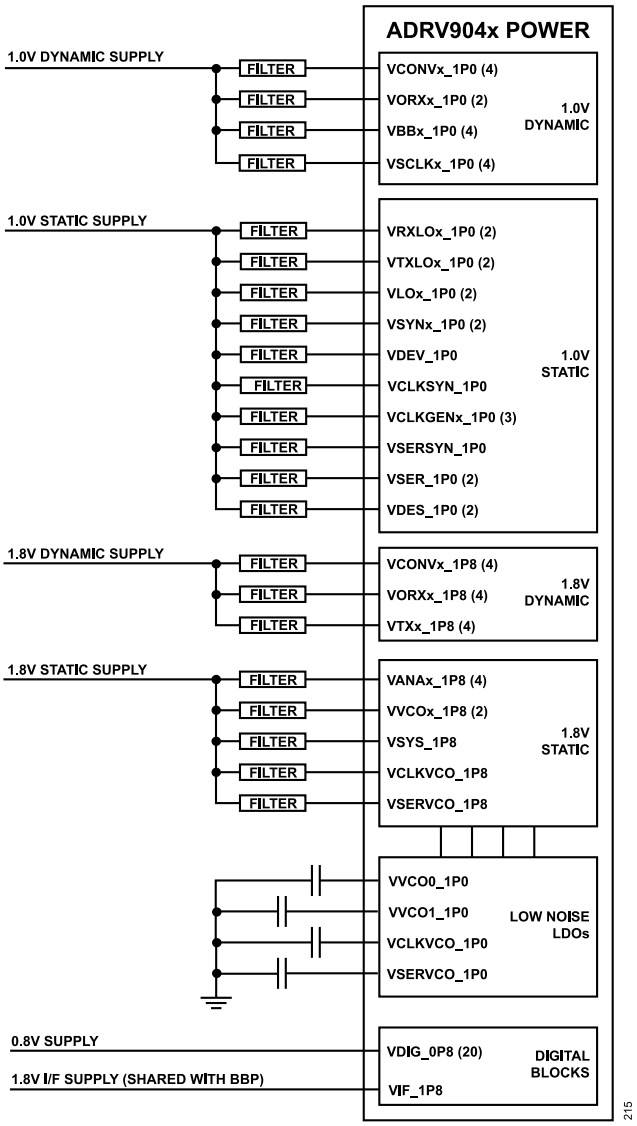


Figure 248. ADRV904x Power Supply Input Distribution

RBIAS SETUP

All internal currents and voltages are based off a master reference. There are two separate master references—one for each half of the device. The master 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.

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 ADRV904x 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 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. The DAC Block reads the rf port *.s1p file. This is the device rf port reflection coefficient.
2. The two equations convert the RF port reflection coefficient to a complex impedance. The end result is the RX_SEDZ variable.
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 249. 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 250. This network has fewer parasitics and fewer total components.

RF PORT IMPEDANCE MATCHING

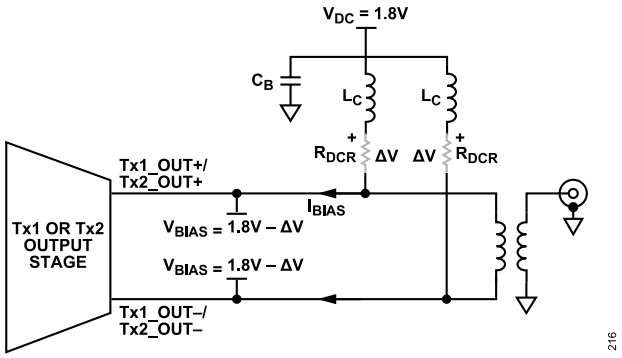


Figure 249. RF DC Bias Configurations Depicting Parasitic Losses Due to Wire Wound Chokes

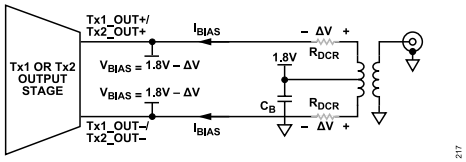


Figure 250. RF DC Bias Configurations Depicting Parasitic Losses Due to Center Tapped Transformers

Figure 251 to Figure 254 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 251. 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.

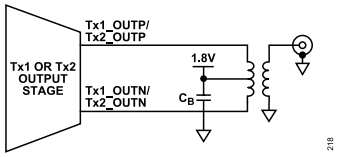


Figure 251. ADRV904x RF Transmitter Interface Configuration A

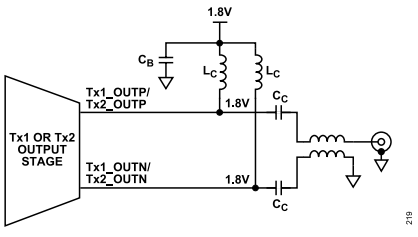


Figure 252. ADRV904x RF Transmitter Interface Configuration B

RF PORT IMPEDANCE MATCHING

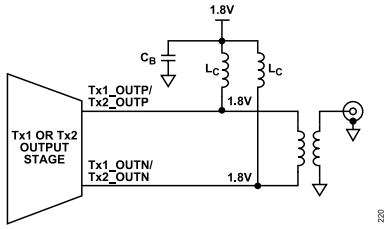


Figure 253. ADRV904x RF Transmitter Interface Configuration C

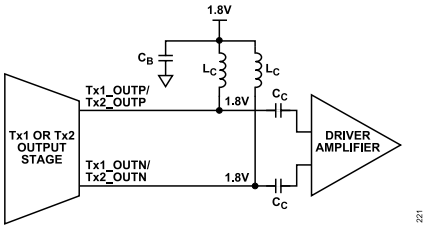


Figure 254. ADRV904x RF Transmitter Interface Configuration D

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 145. 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

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 0.7 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 0.7 V bias to ground.
- Board Design—reference planes, transmission lines, and impedance matching.

Figure 255 shows possible differential receiver port interface circuits. The options in Figure 255 and Figure 256 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.

RF PORT IMPEDANCE MATCHING

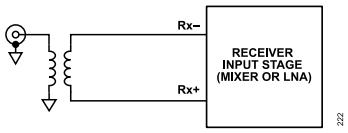


Figure 255. Differential Receiver Input Interface Circuits

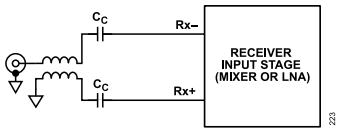


Figure 256. Differential Receiver Input Interface Circuits

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 ADRV904x is 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 ADRV904x 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 ADRV904x 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.

The following information about ADRV904x package is provided in addition to the dimensions provided in the data sheet.

- ADRV904x BGA package is SMD
- Ball pad diameter is 540+/−50 µm
- Pad solder mask opening is 450+/−30 µm
- Ball size is 500+/50 µm

This section discusses the following issues and provides guidelines for system designers to get the best performance out of the ADRV904x 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 ADRV904x 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 ADRV904x 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 ADRV904x 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 257 shows the PCB stack up used for the ADRV904x 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, 15, 16, and 17 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

PCB LAYOUT CONSIDERATIONS

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.

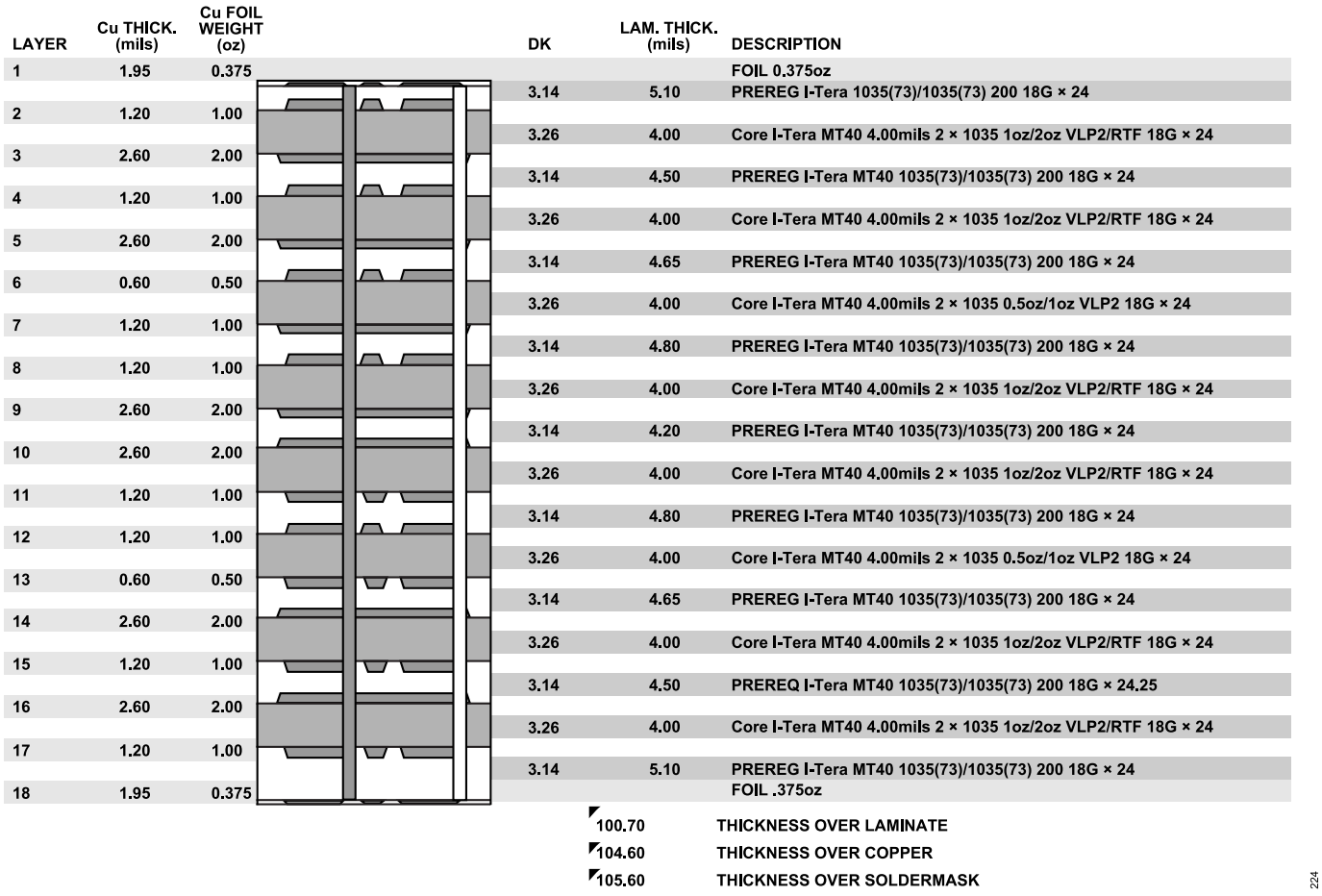


Figure 257. PCB Material Stack Up Diagram

Table 146 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 146. 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 147 describes details about trace impedance controls used in the ADRV904x evaluation board and types of line structures used to obtain desired impedance and performance on, and for, given layers and impedances.

PCB LAYOUT CONSIDERATIONS

Table 147. 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
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 ADRV904x uses a 736-ball BGA, 27 mm × 20 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 258 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 259 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

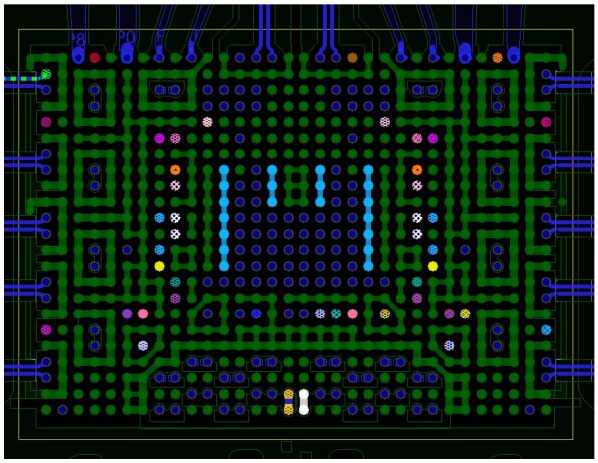


Figure 258. ADRV904x Customer Evaluation Board RF Observation Receiver and Transmitter Fanout and Layout

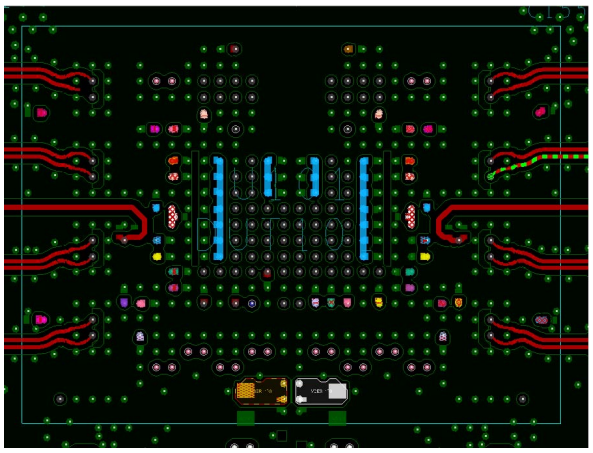


Figure 259. ADRV904x Customer Evaluation Board RF Receiver, Fanout, and Layout

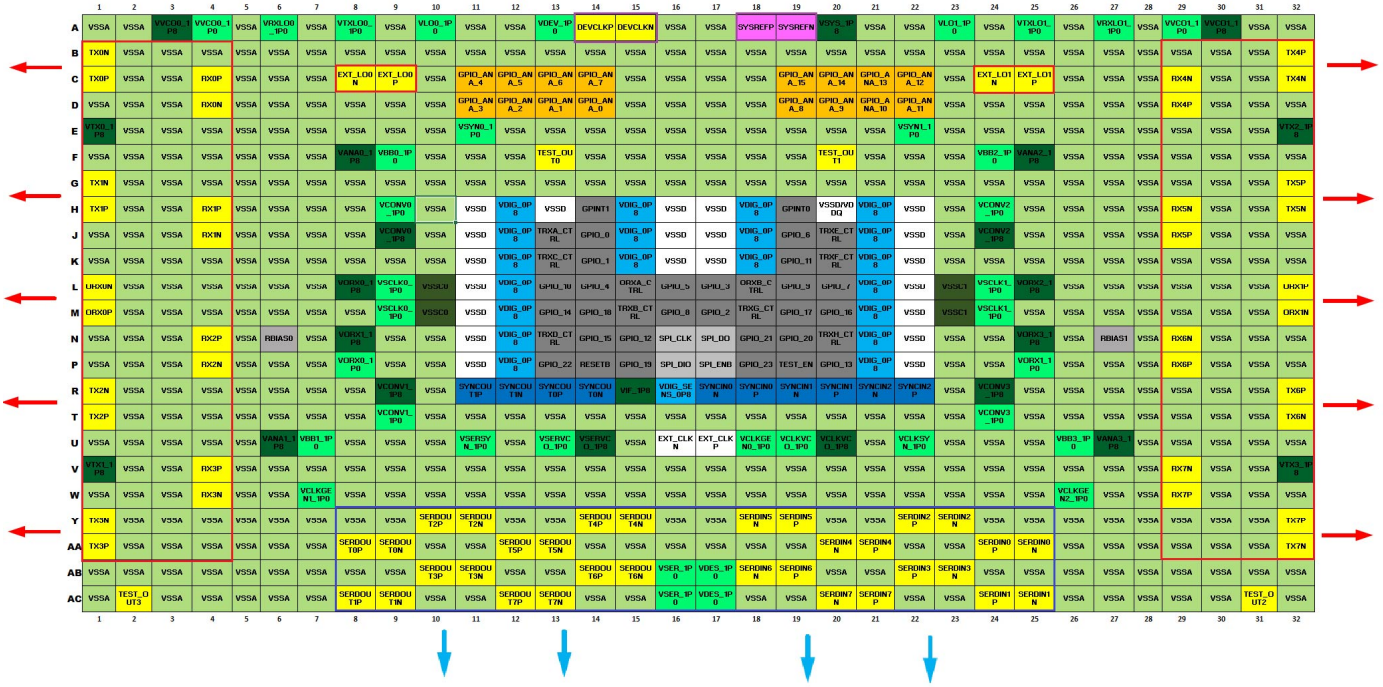
COMPONENT PLACEMENT AND ROUTING GUIDELINES

The ADRV904x 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 260 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.

PCB LAYOUT CONSIDERATIONS



227

Figure 260. 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.

Connect external clock inputs to DEVCLK_P and DEVCLK_N through 0.1 µF ac coupling capacitors. Place a 100 Ω termination near the input pin A14 and pin A15, as shown in Figure 261. 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.

PCB LAYOUT CONSIDERATIONS

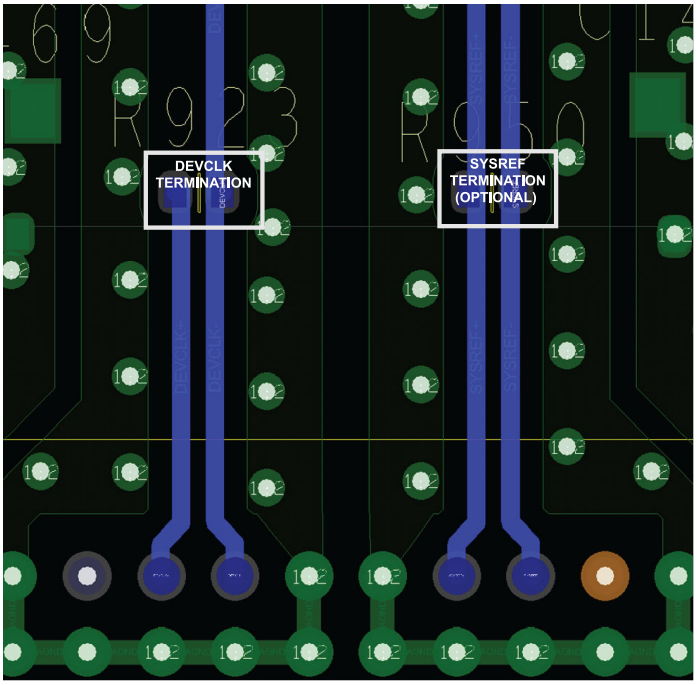


Figure 261. DEVCLK and SYSREF Termination

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 Designsection 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 Designsection to optimize RF and isolation performance.

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 V and column 10 and column 23 down and away from sensitive analog signals on PCB signal layers (see Figure 260 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 ADRV904x.

RF AND JESD TRANSMISSION LINE LAYOUT

RF Routing Guidelines

The ADRV904x 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.

The general RF routing guidelines are as follows:

- 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.

PCB LAYOUT CONSIDERATIONS

- 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 262 shows the routing for the Rx1 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 ADRV904x. If the matching circuit is designed for a single channel bandwidth, it may be possible to eliminate one of the differential pi networks.

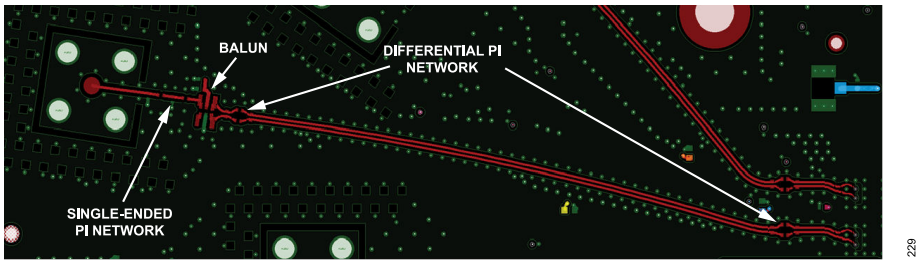


Figure 262. Receiver RF Routing and Matching Network

Figure 263 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.

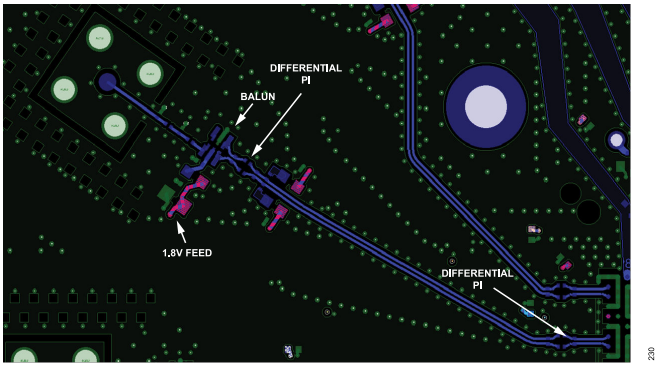


Figure 263. Transmitter RF Routing and Matching Network

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 ADRV904x customer evaluation board as shown in Figure 264.

PCB LAYOUT CONSIDERATIONS

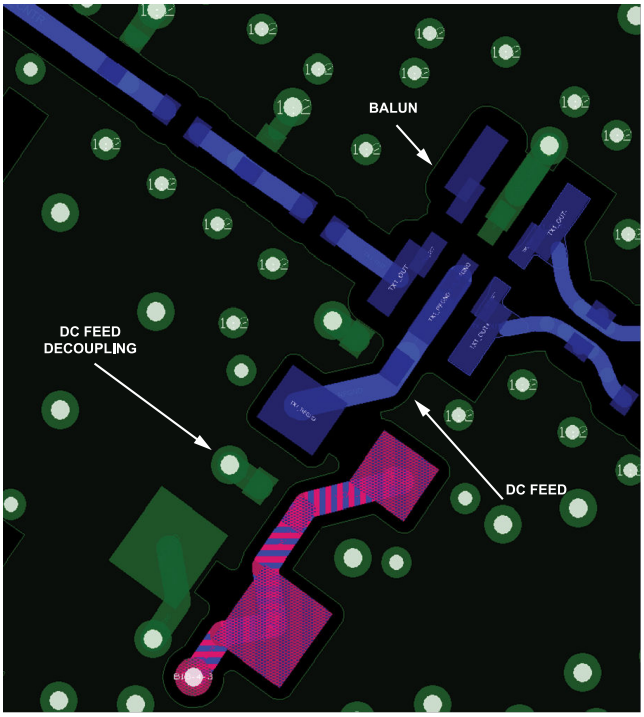


Figure 264. 1.8 V Transmitter Bias Routing at Balun

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.

The ADRV904x 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 265 shows an example of this arrangement.

PCB LAYOUT CONSIDERATIONS

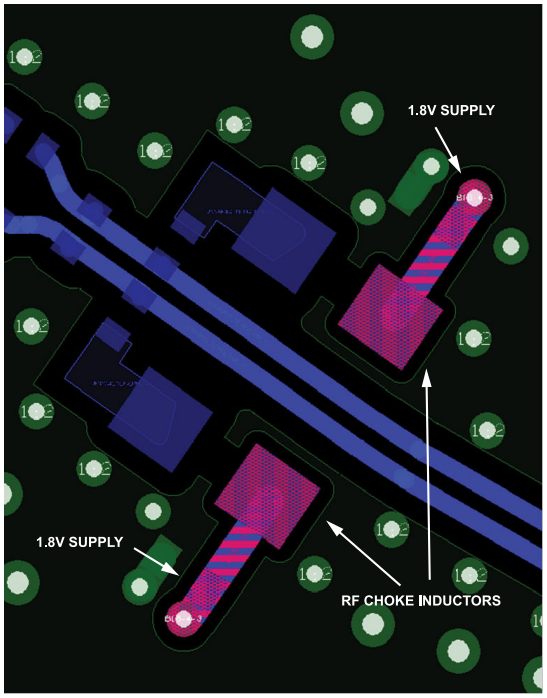


Figure 265. Transmitter Balun RF Choke DC Supply Connection

JESD204B/JESD204C Routing Recommendations

The ADRV904x 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 ADRV904x 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 ADRV904x 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 ADRV904x 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.

Figure 266 and Figure 267 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.

PCB LAYOUT CONSIDERATIONS

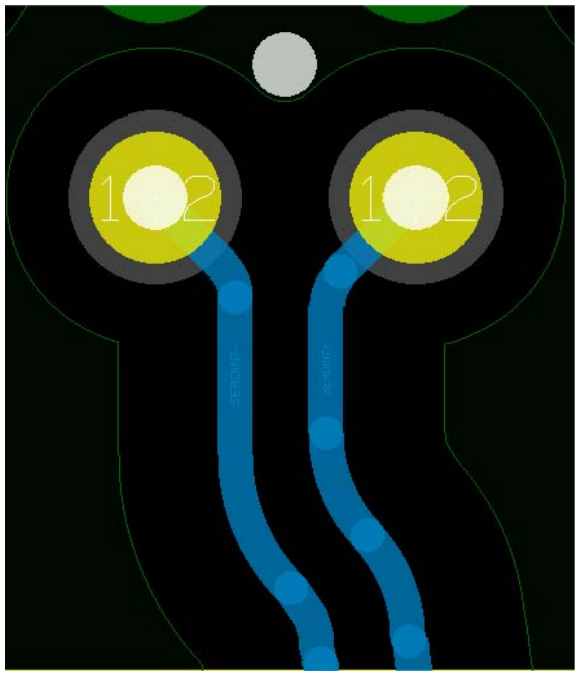


Figure 266. SERDES Signal Launch on Layer 2

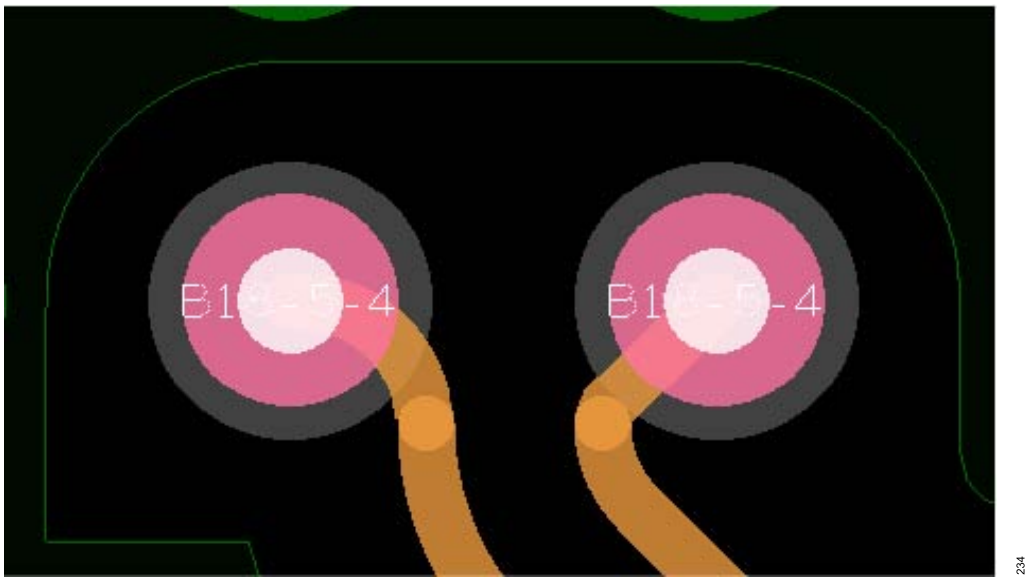


Figure 267. SERDES Signal Launch on Layer 4

ISOLATION TECHNIQUES

Given the density of sensitive and critical signals, significant isolation challenges are faced when designing a PCB for the ADRV904x 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 as follows:

- Magnetic field coupling

PCB LAYOUT CONSIDERATIONS

- Surface propagation
- Cross domain coupling via ground

To reduce the impact of these coupling mechanisms on the ADRV904x 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.

$$Wavelength\left(\mathrm{m}\right) = \frac{300}{frequency\left(\mathrm{MHz}\right) \times \sqrt{\varepsilon_r}} \quad (57)$$

(3)

These techniques are illustrated in Figure 268.

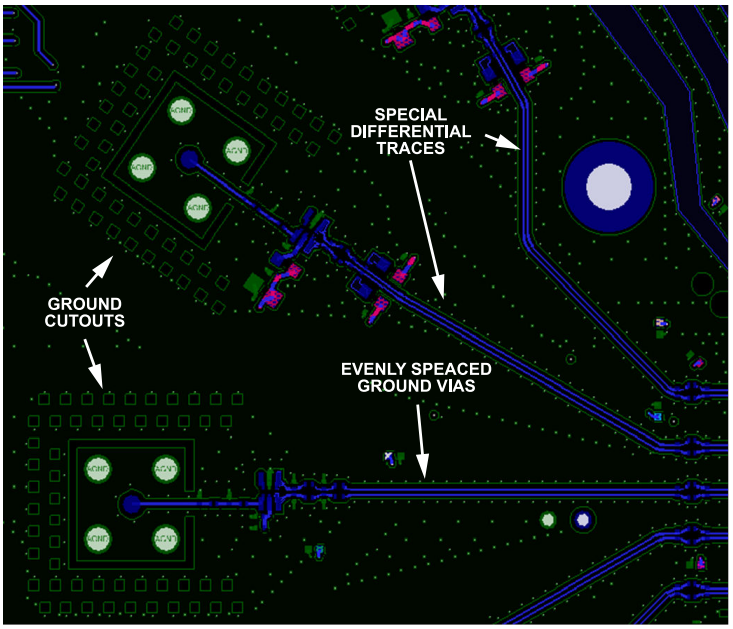


Figure 268. RF I/O Isolation Structures

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 ADRV904x evaluation board based on electromagnetic simulation results that show this to be the best technique for minimizing crosstalk.

Figure 269 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 electromagnetic simulation software.

PCB LAYOUT CONSIDERATIONS

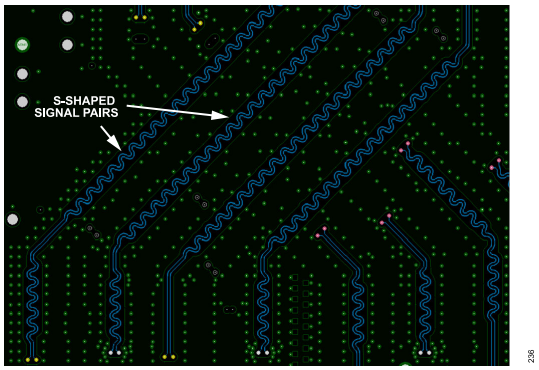


Figure 269. SERDES Lane Routing and Isolation

POWER MANAGEMENT LAYOUT DESIGN

Due to the complexity and high level of integration in the ADRV904x, power supply routing is critical to achieve optimum RF performance. The device is designed with 75 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 ADRV904x 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 ADRV904x 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 270 through Figure 272 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 273 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.

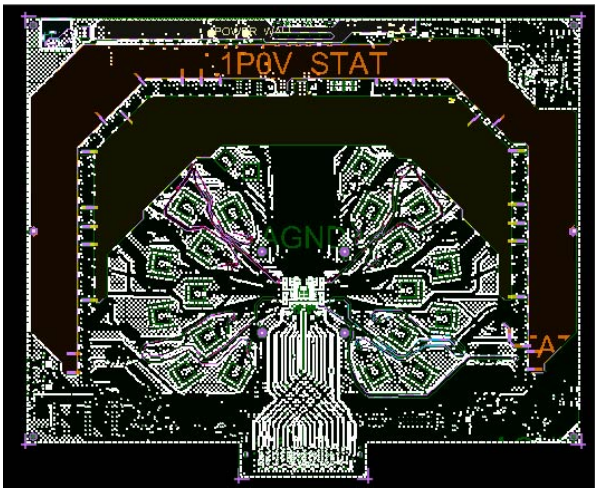
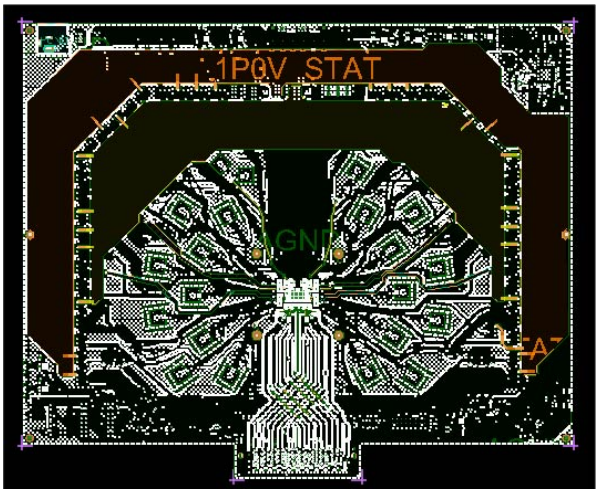


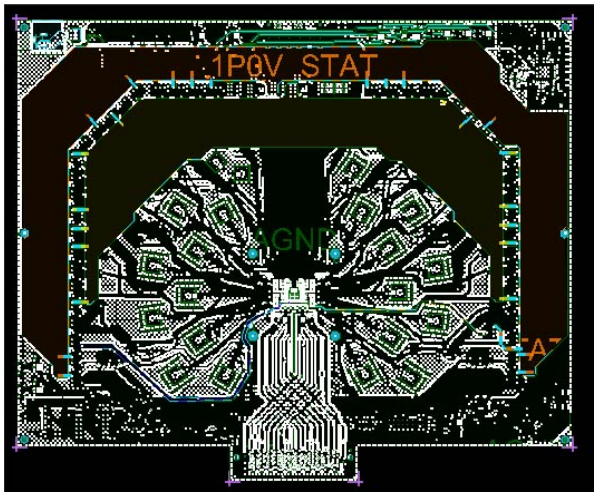
Figure 270. 1.0 V Analog Power Ring Layout Approach—Layer 3

PCB LAYOUT CONSIDERATIONS



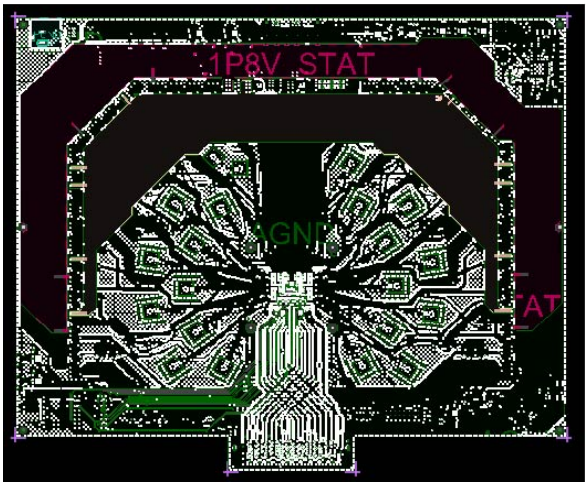
238

Figure 271. 1.0 V Analog Power Ring Layout Approach—Layer 4



239

Figure 272. 1.0 V Analog Power Ring Layout Approach—Layer 5



240

Figure 273. 1.8 V Analog Power Ring Layout Approach—Layer 10

PCB LAYOUT CONSIDERATIONS

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 274 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.

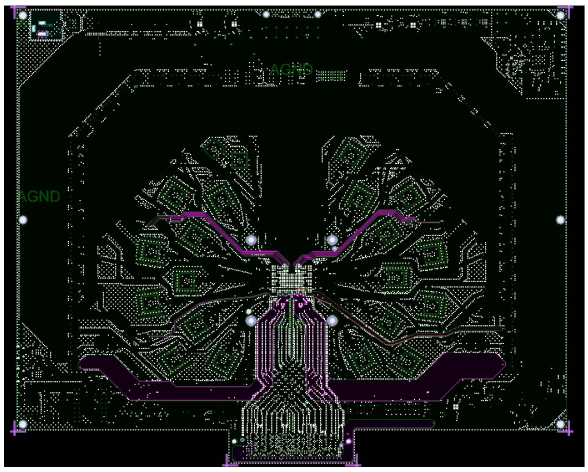


Figure 274. 1.8 V Supply Routing Using Star Connections

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 — 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 — 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 impedance. Figure 275 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.

PCB LAYOUT CONSIDERATIONS

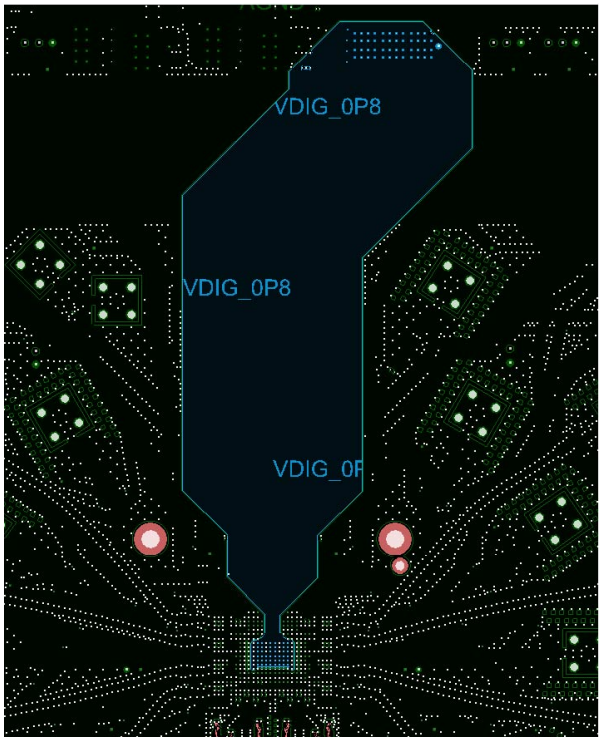


Figure 275. Digital Supply Routing

Figure 276 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

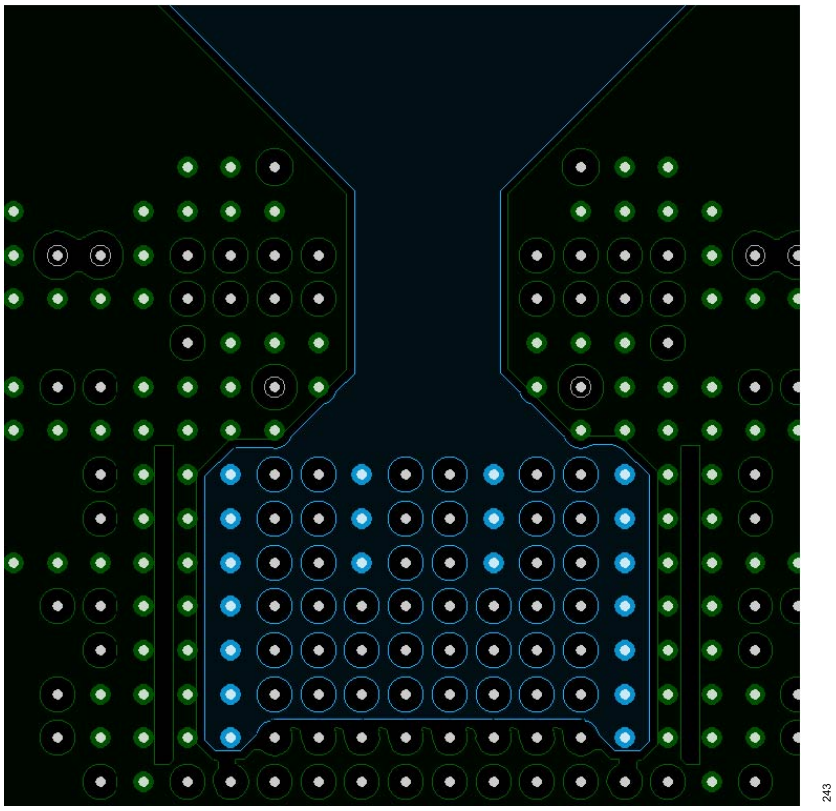


Figure 276. Digital Supply Connection to SIX VDIG_0P8V Input Pins

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 —. 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

PCB LAYOUT CONSIDERATIONS

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.

- Figure 277 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.

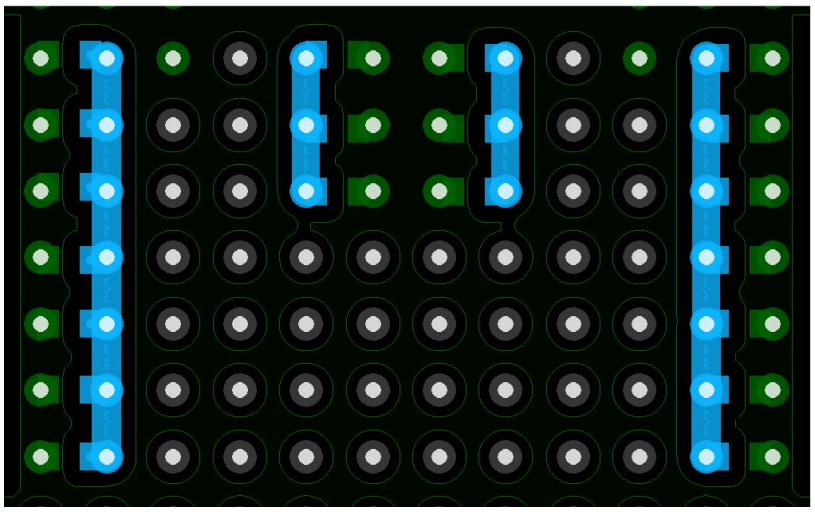


Figure 277. Power Supply Decoupling Capacitor Example

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 ADRV904x. 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 ADRV904x device 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 148 for unused pins.

PCB LAYOUT CONSIDERATIONS

Table 148. Recommendations for Unused Pins

Pin No.	Type[1]	Mnemonic	When Pins Are Not Used
B1, C1, C32, B32, G1, H1, H32, G32, R1, T1, T32, R32, Y1, AA1, AA32, Y32	O	TX0N, TX0P, TX4P, TX4N, TX1N, TX1P, TX5P, TX5N, TX2N, TX2P, TX6P, TX6N, TX3N, TX3P, TX7P, TX7N	Do not connect.
D4, C4, C29, D29, J4, H4, H29, J29, P4, N4, N29, P29, W4, V4, V29, W29	I	RX0P, RX0N, RX4N, RX4P, RX1P, RX1N, RX5N, RX5P, RX2P, RX2N, RX6N, RX6P, RX3P, RX3N, RX7N, RX7P	Connect to VSSA.
L1, M1, M32, L32	I	ORX0N, ORX0P, ORX1P, ORX1N	Connect to VSSA.
D13, D14, D12, D21, D19, D20, C13, C11, C22, C20, C12, D11, C14, C19, D22, D11	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.
J13, J20, M15, K20, K13, M18, N13, N20, L15, L18	I	TRXA_CTRL, TRXE_CTRL, TRXB_CTRL, TRXF_CTRL, TRXC_CTRL, TRXG_CTRL, TRXD_CTRL, TRXH_CTRL, ORXA_CTRL, ORXB_CTRL	Connect to VSSA.
C8, C9, C24, C25	I	EXT_LO0N, EXT_LO0P, EXT_LO1N, EXT_LO1P	Do not connect.
J14, K14, M17, L17, L14, L16, J19, L20, M16, L19, L13, K19, N15, P20, M13, N14, M20, M19, M14, P15, N19, N18, P13, P18	I/O	GPIO_0 to GPIO_23	Connect to VSSA with a 10 kΩ resistor or configure as outputs, drive low, and leave disconnected.
H19, H14	O	GPINT0, GPINT1	Do not connect.
P16	O	SPI_DO	Do not connect.
P19	I	TEST_EN	Connect to VSSA.
R17, R18, R19, R20, R21, R22	I	SYNCIN0N, ,SYNCIN0P, SYNCIN1N, ,SYNCIN1P, SYNCIN2N, ,SYNCIN2P	Connect to VSSA.
R14, R13, R12, R11	O	SYNCOUT1P, SYNCOUT1N, SYNCOUT0P, SYNCOUT0N	Do not connect.
AC9, AC8, Y15, Y14, AA9, AA8, AA13, AA12, AB11, AB10, AB15, AB14, Y11, Y10, AC13, AC12	O	SERDOUT1P, SERDOUT1N, SERDOUT4P, SERDOUT4N, SERDOUT0P, SERDOUT0N, SERDOUT5P, SERDOUT5N, SERDOUT3P, SERDOUT3N, SERDOUT6P, SERDOUT6N, SERDOUT2P, SERDOUT2N, SERDOUT7P, SERDOUT7N	Do not connect.
Y18, Y19, AC25, AC24, AA20, AA21, AA25, AA24, AB18, AB19, AB23, AB22, AC20, AC21, Y23, Y22	I	SERDIN5N, SERDIN5P, SERDIN1P, SERDIN1N, SERDIN4N, SERDIN4P, SERDIN0P, SERDIN0N, SERDIN6N, SERDIN6P, SERDIN3P, SERDIN3N, SERDIN7N, SERDIN7P, SERDIN2P, SERDIN2N	Do not connect.

[1] O = output, I = input, and I/O = input/output.