ADRV9001 System Development User Guide for the RF Agile Transceiver Family

ADRV9001 SYSTEM DEVELOPMENT USER GUIDE OVERVIEW

The ADRV9001 system development user guide covers the following:

- ADRV9002
- ADRV9003
- ADRV9004
- ADRV9005
- ADRV9006

The ADRV9001 is the part number designation used throughout this user guide to refer to all products comprising the ADRV9001 family shown above. Some family members do not include all the features or functions. For each feature and function, refer to the individual product data sheet. A synopsis of the main differentiators is also given in Table 1.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

REVISION HISTORY

8/2024—Rev. 0 to Rev. A

Changes to ADRV9001 System Development User Guide Overview Section....1
Added ADRV9005 Section....11
Added ADRV9006 Section....11
Added ADRV9001 Product Family Comparison Section and Table 1; Renumbered Sequentially....12
Changes to API Initialization Sequence Section....46
Changes to Warm Boot Section and Table 15....50
Added Operating Temperature Considerations Section....52
Changes to Table 16....54
Changes to Warm Boot Boot-Up Section....55
Added Table 17....55
Added SPI Broadcast Mode Section and Figure 27; Renumbered Sequentially....57
Added Table 19....58
Changes to Timing Diagrams Section....60
Changes to Table 24....64
Changes to Enhanced Rx SSI Mode Section....76
Added Figure 58....77
Changes to MCS Procedure and Status Check Section....105
Changes to RF Synthesizer Section....110
Added RF Synthesizer Frequency Accuracy Section....111
Added Example Frequency Accuracy Calculation Section....111
Changes to Local Oscillator (LO) Change Procedure Section....114
Changes to Modes of Operation Section and Table 48....117
Changes to Example 1: Load New Frequencies with Automatic Ping Pong Section....122
Changes to Example 2: Loading a Larger Set of Frequencies with Manual Table Switch Section....123

TABLE OF CONTENTS

TABLE OF CONTENTS

5/2023—Revision 0: Initial Version

HOW TO USE THIS DOCUMENT

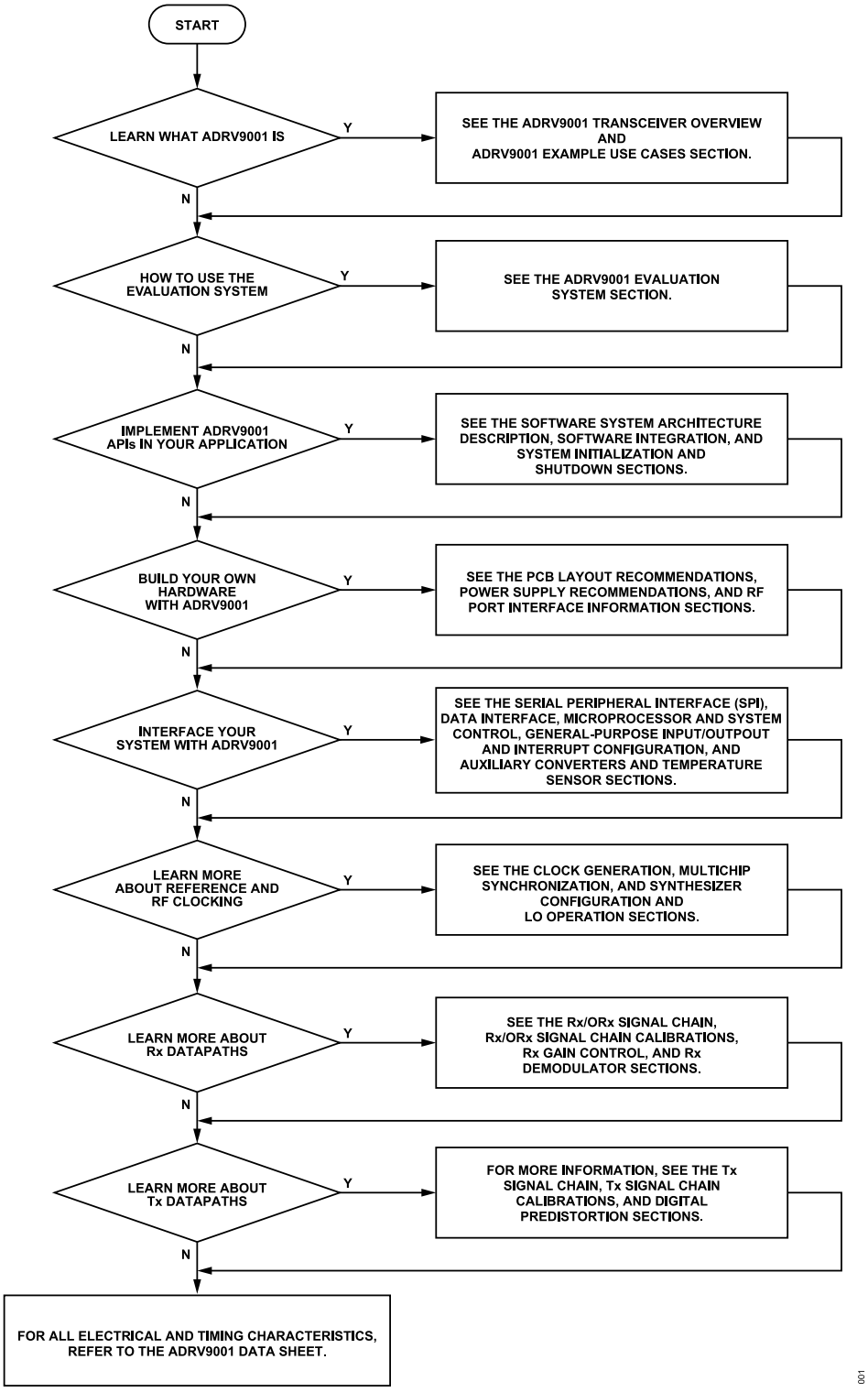


Figure 1. Flowchart for Document Navigation

BLOCK DIAGRAM

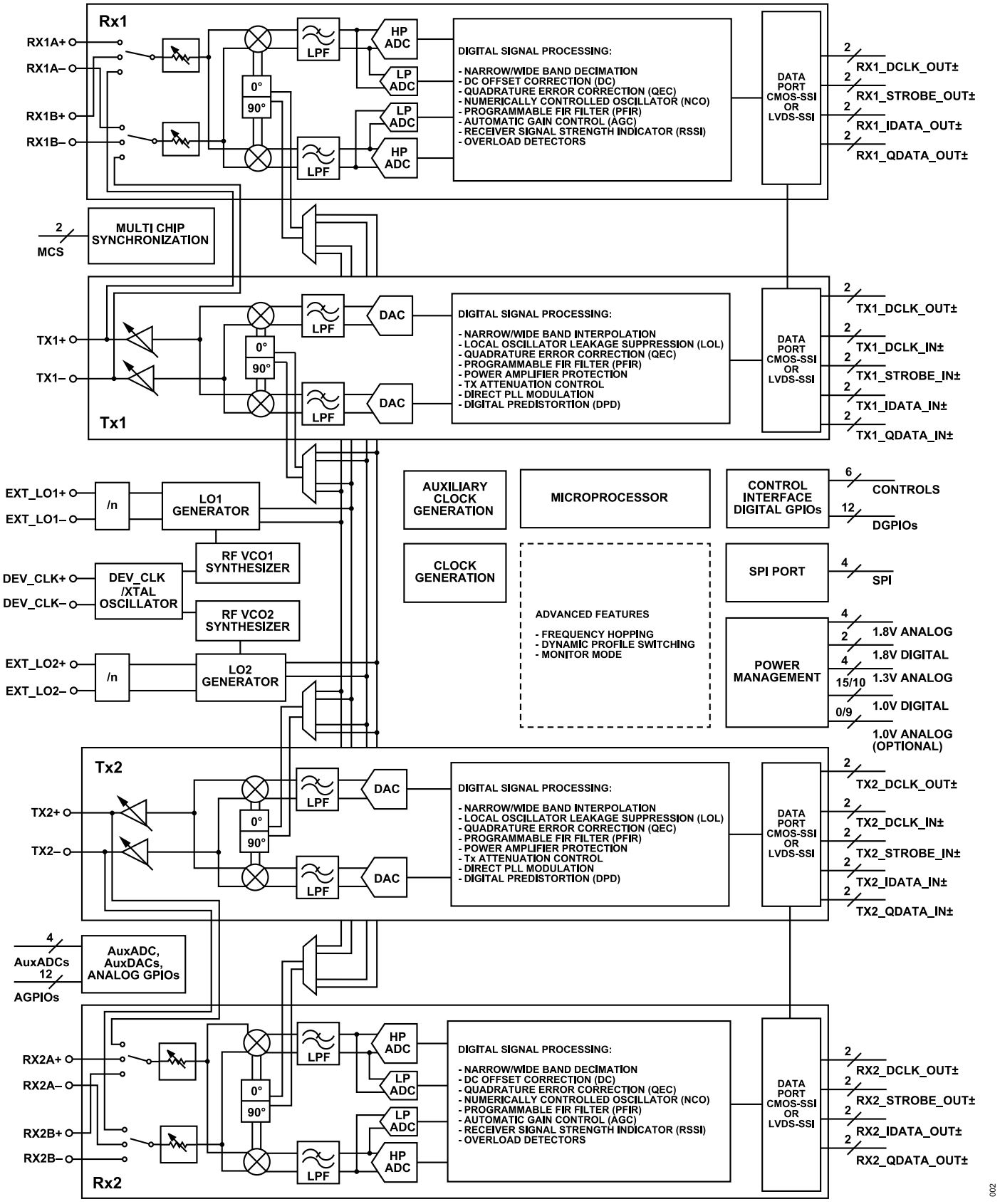


Figure 2. ADRV9002 Block Diagram

PRODUCT HIGHLIGHTS

ADRV9002

The ADRV9002 delivers a versatile combination of high performance and low power consumption required by battery-powered radio equipment. It can operate in both the frequency division duplex (FDD) and time division duplex (TDD) modes. The ADRV9002 operates from 30 MHz to 6000 MHz, which covers the very high frequency (VHF), licensed and unlicensed cellular bands, and industrial, scientific, and medical (ISM) bands. The IC supports both narrowband and wideband standards up to 40 MHz bandwidth on both a receiver and transmitter.

The transceiver consists of direct conversion signal paths with state-of-the-art noise figure and linearity. Each complete receiver and transmitter sub-system includes DC offset correction, quadrature error correction (QEC), and programmable digital filters, which eliminate the need for these functions in the digital baseband. In addition, several integrated auxiliary functions such as an auxiliary analog-to-digital converter (ADC), auxiliary digital-to-analog converters (DACs), and general-purpose inputs/outputs (GPIOs) provide additional monitoring and control capability.

The fully-integrated phase locked loops (PLLs) provide high performance, low power fractional-N frequency synthesis for the transmitter, receiver, and clock sections. Careful design and layout techniques provide the isolation required in high performance mobile radio applications.

All integrated voltage controlled oscillator (VCO) and loop filter components minimize the external component count. The local oscillators (LOs) have flexible configuration options and include fast lock modes.

The transceiver includes low-power sleep and monitor modes to save power, which extends the battery life of portable devices while monitoring communication.

The fully-integrated low power DPD is supported by the ADRV9002. It can linearize wideband and narrowband signals to enable the linearization of high efficiency power amplifiers. In use cases where the integrated DPD is used, main receivers are used as a power amplifier observation path.

The power supply for the ADRV9002 is distributed across four or five different voltage supplies: 2 or 3 analog and 2 digital. The analog supplies are 1.8 V, 1.3 V, and 1.0 V (in internal low dropout (LDO) bypass mode). The 1.3 V domain directly feeds some blocks and internal LDO regulators for some functions as well for maximum performance. The 1.8 V analog domain optimizes transmitter and auxiliary converter performance. The digital processing blocks are supplied by a 1.0 V source. In addition, a 1.8 V supply is used to supply all GPIO and interface ports that connect with the baseband processor.

High and low data rate interfaces are supported using configurable complementary metal oxide semiconductor (CMOS) or low voltage differential signaling (LVDS) synchronous-serial interface (SSI) choice.

The core of the ADRV9002 is controlled through a standard 3-wire or 4-wire serial port. All software control is communicated through this interface. There is also a control interface that uses GPIO lines to provide hardware control to and from the device. These pins can be configured to provide dedicated sets of functions for different application scenarios.

The block diagram in Figure 2 shows a high level view of the functions in the ADRV9002. The subsequent sections of this document describe each block with setup and control details.

Bandwidth and Sample Rate Support

The ADRV9002 supports the reception and transmission of channels up to 40 MHz bandwidth. The available standard sample rates are 24 kHz (typically for narrowband FM waveforms), 144 kHz, and 288 kHz (typically for terrestrial trunked radio (TETRA) signals), and 1.92 MHz, 3.84 MHz, 7.68 MHz, 15.36 MHz, 23.04 MHz, 30.72 MHz, and 61.44 MHz (typically for long-term evolution (LTE) signals).

In addition, the ADRV9002 supports an almost continuous range of sample rates between 24 kHz and 61.44 MHz. It does not support some sample rates due to internal clocking constraints.

The sample rate is scaled by enabling or disabling decimation or interpolation filters in the digital signal chain.

Data Interfaces

The ADRV9002 supports both the CMOS and LVDS electrical interfaces for its data lanes. All data lanes support both the electrical interfaces, but do not support their concurrent operation. Each transmit and receive channel has a dedicated set of lanes to transfer the information.

The CMOS bus speed is limited to 80 MHz. The CMOS-SSI electrical interface has two operating modes. For low sample rates, the mode has 32 bits (16 bits of I and Q data each) that are serialized over a single lane with two additional lanes total required for a clock (single data rate (SDR) or double data rate (DDR)) and a frame synchronization signal, which supports a maximum sample rate of 2.5 MHz.

PRODUCT HIGHLIGHTS

For sample rates above 2.5 MHz, single channel data is serialized over four lanes with two additional lanes required for a clock (SDR or DDR) and a frame synchronization signal, which supports a maximum sample rate of 20 MHz.

The LVDS electrical interface supports two modes of operation. The 32 total bits of I and Q data are serialized over one LVDS lane (32 bits composed of 16 bits of I and 16 bits of Q data) or two LVDS-SSI lanes (each dedicated to 16 bits of I or Q data), with two additional lanes total required for a DDR clock and a frame synchronization signal. The sample rates ranging from 24 kHz to 61.44 MHz are supported through the LVDS-SSI interface, resulting in a maximum lane rate of 983.04 MHz.

Note that in the LVDS-SSI mode, 12-bit I and Q words are supported for most sample rates.

RF LO Frequency Range and Multiplexing

The ADRV9002 supports an RF LO range from 30 MHz to 6 GHz. RF LOs can be generated through two internal PLLs or applied externally to the device.

An LO multiplexing scheme on the ADRV9002 routes either of the RFPLLs to any of the transmit or receive channels. The RF channels and RFPLLs can operate concurrently and independently, off a common reference clock, thus enabling FDD operation, single or dual frequency repeater operation, multiband TDD operation, and diversity operation among various other configurations.

Frequency Hopping

The ADRV9002 supports various forms of frequency hopping with the main distinguishing factor among them being the frequency transition time.

There is a fast frequency hopping (FFH) mode to support up to 128 hop frequencies or less, and these are preloaded through two hopping tables onto the ADRV9002 at power-up. All the frequencies can be cycled through in a circular buffer fashion.

A random access of the hopping frequencies is also supported, whereby a finite set of frequencies already preloaded onto the ADRV9002 can be hopped between in a random manner determined by the user. Select the next frequency to hop by asserting a frequency index word onto the GPIO bus.

In addition, the ADRV9002 also supports the loading of hopping frequencies on the fly. In this mode, change the hopping frequency while performing hopping.

Profile Switching

The ADRV9002 supports rapid switching among different RF channel profiles. A transmit or receive RF channel profile contains settings such as bandwidth, sample rate, filtering, input port selection, automatic gain control (AGC) settings, and algorithm configuration. The profile switching supports switching among a set of waveforms with different sampling rates on the fly.

Low IF Reception

The receiver digital datapath on the ADRV9002 contains an optional digital mixer driven by a programmable numerically controlled oscillator (NCO). The RX LO is offset from the frequency of the desired channel. Then, the digital mixer and NCO are used to down convert the signal to baseband before being processed by a baseband processor.

There are several advantages to offset the RX LO from the frequency of the desired channel: Impairments in the the RX LO, such as LO-leakage, can be avoided. The effect of flicker noise from baseband circuits can be reduced as the received signal is offset from the DC in the analog signal path. Also, image rejection can be improved if the RX LO is offset enough from the required channel, such that the image frequency lies in the attenuation region of the external RF filter.

The low IF reception mode predominately targets low bandwidth channels, which supports offsets in the range of ± 20 MHz from the receiver LO.

Receive Dynamic Range and Blocking

As shown in Figure 2, the ADRV9002 receive path consists of an input mixer followed by a baseband filter that drives an ADC. A highly programmable digital decimation and filtering datapath follows the ADC. There is RF analog gain control in an analog attenuator, and additional gain in the digital datapath through AGC loops.

PRODUCT HIGHLIGHTS

The ADC in the receive chain possesses a high dynamic range. Assuming 0 dB attenuation, the ADC's noise and maximum input power referred to the RF input are -142 dBm/Hz and 8.6 dBm, respectively. These levels translate into a dynamic range in excess of 150 dB on a per Hertz basis. A greater dynamic range can be achieved by considering the digital filtering and AGC loops.

Given the high dynamic range of the receiver ADC, very little channelization, or blocker filtering occurs in the analog signal chain as the ADC can simultaneously absorb weak signals and large blockers. The blocker suppression and channelization are then achieved in the digital signal path.

Reciprocal mixing of the RX LO phase noise by a large blocker close to the desired channel significantly degrades the blocking performance. If high blocker tolerance performance is required, use a lower phase noise external LO source in place of the on-board RFPLLs.

The receiver path also contains two types of ADCs connected to the chip's RF front end for tradeoff between power consumption and dynamic range: a high performance ADC with high linearity performance and a low power ADC with degraded linearity performance but less power consumption. Trade off receive channel linearity performance and power consumption by selecting either set of ADCs.

Power Consumption Modes

The ADRV9002 provides various levels of power control. The power scaling on individual analog signal path blocks can trade-off power and performance. In addition, enabling and disabling various blocks in TDD RX and TX frames to reduce power can be customized, at the expense of RX/TX or TX/RX turnaround time.

A specialized "RX Monitor mode" allows the ADRV9002 to autonomously poll a region of the spectrum for the presence of a signal while in a low power state. In this mode, the chip continuously cycles through sleep-detect-sleep states controlled by an internal state machine. The power savings are achieved by ensuring the sleep duty cycle is greater than the "detect" duty cycle.

In the "sleep" state, the chip is in a minimal power consumption configuration with only few functions enabled, and it also allows the baseband process to go to sleep. After a predetermined period, the chip enters the "detect" state. In this state, the chip enables a receiver and performs signal detection over a bandwidth and at an RX LO frequency determined by the user. If a signal is detected, the "Monitor Mode" state machine exits its cycle. Following the loop exit, an interrupt is provided through a GPIO pin to wake the baseband processor, and the entire receiver analog and digital chains within the ADRV9002 are powered up.

If the power measured over the bandwidth is less than the user-determined threshold, the chip resumes its sleep-detect-sleep cycle. The sleep-detect duty cycle and durations and signal detection method are user-programmable, and are set before enabling the monitor mode.

ADRV9003

The ADRV9003 delivers all the features and performance of the ADRV9002 transceiver. The differences between the ADRV9002 and ADRV9003 are:

- RF IOs. The ADRV9003 has two receivers and one transmitter.
- The ADRV9003 does not support the digital predistortion functionality.

ADRV9004

The ADRV9004 delivers all features and performance of the ADRV9002 transceiver. The difference between the ADRV9002 and ADRV9004 is:

- The ADRV9004 does not support the digital predistortion functionality.

ADRV9005

The ADRV9005 delivers all the features and performance of the ADRV9002 transceiver. The differences between the ADRV9002 and the ADRV9005 are the following:

- RF IOs. The ADRV9005 has one receiver and one transmitter.
- Tx Tracking calibrations can only be performed in TDD mode. Not available in FDD mode.

ADRV9006

The differences between the ADRV9002 and the ADRV9006 are as follows:

- The ADRV9006 only supports the frequency hopping functionality with 'normal' speed.

PRODUCT HIGHLIGHTS

- The ADRV9006 does not support the fast profile switching functionality.
- The ADRV9006 does not support the digital predistortion functionality.
- The ADRV9006 does not support the monitor mode functionality.
- The ADRV9006 only supports the Low Power ADC functionality.
- The ADRV9006 does not support the external LO functionality.

ADRV9001 PRODUCT FAMILY COMPARISON

Table 1 shows the differences between all products in the ADRV9001 family.

Table 1. ADRV9001 Product Family Comparison

Feature	ADRV9002	ADRV9003	ADRV9004	ADRV9005	ADRV9006
Channel Count	2T2R	1T2R	2T2R	1T1R	2T2R
RF Range	30 MHz to 6 GHz	30 MHz to 6 GHz	30 MHz to 6 GHz	30 MHz to 6 GHz	30 MHz to 6 GHz
Bandwidth	12kHz to 40 MHz	12kHz to 40 MHz	12kHz to 40 MHz	12kHz to 40 MHz	12kHz to 40 MHz
Rx Tracking Cals	Yes	Yes	Yes	Yes	Yes
Tx Tracking Cals	Yes	Yes	Yes	Yes[1]	Yes
Frequency Hopping	Yes	Yes	Yes	Yes	Yes
Frequency Hopping Speed	Fast/Normal	Fast/Normal	Fast/Normal	Fast/Normal	Normal
Fast Profile Switching	Yes	Yes	Yes	Yes	No
Multi-Chip Synchronization	Yes	Yes	Yes	Yes	Yes
Digital Pre-Distortion	Yes	No	No	Yes	No
Automatic Gain Control	Yes	Yes	Yes	Yes	Yes
Closed Loop Gain Control	Yes	Yes	Yes	Yes	Yes
Monitor Mode	Yes	Yes	Yes	Yes	No
Power Saving Modes	Yes	Yes	Yes	Yes	Yes
ADC Option	HP/LP	HP/LP	HP/LP	HP/LP	LP
External LO Option	Yes	Yes	Yes	Yes	No

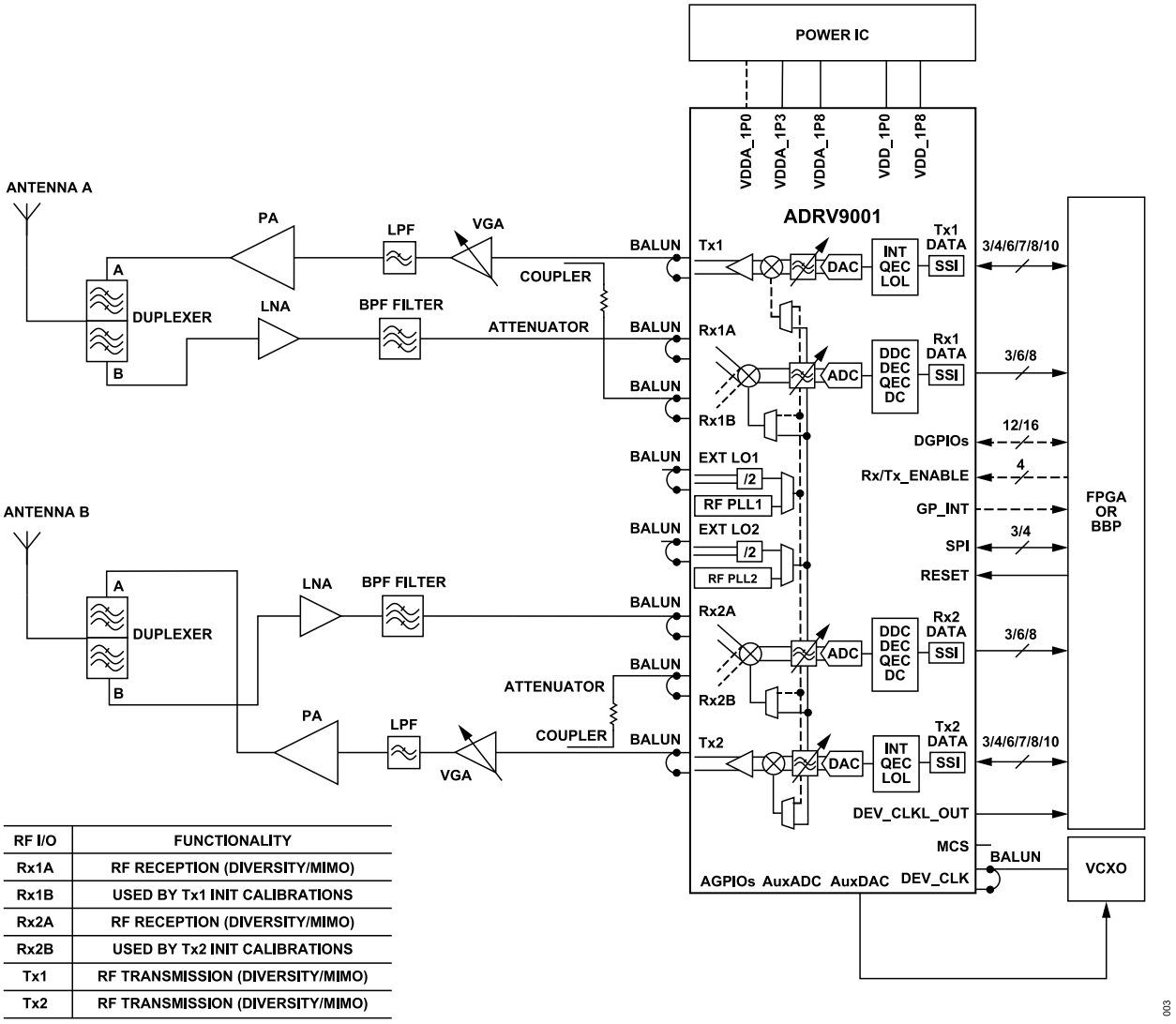
[1] ADRV9005 can support Tx tracking cals when configured for TDD profiles. Not available during FDD operation.

ADRV9001 EXAMPLE USE CASES

The section provides an overall idea on how an ADRV9001 integrated transceiver can operate as an RF front end in different applications. The list is not exhaustive, and there are other applications the ADRV9001 can serve.

Each example is accompanied with a table that explains the main limitations and highlights when implementing the ADRV9001 in the end application.

ADRV9001 IN A SINGLE-BAND 2T2R FDD TYPE SMALL-CELL APPLICATION



RF I/O	FUNCTIONALITY
Rx1A	RF RECEPTION (DIVERSITY/MIMO)
Rx1B	USED BY Tx1 INIT CALIBRATIONS
Rx2A	RF RECEPTION (DIVERSITY/MIMO)
Rx2B	USED BY Tx2 INIT CALIBRATIONS
Tx1	RF TRANSMISSION (DIVERSITY/MIMO)
Tx2	RF TRANSMISSION (DIVERSITY/MIMO)

Figure 3. ADRV9001 in a Single-Band 2T2R FDD Type Small-Cell Application

Single-Band 2T2R FDD Overview

With a minimum number of external components, the ADRV9001 transceiver can be used to build complete RF-to-bits signal chains that can serve as the RF front end in small-cell type applications. The ADRV9001 dual receiver and transmitter signal chains allow to implement multiple-input multiple-output (MIMO) or diversity in the system. The ADRV9001 internal AGC can be used to autonomously monitor and set appropriate gain levels for the receiver signal chains. For non-time critical FDD type applications, the ADRV9001 transceiver can be controlled through application programming interface (API) commands that use the serial peripheral interface (SPI).

Table 2. Constraints and Limitations in a Single-Band 2T2R FDD Type Small-Cell Application

Functionality	Constraints and Limitations
Receiver Signal Path	Ensure an appropriate level of isolation between Rx1 and Rx2 as well as receiver to transmitter at the system level. In the example, the RxB inputs are used only during initialization calibrations.

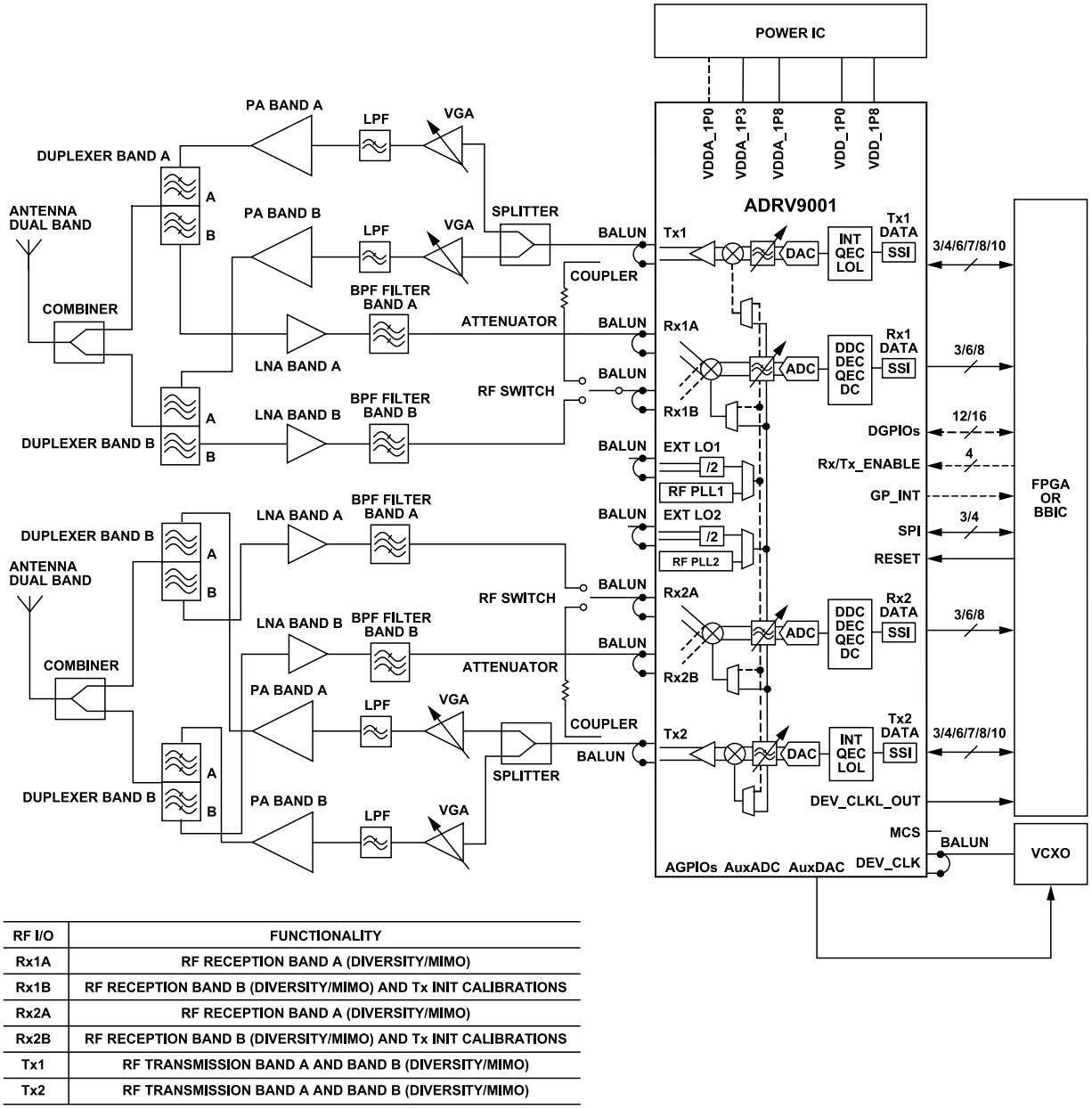
ADRV9001 EXAMPLE USE CASES

Table 2. Constraints and Limitations in a Single-Band 2T2R FDD Type Small-Cell Application (Continued)

Functionality	Constraints and Limitations
Transmitter Signal Path	Ensure an appropriate level of isolation between Tx1 and Tx2 as well as receiver to transmitter at the system level.
LO Generation	In FDD type small-cell applications, the ADRV9001 can use its internal LO to generate RF LO1 for uplink (Rx1 and Rx2) and RF LO2 for downlink (Tx1 and Tx2). It is also possible to use external LO inputs in this mode of operation.
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality is not available when the ADRV9001 operates in the 2T2R FDD mode.
Calibrations	During the receiver initialization sequence, ensure that there are no signals present at the receiver input (external low noise amplifier (LNA) must be disabled), and appropriate termination must be present at the LNA output to avoid reflections of receiver calibration tones. During the transmitter initialization sequence, ensure that the power amplifier is powered down to avoid unwanted emission of transmitter calibration tones at the antenna. No transmitter tracking calibrations are available when the ADRV9001 operates in the 2T2R FDD mode.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels in the end user system. AGPIOs can be used to control states of external components or read back digital logic levels from external components.
DGPIOs	Digital GPIOs can be used to perform real-time monitoring of states of internal ADRV9001 blocks. Digital GPIOs operating as inputs can control receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.95 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV9001 device clock and control any circuitry that requires analog control voltage up to 1.75 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power-up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization (MCS), initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN A DUAL-BAND 2T2R FDD TYPE SMALL-CELL APPLICATION



RF I/O	FUNCTIONALITY
Rx1A	RF RECEPTION BAND A (DIVERSITY/MIMO)
Rx1B	RF RECEPTION BAND B (DIVERSITY/MIMO) AND Tx INIT CALIBRATIONS
Rx2A	RF RECEPTION BAND A (DIVERSITY/MIMO)
Rx2B	RF RECEPTION BAND B (DIVERSITY/MIMO) AND Tx INIT CALIBRATIONS
Tx1	RF TRANSMISSION BAND A AND BAND B (DIVERSITY/MIMO)
Tx2	RF TRANSMISSION BAND A AND BAND B (DIVERSITY/MIMO)

Figure 4. ADRV9001 in a Dual-Band 2T2R FDD Type Small-Cell Application

Dual-Band 2T2R FDD Overview

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete dual and RF-to-bits signal chain that can serve as RF front end in small-cell type applications. Note that in the proposed solution, only one band can be used at a time. The ADRV9001 dual receiver and transmitter signal chains enables users to implement multiple input multiple output (MIMO) or diversity in their system. The ADRV9001 internal AGC can be used to autonomously monitor and set appropriate gain level for receiver signal chains. For none-time critical FDD type applications, the ADRV9001 transceiver can be controlled through API commands that use the serial peripheral interface (SPI).

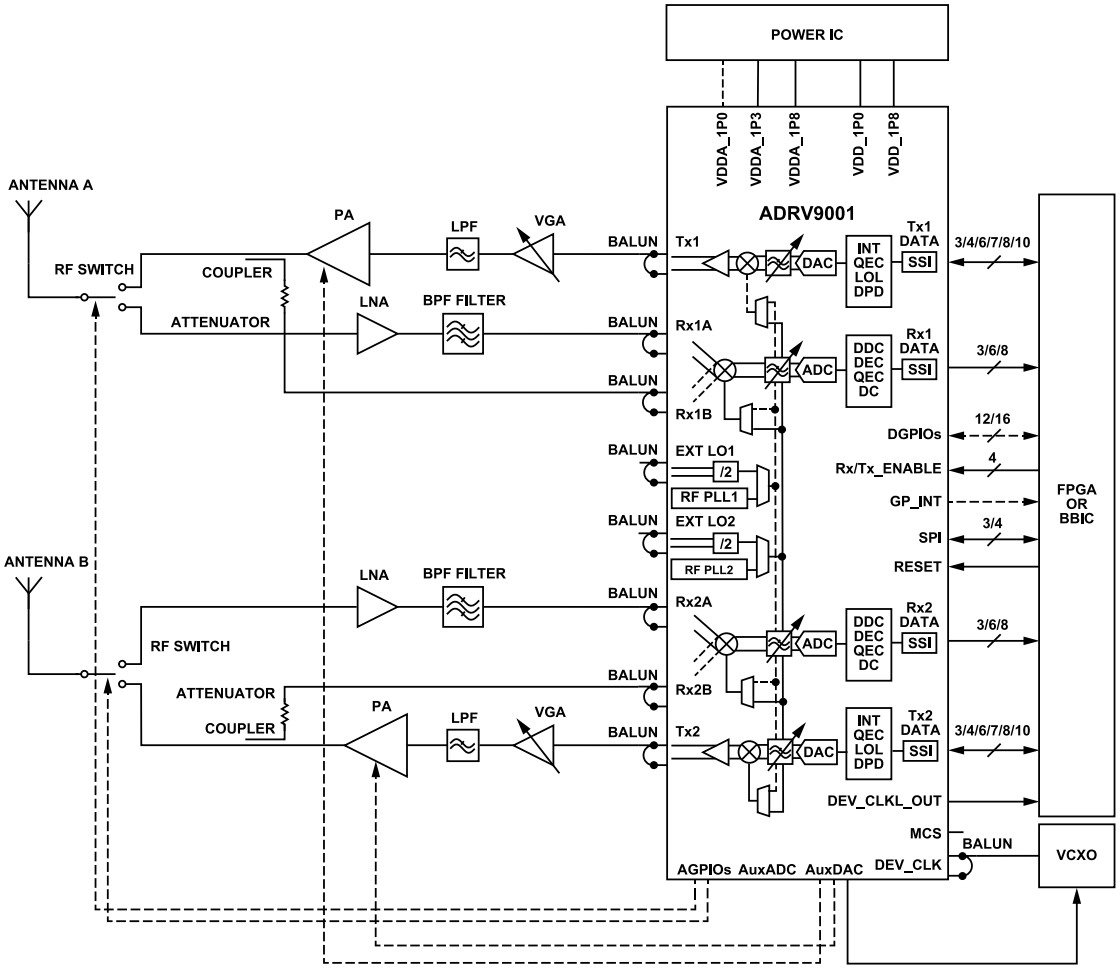
ADRV9001 EXAMPLE USE CASES

Table 3. Constraints and Limitations in a Dual-Band 2T2R FDD Type Small-Cell Application

Functionality	Constraints and Limitations
Receiver Signal Path	Ensure an appropriate level of isolation between Rx1 and Rx2 as well as receiver to transmitter at the system level. In the previous example, RxB inputs are used to work with receiver Band B signals as well as during initialization calibrations. In this scenario, the RF Balun selected for RxB inputs must work with both Band B and transmitter bands.
Transmitter Signal Path	Ensure an appropriate level of isolation between Tx1 and Tx2 as well as receiver to transmitter at the system level.
LO Generation	In FDD type small-cell applications, the ADRV9001 can use its internal LO to generate RF LO1 for uplink (Rx1 and Rx2) and RF LO2 for downlink (Tx1 and Tx2). It is also possible to use external LO inputs in this mode of operation. Note that only one set of the receiver inputs can be used at a time. This system can operate with two different FDD bands, but only one of those bands can be active at a particular moment in time.
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) are not affecting overall system performance.
DPD	The DPD functionality is not available when the ADRV9001 operates in the 2T2R FDD mode.
Calibrations	During the receiver initialization sequence, ensure that there are no signals at the receiver input (external LNA needs to be disabled), and appropriate termination must be present at LNA output to avoid reflections of receiver calibration tones. During the transmitter initialization sequence, ensure that the power amplifier is powered down to avoid unwanted emission of transmitter calibration tones at the antenna. No transmitter tracking calibrations are available when the ADRV9001 operates in the 2T2R FDD mode.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels in the end user system. AGPIOs can be used to control states of external components or read back digital logic levels from external components.
DGPIOs	Digital GPIOs can be used to perform real-time monitoring of states of internal ADRV9001 blocks. Digital GPIOs operating as inputs control receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.95 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV9001 device clock, and control any circuitry that requires analog control voltage up to 1.75 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide a reference clock signal to the digital components in the overall system. This output can be configured to be active after power-up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization, initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN A SINGLE-BAND 2T2R TDD TYPE SMALL-CELL APPLICATION



RF I/O	FUNCTIONALITY
Rx1A	RF RECEPTION (DIVERSITY/MIMO)
Rx1B	USED BY Tx1 CALIBRATIONS
Rx2A	RF RECEPTION (DIVERSITY/MIMO)
Rx2B	USED BY Tx2 CALIBRATIONS
Tx1	RF TRANSMISSION (DIVERSITY/MIMO)
Tx2	RF TRANSMISSION (DIVERSITY/MIMO)

Figure 5. ADRV9001 in a Single-Band 2T2R TDD Type Small-Cell Application

Single-Band 2T2R TDD Overview

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete RF-to-bits signal chain that can serve as RF front end in TDD type small cell type applications. The ADRV9001 dual receiver and transmitter signal chains enable users to implement MIMO or diversity in their system. In TDD type applications, internal DPD block can be used to linearize the external power amplifier and improve the overall system efficiency. The ADRV9001 internal AGC can be used to autonomously monitor and set appropriate gain level for the receiver signal chains. For time-critical TDD type applications, the ADRV9001 transceiver can be controlled by toggling control lines. The ADRV9001 can control the external receiver/transmitter switch using its analog GPIOs and can provide power amplifier bias voltage using AuxDAC outputs.

Table 4. Constraints and Limitations in a Single-Band 2T2R TDD Type Small-Cell Application

Functionality	Constraints and Limitations
LO Generation	In TDD type small cell applications, the ADRV9001 can use its internal LO to generate RF LO1 for both uplink and downlink. It is also possible to use external LO inputs in this mode of operation.

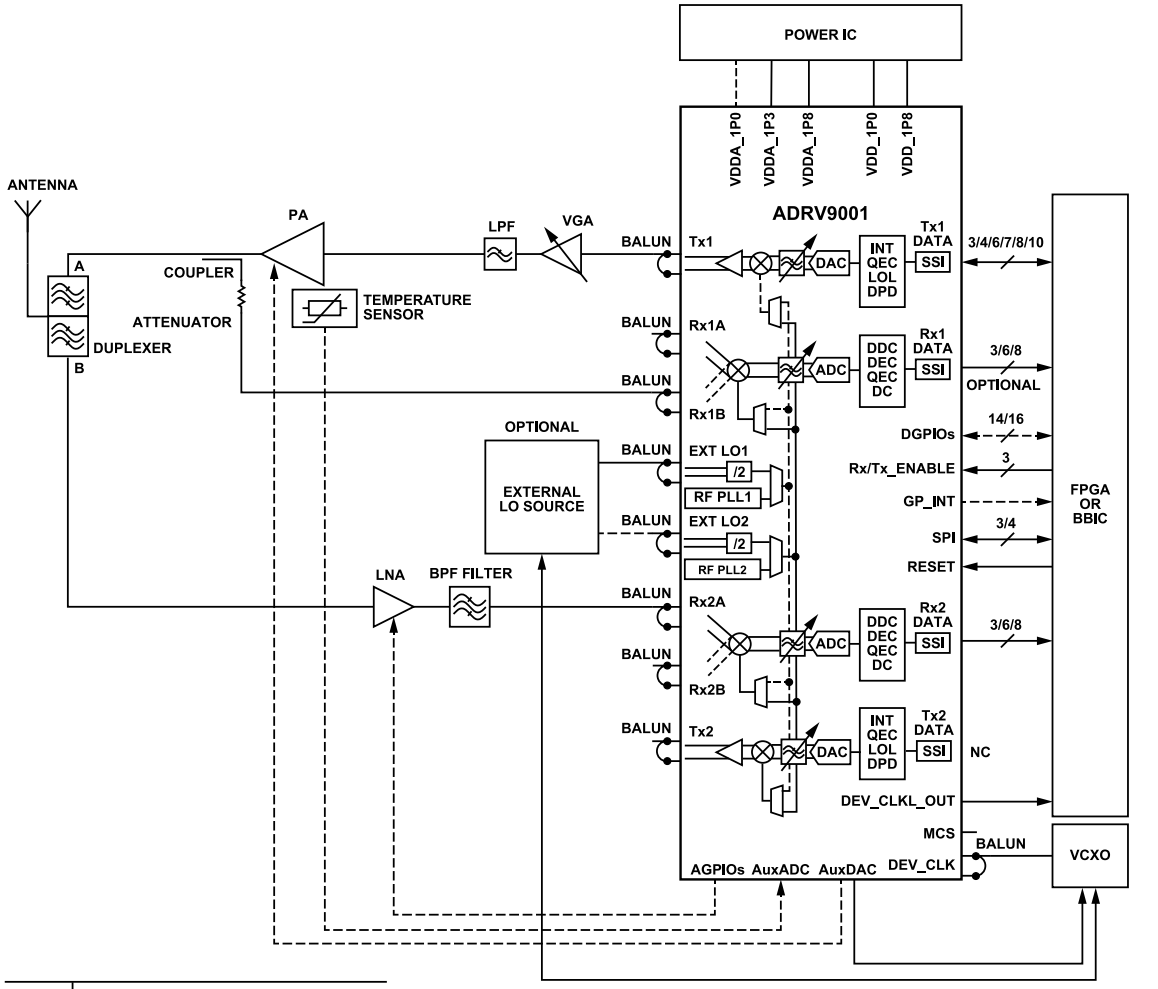
ADRV9001 EXAMPLE USE CASES

Table 4. Constraints and Limitations in a Single-Band 2T2R TDD Type Small-Cell Application (Continued)

Functionality	Constraints and Limitations
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates a square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality can be used in the 2T2R TDD mode. The ADRV9001 can perform the DPD operation or observation receiver data can be sent to the baseband processor through the receiver data port during transmitting operation. The receiver path used during DPD operation to perform transmitter observation is also used by the transmitter tracking calibrations. In case of external DPD, ensure that access to the receiver path during transmitting slots is time-shared between the DPD operation and transmitter calibrations.
Calibrations	During the receiver initialization sequence, ensure there are no signals present at the receiver input (external LNA must be disabled) and appropriate termination is present at the LNA output to avoid reflections of receiver calibration tones. During the transmitter initialization sequence, ensure that the power amplifier is powered down to avoid unwanted emission of transmitter calibration tones at the antenna. The ADRV9001 must access receiver datapath during transmitter time slots to operate the transmitter tracking calibration. If a user uses transmitter observation path with DPD functionality performed by the baseband processor, then access to the receiver datapath during transmitter slots must be time-shared between the DPD operation and transmitter calibrations.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels in the end user system. AGPIOs can be used to control the states of external components or read back digital logic levels from the external components.
DGPIOs	Digital GPIOs can be used to perform real-time monitoring of states of internal ADRV9001 blocks. Digital GPIOs operating as inputs control receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.95 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV9001 device clock, generate a preconfigured ramp up/down signal that can be used to control power amplifier bias, and control any circuitry that requires analog control voltage up to 1.75 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization, initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN 1T1R FDD WITH DPD TYPE APPLICATION



RF I/O	FUNCTIONALITY
Rx1A	NOT USED
Rx1B	DPD + Tx CALIBRATION
Rx2A	RF RECEPTION
Rx2B	NOT USED
Tx1	RF TRANSMISSION
Tx2	NOT USED

Figure 6. ADRV9001 in 1T1R FDD with DPD Type Application

1T1R FDD with DPD Overview

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete RF-to-bits signal chain that can serve as RF front end in FDD type applications that require DPD. Internal DPD block can be used to linearize the external power amplifier and improve the overall system efficiency. For systems that demand superior LO phase noise performance, the ADRV9001 allows users to apply eternal RF LO. The ADRV9001 internal AGC can be used to autonomously monitor and set the appropriate gain level for the receiver signal chain. The ADRV9001 can control external LNA using its analog GPIOs as well as provide power amplifier bias voltage using AuxDAC outputs.

Table 5. Constraints and Limitations in 1T1R FDD with DPD Type Application

Functionality	Constraints and Limitations
LO Generation	In 1T1R FDD+DPD type applications, the ADRV9001 can use its internal LO to generate RF LO1 for uplink and RF LO2 for downlink. For applications with stringent RF LO requirements, the use external LO inputs.
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be

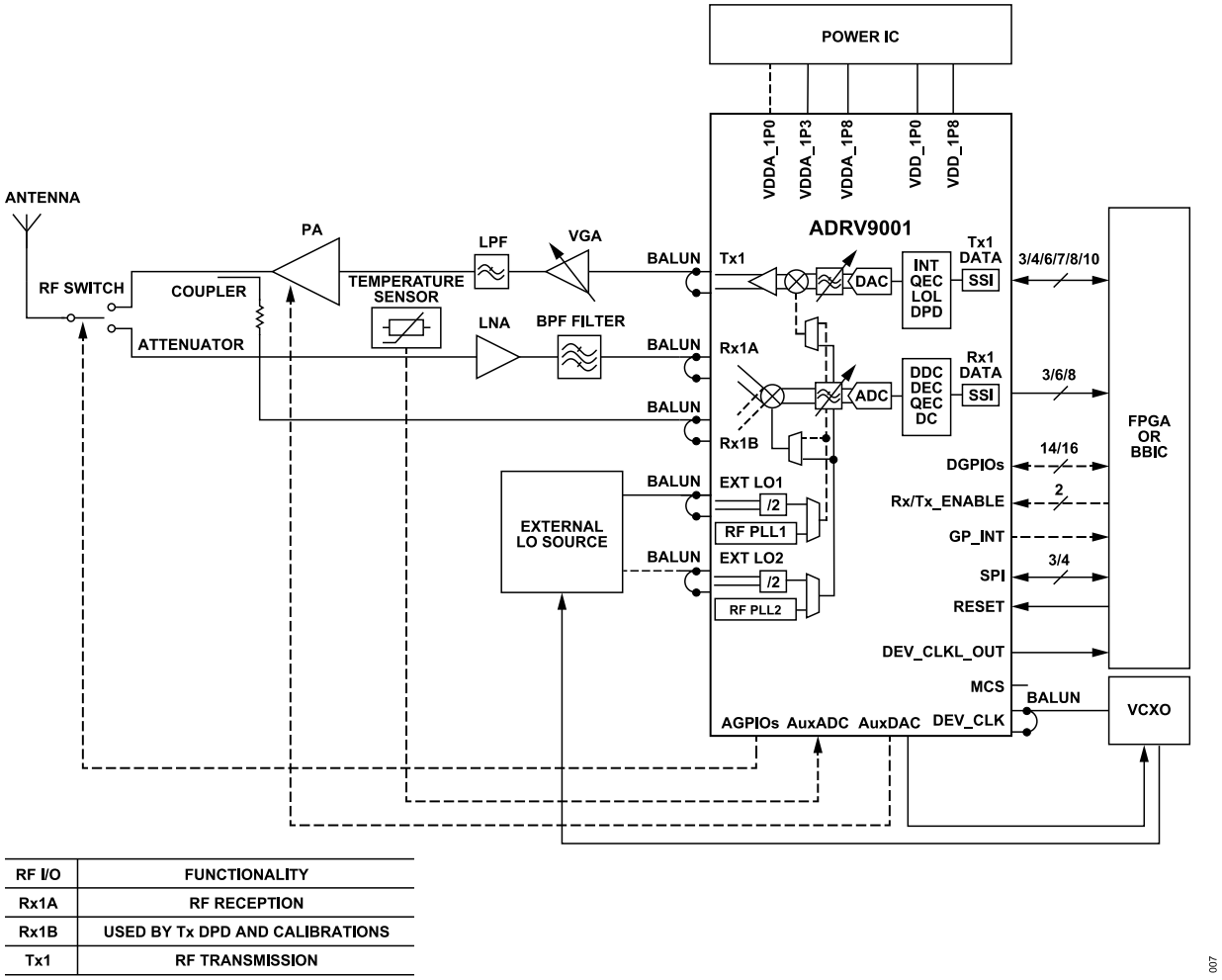
ADRV9001 EXAMPLE USE CASES

Table 5. Constraints and Limitations in 1T1R FDD with DPD Type Application (Continued)

Functionality	Constraints and Limitations
	as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality can be used in the 1T1R FDD mode with the second transmitter being disabled. The DPD operation can be performed by the ADRV9001 or receiver data can be sent to the baseband processor through receiver data port serving as observation receiver. The receiver path used during DPD operation to perform the transmitter observation is also used by the transmitter tracking calibrations. In case of external DPD, ensure that access to the receiver path during transmitter slots is time-shared between the external DPD operation and internal transmitter calibrations.
Calibrations	During the receiver initialization sequence, ensure there are no signals present at the receiver input (external LNA must be disabled), and appropriate termination must be present at the LNA output to avoid reflections of the receiver calibration tones present at the receiver input. During the transmitter initialization sequence, ensure that the power amplifier is powered down to avoid unwanted emission of transmitter calibration tones at the antenna. The ADRV9001 must access the receiver datapath during transmitter time slots to operate the transmitter tracking calibration. If a user uses transmitter observation path with DPD functionality performed by the baseband processor, then access to the receiver datapath during transmitter slots must be time-shared between the DPD operation and transmitter calibrations.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels in the end user system. AGPIOs can be used to control the states of external components (for example, RF Switch, LNA) or read back digital logic levels from the external components.
DGPIOs	Digital GPIOs can be used to perform the real-time monitoring of states of internal ADRV9001 blocks. Digital GPIOs operating as inputs control the receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.9 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV9001 device clock, generate a preconfigured ramp up/down signal that can be used to control power amplifier bias, or control any circuitry that requires analog control voltage up to 1.8 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide a reference clock signal to the digital components in the overall system. This output can be configured to be active after power up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization, initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN A TETRA TYPE PORTABLE RADIO APPLICATION



RF I/O	FUNCTIONALITY
Rx1A	RF RECEPTION
Rx1B	USED BY Tx DPD AND CALIBRATIONS
Tx1	RF TRANSMISSION

Figure 7. ADRV9001 in a TETRA Type Portable Radio Application

TETRA Type Portable Radio Overview

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete RF-to-bits signal chain that can serve as RF front end in TETRA type applications. Internal DPD block can be used to linearize the external power amplifier and improve overall system efficiency. For systems that demand superior LO phase noise performance, the ADRV9001 allows users to apply external RF LO. The ADRV9001 internal AGC can be used to autonomously monitor and set appropriate gain level for the receiver signal chain. For time-critical TDD type applications, the ADRV9001 transceiver can be controlled by toggling control lines. The ADRV9001 can control external receiver/transmitter switch using its analog GPIOs as well as provide power amplifier bias voltage using AuxDAC outputs.

Table 6. Constraints and Limitations in a TETRA Type Portable Radio Application

Functionality	Constraints and Limitations
LO Generation	In a portable radio, TETRA type application, the ADRV9001 can use its internal LO to generate RF LO1 for both uplink and downlink. For applications with stringent RF LO requirements, use external LO inputs.
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates a square wave type signal. A square wave LO produce harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality can be used in the 1T1R TDD mode. The DPD operation can be performed by the ADRV9001 or receiver data can be sent to the baseband processor through receiver data port during transmitter operation. The receiver path used during DPD operation to perform

ADRV9001 EXAMPLE USE CASES

Table 6. Constraints and Limitations in a TETRA Type Portable Radio Application (Continued)

Functionality	Constraints and Limitations
	the transmitter observation is also used by the transmitter tracking calibrations. In case of external DPD, ensure that access to the receiver path during transmitter slots is time-shared between the external DPD operation and transmitter calls.
Calibrations	During the receiver initialization sequence, ensure there are no signals present at the receiver input (external LNA must be disabled), and appropriate termination must be present at the LNA output to avoid reflections of the receiver calibration tones. During the transmitter initialization sequence, ensure that the power amplifier is powered down to avoid unwanted emission of the transmitter calibration tones at the antenna. The ADRV9001 must access the receiver datapath during transmitter time slots to operate the transmitter tracking calibration. If a user uses the transmitter observation path with DPD functionality performed by the baseband processor, then access to the receiver datapath during transmitter slots must be time-shared between the DPD operation and transmitter calibrations.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels in the end user system. AGPIOs can be used to control the states of external components (for example, RF Switch) or read back digital logic levels from the external components.
DGPIOs	Digital GPIOs can be used to perform the real-time monitoring of states of internal ADRV9001 blocks. Digital GPIOs operating as inputs control the receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.95 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV9001 device clock, generate a preconfigured ramp up/down signal that can be used to control power amplifier bias, or control any circuitry that requires analog control voltage up to 1.75 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization, initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN A DMR TYPE PORTABLE RADIO APPLICATION

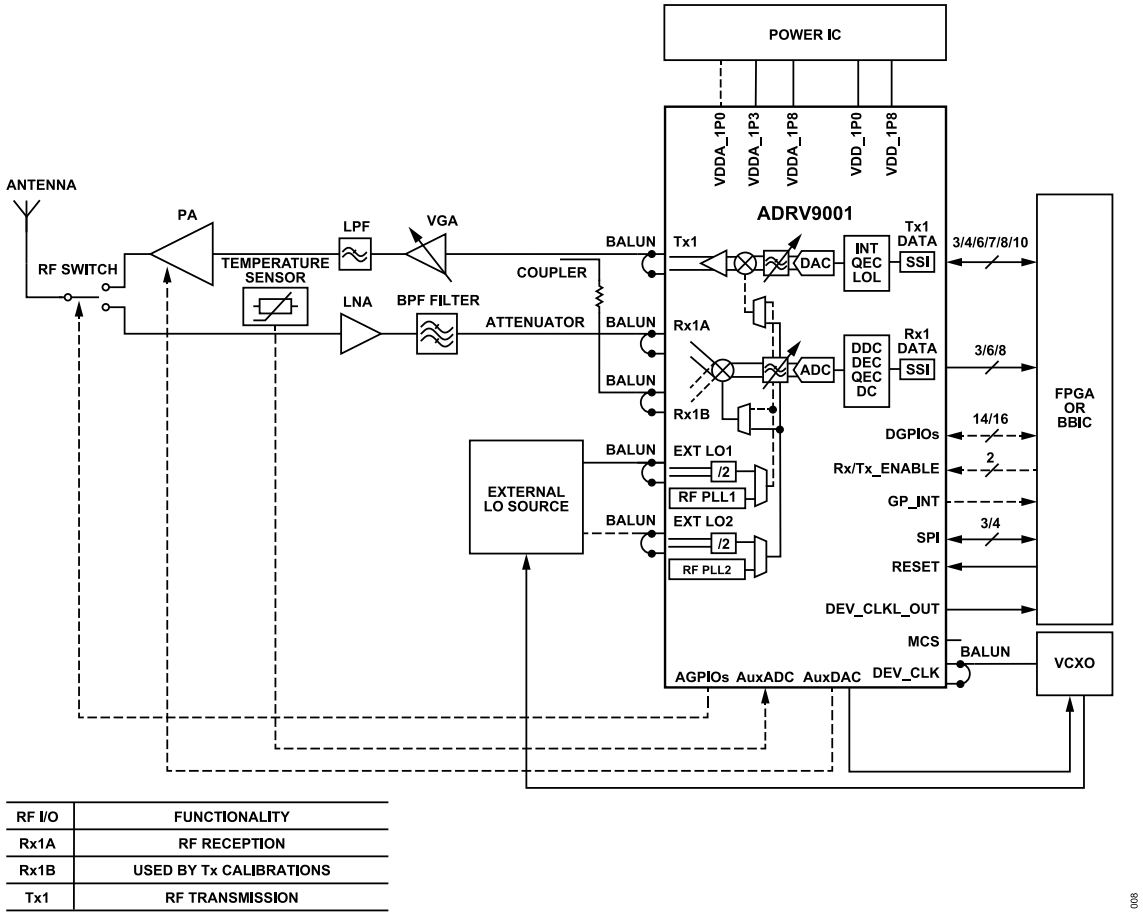


Figure 8. ADRV9001 in a DMR Type Portable Radio Application

DMR Type Portable Radio Overview

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete RF-to-bits signal chain that can serve as RF front end in DMR type applications. For systems that demand superior LO phase noise performance, the ADRV9001 allows users to apply external RF LO. The ADRV9001 internal AGC can be used to autonomously monitor and set the appropriate gain level for the receiver signal chain. For time-critical TDD type applications, the ADRV9001 transceiver can be controlled by toggling control lines. The ADRV9001 can control the external receiver/transmitter switch using its analog GPIOs as well as provide power amplifier bias voltage using AuxDAC outputs.

Table 7. Constraints and Limitations in a DMR Type Portable Radio Application

Functionality	Constraints and Limitations
Receiver Signal Path	Ensure an appropriate level of isolation between Rx1 and Rx2 as well as receiver to transmitter at the system level. In the example, RxB input is used during initialization and tracking calibrations. The LNA connected to the Rx1A must be powered down during transmitter slots for proper operation of the transmitter calibration path (connected to the Rx1B). Ensure that appropriate attenuation is present in the line to prevent receiver being overloaded by the transmitter signal.
LO Generation	In a portable radio, DMR type application, the ADRV9001 can use its internal LO to generate RF LO1 for both uplink and downlink. For applications with stringent RF LO requirements, use external LO inputs.
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates a square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality is not available when the ADRV9001 operates in the 1T1R mode.

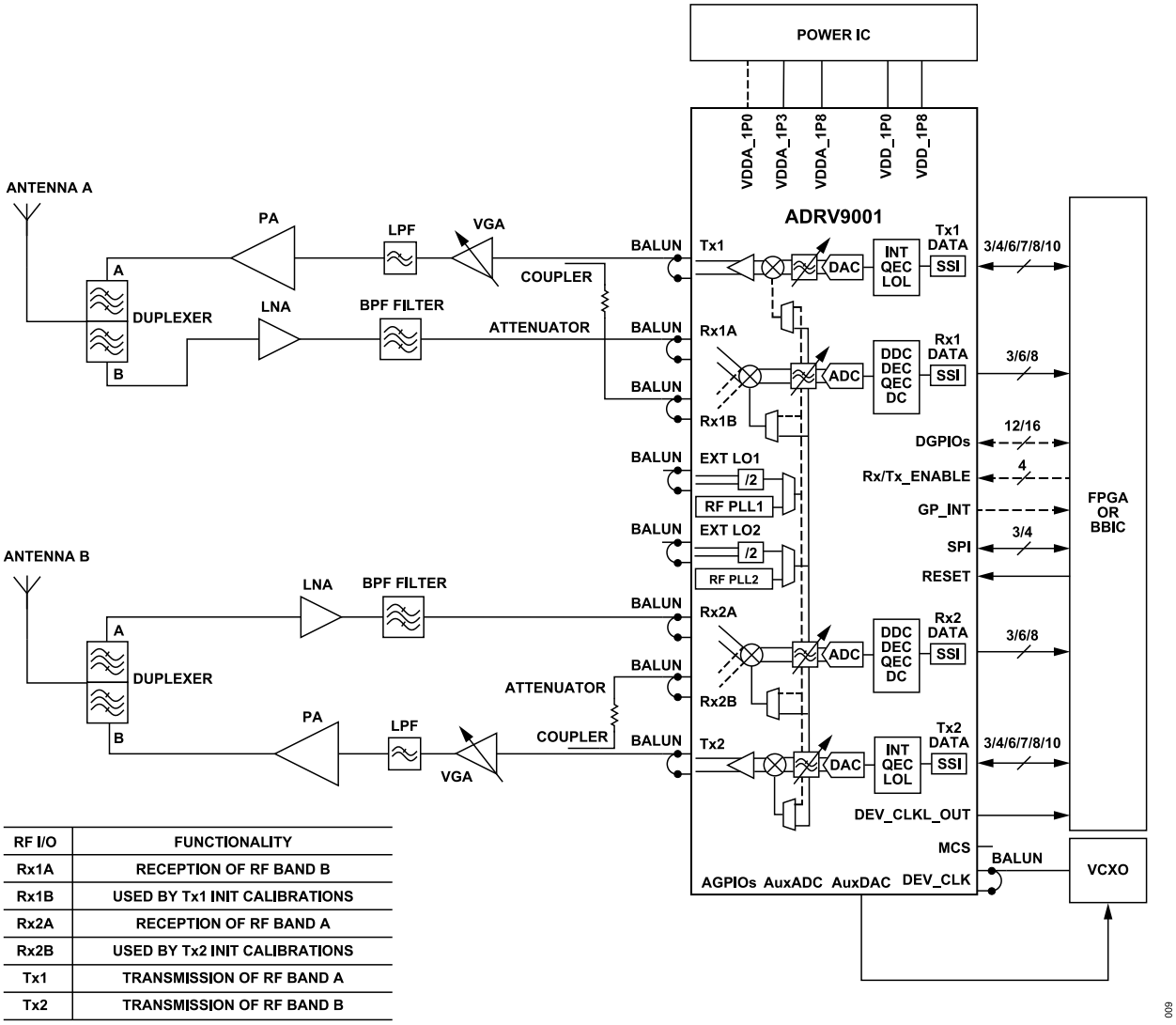
ADRV9001 EXAMPLE USE CASES

Table 7. Constraints and Limitations in a DMR Type Portable Radio Application (Continued)

Functionality	Constraints and Limitations
Calibrations	During the receiver initialization sequence, ensure there are no signals present at the receiver input (external LNA must be disabled), and appropriate termination must be present at the LNA output to avoid reflections of the receiver calibration tones. During the transmitter initialization sequence, ensure the power amplifier is powered down to avoid unwanted emission of the transmitter calibration tones at the antenna. For the transmitter tracking calibrations to operate, the ADRV9001 must access the receiver datapath during transmitter time slots.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels in the end user system. AGPIOs can be used to control the states of external components (for example, RF Switch) or read back digital logic levels from the external components.
DGPIOs	For DMR type applications, the ADRV9001 supports RF Monitor mode of operation. DGPIO pins are used to send wake-up signal to the baseband processor, and allow the baseband processor to move the ADRV9001 into Monitor mode using hardware pins (instead of API command). Digital GPIOs can also be used to perform real-time monitoring of states of internal ADRV9001 blocks. Digital GPIOs operating as inputs control the receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). AuxADC input voltage must not exceed 0.95 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV9001 device clock, generate a preconfigured ramp up/down signal that can be used to control power amplifier bias, or control any circuitry that requires analog control voltage up to 1.75 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization, initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN AN FDD TYPE REPEATER APPLICATION



RF I/O	FUNCTIONALITY
Rx1A	RECEPTION OF RF BAND B
Rx1B	USED BY Tx1 INIT CALIBRATIONS
Rx2A	RECEPTION OF RF BAND A
Rx2B	USED BY Tx2 INIT CALIBRATIONS
Tx1	TRANSMISSION OF RF BAND A
Tx2	TRANSMISSION OF RF BAND B

Figure 9. ADRV9001 in an FDD Type Repeater Application with Baseband Processor Analyzing Traffic Data

ADRV9001 in an FDD Type Repeater Application with Baseband Processor

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete RF-to-bits signal chain that can serve as RF front end in repeater or frequency translator type applications. The ADRV9001 internal AGC can be used to autonomously monitor and set the appropriate gain level for the receiver signal chains. For none-time critical FDD type applications, the ADRV9001 transceiver can be controlled through API commands that use the serial peripheral interface (SPI).

Table 8. Constraints and Limitations in an FDD Type Repeater Application with Baseband Processor Analyzing Traffic Data

Functionality	Constraints and Limitations
Receiver Signal Path	Ensure an appropriate level of isolation between Rx1 and Rx2 as well as receiver to transmitter at the system level. In the example, RxB inputs are used only during initialization calibrations.
Transmitter Signal Path	Ensure an appropriate level of isolation between Tx1 and Tx2 as well as receiver to transmitter at the system level.
LO Generation	In the FDD type Repeater application, the ADRV9001 can use its internal LO to generate RF LO1 for uplink (example, Tx1, and Rx2) and RF LO2 for downlink (example, Tx2, and Rx1). It is also possible to use external LO inputs in this mode of operation.
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates a square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be

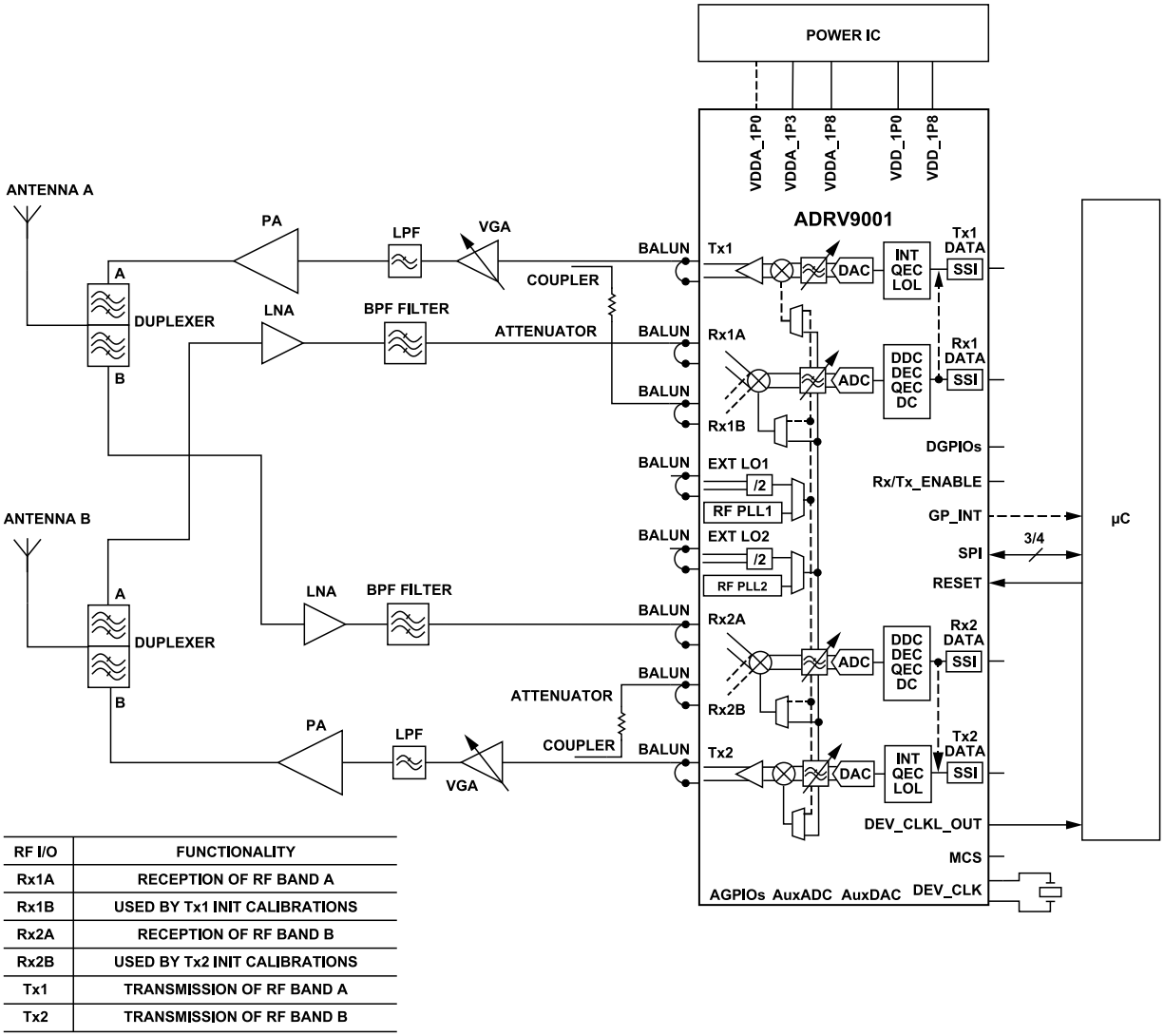
ADRV9001 EXAMPLE USE CASES

Table 8. Constraints and Limitations in an FDD Type Repeater Application with Baseband Processor Analyzing Traffic Data (Continued)

Functionality	Constraints and Limitations
	as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality is not available when the ADRV9001 operates in the 2T2R FDD mode.
Calibrations	During the receiver initialization sequence, ensure that there are no signals present at the receiver input (external LNA must be disabled), and appropriate termination must be present at the LNA output to avoid reflections of the receiver calibration tones. During the transmitter initialization sequence, ensure the power amplifier is powered down to avoid unwanted emission of the transmitter calibration tones at the antenna. No transmitter tracking calibrations are available when the ADRV9001 operates in the 2T2R FDD mode.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels of in the end user system. AGPIOs can be used to control the states of external components or read back digital logic levels from the external components.
DGPIOs	Digital GPIOs can be used to perform real-time monitoring of states of internal ADRV9001 blocks. Digital GPIOs operating as inputs control the receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.9 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV9001 device clock, or control any circuitry that requires analog control voltage up to 1.8 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization, initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN AN FDD TYPE REPEATER APPLICATION USING INTERNAL LOOPBACKS



RF I/O	FUNCTIONALITY
Rx1A	RECEPTION OF RF BAND A
Rx1B	USED BY Tx1 INIT CALIBRATIONS
Rx2A	RECEPTION OF RF BAND B
Rx2B	USED BY Tx2 INIT CALIBRATIONS
Tx1	TRANSMISSION OF RF BAND A
Tx2	TRANSMISSION OF RF BAND B

Figure 10. ADRV9001 in an FDD Type Repeater Application Without Baseband Processor Analyzing Traffic Data

FDD Type Repeater Using Internal Loopbacks

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete RF-to-RF signal chain that can serve as a repeater or a frequency translator. The ADRV9001 internal AGC can be used to autonomously monitor and set the appropriate gain level for the receiver signal chains. For none-time critical FDD type applications, the ADRV9001 transceiver can be controlled through API commands that use the serial-peripheral interface (SPI). The support of external crystal enables a very compact solution, where the ADRV9001 provides clock for a microprocessor that programs and monitors the ADRV9001 operation.

Table 9. Constraints and Limitations in an FDD Type Repeater Application Without Baseband Processor Analyzing Traffic Data

Functionality	Constraints and Limitations
Receiver Signal Path	Ensure an appropriate level of isolation between Rx1 and Rx2 as well as receiver to transmitter at the system level. In the example, RxB inputs are used only during initialization calibrations.
Transmitter Signal Path	Ensure an appropriate level of isolation between Tx1 and Tx2 as well as receiver to transmitter at the system level.
LO Generation	In the FDD type repeater application, the ADRV9001 can use its internal LO to generate RF LO1 for uplink (example, Tx1, and Rx2) and RF LO2 for downlink (example, Tx2, and Rx1). It is also possible to use external LO inputs in this mode of operation.

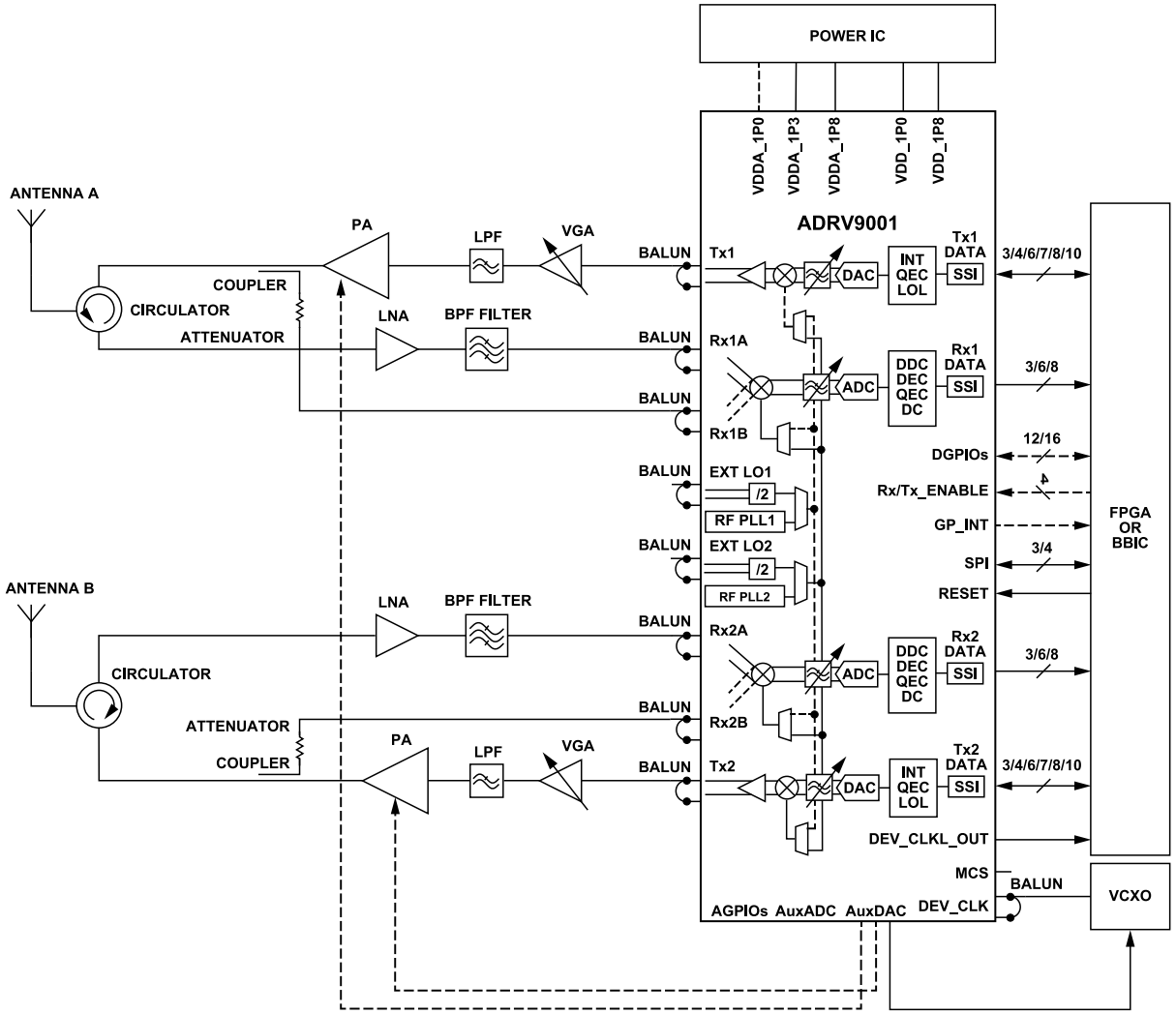
ADRV9001 EXAMPLE USE CASES

Table 9. Constraints and Limitations in an FDD Type Repeater Application Without Baseband Processor Analyzing Traffic Data (Continued)

Functionality	Constraints and Limitations
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality is not available when the ADRV9001 operates in the 2T2R FDD mode.
Calibrations	During the receiver initialization sequence, ensure that there are no signals at the receiver input (external LNA must be disabled), and appropriate termination must be present at the LNA output to avoid reflections of the receiver calibration tones. During the transmitter initialization sequence, ensure that the power amplifier is powered down to avoid unwanted emission of the transmitter calibration tones at the antenna. No transmitter tracking calibrations are available when the ADRV9001 operates in the 2T2R FDD mode.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels of in the end user system. AGPIOs can be used to control states of external components or read back digital logic levels from external components.
DGPIOs	Unused in this application example.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.9 V.
AuxDAC	AuxDAC can be used to control any circuitry that requires analog control voltage up to 1.8 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization, initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN A TDD TYPE REPEATER APPLICATION



RF I/O	FUNCTIONALITY
Rx1A	RECEPTION FROM ANTENNA A
Rx1B	USED BY Tx1 DPD AND CALIBRATIONS
Rx2A	RECEPTION FROM ANTENNA B
Rx2B	USED BY Tx2 DPD AND CALIBRATIONS
Tx1	TRANSMISSION ON ANTENNA A
Tx2	TRANSMISSION ON ANTENNA B

Figure 11. ADRV9001 in a TDD Type Repeater Application with Baseband Processor Analyzing Traffic Data

TDD Type Repeater Overview

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete RF-to-bits signal chain that can serve as RF front end in TDD type repeater or frequency translator applications. In TDD type applications, internal DPD block can be used to linearize external power amplifier and improve overall system efficiency. The ADRV9001 internal AGC can be used to autonomously monitor and set appropriate gain level for the receiver signal chains. Field programmable gate array (FPGA) or baseband processor is responsible for appropriate time alignment of the receiver and transmitter time slots. The ADRV9001 receiver and transmitter signal chains can be controlled by toggling control lines. The ADRV9001 can provide power amplifier bias voltage using AuxDAC outputs.

ADRV9001 EXAMPLE USE CASES

Table 10. Constraints and Limitations in a TDD Type Repeater Application with Baseband Processor Analyzing Traffic Data

Functionality	Constraints and Limitations
LO Generation	In the TDD type repeater application, the ADRV9001 can use its internal LO to generate RF LO1 for both uplink and downlink. It is also possible to use external LO inputs in this mode of operation.
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality can be used in the 2T2R TDD mode. The DPD operation can be performed by the ADRV9001 or observation receiver (ORx) data can be sent to the baseband processor through the receiver data port during the transmitter operation. The receiver path used during DPD operation to perform the transmitter observation is also used by the transmitter tracking calibrations. In case of external DPD, ensure that access to the receiver path during transmitter slots is time-shared between the DPD operation and transmitter calibrations.
Calibrations	During the receiver initialization sequence, ensure that there are no signals present at the receiver input (external LNA must be disabled), and appropriate termination must be present at the LNA output to avoid reflections of the receiver calibration tones. During the transmitter initialization sequence, ensure that the power amplifier is powered down to avoid unwanted emission of the transmitter calibration tones at the antenna. The ADRV9001 must access the receiver datapath during transmitter time slots to operate the transmitter tracking calibration. If a user uses the transmitter observation path with DPD functionality performed by the baseband processor, then access to the receiver datapath during transmitter slots must be time-shared between the DPD operation and transmitter calibrations.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels of in the end user system. AGPIOs can be used to control the states of external components or read back digital logic levels from the external components.
DGPIOs	Digital GPIOs can be used to perform real-time monitoring of the states of internal ADRV9001 blocks. Digital GPIOs operating as inputs control the receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.9 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV9001 device clock, generate a preconfigured ramp up/down signal that can be used to control power amplifier bias, or control any circuitry that requires analog control voltage up to 1.8 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power up and before the ADRV9001 configuration stage.
Multichip Sync	If there is no need for multichip synchronization, initialize the ADRV9001 using API functions only.

ADRV9001 EXAMPLE USE CASES

ADRV9001 IN A RADAR TYPE APPLICATION

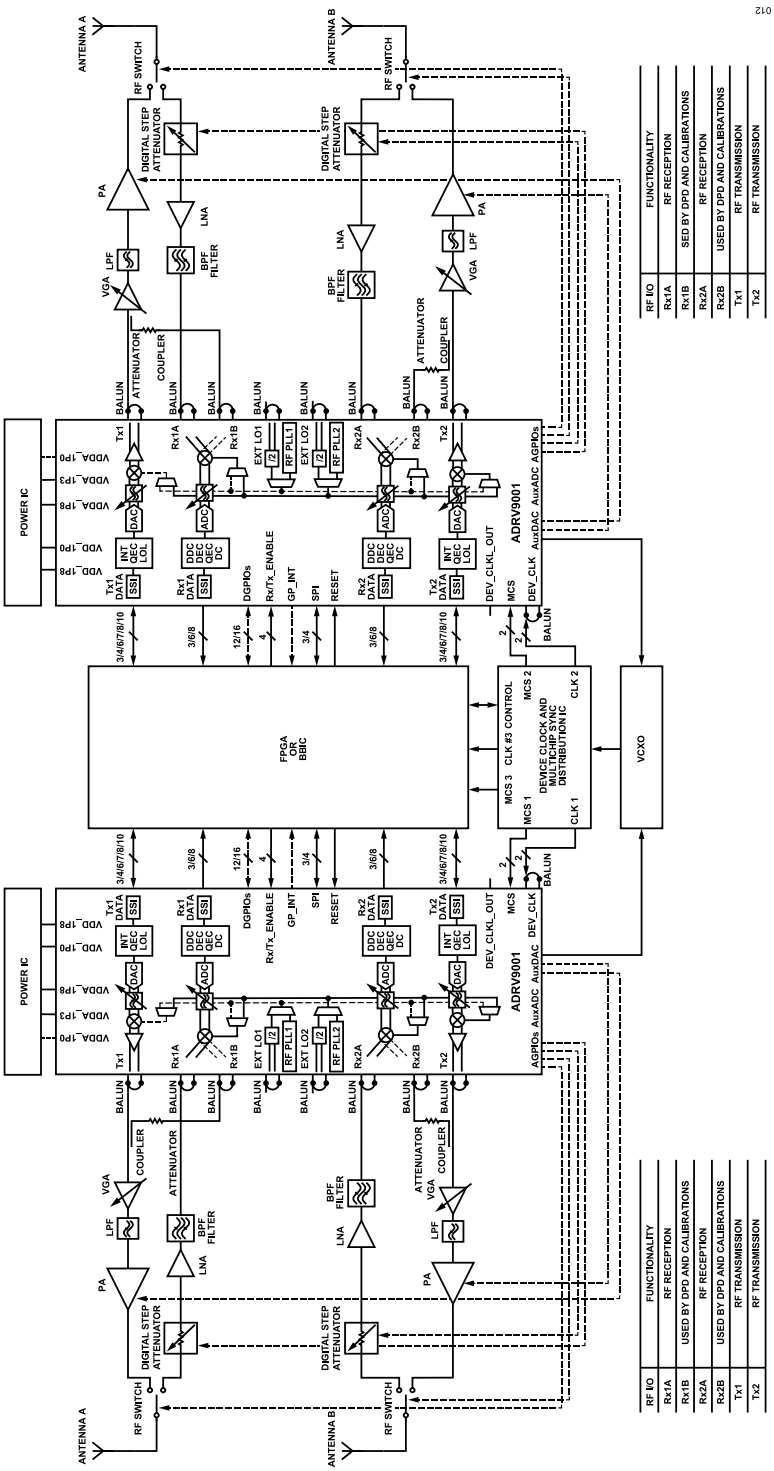


Figure 12. ADRV9001 in a Radar Type Application

Radar Type Application Overview

With a minimum number of external components, the ADRV9001 transceiver can be used to build a complete RF-to-bits signal chain that can serve as RF front end building block in Radar type applications. The ADRV9001 internal AGC can be used to autonomously monitor and set the appropriate gain level for the receiver signal chains. Internal AGC can use analog general purpose input/output (GPIO) interface to control

ADRV9001 EXAMPLE USE CASES

external DSA in the receiver signal chains. For time-critical TDD type applications, the ADRV9001 transceiver can be controlled by toggling control lines. The ADRV9001 can control external receiver/transmitter switch using its analog GPIOs as well as provide power amplifier bias voltage using AuxDAC outputs. Multichip Sync signal together with DEV_CLK can be used to synchronize multiple ADRV9001 in the end system.

Table 11. Constraints and Limitations in a Radar Type Application

Functionality	Constraints and Limitations
Receiver Signal Path	Ensure an appropriate level of isolation between Rx1 and Rx2 as well as receiver to transmitter at the system level. In the example, RxB input is used during transmitter observation. The LNA connected to the Rx1A must be powered down during transmitter slots to ensure proper operation of the transmitter observation path (connected to the Rx1B). Ensure that an appropriate attenuation is present in line to prevent the receiver input being overloaded by the transmitter signal.
LO Generation	In a Radar type application, the ADRV9001 can use its internal LO to generate RF LO1 for both uplink and downlink. For applications with stringent RF LO requirements, use external LO inputs.
RF Front End	For LO generation, the ADRV9001 uses internal VCO that generates square wave type signal. A square wave LO produces harmonics. For example, depending on RF matching used on the RF ports, the second LO harmonic can be as high as −50 dBc, and the third harmonic can be as high as −9 dBc. Therefore, the RF filtering on the receiver and transmitter path must ensure that signals at the LO harmonic frequencies (up to ninth in some cases) do not affect overall system performance.
DPD	The DPD functionality can be used in the 2T2R TDD mode. The DPD operation can be performed by the ADRV9001 or ORx data can be sent to the baseband processor through the receiver data port during transmitter operation. The receiver path used during DPD operation to perform the transmitter observation is also used by the transmitter tracking calibrations. In case of external DPD, ensure that access to the receiver path during transmitter slots is time-shared between the DPD operation and transmitter calibrations.
Calibrations	During the receiver initialization sequence, ensure that there are no signals present at the receiver input (external LNA must be disabled), and appropriate termination must be present at the LNA output to avoid reflections of the receiver calibration tones. During the transmitter initialization sequence, ensure that the power amplifier is powered down to avoid unwanted emission of the transmitter calibration tones at the antenna.
	The ADRV9001 must access the receiver datapath during transmitter time slots to operate the transmitter tracking calibration. If a user uses DPD in its system, then access to the receiver datapath during transmitter slots must be time-shared between the DPD operation and Tx calibrations.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels of in the end user system. AGPIOs can be used to control the states of external components (for example, RF Switch) or read back digital logic levels from the external components.
DGPIOs	Digital GPIOs can be used to perform real-time monitoring of the states of internal ADRV9001 blocks. Digital GPIOs operating as inputs control the receiver gain, transmitter attenuation, AGC operation, and other elements of the ADRV9001 transceiver. Depending on the ADRV9001 operation, up to 4 GPIOs may be used by the data port interface.
AuxADC	AuxADC can be used to monitor analog voltage (for example, a temperature sensor). The maximum AuxADC input voltage must not exceed 0.95 V.
AuxDAC	AuxDAC can be used to: control VCXO responsible for generating device clock, generate a preconfigured ramp up/down signal that can be used to control power amplifier bias, or control any circuitry that requires analog control voltage up to 1.75 V.
DEV_CLK_OUT	The ADRV9001 provides a divided down version of the DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power up and before the ADRV9001 configuration stage.
Multichip Sync	The ADRV9001 allows to synchronize multiple transceivers used in a single system. The ADRV9001 provides the capability to accept an external reference clock and synchronize operation with other devices using simple control logic. Logical pulses applied at MCS input align each device's data clock with a common reference. Preserve the relationship of MCS pulse to the DEV_CLK edge at the ADRV9001 pins. For correct operation, it is critical to match the length of PCB traces that carry the DEV_CLK and MCS signals to each ADRV9001 device.

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

This section provides information about the device driver application programming interface (API) software developed by Analog Devices for the ADRV9001 transceivers. It also outlines the overall architecture, folder structure, and methods to use the API software on the user platform.

Note:

- This document does not explain the API library functions. Detailed information on the API functions is in the Doxygen document included with the software development kit (SDK) (**ADRV9001_API.chm**) located at **/pkg/production/**.
- The **ADRV9001_API.chm** is in compressed HTML format. For security reasons, open the **.chm** files only from a local drive. If there is an attempt to open from a network drive, the file may look empty.
- The ADRV9001 is a baseline device for the product family; therefore, all API and evaluation systems use this product number to describe the product.
  - Figure 13 shows a simple flowchart to the users new to this product family and guides them through the evaluation, testing, and development of the Analog Devices platform and their system.

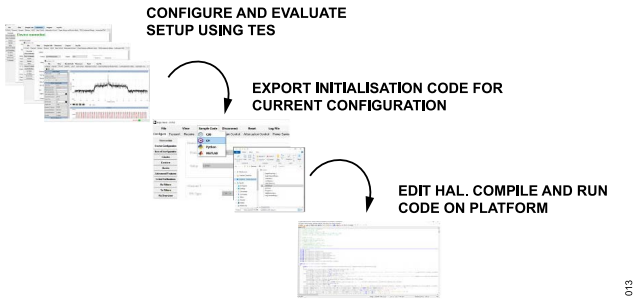


Figure 13. Development Flowchart

SOFTWARE ARCHITECTURE

Figure 14 shows the software architecture for the ADRV9001 evaluation platform from Analog Devices. This chapter takes a high-level overview of the changes to make to this architecture once the baseline setup for the evaluation board is explained.

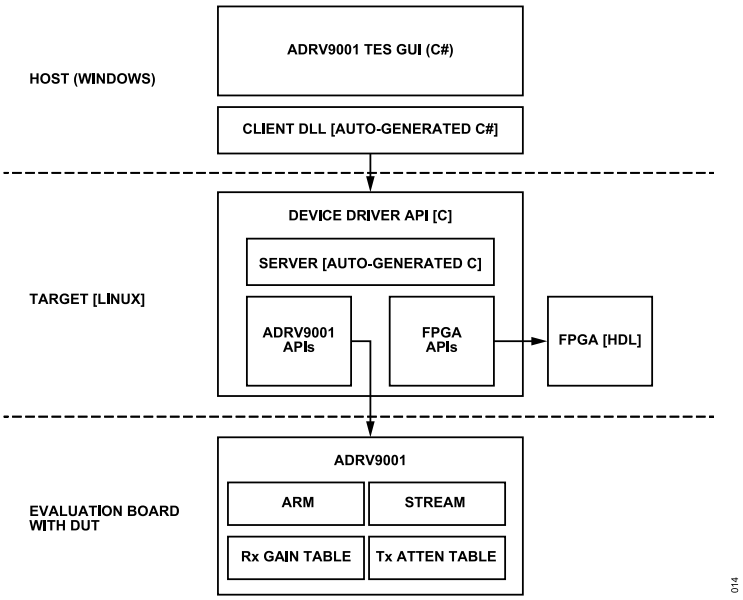


Figure 14. ADRV9001 API Software Architecture (ADI Evaluation Platform)

Layering the system, as shown in Figure 14, allows for great flexibility in the system platform. Any user who starts the development using the transceiver evaluation software (TES) package can swap elements in the Target layer of this system architecture with only minor edits to the code base.

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

FOLDER STRUCTURE

Analog Devices provides the source files in the folder structure shown in Figure 15. Each subfolder is explained in the following sections. Analog Devices understands that the developer may want to use a different folder structure. Although Analog Devices provides the ADRV9001 API source code releases in the folder structure shown in Figure 15, the developer can organize the ADRV9001 API into a custom folder organization, if required. This operation, however, does not permit the developer to modify the content of the ADRV9001 API source code, except for the hardware abstraction layer (HAL) placeholder files, which are detailed later in this chapter and in Figure 15.

Figure 15. API Folder Structure

/c_src/common:

A common code is shared between all the devices. It contains error handling facilities, logging facilities, and HAL access facilities. Note: Users cannot edit this folder when implementing their own HAL. The files under **common** are designed to work together with any HAL provided to the code base and do not need to be modified. Users cannot add any new files here as well.

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

/c_src/devices:

The device folder includes the main API code for the ADRV9001 transceiver as well as the auxiliary devices the APIs used for the demo of the ADRV9001. The **/ADRV9001** folder contains the high-level function prototypes, data types, macros, and source code used to build the final software system. Do not modify the files contained in the **/ADRV9001** or other devices in this section. There is no software support when these files are modified. Analog Devices maintains this code. The only exception is that the user can modify #define macros in **adi_ADRV9001_user.h**, such as modifying polling timeouts and interval settings for various functions.

/c_src/platforms:

The **/platforms** folder allows a developer to insert custom platform hardware driver code for system integration with the ADRV9001 API. The HAL interface is described later in this document. The **adi_platform.c/.h** files contain function pointers and the required prototypes necessary for the ADRV9001 API to work correctly. Do not modify the function prototypes in **adi_platform.c**. The developer is responsible to implement the code for each function to ensure the correct hardware drivers are called for the platform hardware. In the example code provided by ADI in the **customer** folder, there are placeholder functions left empty to fill their platform-specific code.

/c_src/third_party:

This section contains third-party APIs to help the field programmable gate array (FPGA) control the system. For example, this includes a JSON parser and the FMC field replaceable unit (FRU) info manipulator.

CUSTOMIZING THE SYSTEM ARCHITECTURE AND FILE STRUCTURE

The system architecture and file set are uniquely suited for the evaluation platform, and change when users progress to developing their own unique solution. This paragraph is designed to give users at this stage of development a good starting point to build from, as well as build up to more in-depth details, which are provided in the next chapter (Software Integration). Figure 16 shows the steps to begin the development on a custom platform. The goal here is to migrate from the evaluation platform to a custom, bespoke platform. To do so, take the evaluation platform APIs and HDL out of the TARGET compilation and replace them with the APIs specific to the bespoke platform. To accommodate this migration, use the TES in a slightly different way. Use the TES package to generate C code rather than connecting directly to the TARGET platform through a client-server setup. Make minor edits to the generated files and then deploy the files to the target platform.

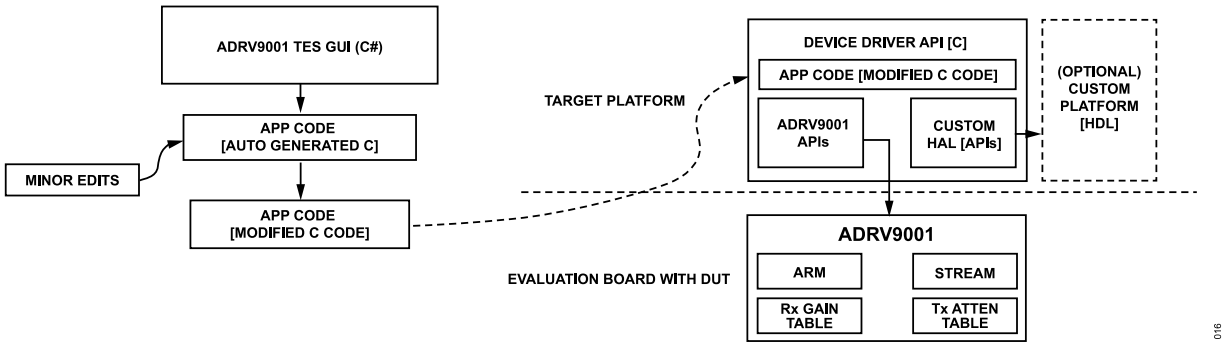


Figure 16. Modified ADRV9001 API Software Architecture

These **Minor Edits** listed in Figure 16 consist mostly of removing function calls targeting unused hardware components. For example, by default, the generated code seeks to initialize the Evaluation Platform by making calls to **adi_fpga9001_** functions. Given the migration away from this platform, it is necessary to remove those function calls. In a similar fashion, insert any initialization functions needed for the user platform into the generated C code, where they are necessary.

Most **Minor Edits** occur in the **main.c** and **initialize.c** files of the generated code base. For example, near the beginning of the **main(...)** function, the **linux_uio_initialize(...)** function is called. This function is defined in **linux_uio_init.c**, found under **platforms\linux_uio**. Inspecting this function, the following code snippet is found early in the definition:

```
if (NULL != fpga9001)
{
  fpga9001->common.devHalInfo = linux_uio_fpga9001_open();
  if (NULL != adrv9001)
  {
```

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

```
    ((adi_adrv9001_hal_linux_uio_Cfg_t *)adrv9001->common.devHalInfo)->fpga9001 = fpga9001;
  }
  if (NULL == fpga9001->common.devHalInfo)
  {
    return -1;
  }
}
```

Simply put, this code snippet uses the **linux_uio** FPGA code base to define an FPGA structure if the FPGA pointer argument is not NULL. The application uses this structure to interact with the DMA, the SSI ports, and other elements of functionality. Figure 17 highlights the areas of the file structure to focus on during the platform development phase. The edits required for custom platform development involve replacing functions such as **linux_uio_initialize(…)** with custom code that performs the same tasks relative to the platform device in these highlighted folders.

The **adi_platform.c** file is where the HAL is chosen. For the default HAL provided by Analog Devices, it is also implemented in the **adi_platform.c file**. However, make no edits to the default HAL stored under **linux_uio**. Users have placeholder files in the **customer** folder. Here, all the necessary functions are declared and left empty for the users to fill their platform-specific code.

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

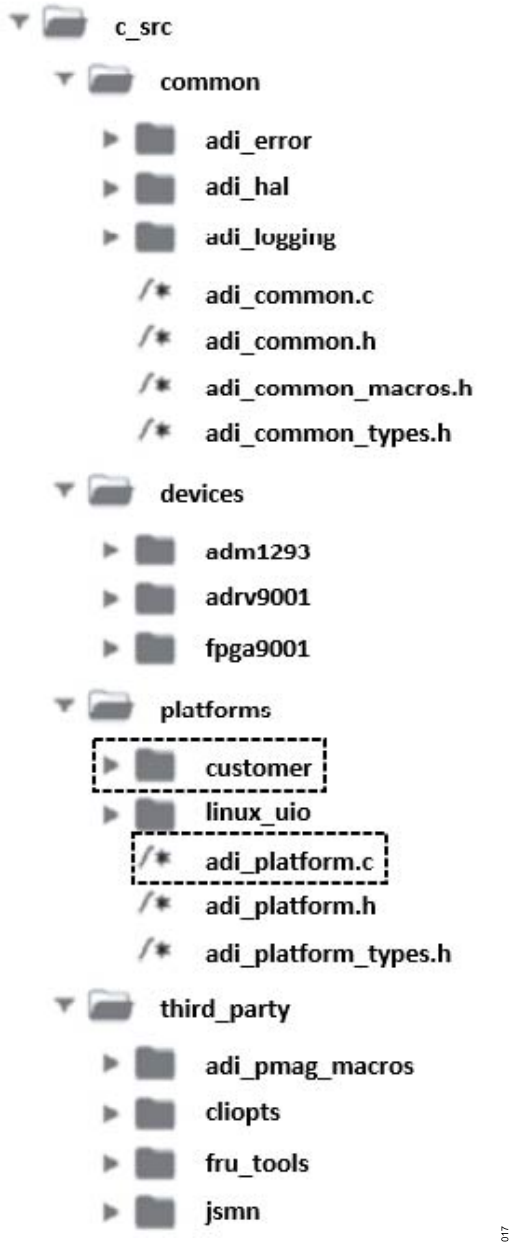


Figure 17. API Folder Structure with Customer Interaction Points Highlighted

More details are provided in the **customer** folder in the Software Integration chapter, which goes into more specifics on the HAL. At this point of development, read the files under the highlighted directories in Figure 17 (primarily the **customer** folder) before editing them.

SOFTWARE INTEGRATION

The ADRV9001 API package was developed using the ZC706 or ZCU102 Evaluation platform. This section describes how to use the ADRV9001 API in a custom hardware/software environment. This is readily accomplished because the API was developed abiding by C99 constructs while maintaining Linux system call transparency. The TES package can be used to produce C99 code to replicate a user's application while maintaining agnostic processor and operating system integration with the ADRV9001 API code.

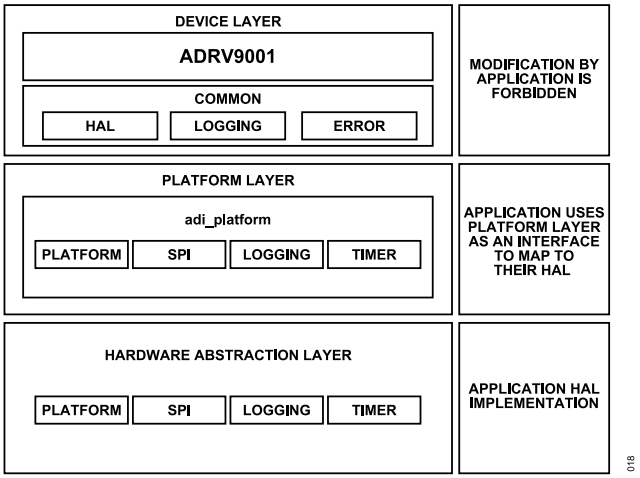


Figure 18. Evaluation System Software Stack

HARDWARE ABSTRACTION LAYER

The users who develop code to target custom hardware platforms use different drivers for the peripherals, such as the SPI and GPIO, compared to the drivers chosen for the ADI evaluation platform. The HAL interface is a set of function pointers that the ADRV9001 API uses to access the target platform hardware. Each device driver (ADM1293, ADRV9001, and FPGA9001) has its own HAL, defined in the respective **adi_<device>_hal.h** files. This allows to select only the required components. The user is responsible to implement the interface defined in each HAL to use the corresponding device driver with their specific platform. The **adi_platform.c** file maps function pointers to the platform-specific HAL implementations. The implementation of this interface is platform dependent and must be implemented by the user. The function pointers associated with the user HAL layer must be set in **adi_platform.c**.

Following is a code snippet from the beginning of **adi_platform.c**:

```
#include "adi_platform.h"
#ifndef CUSTOMER_PLATFORM
#include "adi_linux_uio_logging.h"
#include "adi_linux_uio_timer.h"
#include "adi_adm1293_hal_linux_uio.h"
#include "adi_adrv9001_hal_linux_uio.h"
#include "adi_fpga9001_hal_linux_uio.h"
/* Logging interface */
int32_t(*adi_hal_LogWrite)(void *devHalCfg, uint32_t logLevel, const char *comment, va_list args) =
linux_uio_LogWrite;
/* Timer interface */
int32_t(*adi_hal_Wait_us)(void *devHalCfg, uint32_t time_us) = linux_uio_TimerWait_us;
…
```

The file continues from this point to set the functions, which handle the I^2C reading and writing, the SPI reading and writing, and assigning the RESET pin, etc. Taking note of the second line of the file, define only a **CUSTOMER_PLATFORM** variable for the preprocessor to use own platform code. Doing so changes the operation of the **adi_platform.c** file as follows:

```
#else
#include "adi_common_hal_customer.h"
#include "adi_adm1293_hal_customer.h"
#include "adi_adrv9001_hal_customer.h"
```

SOFTWARE INTEGRATION

```
#include "adi_fpga9001_hal_customer.h"
/* Logging interface */
int32_t(*adi_hal_LogWrite)(void *devHalCfg, uint32_t logLevel, const char *comment, va_list args) =
customer_LogWrite;
/* Timer interface */
int32_t(*adi_hal_Wait_us)(void *devHalCfg, uint32_t time_us) = customer_TimerWait_us;
…
```

Once done, the **adi_platform.c** code automatically switches to use of the placeholder customer code under the **customer** folder.

Following is a code snippet from **adi_adrv9001_hal_customer.c**, located under **customer\adrv9001**. It shows each function required by the HAL to support a customer-specific platform. Read the example HAL implementation provided under the **linux_uio** folder to understand the purpose of each function, as well as the acceptable return values. Function names may be modified, if required, provided the pointer assignments in **adi_platform.c** are updated accordingly. However, the function parameters may not be modified as this causes compilation errors.

```
#include "adi_adrv9001_hal_customer.h"
int32_t customer_adi_adrv9001_hal_open(void *devHalCfg)
{
  /* Customer code goes here */
  return 0;
}
int32_t customer_adi_adrv9001_hal_close(void *devHalCfg)
{
  /* Customer code goes here */
  return 0;
}
int32_t customer_adi_adrv9001_hal_spi_write(void *devHalCfg, const uint8_t txData[], uint32_t numTx▶
Bytes)
{
  /* Customer code goes here */
  return 0;
}
int32_t customer_adi_adrv9001_hal_spi_read(void *devHalCfg, const uint8_t txData[], uint8_t rxData[],
uint32_t numTxRxBytes)
{
  /* Customer code goes here */
  return 0;
}
int32_t customer_adi_adrv9001_hal_resetbPin_set(void *devHalCfg, uint8_t pinLevel)
{
  /* Customer code goes here */
return 0;
}
```

When going through platform development, be aware of all changes made to the HAL implementation with each release of the SDK. Being familiar with **README.md** and **CHANGELOG.md** under the **production** folder is the best way to keep a system up to date.

/c_src/platforms/customer/

Figure 19 shows an expansion of the **customer/** folder. This folder contains all the necessary files to build a custom platform with support for the ADRV9001 Eval Board hardware.

SOFTWARE INTEGRATION

Figure 19. Customer Folder Expanded

Most of these files can safely be left at their defaults. Most function calls defined in this section of the code base are optional. For example, the **adm1293** folder houses the code to interact with the ADM1293 device (used for power monitoring) through I^2C. However, this device is only used by TES to provide power measurements. It has no essential function in the operation of the Eval Board. Similarly, **common** houses the code for a Timer(…) and a Log(…) function, both of which are more useful than essential. The **fpga9001** folder houses the function designed to access and manage memory on the user's platform, as well as handle the synchronous serial interface (SSI) type configuration, direct memory access (DMA), and random access memory (RAM) readings. These functions are necessary if a user's custom application requires a platform connected to the SSI ports with access to RAM, DMA, and register memory. However, not every application needs such a complex platform. Simpler applications, which do not need access to these elements of functionality, can safely ignore this folder.

The only essential folder is the **ADRV9001** folder, which houses the code necessary to instantiate and deconstruct the customer HAL, SPI functionality, and RESET pin control. These are the minimum functions required to interact with the ADRV9001 device, provided a user can power it. The Raspberry Pi HAL section provides an example customer HAL file to operate the ADRV9001 device using a Raspberry Pi.

SPI Access

As stated in the previous section, programming the API to use custom SPI code in execution is as easy as calling #define **CUSTOMER_PLATFORM** in **adi_platform.c**; however, some thought is given to the SPI code. A common issue encountered at this stage in development is the lack of response from the device over the SPI lines, sometimes accompanied by an **ARM Boot Up Timed Out** error message. These issues can often be very confusing as they happen directly after the **Hw_Open()** runs successfully.

The issue arises due to a discrepancy between the “standard” bit field control of SPI settings (clock polarity (CPOL), clock phase (CPHA), etc.) and the approach Analog Devices takes with SPI control of the devices. Where most devices use normal Binary encodings for their SPI control (00, 01, 10, 11), Analog Devices uses Gray Code for SPI control (00, 01, 11, 10).

To register the access, as mentioned previously, Analog Devices does not provide the register map for this device. All register access is handled through the APIs. This means that as Register maps change and update, the customer code remains valid simply by updating the SDK.

Confirm the SPI operation for custom platforms with a few simple tests. The Serial-Peripheral Interface (SPI) section, shows that the ADRV9001 transceivers use 3-Byte interactions for SPI communication: **{Command}{Address}{Data}**. By knowing this, monitor the SPI interactions using a scope and record the interactions between the default platform (ZC706 or ZCU102) and the ADRV9001 device. Do the same for any custom platform. Provided the device setup is identical between the default and custom platform, the SPI interactions should also be identical.

In the absence of any external equipment, there are APIs that can verify the SPI operation. One such API is the **adi_adrv9001_spi_Verify(…)**, which performs the following functions:

1. Reads read-only register to check SPI read operation.
2. Writes scratchpad register with 10110110, reads back the data.
3. Writes scratchpad register with 01001001, reads back the data.

Such APIs are often described in the documentation as “This function is a helper function and does not need to be called directly by the user.” However, there is nothing to prevent from calling them early in the platform setup.

SOFTWARE INTEGRATION

Raspberry Pi HAL

Following are some basic examples to fill the functions listed previously for a Raspberry Pi platform. The Broadcom SPI library for the Raspberry Pi handles all configurations and interactions with the ADRV9001. Given that the Raspberry Pi's connectivity is limited to the GPIO pin headers, individual pins are used for the chip-enable signal and the RESET signal. This is done for fine control on the interaction timings. In the **customer_adi_adrv9001_hal_open(...)** function, the Raspberry Pi's SPI is initialized and configured to match the required behavior, and the chip-enable pin is configured as an output. The **customer_adi_adrv9001_hal_close(...)** function, by contrast, simply ends all SPI activity.

Next are the **customer_adi_adrv9001_hal_spi_write(...)** and **customer_adi_adrv9001_hal_spi_read(...)** functions. In accordance with the workings of the bcm2835 library, these functions write data to the ADRV9001 after driving the chip-enable line low. For the **customer_adi_adrv9001_hal_spi_read(...)** function, data is also accepted from the device on the DO line (MISO on the Raspberry Pi).

Lastly is the **customer_adi_adrv9001_hal_resetPin(...)** function. When provided with a value (1 or 0), this function drives the RESET pin high or low.

```
#include "adi_adrv9001_hal_customer.h"
#include "bcm2835.h" //rpi spi library
#define CE_PIN RPI_BPLUS_GPIO_J8_37
#define RESET_PIN RPI_BPLUS_GPIO_J8_36

int32_t customer_adi_adrv9001_hal_open(void *devHalCfg) {
  /* Customer code goes here */
  if (!bcm2835_init())
  {
    printf("bcm2835_init failed. Are you running as root??\n");
    return 1;
  }
  if (!bcm2835_spi_begin())
  {
    printf("bcm2835_spi_begin failed. Are you running as root??\n");
    return 1;
  }

  bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST); // The default
  bcm2835_spi_setDataMode(BCM2835_SPI_MODE0); // The default
  bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_32768); // 12kHz
  bcm2835_spi_chipSelect(BCM2835_SPI_CS_NONE); //
  bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, HIGH); // The default

  // Set the CE pin to be an output
  bcm2835_gpio_fsel(CE_PIN, BCM2835_GPIO_FSEL_OUTP);
  bcm2835_gpio_write(CE_PIN, HIGH);
  delay(1);
  return 0;
}

int32_t customer_adi_adrv9001_hal_close(void *devHalCfg) {
  /* Customer code goes here */
  bcm2835_spi_end();
  return 0;
}

int32_t customer_adi_adrv9001_hal_spi_write(void *devHalCfg, const uint8_t txData[], uint32_t numTx▶
Bytes)
{
  /* Customer code goes here */
  bcm2835_gpio_write(CE_PIN, LOW);
```

SOFTWARE INTEGRATION

```
  delay(1);
  bcm2835_spi_begin();
  bcm2835_spi_transfern(txData, numTxBytes);
  bcm2835_spi_end();
  delay(1);
  bcm2835_gpio_write(CE_PIN, HIGH);
  return 0;
}

int32_t customer_adi_adrv9001_hal_spi_read(void *devHalCfg, const uint8_t txData[], uint8_t rxData[],
uint32_t numTxRxBytes)
{
  /* Customer code goes here */
  bcm2835_gpio_write(CE_PIN, LOW);
  delay(1);
  bcm2835_spi_begin();
  bcm2835_spi_transfernb(txData, rxData, numTxRxBytes);
  bcm2835_spi_end();
  delay(1);
  bcm2835_gpio_write(CE_PIN, HIGH);
  return 0;
}

int32_t customer_adi_adrv9001_hal_resetbPin_set(void *devHalCfg, uint8_t pinLevel)
{
  /* Customer code goes here */
  if (pinLevel == 1)
  {
    bcm2835_gpio_write(RESET_PIN, HIGH);
  }
  else
  {
    bcm2835_gpio_write(RESET_PIN, LOW);
  }
  return 0;
}
```

In this example of the Raspberry Pi platform, alter the Makefile to suit. As this example uses the bcm2835 library, make an additional link in the Makefile, as shown here:

```
-lm -lbcm2835 -lpthread
```

This link allows the compiler to access the bcm2835 library, provided it is installed correctly on the Raspberry Pi. Take similar measures on a user platform if additional libraries are needed to control the platform.

Once the auto-generated code is verified as operational, and then modified to suit the user's platform, it is safe to remove all files, folders, and function calls (including needless print statements) containing code not pertaining to the user's platform. To do so, the be certain that all calls to the removed functions are replaced by the bespoke code or removed altogether. Once finished, review the Makefile again and remove any unnecessary links and inclusions from the compilation process.

API Error Handling and Debug

Logging Functions

The API provides a simple logging feature function for debugging. The available logging levels are given by **adi_common_LogLevel_e**, as shown in Table 12.

SOFTWARE INTEGRATION

Table 12. Logging Level

Function Name	Purpose
ADI_LOGLEVEL_TRACE	Logs everything in exhaustive detail. Use only for development.
ADI_LOGLEVEL_DEBUG	Logs diagnostic information.
ADI_LOGLEVEL_INFO	Logs state changes in the application.
ADI_LOGLEVEL_WARN	Logs bad, but recoverable events.
ADI_LOGLEVEL_ERROR	Logs events that cannot be recovered from.
ADI_LOGLEVEL_FATAL	Logs events likely to be fatal.
ADI_LOGLEVEL_NONE	Disables all logging.

When logging is enabled, the APIs log various messages to the system through the HAL. This feature requires an implementation for the **customer_LogWrite** function. To enable logging, set #define **ADI_COMPILED_LOGLEVEL** to one of the defined log levels other than **ADI_LOGLEVEL_NONE**. The application must do any log initialization (for example, opening files) before calling any APIs, or messages do not log properly. Log levels increase in severity numerically. When a given level is used, messages logged with a log level at least as severe (greater than or equal to) as the set level is published. Less severe logging calls are compiled out of the API. By default, **ADI_COMPILED_LOGLEVEL** is set to **ADI_LOGLEVEL_WARN**, which means that only messages with severities of **ADI_LOGLEVEL_WARN**, **ADI_LOGLEVEL_ERROR**, and **ADI_LOGLEVEL_FATAL** are included. Log levels are passed to **customer_LogWrite** as it may be required to log messages of differing severity in different manners (for example, insert a message prefix for each level, log errors to standard error, etc.).

Error Handling

Each ADRV9001 API function returns an **int32_t** value representing a recovery action, with 0 being no action or success. Recovery actions are divided into:

- Warning actions that do not have an impact when executing the device API but can cause performance issues or logging problems. The value of this action is positive.
- Error actions that cause the API unable to run. An action is required for the API to go back to a good state. The value of this action is negative.

DEVELOPING THE APPLICATION

The user application must allocate the init (**adi_adrv9001_Init_t**) and device (**adi_adrv9001_Device_t**) structures. Consider allocating memory from the heap for the **adi_adrv9001_Device_t** and **adi_adrv9001_Init_t**, as the structures have members with a large memory size. As part of the SDK, there is a **memory_profile.py** script designed to profile the storage of application variables. Users concerned about memory usage or limitations may find this script beneficial. There is a section of the **README.md** under the **production** folder dedicated to **Running the memory profiler**.

Also note, the ARM binary and Stream Image are now factored out into separate files, which help mitigate memory concerns.

The **adi_adrv9001_Init_t** structure contains the user profile initialization settings to configure the ADRV9001 device. This 'init' structure is passed to the ADRV9001 API 'init' functions during the initialization phase. This structure contains the device profile settings, system clock settings, data interface settings, and ADRV9001-specific SPI target controller settings. The application layer passes a pointer to an instance of the **adi_adrv9001_Init_t** structure for a particular ADRV9001 device to handle most of the device core initialization. After initialization, the **adi_adrv9001_Init_t** structure may be deallocated, if required.

The **adi_adrv9001_Device_t** data structure contains information for a particular ADRV9001 device, including devHalInfo, error, and caching structures. To support multiple ADRV9001 devices, the application must instantiate multiple **adi_adrv9001_Device_t** structures to describe each physical ADRV9001 device. Multiple ADRV9001 devices can have their own **adi_adrv9001_Init_t** or can share a common **adi_adrv9001_Init_t** if they are to be configured identically.

devHalInfo

The **devHalInfo** is a structure to define and pass any platform hardware settings to the platform HAL layer functions. The common device structure **adi_common_Device_t** contains the **devHalInfo**. The **devHalInfo** is passed to the platform specific HAL function as a void ***devHalCfg**. The ADRV9001 API functions do not read or write the **devHalInfo** but pass it as a parameter to all HAL function calls.

SOFTWARE INTEGRATION

The application developer must define the **devHalInfo** per system HAL implementation requirements. The application developer may implement any structure to pass any hardware configuration information that the hardware requires between the application and platform layer. For example, the **devHalInfo** contains SPI chip select information to use for the physical ADRV9001 device.

Note that the API functions are shared across all instances of physical ADRV9001 devices. The **devHalInfo** structure defined by the developer identifies which physical ADRV9001 device is targeted (SPI chip select) when a particular ADRV9001 API function is called. The developer may need to store other hardware information unique to a particular ADRV9001 device in this structure such as timer instances or log file information.

Note for the ADRV9001 API, only one thread must control and configure a specific device instance at any given time.

devStateInfo

The **devStateInfo** member is of type **adi_adrv9001_Info_t** and is a run-time state container for the ADRV9001 API. The application layer must allocate memory for this structure, but only the ADRV9001 API writes to the structure. The application layer allocates the **devStateInfo** structure with all zeroes. The API uses the **devStateInfo** structure to keep up with the current state of the API (is it initialized, ARM loaded, etc.), as well as a debug store for any run-time data, such as error codes, error sources, and so forth. The application layer must not access the **devStateInfo** member directly because there are API functions to access the information of the last error.

Private vs. Public API Functions

The API is made up of multiple **.c** and **.h** files. As the API is written in C, there are no language modifiers to identify a function as private or public as commonly used in object-oriented languages. Per the ADI coding standard, the public API functions are denoted by the function name prefixed with **adi_adrv9001_** (for example, **adi_adrv9001_Rx_Gain_Set()**). The private helper functions lack the **adi_ prefix** and are not to be called by the user application.

Most functions in the ADRV9001 API are prefixed with **adi_adrv9001_** and are for the public use. However, many of these functions are never called directly from the application. The utility functions that abstract some common operations, specifically the initialization of the ADRV9001, are provided in **adi_adrv9001_utilities.c**. So, much of the initialization and other helper functions are separated from the top-level **adi_adrv9001.c/ adi_adrv9001.h** files to help the developer focus on the functions most commonly used by the application.

Include Files

For each major function block, there are generally three files: **adi_[feature].c**, **adi_[feature].h**, and **adi_[feature]_types.h**. The ADRV9001 API places 'typedef' definitions in files with **_types** suffix such as **adi_adrv9001_types.h**. These **_types.h** files are included within their corresponding **.h** files and do not need to be manually included in the application layer code.

Note that the **adi_adrv9001_user.h** contains the #defines for API timeouts and SPI read intervals, which may be set as needed by the user platform. The ADRV9001 user files are the only API files the developer may change.

Restrictions

Analog Devices maintains the code in the **/c_src/devices/*** folders. Application developers must not modify this code. Direct SPI read/write operation is forbidden when configuring an ADRV9001 or any other ADI devices used to evaluate the ADRV9001. Developers should only use the high-level API functions defined in the public ***.h** files. Developers should not directly use any SPI read/write functions in the application for the ADRV9001 configuration or control. Analog Devices does not support any application containing SPI writes reverse engineered from the original ADRV9001 API.

Delays, Waits, and Sleeps

A subset of the ADRV9001 APIs requires delays to allow the hardware to complete internal configurations. These ADRV9001 APIs request the system to perform a wait or sleep by calling the HAL interface function **adi_common_Wait_us**. If the target platform's HAL interface implementation chooses to implement a thread-sleep, the application is not permitted to call another API targeting the same ADRV9001 device. The application must wait/sleep for the API to complete before continuing with the configuration of the device.

Wait/sleep periods are defined in the **adi_adrv9001_user.h**. The timeout period values are the recommended period required to complete the operation. Modifying these values is not recommended and may impact performance. During this timeout period, the status of the ADRV9001 is polled. The frequency of the polling the status during this timeout period may be modified by adjusting the value of the polling interval.

SYSTEM INITIALIZATION AND SHUTDOWN

There is a graphical user interface (GUI)-based TES to initialize and interact with the ADRV9001 device for evaluation. Through this TES, provide high level system configuration parameters such as signal bandwidth, sample rate, and initial gain control settings to initialize the device. The TES uses these parameters to set up an initialization C structure and then makes multiple API calls in a proper order to initialize the device. During the normal operation of the device, the TES allows further user interaction with the device, such as adjusting the transmit/receive gain on the fly. When the operation is completed, safely shut down the device through TES. Figure 20 shows the high-level flow of the device operation sequence and the user interaction through TES.

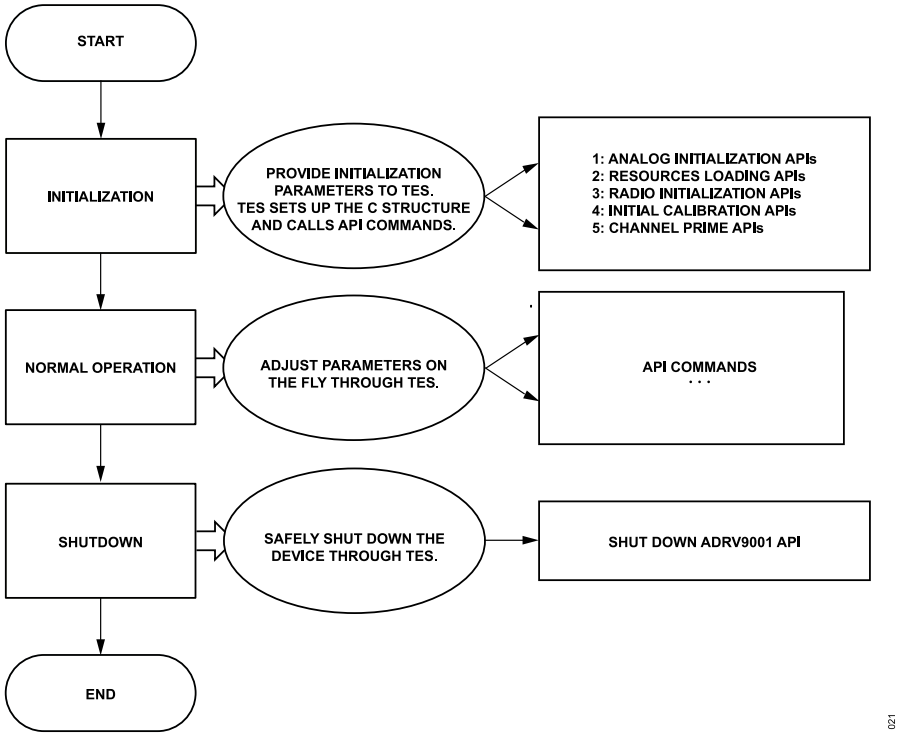


Figure 20. High Level Flowchart of the Device Operation and User Interaction Through TES

As shown in Figure 20, this section provides information about the initialization and shutdown process for the ADRV9001 device using the APIs developed by ADI. Figure 20 lists all high level operations used for initialization and shutdown through one or a set of APIs.The further sections discuss the major steps associated with each high level operation. Note with the SDK, user can initialize the ADRV9001 device through own software program independent of the TES. However, it should follow the same API calling procedure described in this document.

Note: This section does not explain every related API function. For detailed information on the API functions, refer to the ADRV9001 Device API Doxygen document. In addition, this section does not describe API integration and the hardware abstraction Interface. For more details, see the Software Integration section. For more details about the TES, see the Transceiver Evaluation Software section.

TES CONFIGURATION AND INITIALIZATION

The TES provides a **Config** tab that contains all the setup options for the ADRV9001. Under the **Config** tab, configure each channel of the device for a required profile under the **Device Configuration** subtab, which sets high level parameters such as duplex mode, data port sample rates, and RF channel bandwidth. Further initialize the options used by the device during start-up under other subtabs, such as the carrier frequencies, ADC type, and initial calibrations.

Based on the set parameters, the TES formulates an initialization structure, **adi_adrv9001_Init_t**, to contain all the required settings to configure the device. This structure contains the settings for the system configuration, system clock, transmit/receive data structure, and programmable FIR filter. For more details, refer to the Doxygen document.

When all tabs are configured, press the **Program** button in TES. This starts the initialization programming. TES sends a series of API commands executed by a dedicated Linux application on the platform. This initialization structure is passed to the ADRV9001 API initialization functions during the initialization phase. When programming is completed, the system is fully calibrated. With a few additional API calls, the device is ready to operate.

SYSTEM INITIALIZATION AND SHUTDOWN

The TES also generates MATLAB, Python (.py), C# or C code, which includes all high level API initialization calls. Store these automatically generated codes in a location of choice for future use.

API INITIALIZATION SEQUENCE

As previously mentioned, the initialization sequence consists of a serial of API calls intermixed with user-defined function calls specific to the hardware platform. The API functions perform all the necessary tasks for device configuration, calibration, and control. Figure 21 shows the state machine of the device from power-up to RF enabled. Moving back to the 'STANDBY' state is not currently supported without a device reset.

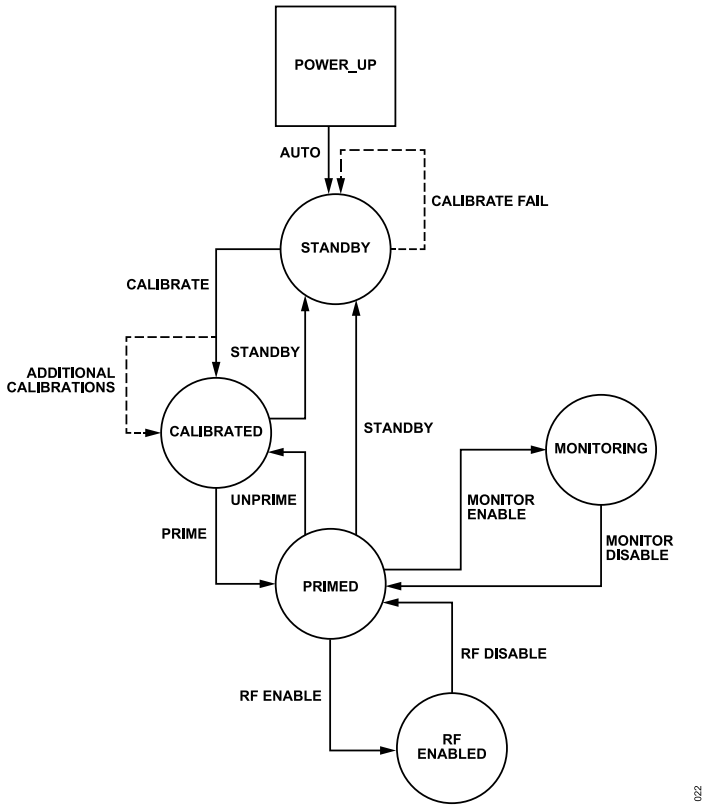


Figure 21. Device State Machine

As shown in Figure 21, after power-up, the device automatically enters the standby state. Then, the device initialization and calibration begin. If it is successful, the device moves to the calibrated state, or else, it remains in the standby state. Note that when TES completes programming, the device is in the calibrated state. After the device is calibrated, move it to the primed state, which indicates the device is ready for operation. Then, through SPI or PIN mode, move the device to the RF-enabled state by enabling the transmit/receive channels so transmission and reception can start. In TES, after programming, playing the receiver or transmitter moves the device from the calibrated state to the primed state and then to the RF-enabled state.

This section mainly discusses the device initialization procedure from the standby state to the RF_ENABLED state. The following subsections discuss the related high-level API functions briefly. For details of each API function, refer to the Doxygen document.

Note for MIMO systems with multiple input and output channels, multiple ADRV9001 devices might be involved. To synchronize among all the devices, it requires a common device clock (DEV_CLOCK) and an MCS signal so that all the internally generated analog and digital clocks are aligned among all the devices. In addition, the MCS is used to synchronize the device and baseband processor data interface for all devices. For simplicity, in the following descriptions, MCS operations are excluded from the initialization steps. For more details, see the Multichip Synchronization section.

SYSTEM INITIALIZATION AND SHUTDOWN

Analog Initialization

Analog initialization API **adi_adrv9001_InitAnalog()** is the very first API call to configure the device after all dependent data structures are initialized. It mainly sets the controller bias, enables the reference clock, and validates the profile settings.

Resource Loading

After analog initialization, a set of APIs is used to load required resources such as stream image, ARM image, programmable FIR (PFIR) coefficients, and so on. It also enables the internal microprocessor and initializes the digital clocks.

Figure 22 shows the major APIs and their functionalities (arranged sequentially from left to right based on the order of the API calls).

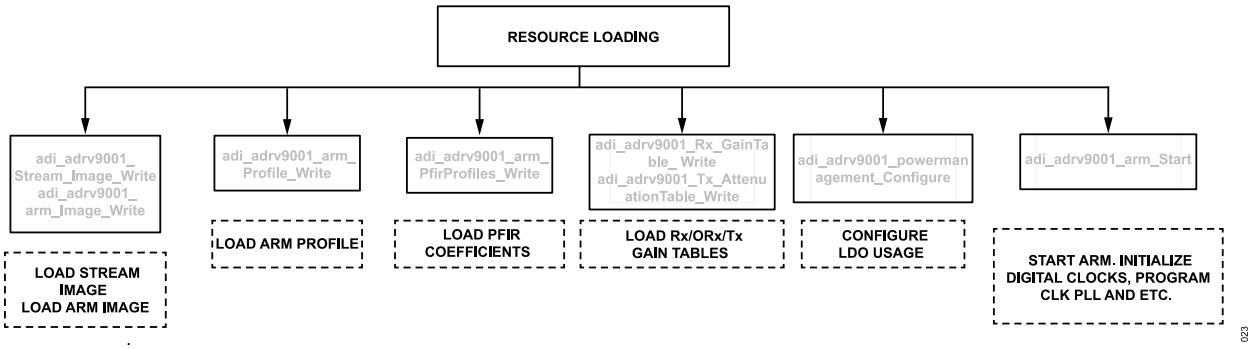


Figure 22. Load Resources and Digital Initialization

Radio Initialization

After digital initialization, the next step is radio initialization through a set of API calls, which is used to load any radio configuration data not passed by profile before performing initial calibrations, such as GPIO configuration, phase locked loop (PLL) filter configuration, carrier frequencies, time division duplex (TDD) timing parameters, power management configurations, MCS delay configurations, etc.

Figure 23 shows the major APIs.

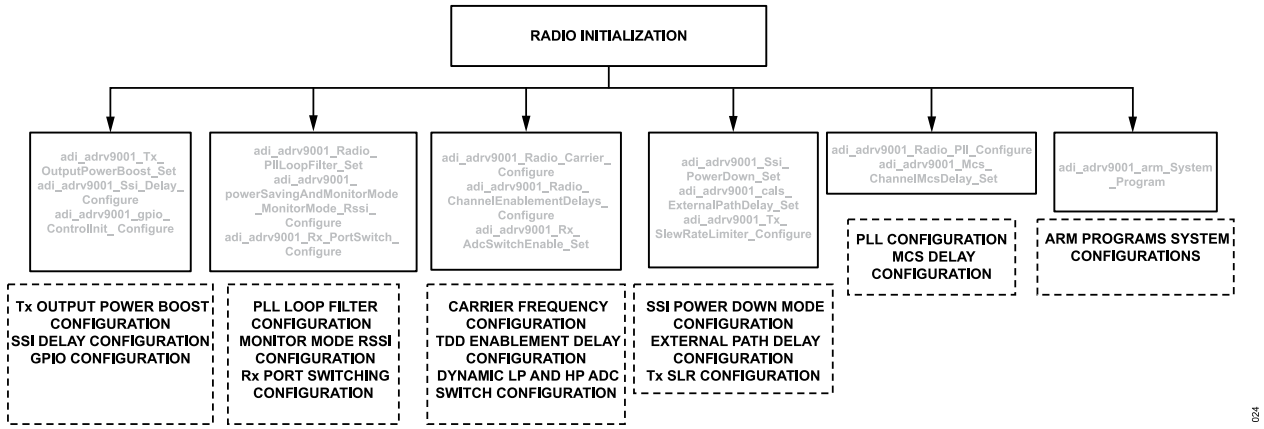


Figure 23. Radio Control Initialization

Calibrations Initialization

The next step in initialization is to perform initial calibrations through API call **adi_adrv9001_cals_InitCals_Run()** based on calibration mask. To understand calibration mask, see the Transmitter/Receiver/Observation Receiver Signal Chain Calibrations section. When initial calibrations are correctly performed, the channel state transitions from the standby to calibrated state. If MCS is enabled, after initial calibration, the next step is to configure and perform MCS by calling APIs **adi_fpga9001_Mcs_Configure()** and **adi_fpga9001_Mcs_Start()**. For more information, see the Multichip Synchronization section.

SYSTEM INITIALIZATION AND SHUTDOWN

Prime and RF Enable

The last step in initialization is to move the device from the calibrated to primed state through API call **adi_adrv9001_Radio_Channel_Prime()** or **adi_adrv9001_Radio_Channel_ToPrimed()**. The primed state indicates the system is ready for operation when the transmit and receive channels are enabled.

Note that after the channel is primed, to start the normal transmit or reception activities, further transition it from the primed to RF-enabled state with a set of API calls. There are two modes to enable channels: the PIN and SPI modes.

PIN Mode

1. Call **adi_adrv9001_Radio_ChannelEnableMode_Set( )** to set the PIN mode.
2. Toggle the pins (for example, Rx1_ENABLE and Tx1_ENABLE pins for Channel 1) to transition the channel to the RF-enabled state.

SPI Mode

1. Call **adi_adrv9001_Radio_ChannelEnableMode_Set( )** to set the SPI mode.
2. Call **adi_adrv9001_Radio_Channel_EnableRf( )** to transition the channel to the RF-enabled state.

STATE CHANGE TIMING

Table 13 shows the approximate time taken to change the state as well as the APIs used to perform it. It is also possible to call "**adi_adrv9001_Radio_Channel_State_Get()**" to verify the state of the device. Note that the test was performed under two profiles: LTE and DMR. These are the default profiles as configured in the TES.

The time taken to calibrate the device varies significantly from profile to profile and can take several seconds to run. This is due to varying initial calibrations used in different profiles. The Warm Boot section details a method to significantly reduce the time to calibrate the device.

Table 13. Time Taken to Change State

State Change	API	LTE: Time Taken (µs)	DMR: Time Taken (µs)
Standby -> Calibrated	adi_adrv9001_cals_InitCals_Run()	5,287,000	2,360,000
Calibrated -> Primed	adi_adrv9001_Radio_Channel_Prime()	2,100	2,100
Primed -> Calibrated	adi_adrv9001_Radio_Channel_ToCalibrated()	2,100	2,100
Primed -> RF_Enabled	adi_adrv9001_Radio_Channel_EnableRf()	23	23
RF_Enabled -> Primed	adi_adrv9001_Radio_Channel_EnableRf()	23	23

SHUTDOWN SEQUENCE

After completing all the operations, call API **adi_adrv9001_Shutdown()** through the TES to safely shut down the ADRV9001 device. It performs hardware reset to reset the ADRV9001 device into a safe state for shutdown or reinitialization.

SYSTEM DEBUGGING

The "System Debugging" feature runs a system check on the ADRV9001 to ensure that all main functionalities of the device are working correctly. The system check is primarily a debugging tool to assist in narrowing down any issues with the system. It can also be run at each start-up to check if the device booted correctly. The system debug can help to resolve the most common issues associated with producing a new system with the ADRV9002.

The system debug performs the following checks:

- Checks the API version.
- Configures/checks the SPI.
- Checks the power supply.
- Loads/checks the stream processor image.
- Loads/checks Arm image.

SYSTEM INITIALIZATION AND SHUTDOWN

- Checks the device clock.
- Checks the RF PLLs.

A detailed report shows a list of all the tests run, the success or non-success of the tests, and some guidance on the cause of an unsuccessful test. The following is a sample successful report:

**** ADRV9001 Pre-Calibrate System Debugging Started ****

--> . Hardware Reset

OK

--> . API Version

OK - API Version = 67.3.2

--> . Configure SPI

OK

--> . Check SPI

OK

--> . Check Power Supplies

OK

--> . ARM Image Load

Started Reading Binary Image

Completed Reading Binary Image

OK

--> . Check ARM Image Load

OK

--> . Check Dev_Clk

OK

--> . Load SP

Started Reading Binary Image

Completed Reading Binary Image

OK

--> . Check SP

OK

**** ADRV9001 Pre-Calibrate System Debugging Completed ****

**** ADRV9001 Post-Calibrate System Debugging Started ****

--> . Check RF PLLs

OK

**** ADRV9001 Post-Calibrate System Debugging Completed ****

Table 14 shows the two APIs associated with this feature: one each for precalibration and post-calibration checks.

SYSTEM INITIALIZATION AND SHUTDOWN

Table 14. APIs for Precalibration and Post-Calibration Checks

API Function	Description
adi_adrv9001_Utilities_SystemDebugPreCalibrate()	Function for in-system debugging (precalibration)
adi_adrv9001_Utilities_SystemDebugPostCalibrate()	Function for in-system debugging (post-calibration)

WARM BOOT

As described in the Calibrations Initialization section, initial calibration is performed during the initialization of the ADRV9001 device. Depending on the profile configured, initial calibration can take a significant time to complete, especially for complicated profiles such as frequency hopping with dynamic receiver port switching. Some example timings are given in the Warm Boot Boot-Up section. Each time after resetting such a system, performing the same initial calibration is required, which is unacceptable for a system that requires a short boot-up time.

Warm boot is developed to shorten the initialization time by applying pre-saved initial calibration results to the device when it is reset and then reinitialized to the same profile. Therefore, the initial calibration procedure is skipped during device initialization stage for a much shorter boot-up time. To achieve this, the ADRV9001 device provides user APIs to save the initial calibration coefficients after performing the first initial calibration (cold boot) and then loads the saved initial calibration coefficients to the device before the subsequent boot-ups (warm boot).

Figure 24 shows the warm boot procedures.

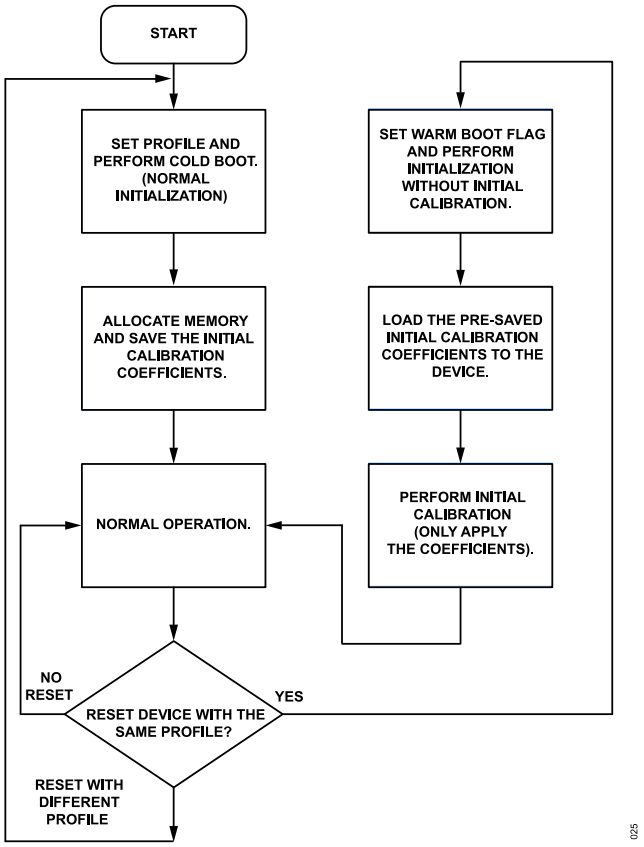


Figure 24. A Block Diagram of Warm Boot Procedures

There are three sets of APIs to use warm boot: a memory-optimized version, a non-memory-optimized version and a chunk-loading version. The non-memory-optimized APIs require the allocation of memory for the maximum number of calibration coefficients (936 kB), whereas the memory-optimized version requires the allocation of memory only for the calibrations coefficients needed in the user's specific profile. The memory-optimized APIs are recommended for memory-constrained systems. The chunk-loading APIs load the coefficients in user-defined blocks to optimize the host controllers stack usage. Use the chunk-loading APIs when there is limited stack space available on the host controller.

SYSTEM INITIALIZATION AND SHUTDOWN

Table 15. Warm Boot APIs

API	Description
adi_adrv9001_cals_InitCals_WarmBoot_Coefficients_MaxArray_Get()	Read the InitCal coefficients needed for warm boot (Non-memory-optimized).
adi_adrv9001_cals_InitCals_WarmBoot_Coefficients_MaxArray_Set()	Write the InitCal coefficients needed for warm boot (Non-memory-optimized).
adi_adrv9001_cals_InitCals_WarmBoot_Coefficients_UniqueArray_Get()	Read the InitCal coefficients needed for warm boot and place in allocated memory. (Memoy-optimized).
adi_adrv9001_cals_InitCals_WarmBoot_Coefficients_UniqueArray_Set()	Write the InitCal coefficients needed for warm boot from allocated memory. (Memoy-optimized).
adi_adrv9001_cals_InitCals_WarmBoot_UniqueEnabledCals_Get()	Return the unique initCals enabled and the required memory size for this device configuration. (Memoy-optimized).
adi_adrv9001_Utilities_InitCals_WarmBoot_Coefficients_MaxArrayChunk_Get()	Read the InitCal coefficients needed for Warmboot in chunks.
adi_adrv9001_Utilities_InitCals_WarmBoot_Coefficients_MaxArrayChunk_Set()	Write the InitCal coefficients needed for Warmboot in chunks.
adi_adrv9001_Utilities_InitCals_WarmBoot_Coefficients_VectTblChunkRead()	Read the InitCal cal coefficient table.

After the first initial calibration of the device, follow the below procedure for the desired warm boot method.

Procedure for non-memory-optimized APIs is as follows:

- Allocate memory for the warm boot coefficients by using the **adi_adrv9001_Warmboot_Coeff** data structure.
- Call **adi_adrv9001_cals_InitCals_WarmBoot_Coefficients_UniqueArray_Get()** after the first initialization at “Calibrated,” “Primed”, or “RF_enabled” channel state. Also provide the channel 1 and 2 initial calibration masks when calling the 'get' API.
- When performing reinitialization with the same profile after resetting the system, set the warm boot flag in the profile.
- Perform all the initialization steps without running initial calibration.
- Call **adi_adrv9001_cals_InitCals_WarmBoot_Coefficients_MaxArray_Set()** to load the pre-saved initial calibration coefficients to the device.
- Call the initial calibration API. Note that based on the warm boot flag, this API only applies the loaded coefficients without actually performing the initial calibration.

Procedure for memory-optimized APIs is as follows:

- Retrieve the memory requirements for the warm boot coefficients using **adi_adrv9001_cals_InitCals_WarmBoot_UniqueEnabledCals_Get()**, and then allocate this memory.
- Call the get API after the first initialization at “Calibrated,” “Primed”, or “RF_enabled” channel state. Also provide the channel 1 and 2 initial calibration masks when calling the 'get' API.
- When performing reinitialization with the same profile after resetting the system, set the warm boot flag in the profile.
- Perform all the initialization steps without running initial calibration.
- Call the **adi_adrv9001_cals_InitCals_WarmBoot_Coefficients_UniqueArray_Set()** API to load the pre-saved initial calibration coefficients to the device.
- Call the initial calibration API. Note that based on the warm boot flag, this API only applies the loaded coefficients without actually performing the initial calibration.

Procedure for chunk-loading APIs is as follows:

- Use **adi_adrv9001_cals_InitCals_WarmBoot_UniqueEnabledCals_Get()**, to retrieve the number of cals enabled.
- For getting warm boot coeffients:
  - Iterate through each cal.
  - Read the vector table data for that cal using **adi_adrv9001_Utilities_InitCals_WarmBoot_Coefficients_VectTblChunkRead()**.
  - Using vector table information, retrieve the warm boot coefficient for that cal using **adi_adrv9001_Utilities_InitCals_WarmBoot_Coefficients_MaxArrayChunk_Get()**. Either retrieve the cal in one chunk or optionally split it into smaller chunks.
- When performing reinitialization with the same profile after resetting the system, set the warm boot flag in the profile.
- Perform all the initialization steps without running initial calibration.
- For setting warm boot coeffients:
  - Iterate through each cal.

- Read the vector table data for that cal using **adi_adrv9001_Utilities_InitCals_WarmBoot_Coefficients_VectTblChunkRead()**.
- Using vector table information, load the warmboot coefficient for that cal using **adi_adrv9001_Utilities_InitCals_WarmBoot_Coefficients_MaxArrayChunk_Set()**. Either load the cal in one chunk or optionally split it into smaller chunks.

- Call the initial calibration API. Note that based on the warm boot flag, this API only applies the loaded coefficients without actually performing the initial calibration.

Evaluate the warm boot functionality through the TES by following the procedures shown in Figure 25.

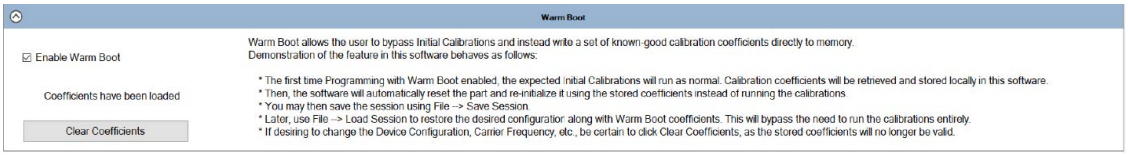


Figure 25. Performing Warm Boot in TES

Note: Perform cold boot for several different profiles one by one and save the corresponding coefficients of each profile (assuming there is sufficient memory to save all the coefficients). During transceiver operations, use the warm boot feature to reduce the initialization time required to change to another profile by simply loading the corresponding coefficients of the desired profile. This allows quick user switch among a set of predefined profiles for agile transceiver operations.

Operating Temperature Considerations

Since calibration coefficients are dependent on the ambient temperature during calibration, this might not give optimal performance if operating at a different temperature. For example, if the initial calibrations were taken in a lab environment at 25°C, but the device is operating at a higher temperature of 40°C, the initial calibrations alone might not give optimal performance. There are two methods which could be considered to supplement the warm boot coefficients in this scenario as follows:

- Tracking calibrations: as discussed in the Tracking Calibrations section, tracking calibrations are effective for improving the performance of the initial calibrations over changing temperatures. In a similar way, the tracking calibrations can be used to improve the performance of warm boot initial calibrations even if operating at a different temperature. The tracking calibrations will take some time to improve performance after the start of operation.
- Multiple warm boot coefficients: a device could be calibrated over a range of temperatures in a lab environment and store the calibration coefficients for each temperature. The corresponding calibration coefficients could be loaded using warm boot based on the operating temperature at boot-up. For example, the device could be calibrated at 'hot', 'ambient' and 'cold' temperatures and the coefficients for each stored. If operating close to the 'hot' temperature, these calibration coefficients could be then loaded at boot-up.

Refer to the ADRV9001 data sheets for specific performance results.

BOOT-UP TIMING

Many users have a vested interest in knowing the time needed for the ADRV9001 devices to perform certain tasks, such as Init Calibrations, ARM image loading, profile configuration, etc. It is also common that the RF Bring-Up time (time from HW RESET to RF Enabled) must be within a limit for certain applications. For those users who need timing data such as this, it is explained in further sections. The following sections detail the measurement method, how this method is adjusted to suit the user's application, how to interpret the results from this measurement, and a selection of the data from the user's own tests.

Measurement Method

Irrespective of the application, the methodology remains the same. The idea is to generate code that programs the setup to use with the ADRV9001 product, modify it to include functionality unique to an application, if necessary, and then time its execution. There are several aspects to consider with this idea, the foremost being the choice of programming language. TES can autogenerate code for four languages: C99, C#, Python, and MATLAB. For timing, use the C99 code running on the platform as it yields the most accurate results without incurring TCP/IP communication overhead.

SYSTEM INITIALIZATION AND SHUTDOWN

Following is an example of how to measure the time taken by the main method. The time is found at the beginning and end of the function, and the difference between the start and end time is calculated to be the time taken for the program to run. The code in this snippet is taken from the autogenerated C99 code from TES.

```
int main()
{
    // Start Timer
    struct timeval start_t, end_t;
    gettimeofday(&start_t, NULL);

    int32_t error_code = 0;

    adi_adrv9001_Device_t * adrv9001Device_0 = (adi_adrv9001_Device_t *) calloc(1,

sizeof(adi_adrv9001_Device_t));

      adi_fpga9001_Device_t * fpga9001Device_0 = (adi_fpga9001_Device_t *) calloc(1,

sizeof(adi_fpga9001_Device_t));

    error_code = linux_uio_initialize(adrv9001Device_0, fpga9001Device_0, NULL);

    AUTOGENERATOR_ERROR_HANDLER(error_code);

    error_code = initialize(adrv9001Device_0, fpga9001Device_0);

    AUTOGENERATOR_ERROR_HANDLER(error_code);

    .
    .
    .

    error_code = stopTransmitting(fpga9001Device_0);

    AUTOGENERATOR_ERROR_HANDLER(error_code);

    //End Timer
    gettimeofday(&end_t, NULL);
    //Print Out Time Taken
    printf("Time taken to run is : %ld micro seconds\n",
                ((end_t.tv_sec * 1000000 + end_t.tv_usec) -
                (start_t.tv_sec * 1000000 + start_t.tv_usec)));



return error_code;
}
```

Example Results

During experiments, Analog Devices focused on characterizing the time taken for the device to boot-up from RESET to RF ENABLED using a given digital mobile radio (DMR) configuration (note: this configuration is just an example use case. It does not represent a “best case” scenario):

- TDD mode of operation

SYSTEM INITIALIZATION AND SHUTDOWN

- DMR profile
- 1 transmitter
- 1 receiver
- 24 kSPS
- LO = 900 MHz
- Configuration details:
  - Low power clock PLL
  - Low power receiver ADC
  - LVDS SSI
  - Processor clock divider = /2
  - Low power mode for RF PLL LO
  - Receiver transimpedance amplifier (TIA) in low power mode
  - Transmitter band pass filter (BPF) in low power mode

In TES, program the transceiver and bring the receiver signal chain to RF_ENABLED. Then, disable the receiver and reset the part. The sample code is autogenerated from TES in C99.

To this code, add a series of timers to monitor the time taken to run each file using the method outlined previously. The autogenerated code is mostly unedited, although the print statements and user inputs are removed.

Table 16 and Figure 26 show the result of this experiment. Note that the timing for the "prime.c" and "beginReceiving.c" state changes took comparatively little time. So, the data is retrieved from the main chart.

Table 16. Time Taken to Run File at Boot-Up

File	Time (ms)
initialize.c	2539.481
calibrate.c	2776.808
configure.c	305.183
prime.c	4.215
beginReceiving.c	1.162
Total Boot-Up Time	5628.8

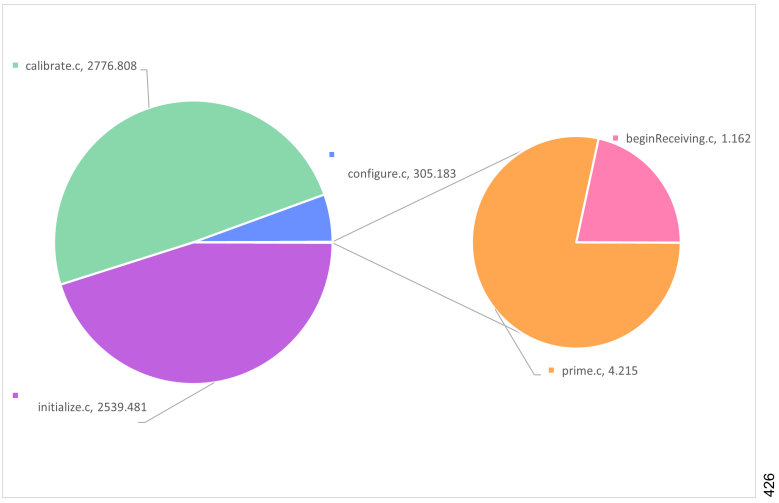


Figure 26. Time Taken to Boot-Up Device (ms)

Figure 26 shows that most of the time taken to boot the device comes from the intitialize.c and calibrate.c files. Also, the most significant contributing factor to this is loading values to the memory (such as the ARM image) and running the initial calibrations. Loading values to the

SYSTEM INITIALIZATION AND SHUTDOWN

memory can improve with a faster SPI implementation. Also note that the SPI implementation on the Analog Devices platform is not necessarily optimized for speed. So, results for a custom platform can improve. See the Warm Boot section for a method to reduce the boot-up time.

Warm Boot Boot-Up

Warm boot significantly reduces the time to calibrate the device. Warm boot stores and re-loads calibration coefficients without needing to run initial calibrations as described in the Warm Boot section.

Table 17 shows the time to calibrate the device with and without using warm boot. Regardless of the complexity of the profile, warm boot can reduce the INITIALIZED->CALIBRATED state change portion of device boot-up to less than 20 ms.

Table 17. Warm Boot Initialization Timing

Profile	Without Warm Boot Time (ms)	With Warm Boot Time (ms)
Narrowband Profile	2100	~16
Wideband Profile	2500	~16
2-Channel Profile	5100	~16
2-Channel Frequency Hopping Profile	8700	~16

For this test, enable the warm boot and repeat the test mentioned in the Example Results section. The memory optimized APIs **adi_adrv9001_cals_InitCals_WarmBoot_Coefficients_UniqueArray_Get/Set()** are used for best results.

SERIAL-PERIPHERAL INTERFACE (SPI)

The SPI bus provides the mechanism for digital control by a baseband processor. Each SPI register is 8 bits wide, and each register contains control bits, status monitors, or other settings that control all device functions . This section is mainly an information-only section meant to understand the hardware interface used by the baseband processor 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 CONFIGURATION

Configure the SPI settings for the device with different SPI controller configurations by configuring the member values of the **adi_adrv9001_SpiSettings_t data** structure. The **adi_adrv9001_SpiSettings_t data** structure contains:

```
typedef struct adi_adrv9001SpiSettings{
uint8_t msbFirst;
uint8_t enSpiStreaming;
uint8_t autoIncAddrUp;
uint8_t fourWireMode;
adi_adrv9001_CmosPadDrvStr_e cmosPadDrvStrength;
} adi_adrv9001_SpiSettings_t;
```

Table 18 lists the parameters for this structure.

Table 18. SPI Settings Data Structure

Structure Member	Value	Function	Default
MSBFirst	0x00	Least significant bit first.	0x01
	0x01	Most significant bit first.	
enSpiStreaming	0x00	Disable SW feature. Section Multibyte Data Transfer (SPI Streaming) describes this mode of operation.	0x00
	0x01	Enable SW feature to improve SPI throughput. Section Multibyte Data Transfer (SPI Streaming) describes this mode of operation. Not recommended as most registers in the ADRV9001 API are not consecutive.	
autoIncAddrUp	0x00	Autodecrement. Functionality to be used with SPI Streaming. Sets address autodecrement -> next addr = addr -1	0x01
	0x01	Autoincrement. Functionality to be used with SPI Streaming. Sets address autoincrement -> next addr = addr +1	
fourWireMode	0x00	SPI hardware implementation ofusing 3-wire SPI (SDIO pin is bidirectional). Note: ADI's FPGA platform always uses 4-wire mode.	0x01
	0x01	SPI hardware implementation of using 4-wire SPI. Note: Default mode for ADI's FPGA platform is 4-wire mode.	
cmosPadDrvStrength	0x00	5 pF load at 75 MHz	0x01
	0x01	100 pF load at 20 MHz	

Note: Any value not listed in the table is invalid.

For more details, refer to the ADRV9001_API Doxygen file provided in the ADRV9001 SDK package.

SPI BUS SIGNALS

The SPI bus consists of the following signals:

- Serial clock (SCLK)
- Chip select bar (CSB)
- Serial data input/output and serial data output (SDIO and SDO)

SERIAL-PERIPHERAL INTERFACE (SPI)

Serial Clock (SCLK)

SCLK is the serial interface reference clock driven by the baseband processor (uses the SPI_CLK pin). It is only active while CSB is low. The maximum SCLK frequency is 45.45 MHz.

Chip Select Bar (CSB)

CSB is the active-low chip select that functions as the bus enable signal driven from the baseband processor to the device (uses the SPI_EN pin). CSB is driven low before the first SCLK rising edge and is normally driven high again after the last SCLK falling edge. The device ignores the clock and data signals while CSB is high. CSB also frames communication to and from the device and returns the SPI to the ready state when it is driven high.

Forcing CSB high in the middle of a transaction stops part or all of the transaction. If the transaction stops before the instruction is complete or in the middle of the first data-word, the whole transaction stops and the state machine returns to the ready state. Any complete data byte transfers before CSB deserting are valid, but all subsequent transfers in a continuous SPI transaction stop.

Serial Data Input/Output and Serial Data Output (SDIO and SDO)

When configured as a 4-wire bus, the SPI uses two data signals: SDIO and SDO. SDIO is the data input line driven from the baseband processor (uses the SPI_DIO pin) and SDO is the data output from the device to the baseband processor in this configuration (uses the SPI_DO pin). When configured as a 3-wire bus, SDIO is used as a bidirectional data signal that both receives and transmits serial data. The SDO port is disabled in this mode.

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

The SDO and SDIO pins transition to a high-impedance state when the CSB input is high. The device does not provide any weak pullups or pulldowns on these pins. When SDO is inactive, it is floated in a high-impedance state. If a valid logic state on SDO is required at all times, add an external weak pullup/pulldown (10 kΩ value) on the PCB.

SPI BROADCAST MODE

Broadcast mode is used to initialize multiple ADRV9001 devices at the same time, in the initialization stage. . When SPI broadcast is not enabled, multiple devices will be controlled in series where each device is controlled one at a time e.g. load the arm image on device one, then on device two and so on. Broadcast mode allows the devices to be initialized in parallel, so each device is controlled at the same time e.g. the arm image is loaded to all devices at the same time. This means that from the RESET to the CALIBRATED states, the devices will be all controlled together.

The main advantage of this mode is to save time in multi ADRV9001 applications. Where N devices are all initialized using broadcast mode, the time taken to initialize all the devices is reduced by a factor of N.

The expected hardware setup is similar to the Figure 27 with a ‘main’ device (BBIC) and 2+ ‘subordinate’ devices (ADRV9001) sharing common clk + data lines. In broadcast mode, user would simultaneously activate all chip select lines and then send the data to all ‘sub’ devices at the same time. This means that SPI is operating in write only mode and there will be no SPI reads. Since there is no readback from the devices during broadcast mode, this is not recommended during the development/debug stage since readback is useful for debugging. Any readback or verification per device can be done after the full broadcast initialization procedure has completed.

SERIAL-PERIPHERAL INTERFACE (SPI)

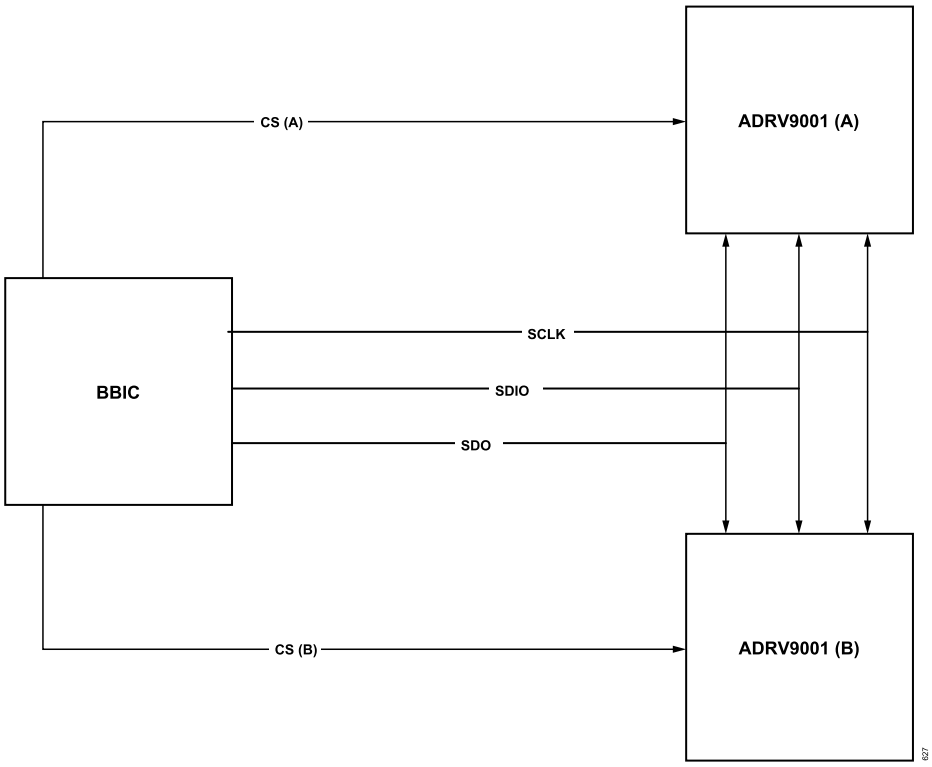


Figure 27. SPI Broadcast Mode Block Diagram

To enable broadcast mode, users must enable it in the **adi_adrv9001_user.h** file. The file is located in the ADRV9001 SDK at **adrv9001-sdk\pkg\production\c_src\devices\adrv9001\public\include**.

Table 19. Defines For Broadcast Mode Enablement

Define	State
#define ADI_ADRV9001_PRE_MCS_BROADCAST_DISABLE 1	Broadcast mode is disabled (default)
#define ADI_ADRV9001_PRE_MCS_BROADCAST_DISABLE 0	Broadcast mode is enabled

Users will also need to develop a way allowing SPI writes to multiples devices at the same time in their HAL. The 'chip select' SPI line will need to be triggered for all devices to allow for broadcast mode to operate.

SPI DATA TRANSFER PROTOCOL

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

There are two phases to a communication cycle. Phase 1 is the control cycle, which is the writing of a control word into the device. The control word provides the serial port controller with information regarding the data field transfer cycle, which is Phase 2 of the communication cycle. The Phase 1 control field defines whether the upcoming data transfer is read or write. It also defines the register address being accessed.

Phase 1 Instruction Format

The 16-bit control field contains the information shown in Table 20.

Table 20. SPI Control Field

MSB	D14:D0
R/Wb	A<14:0>

SERIAL-PERIPHERAL INTERFACE (SPI)

R/Wb - 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 zero 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 starting and internally generated addresses, are assumed to be valid, i.e., if an invalid address (undefined register) is accessed, the IO operation continues as if the address space is 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, choose a single-byte data transfer. In this mode, CSB goes active-low, the SCLK signal activates, and the address is transferred from the baseband processor to the device.

In LSB mode, the LSB of the address is the first bit transmitted from the baseband processor, followed by the next 14 bits in order from the next LSB to MSB. The next bit signifies whether the operation is read (set) or write (clear). If the operation is a write, the baseband processor transmits the next 8 bits LSB to MSB. If the operation is a read, the device transmits the next 8 bits LSB to MSB. Once the final bit is transferred, the data lines return to their idle state, and the CSB line is driven high to end the communication session.

In MSB mode, the first bit transmitted is the R/Wb bit that determines if the operation is a read (set) or write (clear). The MSB of the address is the next bit transmitted from the baseband processor, followed by the remaining 14 bits in order from the next MSB to LSB. If the operation is a write, the baseband processor transmits the next 8 bits MSB to LSB. If the operation is a read, the device transmits the next 8 bits MSB to LSB. Once the final bit is transferred, the data lines return to their idle state, and the CSB line is driven high to end the communication session.

Multibyte Data Transfer

When enSpiStreaming = 1, a multibyte data transfer is allowed. In this mode, data transfers across the bus are possible as long as the CSB pin is low. The autoIncAddrUp controls how the address changes for subsequent writes or reads. When autoIncAddrUp = 1, the address increments from the starting address for each subsequent data transfer until CSB is driven high. If the last register address is reached, the next address accessed is 0x000. When autoIncAddrUp = 0, the address decrements from the starting address for each subsequent data transfer. If address 0x000 is reached, the next address accessed is the last register location defined in the register map. The register address 0x000 sets up the SPI as well as functionality to soft reset the device. Uncontrolled data written to the register address 0x000 can cause SPI misconfigurations or can reset the device. It is strongly recommended to control any data transfer using the multiByte data feature so that 0x000 is only written once at start-up.

For multibyte data transfers in LSB mode, the LSB of the address is the first bit transmitted from the baseband processor, followed by the next 14 bits in order from the next LSB to MSB. The next bit signifies whether the operation is read (set) or write (clear). If the operation is a write, the baseband processor transmits the next 8 bits LSB to MSB. After the MSB is received, the address increments or decrements based on the autoIncAddrUp parameter. The baseband processor then continues to transfer data in 8-bit words, LSB to MSB, until the operation is terminated by CSB being driven high. If the operation is a read, the device transmits the next 8 bits LSB to MSB. It then changes the address and continues to transfer data in 8-bit words, LSB to MSB, until the operation is terminated by CSB being driven high.

For multibyte data transfers in MSB mode, the same process is followed except the first bit transferred indicates whether the operation is read (set) or write (clear). The starting address is then transmitted by the baseband processor, MSB to LSB, followed by the data transfer, MSB to LSB. The autoIncAddrUp parameter still controls the address increment or decrement.

Using the multibyte data transfer mode provides limited benefit because most registers in the device are not consecutive. Users determine if multibyte data transfer enhances device control in their end application compared to the single command format.

Example: LSB-First Multibyte Transfer, Autoincrementing Address

To complete a 4-byte write starting at register address 0x02A and ending with register 0x02D in LSB-first format, follow these instructions when programming the controller:

- Make sure that fourWireMode = 1 - the device is configured to work with the 4-wire interface.
- Make sure that MSBFirst = 0 - SPI operates in LSB first mode.
- Make sure that autoIncAddrUp = 1 - the address pointer automatically increments.
- Make sure that enSpiStreaming = 1 - a multibyte data transfer is allowed.

SERIAL-PERIPHERAL INTERFACE (SPI)

- Force the CSB line low and keep it low until the last byte is transferred.
- Send the instruction word 0101 0100 0000 000_0 (the last 0 indicates a write operation) to select 0x02A as the starting address.
- Use the next 32 clock cycles to send the data to be written to the registers, LSB to MSB, for each 8-bit word.
- Make sure the CSB line is driven high after the last bit is sent to 0x02D to end the data transfer.

Example: MSB-First Multibyte Transfer, Autodecrementing Address

To complete a 4-byte write starting at register address 0x02A and ending with register 0x027 in MSB-first format, follow these instructions when programming the controller:

- Make sure that fourWireMode = 1 - the device is configured to work with the 4-wire interface.
- Make sure that MSBFirst = 1 - SPI operates in MSB first mode.
- Make sure that autoIncAddrUp = 0 - the address pointer automatically decrements.
- Make sure that enSpiStreaming = 1 - a multibyte data transfer is allowed.
- Force the CSB line low and keep it low until the last byte is transferred.
- Send the instruction word 0_000 0000 0010 1010 (the first 0 indicates a write operation) to select 0x02A as the starting address.
- Use the next 32 clock cycles to send the data to be written to the registers, MSB to LSB, for each 8-bit word.
- Make sure the CSB line is driven high after the last bit is sent to 0x027 to end the data transfer.

TIMING DIAGRAMS

Figure 28 and Figure 29 show the SPI bus waveforms for a single-register write operation and a single-register read operation, respectively. In Figure 28, the value 0x55 is written to register 0x00A. In Figure 29, register 0x00A is read, and the value returned by the device is 0x55. If the same operations are performed with a 3-wire bus, the SDO line in Figure 28 eliminates, and the SDIO and SDO lines in Figure 29 combine on the SDIO line. Note that both operations use the MSB-first mode, and all data is latched on the rising edge of the SCLK signal.

Register 0x00A is not user accessible. The SPI write and read operations in Figure 28 and Figure 29 are only for the SPI timing diagram demonstration. Use the scratch register 0x009 for the SPI read/write test.

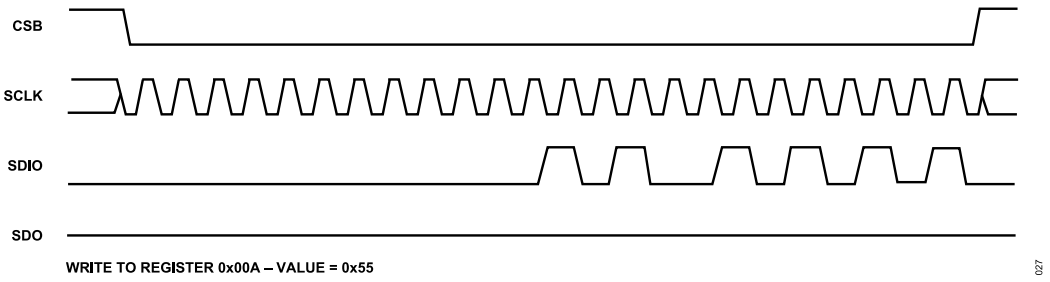


Figure 28. Nominal Timing Diagram, SPI Write Operation

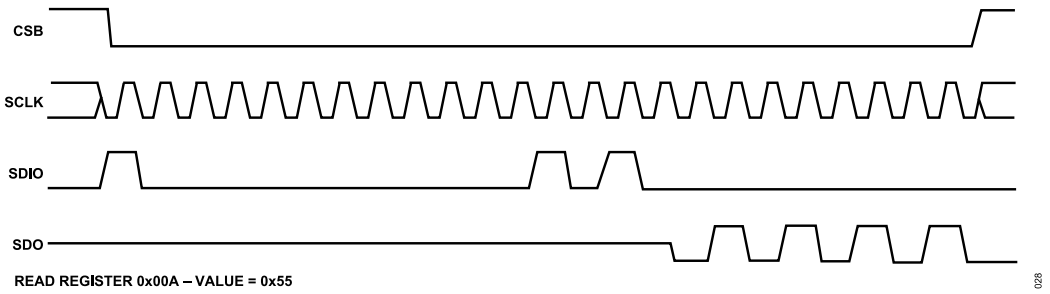


Figure 29. Nominal Timing Diagram, SPI Read Operation

Table 21 lists the timing specifications for the SPI bus. Figure 30 shows the relationship between these parameters. The figure shows a 3-wire SPI bus timing diagram with the device returning a value of 0xD4 from a test register and timing parameters marked. Note: This is a single read operation. So, the figure does not show the bus-ready parameter after the data is driven from the device (t_{HZS}).

SERIAL-PERIPHERAL INTERFACE (SPI)

Table 21. SPI Bus Timing Constraint Values

Parameter	Min	Typical	Max	Description
t_{CP}	28 ns			SCLK period, 3-wire mode
	22 ns			SCLK period, 4-wire mode
t_{MP}	10 ns			SCLK pulse width
t_{SC}	3 ns			CSB setup time to first SCLK rising edge
t_{HC}	0 ns			Last SCLK falling edge to CSB hold
t_S	2 ns			SDIO data input setup time to SCLK
t_H	0 ns			SDIO data input hold time to SCLK
t_{CO}	3 ns		15 ns	SCLK falling edge to output data delay (3-wire mode)
	3 ns		10 ns	SCLK falling edge to output data delay (4-wire mode)
t_{HZM}	t_H		t_{CO} (max)	Bus turnaround time after baseband processor drives the last address bit
t_{HZS}	0 ns		t_{CO} (max)	Bus turnaround time after device drives the last data bit

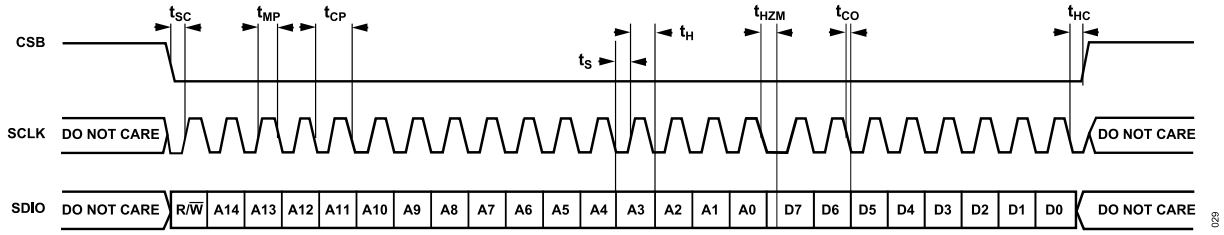


Figure 30. 3-Wire SPI Timing with Parameter Labels, SPI Read

Note: on the ADRV9001 (SPI sub device), SPI data in is read on the clock rising edge while SPI data out is written on the clock falling edge. The SPI main device setup and hold times should account for this SPI data format.

SPI TEST

The ADRV9001 has a scratch SPI register 0x009 for the SPI read/write validation. The following procedures quickly check the SPI function before the relative BBIC drivers are ready:

- Power on the ADRV9001 properly.
- Toggle the reset pin to reset the ADRV9001
- Write register 0x0 with value 0x3C to set the ADRV9001 SPI to the 4-wire mode, or with value 0x24 to set the ADRV9001 SPI to the 3-wire mode.
- Write any value to scratch register 0x009, then read register 0x009 to validate if the read value is the same as the write value.

Use the oscilloscope to probe the SPI bus signal and check if the SPI controller follows the timing diagrams in Figure 28 and Figure 29 when the above SPI validation cannot pass.

Also use the API **adi_adrv9001_spi_Verify()** to validate the SPI after the **adi_adrv9001_spi_Configure()** is set if the BBIC has the available drivers.

SPI MAIN

The ADRV9001 contains an SPI) Main digital block, which can be utilized through the GPIO (either digital or analogue) pins to control external components (for example, a PLL). This is separate from the SPI Sub functionality previously discussed. Table 22 shows the SPI pin mapping on the GPIO pin banks. The MCLK, MOSI, and MISO pins are currently fixed in assignment, and while the SPI Main functionality is enabled, these pins cannot be assigned to any other functions.

Table 22. SPI Main Pin Mapping

Pin	MCS0	MCS1 [1]	MCS2 [1]	MCS3 [1]	MCLK [1]	MISO[2]	MOSI
GPIO0	✓						
GPIO1		✓					
GPIO2			✓				

SERIAL-PERIPHERAL INTERFACE (SPI)

Table 22. SPI Main Pin Mapping (Continued)

Pin	MCS0	MCS1 [1]	MCS2 [1]	MCS3 [1]	MCLK [1]	MISO[2]	MOSI
GPIO3				✓			
GPIO4	✓						
GPIO5		✓					
GPIO6			✓				
GPIO7				✓			
GPIO8	✓						
GPIO9					✓		
GPIO10						✓	
GPIO11							✓

[1] While multiple pin options exist when assigning a chip-select pin, only a single SPI sub device can be used at a time. Currently, MCS0 can be assigned to any available GPIO pin.

[2] Since SPI read functionality is currently not implemented in the SPI Main block, this pin has no functionality. However, it is still reserved when SPI Main is enabled.

The SPI Main functionality is currently incapable of performing read operations. The write operation is triggered using the **adi_adrv9001_spi_Master_Configure()**.

The SPI Main clock is divided down from DEV_CLK by integers ranging from 2 to 64, enabling functionality at SPI clock rates as low as DEV_CLK/64 and as high as DEV_CLK/2. The clock idles at logic low (CPOL = 0), and the data is shifted out on the MOSI pin on the falling clock edge (CPHA = 0), thus operating in SPI Mode 0. This is currently the only mode of operation. The data is sent MSB first.

Following is an example of the SPI main configuration struct, followed by a function call that triggers the SPI transaction:

```
adi_adrv9001_spiMasterConfig_t spiMasterConfig_59 = {
    .numBytes = 2,
    .baudRateDiv = 31,
    .transactionBytes = 2,
    .assertionMode = ADI_ADRV9001_SPI_MASTER_CS_ASSERTION_PER_BYTE,
    .spiSlaveDevicesConnected = ADI_ADRV9001_SPI_MASTER_SLAVE_SINGLE,
    .csSource = ADI_ADRV9001_SPI_MASTER_CS_SOURCE_GPIO_DIGITAL,
    .pin = ADI_ADRV9001_GPIO_DIGITAL_04,
    .triggerSource = ADI_ADRV9001_SPI_MASTER_TRIGGER_SOURCE_ARM,
    .wakeupTimer_us = 0,
    .spiData = { 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }};
error_code = adi_adrv9001_spi_Master_Configure(adrv9001Device_0, &spiMasterConfig_59);
```

Table 23 explains the meaning of each variable.

Table 23. SPI Main Configure Struct Definition

Type	Name	Description
uint8_t	numBytes	Number of bytes to write. Valid range is 1 to 30.
uint8_t	baudRateDiv	SPI baudRate = DEV_CLK_IN / ((baudRateDiv+1) * 2). Valid range is 0 to 31.
uint8_t	transactionBytes	Number of bytes per CS assertion. Valid range is 1 to 30.
adi_adrv9001_spiMaster_TransferMode_e	transferMode	CS transfer during frame duration.
adi_adrv9001_spiMasterSlaveDevices_e	spiSlaveDevicesConnected	Selects the number of devices. Currently, only a single device is supported.
adi_adrv9001_spiMasterSource_e	csSource	GPIO type for all SPI pins. Analog or Digital.
adi_adrv9001_GpioPin_e	pin	CS pin - other SPI signals have dedicated pins (see Table 22).

SERIAL-PERIPHERAL INTERFACE (SPI)

Table 23. SPI Main Configure Struct Definition (Continued)

Type	Name	Description
adi_adrv9001_spiMasterTriggerSource_e	triggerSource	SPI master transaction trigger source.
uint32_t	wakeupTimer_us	Monitor mode for early wakeup timer for SPI Main transaction. Only available if triggerSource = ADI_ADRV9001_SPI_MASTER_TRIGGER_SOURCE_MONITOR.
uint8_t	spiData[SPI_MASTER_TOTAL_BYTES_MAX]	Bytes to be written to the SPI sub device. By default populated with zeros. *Note: SPI Main makes no distinction between op-codes, addresses, and data; these are all treated as one in the spiData array.*

DATA INTERFACE

GENERAL DESCRIPTION

This document defines the synchronous-serial interface (SSI), which transfers the data between the ADRV9001 and a baseband processor. The ADRV9001 SSI consists of two receive channels and two transmit channels. The channels are independent and can be configured either as complementary metal oxide semiconductors (CMOS) signals (CSSI) for applications that have narrow RF signal bandwidths and low data-rate, or as low voltage differential signaling (LVDS) signals (LSSI) for applications that require high-speed, low noise, and longer distance data transfer.

The CSSI supports the following two modes of operation and can be operated as either single data rate (SDR) or double data rate (DDR) data transfer, and the maximum clock frequency is 80 MHz:

- One-lane data mode, I/Q data, or other format data are serialized onto one single lane.
- Four lanes data mode, which is valid only when the ADRV9001 transmits or receives I/Q samples, and the I/Q samples are 16 bits wide. In the four-lane data mode, each sample is split into 8 bit blocks of data and sent over one data lane.

The LSSI also supports the following two modes of operation, and always operates in DDR data transfer. The maximum clock frequency is 491.52 MHz:

- I/Q in one lane (one-lane mode)
  - With I-Q data samples of 16 bits (total of 32 bits for each transfer)
- I/Q in separate lanes (two-lane mode)
  - With I and Q data samples of 16 bits
  - With I and Q data samples of 12 bits

The ADRV9001 SSI has various and flexible work modes to support many kinds of system scenarios. Choose the appropriate work modes according to the interface sample/symbol rate and bit width. Table 24 lists the ADRV9001 SSI work modes and the maximum support I/Q sample rate.

Table 24. ADRV9001 SSI Work Modes

SSI Modes	Data Lanes Per Channel	Serialization Factor Per Data Lane	Maximum Data Lane Rate (MHz)	Maximum Clock Rate (MHz)	Maximum Sample Rate for I/Q (MHz)	Data Type
CSSI 1-Lane	1	32	80	80	2.5	SDR
CSSI 1-Lane	1	32	160	80	5	DDR
CSSI 1-Lane[1]	1	16/8/2	80-SDR/160-DDR	80	Not Applicable	SDR/DDR
CSSI 4-Lane	4	8	80	80	10	SDR
CSSI 4-Lane	4	8	160	80	20	DDR
LSSI 1-Lane[2]	1	32	983.04	491.52	30.72	DDR
LSSI 2-Lane	2	16	491.52	491.52	30.72	SDR
LSSI 2- Lane[2, 3]	2	12	368.64	368.64	30.72	SDR
LSSI 2-Lane	2	16	983.04	491.52	61.44	DDR
LSSI 2- Lane [2, 4]	2	12	737.28	368.64	61.44	DDR

[1] ADRV9001 data port transmit/receive data symbols, see the section CSSI Data Symbols Transmit and Receive.

[2] Currently not implemented.

[3] For user's LVDS data lane rate limitation applications, the receive samples are rounded from 16 bits to 12 bits. The transmit samples are extended from 12 bits to 16 bits.

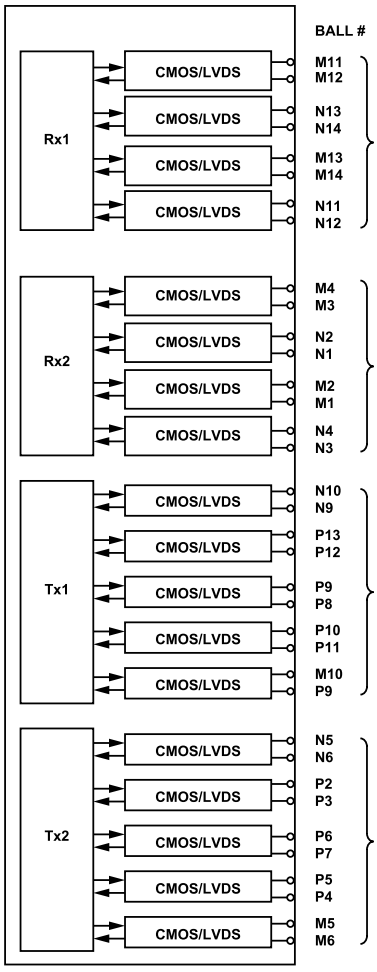
[4] For user's LVDS data lane rate limitation applications, the receive samples are rounded from 16 bits to 12 bits. The transmit samples are extended from 12 bits to 16 bits.

The following sections detail the signals that make up the SSI and their properties when configured for each mode.

ELECTRICAL SPECIFICATION

The ADRV9001 SSI can operate in the standard single-ended CMOS compatible mode, or LVDS compatible mode. Both CMOS SSI and LVDS SSI share the IO pads of the ADRV9001. Figure 31 shows the four channels with their corresponding inputs and outputs in the CMOS and LVDS modes.

DATA INTERFACE



	PIN	LVDS-SSI	CMOS-SSI 1-LANE	CMOS-SSI 4-LANE
Rx CHANNEL 1	M11	RX1_DCLK_OUT+	RX1_DCLK_OUT	RX1_DCLK_OUT
	M12	RX1_DCLK_OUT–		
	N13	RX1_STROBE_OUT+	RX1_STROBE_OUT	RX1_STROBE_OUT
	N14	RX1_STROBE_OUT–		
	M13	RX1_IDATA_OUT+		RX1_IDATA1_OUT
	M14	RX1_IDATA_OUT–	RX1_DATA_OUT	RX1_IDATA0_OUT
	N11	RX1_QDATA_OUT+		RX1_QDATA3_OUT
	N12	RX1_QDATA_OUT–		RX1_QDATA2_OUT
Rx CHANNEL 2	M4	RX2_DCLK_OUT+	RX2_DCLK_OUT	RX2_DCLK_OUT
	M3	RX2_DCLK_OUT–		
	N2	RX2_STROBE_OUT+	RX2_STROBE_OUT	RX2_STROBE_OUT
	N1	RX2_STROBE_OUT–		
	M2	RX2_IDATA_OUT+		RX2_IDATA1_OUT
	M1	RX2_IDATA_OUT–	RX2_DATA_OUT	RX2_IDATA0_OUT
	N4	RX2_QDATA_OUT+		RX2_QDATA3_OUT
	N3	RX2_QDATA_OUT–		RX2_QDATA2_OUT
Tx CHANNEL 1	N10	TX1_DCLK_IN+	TX1_DCLK_IN	TX1_DCLK_IN
	N9	TX1_DCLK_IN–		
	P13	TX1_STROBE_IN+	TX1_STROBE_IN	TX1_STROBE_IN
	P12	TX1_STROBE_IN–	TX1_DCLK_OUT	TX1_DCLK_OUT
	P9	TX1_IDATA_IN+		TX1_IDATA1_IN
	P8	TX1_IDATA_IN–	TX1_DATA_IN	TX1_IDATA0_IN
	P10	TX1_QDATA_IN+		TX1_QDATA3_IN
	P11	TX1_QDATA_IN–		TX1_QDATA2_IN
	M10	TX1_DCLK_OUT+	TX1_DCLK_OUT	TX1_DCLK_OUT
	M9	TX1_DCLK_OUT–	TX1_DCLK_OUT	TX1_DCLK_OUT
Tx CHANNEL 2	N5	TX2_DCLK_IN+	TX2_DCLK_IN	TX2_DCLK_IN
	N6	TX2_DCLK_IN–		
	P2	TX2_STROBE_IN+	TX2_STROBE_IN	TX2_STROBE_IN
	P3	TX2_STROBE_IN–	TX2_DCLK_OUT	TX2_DCLK_OUT
	P6	TX2_IDATA_IN+		TX2_IDATA1_IN
	P7	TX2_IDATA_IN–	TX2_DATA_IN	TX2_IDATA0_IN
	P5	TX2_QDATA_IN+		TX2_QDATA3_IN
	P4	TX2_QDATA_IN–		TX2_QDATA2_IN
	M5	TX2_DCLK_OUT+	TX2_DCLK_OUT	TX2_DCLK_OUT
	M6	TX2_DCLK_OUT–	TX2_DCLK_OUT	TX2_DCLK_OUT

030

Figure 31. ADRV9001 SSI I/Os Mapping

Table 25 shows the CMOS SSI electrical specifications. For optimal performance, keep the load capacitance driven by the CMOS outputs to a minimum. The CMOS output drive strength can be increased to compensate for load capacitance larger than 10 pF to increase the edge rate of the output signal during the transition period. The maximum load capacitance is 30 pF at an 80 MHz clock data rate.

In the LVDS mode, a 100 Ω differential termination resistor is required for each LVDS pair, which must be located as close to the LVDS receiver as possible. The ADRV9001 LVDS input circuit has an optional internal 100 Ω termination resistor, which is enabled by default for LSSI. The ADRV9001 LVDS output circuits do not have internal termination resistors, and the user is expected to implement the appropriate LVDS termination resistors in the LVDS receiver. The default LVDS output circuit produces 350 mV peak at 1.2 V common-mode level. The output swing level can be increased to 450 mV for longer trace (currently not supported). Table 26 shows the LVDS SSI electrical specifications.

It is recommended to closely match the trace lengths of SSI clock, strobe, data signals into each transmit or receive channel. The ADRV9001 SSI has configurable delay cells on the LVDS/CMOS input and output circuits, which allow small adjustments to the phase relationship between the strobe/data and clock. The adjustable phase delay cell increments in approximately 90 ps per step in the LVDS mode, and 170 ps per step in the CMOS mode, up to a maximum of seven steps.

Table 25. CSSI Electrical Specification

Symbol	Parameter	Min	Typ	Max	Units
VDIGIO_1P8	Interface power supply voltage	1.71	1.8	1.89	V
V_{IH}	Input voltage high	VDIGIO_1P8 × 0.65		VDIGIO_1P8 + 0.18	V
V_{IL}	Input voltage low	-0.3		VDIGIO_1P8 × 0.35	V
V_{OH}	Output-voltage high	VDIGIO_1P8 − 0.45		VDIGIO_1P8	V

DATA INTERFACE

Table 25. CSSI Electrical Specification (Continued)

Symbol	Parameter	Min	Typ	Max	Units
V_{OL}	Output-voltage low			0.45	V
f_{CLK}	Clock frequency			80	MHz
C_L at 80 MHz	Load capacitance supported for an 80 MHz clock waveform		10	30	pF

Table 26. LSSI Electrical Specification

Symbol	Parameter	Conditions	Min	Typ	Max	Units
VDIGIO_1P8	Interface power supply voltage		1.71	1.8	1.89	V
V_I	Input voltage range		825		1675	mV
V_{IDTH}	Input differential threshold		−100		+100	mV
R_{IN}	Receiver differential input impedance			100		Ω
V_{OH}	Output-voltage high	R_{LOAD} = 100 Ω ± 1%			1390	mV
V_{OL}	Output-voltage low	R_{LOAD} = 100 Ω ± 1%	1000			mV
$\|V_{OD}\|$	Output differential voltage	R_{LOAD} = 100 Ω ± 1%		360		mV
V_{OS}	Output offset voltage	R_{LOAD} = 100 Ω ± 1%	1150	1200	1250	mV
R_O	Output impedance, single-ended		80	100	120	Ω
I_{SA}, I_{SB}	Output current	Driver shorted to ground			17	mA
I_{SAB}	Output current	Drivers shorted together			4.1	mA
	Clock signal duty cycle	500 MHz	45	50	55	%
T_R,T_F	Output Rise/Fall Time	300 mVp swing		0.371		nsec

CMOS SYNCHRONOUS-SERIAL INTERFACE (CMOS-SSI)

One-Lane Mode CSSI

Receive CSSI

The one-lane mode receives CSSI of each channel (Rx1 and Rx2). These are a 3-wire digital interface consisting of:

- RX_DCLK_OUT: output clock synchronizing data and strobe output signals.
- RX_STROBE_OUT: output signal indicating the first bit of the serial data sample.
- RX_DATA_OUT: output serial data stream.

The I and Q samples are serialized starting with configurable I or Q first and MSB or LSB first. Figure 32 shows the receive CSSI (Rx1 and Rx2) for a 16-bit I/Q data sample with I sample and MSB first configuration.

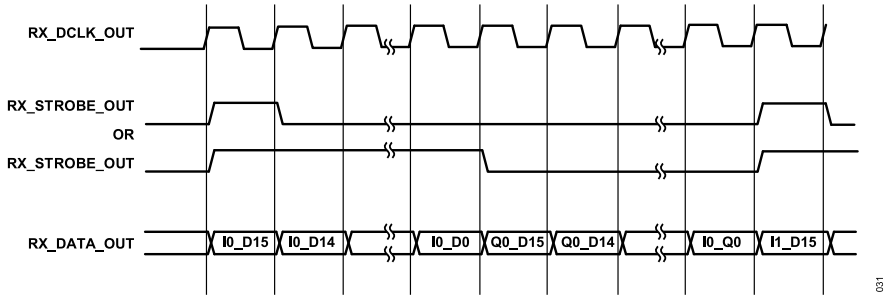


Figure 32. Receive CSSI Timing for 16-Bit I/Q Data Sample (I and MSB First)

The RX_STROBE_OUT signal is aligned with the first bit of the serialized data (I and Q), and can be configured to be high:

- For one clock cycle at the start of I and Q sample transmit. For a 16-bit data sample, RX_STROBE is high for one clock cycle and low for 31 clock cycles.

DATA INTERFACE

- For I data duration and low for Q data duration. For a 16-bit data sample, RX_STROBE is high for 16 clock cycles (I data sample) and low for 16 clock cycles (Q data sample).

Transmit CSSI

The one-lane mode transmits the CSSI of each channel (Tx1 and Tx2). This is a 4-wire digital interface consisting of:

- TX_DCLK_IN: input clock synchronized to the data and strobe inputs.
- TX_STROBE_IN: input signal indicating the first bit of the serial data sample.
- TX_DATA_IN: input serial data stream.
- TX_DCLK_OUT: optional output reference clock provided to the baseband processor to generate all the signals shown in Figure 33. The baseband processor can also use RX_DCLK_OUT as the reference clock when its clock rate is equal with the Transmit SSI clock rate.

The I and Q samples can be deserialized, starting with configurable I or Q first and MSB or LSB first. Figure 33 shows the transmit CSSI (Tx1 and Tx2) for a 16-bit I/Q data sample with I sample and MSB first configuration.

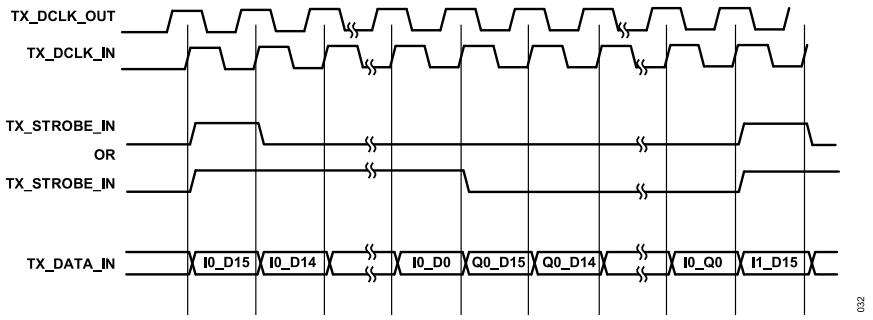


Figure 33. Transmit CSSI Timing for 16-Bit I/Q Data Sample (I and MSB First)

The TX_STROBE_IN signal is aligned with the first bit of the serialized data (I and Q), and can be configured to be high:

- For one clock cycle at the start of I and Q sample transmit. For a 16-bit data sample, the TX_STROBE is high for one clock cycle and low for 31 clock cycles.
- For I data duration and low for Q data duration. For a 16-bit data sample, TX_STROBE is high for 16 clock cycles (I data sample) and low for 16 clock cycles (Q data sample).

CSSI Data Symbols Transmit and Receive

The previous sections describe data transfer with I/Q format with 16-bit width. When the ADRV9001 internal modulation/demodulation is enabled (see the Transmitter Signal Chain and Receiver Demodulator sections), the data transfer between the ADRV9001 and baseband processor is 2 bits or 16 bits I only data (denoted as symbol to differentiate with I/Q complex samples). In a symbol format mode, raw data is transferred through this interface using different data sizes. The CSSI interface supports three additional data formats:

- 2 bits of data
- 8 bits of data
- 16 bits of data

Two bits of data can be transferred over a CSSI with an 8-bit data format with six dummy bits. The clock and strobe behavior are similar to the I/Q format described in the previous sections.

Figure 34 shows the receive CSSI (Rx) for 2-bit data symbols.

DATA INTERFACE

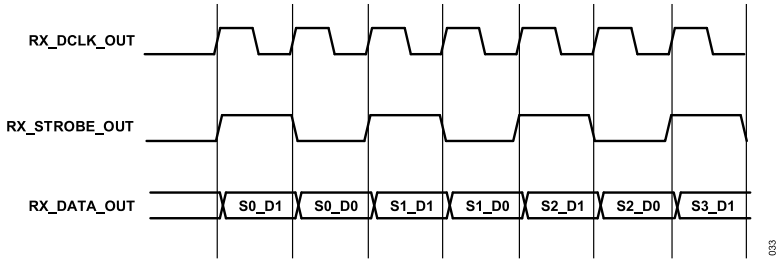


Figure 34. Receive CSSI Timing for 2-Bit Symbols (MSB First)

Figure 35 shows the transmit CSSI (Tx) for 2-bit data symbols.

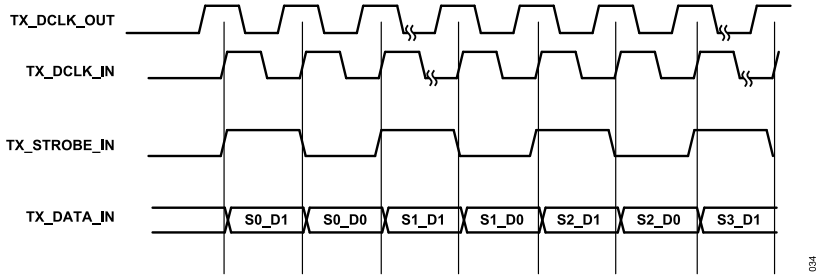


Figure 35. Transmit CSSI Timing for 2-Bit Symbols (MSB First)

Figure 36 shows the receive CSSI (Rx) for 8-bit data symbols.

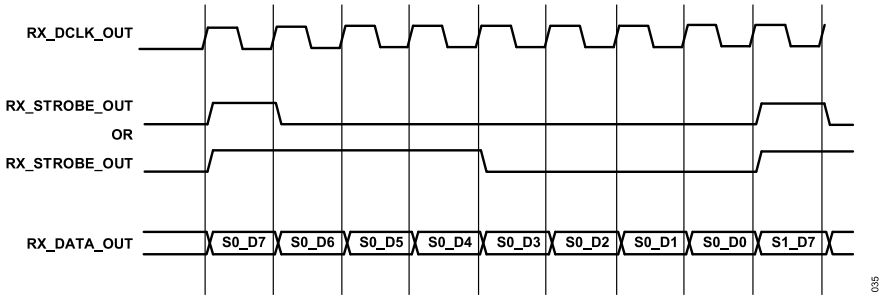


Figure 36. Receive CSSI Timing for 8-Bit Symbols (MSB First)

Figure 37 shows the transmit CSSI (Tx) for 8-bit data symbols.

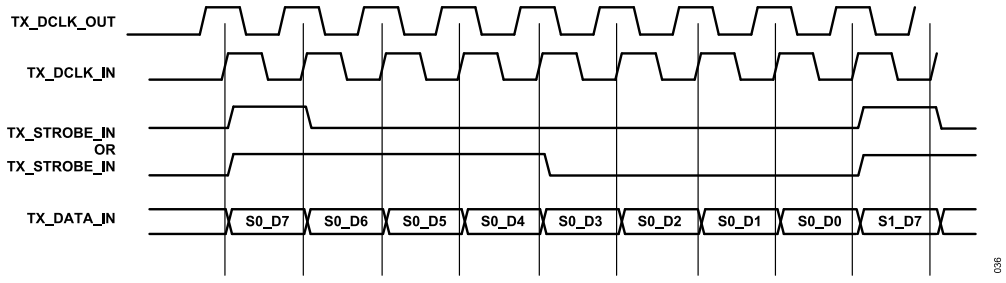


Figure 37. Transmit CSSI Timing for 8-Bit Symbols (MSB First)

Figure 38 shows the receive CSSI (Rx) for 16-bit data symbols.

DATA INTERFACE

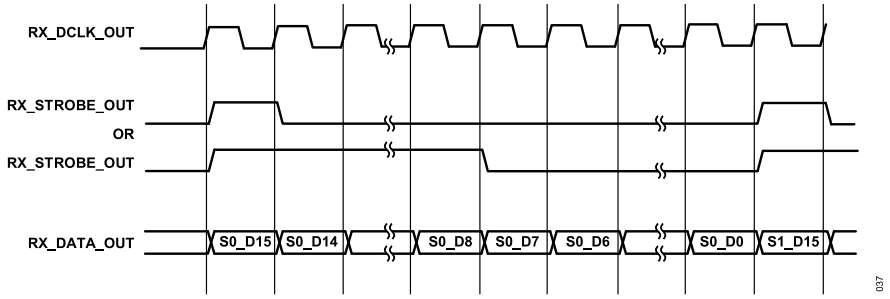


Figure 38. Receive CSSI Timing for 16-Bit Symbols (MSB First)

Figure 39 shows the transmit CSSI (Tx) for 16-bit data symbols.

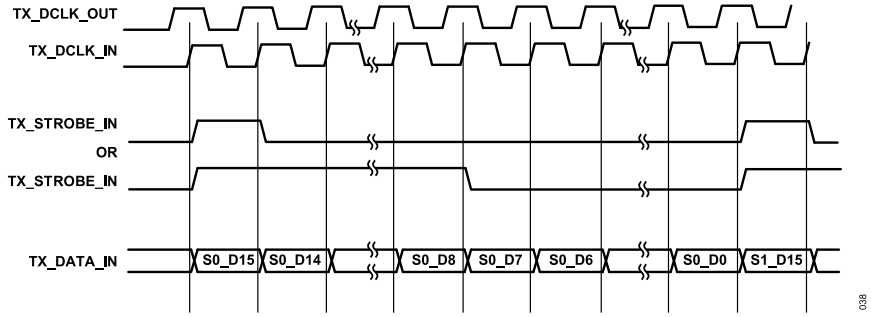


Figure 39. Transmit CSSI Timing for 16-Bit Symbols (MSB First)

Receive CSSI with Two, Four, and Eight Times Data Clock Rates

The ADRV9001 receive CSSI supports two, four, or eight times of the data clock rate for some applications.

Figure 40, Figure 41, and Figure 42 show the receive CSSI (Rx1 and Rx2) for 16-bit I/Q data sample with two, four, and eight times clock rates. The strobe pulse validates the start of the 32-bit I and Q samples, and the remaining data bits are ignored.

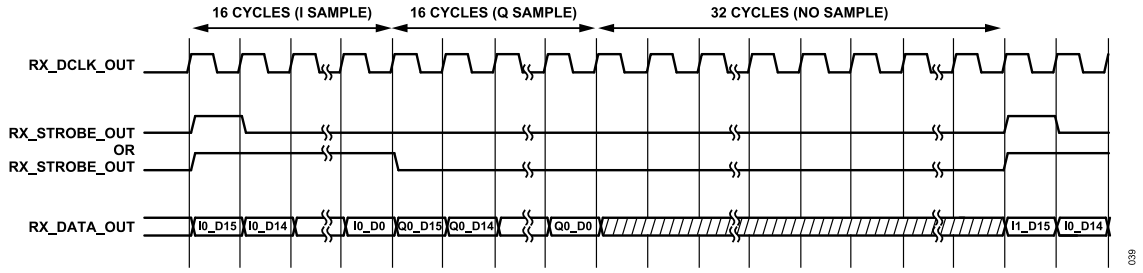


Figure 40. Receive CSSI Timing with 2 Times Data Clock Rate for 16-Bit I/Q Data Sample (MSB First), 32 Cycles

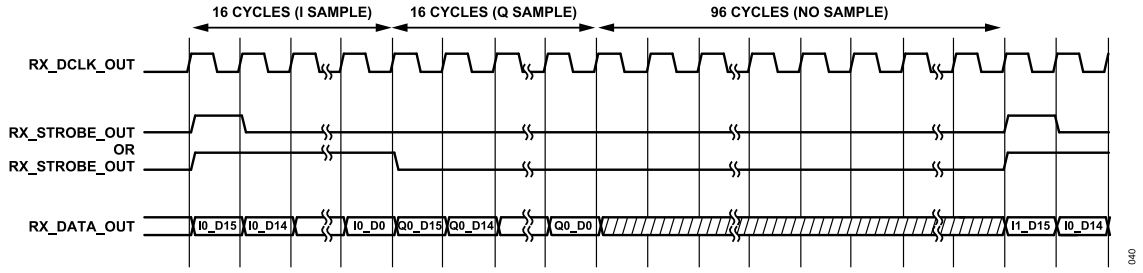


Figure 41. Receive CSSI Timing with 4 Times Data Clock Rate for 16-Bit I/Q Data Sample (MSB First), 96 Cycles

DATA INTERFACE

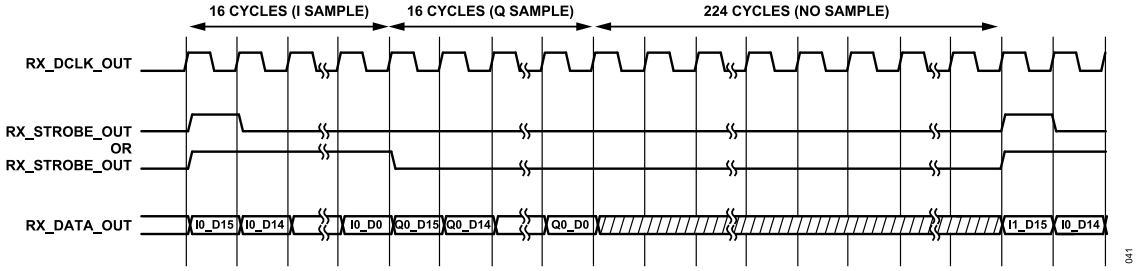


Figure 42. Receive CSSI Timing with 8 Times Data Clock Rate for 16-Bit I/Q Data Sample (MSB First), 224 Cycles

Figure 43, Figure 44, and Figure 45 show the receive CSSI (Rx1 and Rx2) in the frequency deviation mode with 16-bit data symbol with two, four, and eight times clock rates. The strobe pulse validates the start of the 16-bit data symbol, and the remaining data bits are ignored.

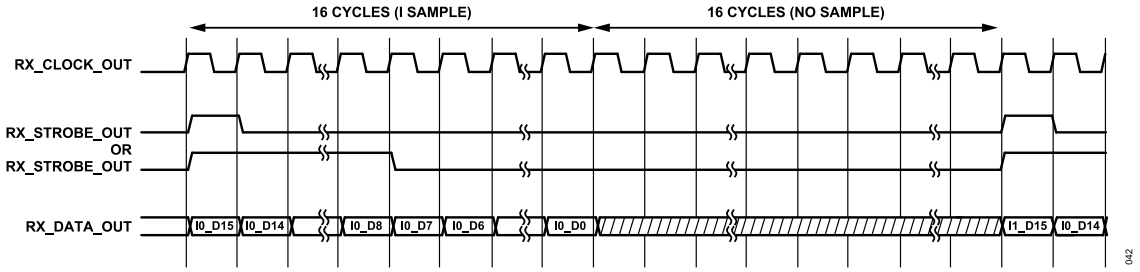


Figure 43. Receive CSSI Timing with 2 Times Data Clock Rate for 16-Bit Data Symbol (MSB First)

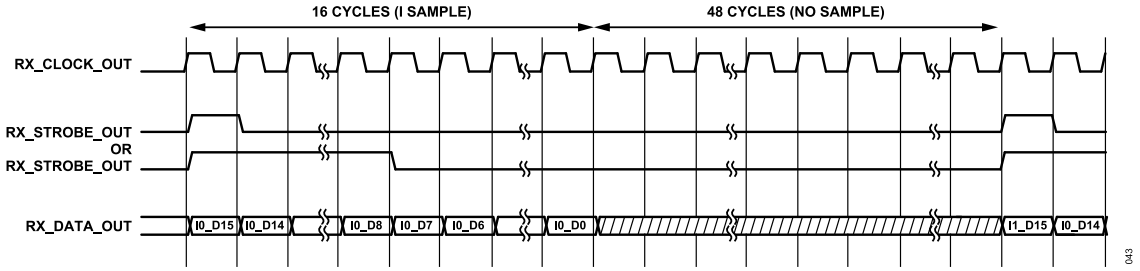


Figure 44. CSSI Receive Timing with 4 Times Data Clock Rate for 16-Bit Data Symbol (MSB First)

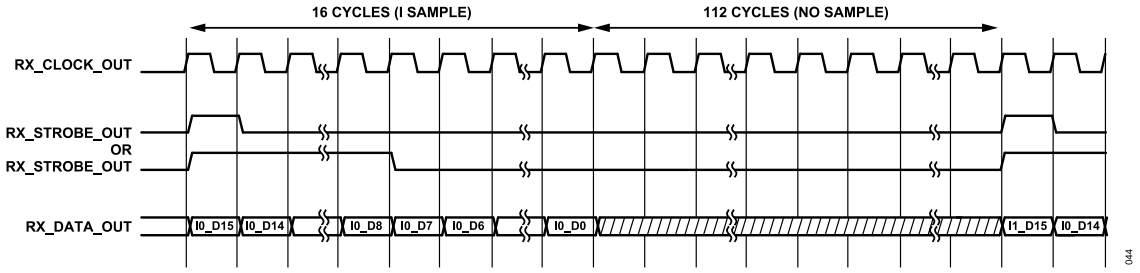


Figure 45. Receive CSSI Timing with 8 Times Data Clock Rate for 16-Bit Data Symbol (MSB First)

Four-Lane Mode CSSI

The four-lane mode receives the CSSI of each channel (Rx1 and Rx2). These are a 6-wire digital interface consisting of:

- RX_DCLK_OUT: output clock synchronous data and strobe output signals.
- RX_STROBE_OUT: output signal indicating the first bit of the serial data sample.
- RX_IDATA0_OUT: output serial data stream of I sample low byte.
- RX_IDATA1_OUT: output serial data stream of I sample high byte.
- RX_QDATA2_OUT: output serial data stream of Q sample low byte.

DATA INTERFACE

- RX_QDATA3_OUT: output serial data stream of Q sample high byte.

Figure 46 shows the receive CSSI (Rx1 and Rx2) for a four-lane format with MSB first configuration.

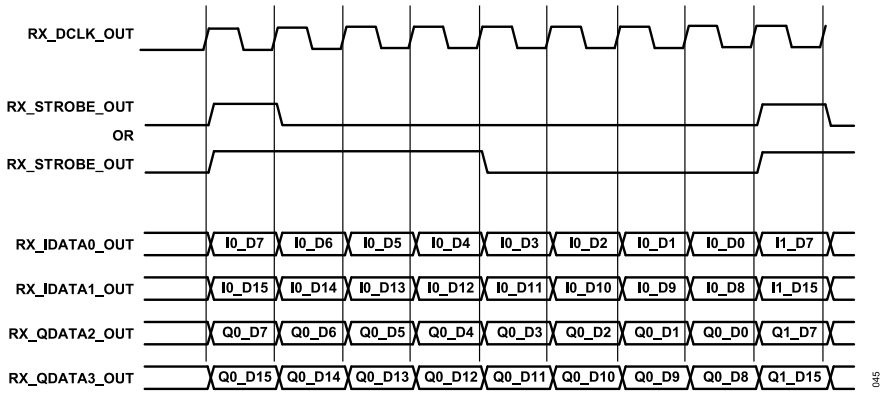


Figure 46. Four-Lane Mode Receive CSSI Timing for 16-Bit I/Q Data Sample (MSB First)

The four-lane mode CSSI transmits the interface of each channel (Tx1 and Tx2). This is a 7-wire digital interface consisting of:

- TX_DCLK_IN: input clock synchronized to the data and strobe inputs.
- TX_STROBE_IN: input signal indicating the first bit of the serial data sample.
- TX_IDATA0_IN: input serial data stream of I sample low byte.
- TX_IDATA1_IN: input serial data stream of I sample high byte.
- TX_QDATA2_IN: input serial data stream of Q sample low byte.
- TX_QDATA3_IN: input serial data stream of Q sample high byte.
- TX_DCLK_OUT: optional output reference clock provided to the baseband processor to generate all the above signals. The baseband processor can also use RX_DCLK_OUT as the reference clock when its clock rate is equal to the transmit SSI clock rate.

Figure 47 shows the transmit CSSI (Tx1 and Tx2) for a four-lane format with MSB first configuration.

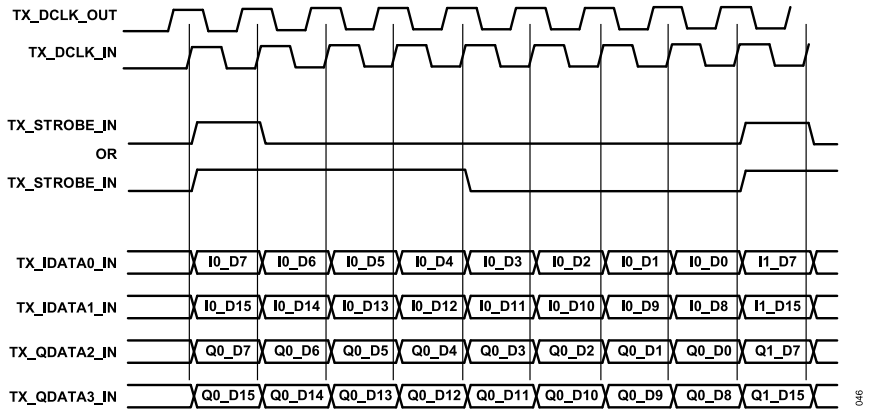


Figure 47. Four-Lane Mode Transmit CSSI Timing for 16-Bit I/Q Data Sample (MSB First)

Transmit and Receive CSSI Using DDR Clock

Transmit and receive CSSI can be operated in either SDR or DDR data transfer.

Figure 48 shows the Rx CMOS SSI with DDR clock in relation to the strobe/data. Each edge of the clock (positive and negative) corresponds to a data sample. The RX DDR clock can be generated in phase with the data/strobe or delayed by a quarter cycle of the clock period. The optional delayed clock eases the timing interface of the baseband processor to meet the setup/hold on the baseband processor.

DATA INTERFACE

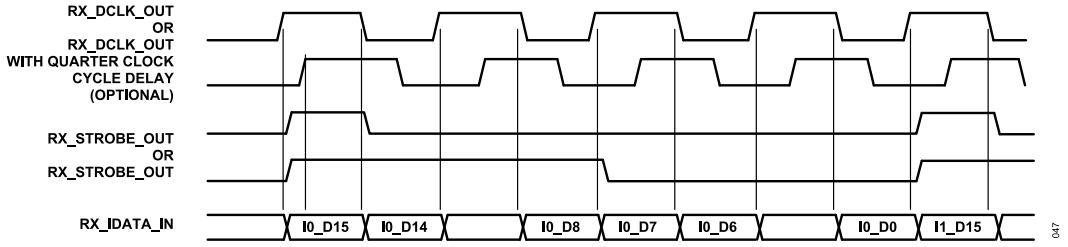


Figure 48. Receive CSSI DDR Clock Relation with Strobe/Data

Figure 49 shows the transmit CMOS SSI with DDR clock in relation to the strobe/data, with respect to the ADRV9001. Each edge of the clock (positive and negative) samples the corresponding strobe/data sample based on the interface setup/hold timing.

When the baseband processor drives out the transmit SSI clock, strobe, and data to the ADRV9001, the output DDR clock can either be in-phase with the strobe/data or delayed by a quarter cycle of the clock period. Regardless of the choice, the relationship between the transmit DDR clock and strobe/data must meet the ADRV9001 setup and hold timing specification.

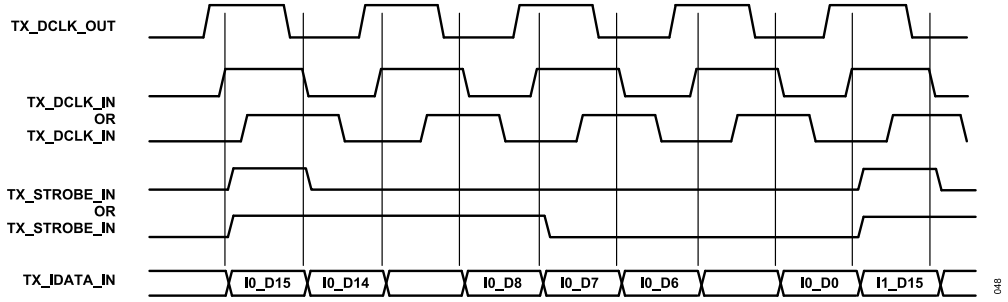


Figure 49. Transmit CSSI DDR Clock Relation with Strobe/Data

Figure 50 and Figure 51 show the timing diagram examples for four-lane mode receive and transmit CSSI with DDR clock and 16-bit I/Q samples.

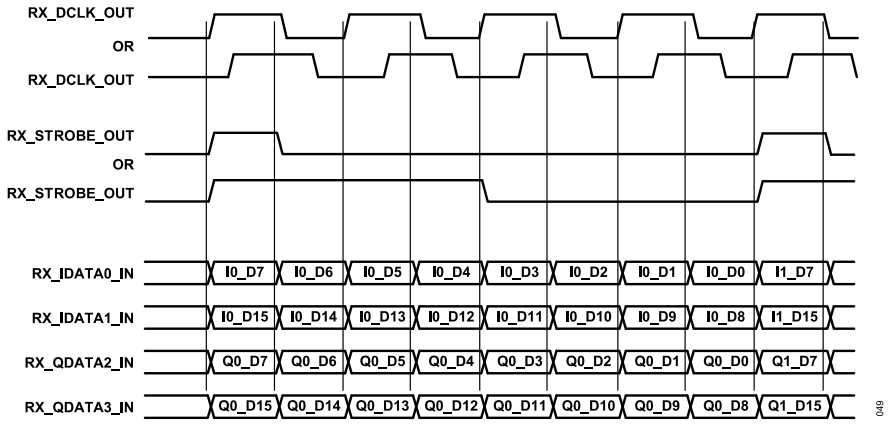


Figure 50. Four-Lane Mode Receive CSSI DDR Timing for 16-Bit I/Q Data Sample

DATA INTERFACE

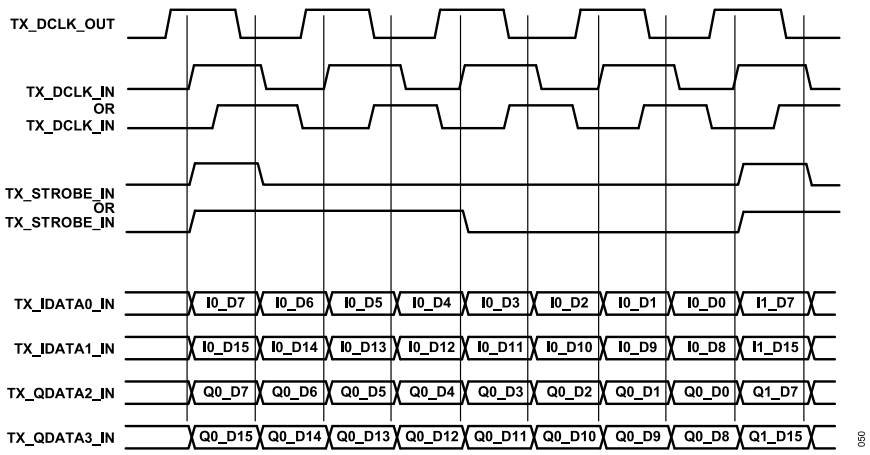


Figure 51. Four-Lane Mode Transmit CSSI DDR Timing for 16-Bit I/Q Data Sample

LVDS SYNCHRONOUS-SERIAL INTERFACE (LVDS-SSI)

Receive LSSI

The LSSI receive interface of each channel (Rx1 and Rx2) is an 8-wire LVDS interface consisting of:

- RX_DCLK_OUT (±): differential output clock.
- RX_STROBE_OUT (±): differential output signal indicating the first bit of the serial data sample.
- RX_IDATA_OUT (±): differential output serial I data stream.
- RX_QDATA_OUT (±): differential output serial Q data stream.

Receive LSSI with Separate Lanes for I and Q

Figure 52 shows the receive LSSI (Rx1 and Rx2) for a 16-bit I/Q data sample with MSB first configuration. Figure 53 shows the receive LSSI for a 12-bit I/Q data sample.

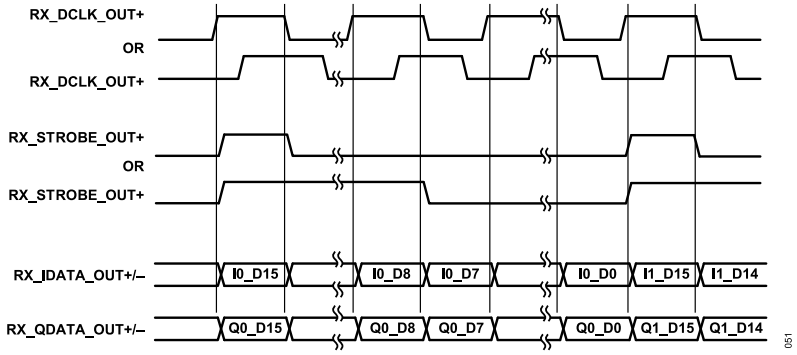


Figure 52. Receive LSSI Timing for 16-Bit I/Q Data Sample over Two Lanes (MSB First)

DATA INTERFACE

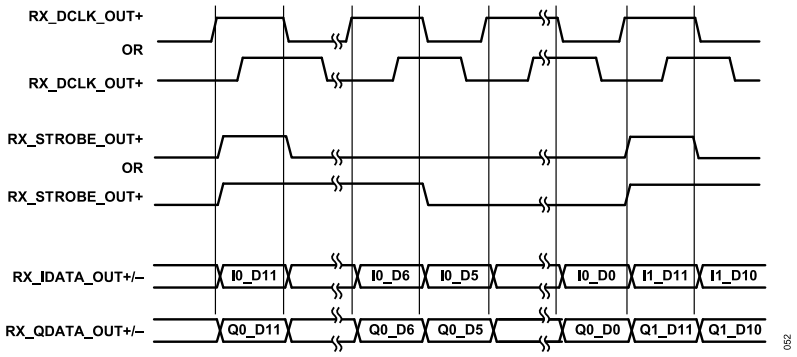


Figure 53. Receive LSSI Timing for 12-Bit I/Q Data Sample over Two Lanes (MSB First)

The RX_STROBE signal is aligned with the first bit of the serialized data (I and Q), and can be configured to be high:

- For a half-clock cycle at the start of I and Q sample transmit. For a 16-bit data sample, RX_STROBE is high for a half-clock cycle and low for a half and 15 clock cycles. For a 12-bit data sample, RX_STROBE is high for a half-clock cycle and low for a half and 11 clock cycles.
- For half of I and Q data duration. For a 16-bit data sample, the RX_STROBE is high for 4 clock cycles, and low for 4 clock cycles (Q data sample). For a 12bit data sample, the RX_STROBE is high for 3 clock cycles and low for 3 clock cycles.

In the 12-bit I/Q mode, 16-bit samples from the receive datapath are cut to 12 bits for LSSI, a configurable option to choose the 12-bit from the LSB or MSB of the 16-bit sample data.

Receive LSSI with One-Lane for I and Q

In this mode, only one-lane is used to transfer I and Q data samples. The I/Q data bits are serialized with configurable I or Q first and MSB or LSB first. The strobe signal can be configured to high for a half-clock cycle to indicate the start of I and Q symbols or for half of the I and Q data duration to distinguish between I and Q data.

Figure 54 shows that the one-lane receives LSSI (Rx1 and Rx2) for a 16-bit I/Q data sample with the I sample and MSB first configuration.

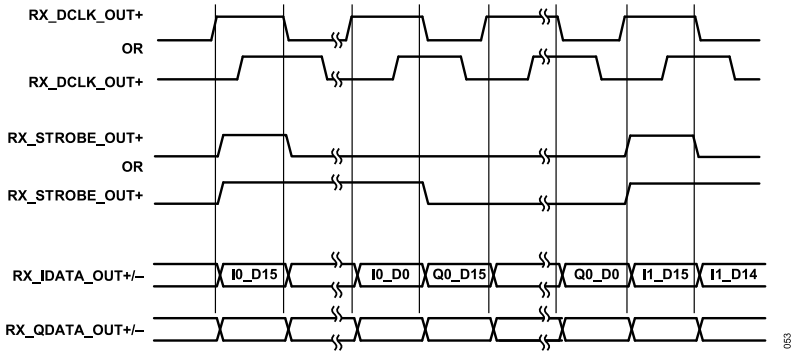


Figure 54. Receive LSSI Timing for 16-Bit I/Q Data Sample over One-Lane (I and MSB First)

Transmit LSSI

The transmit LSSI of each channel (Tx1 and Tx2) is an 8-wire digital interface consisting of:

- TX_DCLK_IN (±): is a differential input clock synchronized to the data and strobe inputs.
- TX_STROBE_IN (±): is a differential input signal indicating the first bit of the serial data sample.
- TX_IDATA_IN (±): is a differential input serial I data stream.
- TX_QDATA_IN (±): is a differential input serial Q data stream.

An additional port can used as a reference clock for the baseband processor to generate the transmit LSSI clock, strobe, and data signal. Use RX1_DCLK_OUT or RX2_DCLK_OUT as a reference clock if these clock frequencies are equal to the TX clock frequency.

DATA INTERFACE

An optional LVDS port (alternative function of Digital GPIO) can also be configured as an output LVDS pad used as a reference clock TX_DCLK_OUT (±) for the baseband processor. Use TX_DCLK_OUT to generate the LSSI clock, strobe, and data signal.

Transmit LSSI with Separate Lanes for I and Q

Figure 55 shows the transmit LSSI (Tx1 and Tx2) for a 16-bit I/Q data sample with MSB first configuration.

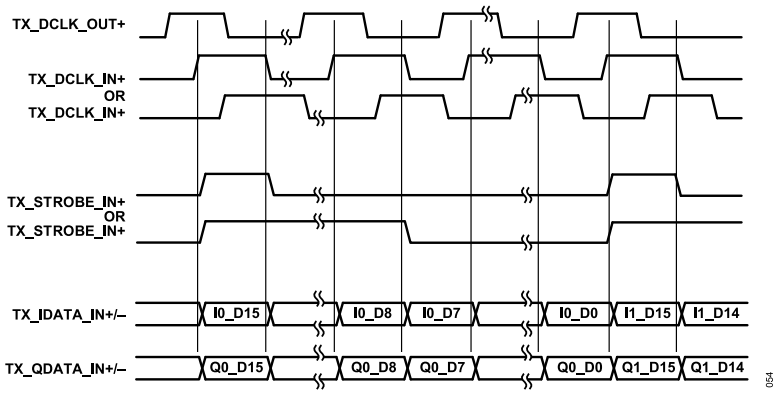


Figure 55. Transmit LSSI Timing for 16-Bit I/Q Data Sample on Separate Lanes

Figure 56 shows the transmit LSSI (Tx1 and Tx2) for a 12-bit I/Q data sample with MSB first configuration.

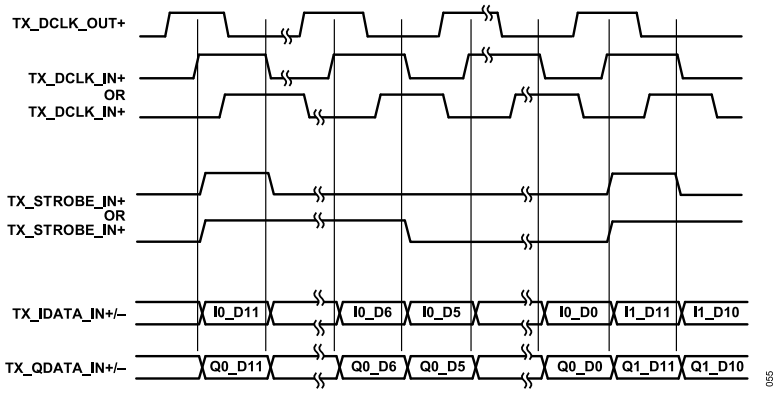


Figure 56. Transmit LSSI Timing for 12-Bit I/Q Data Sample on Separate Lanes

The TX_STROBE signal is aligned with the first bit of the serialized data (I and Q), and can be configured to be high:

- For a half-clock cycle at the start of the I and Q sample transmit. For a 16-bit data sample, the TX_STROBE is high for a half-clock cycle and low for a half and 15 clock cycles. For a 12-bit data sample, the TX_STROBE is high for a half-clock cycle and low for a half and 11 clock cycles.
- For half of I and Q data duration. For a 16-bit data sample, the TX_STROBE is high for 4 clock cycles, and low for 4 clock cycles (Q data sample). For a 12-bit data sample, the TX_STROBE is high for 3 clock cycles and low for 3 clock cycles.

In the 12-bit I/Q mode, 12-bit samples from LSSI are extended to 16 bits by padding four bits zero in LSB for the following transmit datapath process.

Transmit LSSI with One-Lane for I and Q

In this mode, only one-lane is used to transfer I and Q data samples. The I/Q data bits can be deserialized with configurable I or Q first and MSB or LSB first. The strobe signal can be configured to high for a half-clock cycle to indicate the start of the I and Q symbols or for half of the I and Q data duration to distinguish between I and Q data.

Figure 57 shows the one-lane LSSI (Tx1 and Tx2) for a 16-bit I/Q data sample with I sample and MSB first configuration.

DATA INTERFACE

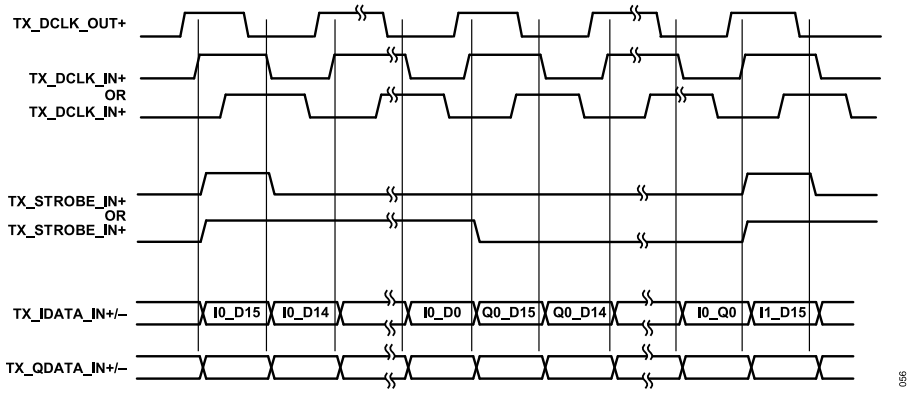


Figure 57. Transmit LSSI Timing for 16-Bit I/Q Data Sample Sharing One-Lane

Receive LSSI with Two, Four, and Eight Times Data Clock Rates

The ADRV9001 receive LSSI supports the two, four, or eight times of the data clock rate for some applications, which is similar with the receiver CSSI mode (see the timing diagrams in the Receive CSSI with Two, Four, and Eight Times Data Clock Rates section).

ENHANCED RX SSI MODE

The Rx SSI LVDS two-lane and CMOS one-lane modes have two enhanced modes to support 22-bit and 15-bit data samples in the I/Q mode.

For the 22-bit data samples, the 32bit interface data bus has the following fields:

- 22 -bit of data sample (I/Q from Rx datapath: unrounded data samples: RxDataPathI/Q[21:0])
- 1-bit = 0 (Constant)
- 1-bit Gain Change (Slicer or Index Gain Change flag)
- 8-bit Gain (Slicer or Index Gain)

It produces the following Interface data with a 32-bit data format for the CMOS and LVDS SSI:

- LVDS 32-bit: 2 lanes (I and Q) of 32-bit each
  - LSSI_DATA_I/Q [31:0] = {RxDataPathI/Q[21:0], b0, Gain_Change, Gain [7:0]}
- CMOS 64-bit: 1 Lane (I and Q)
  - CSSI_DATA [63:0] = {RxDataPathI[21:0], b0, Gain_Change, Gain[7:0], RxDataPathQ[21:0], b0, Gain_Change, Gain[7:0] }

Note: in 22-bit mode, a sample contains 32 bits of I and 32 bits of Q data. This requires the Rx interface to have a 2x clock while the Tx remains at 1x. For example, if in 16-bit mode, Tx/Rx SSI clock runs at 5 MHz, then in 32 bit mode the Tx SSI clock would need to run at 5 MHz and the Rx SSI clock would need to run at 10 MHz.

For the 15-bit data samples, the 16-bit interface data bus has the following fields:

- 15-bit of data sample (15-bit I/Q rounded from 22-bit Rx datapath samples)
- 1-bit Gain Change (Slicer or Index Gain Change flag)

It produces the following interface data with 16-bit data format for the CMOS and LVDS SSI:

- LVDS 16-bit: 2 lanes (I and Q) of 16-bit each
  - LSSI_DATA_I/Q [15:0] = {RxDataPathI/Q rounded[14:0], Gain_Change}
- CMOS 32-bit: 1 Lane (I and Q)
  - CSSI_DATA [31:0] = {RxDataPathI rounded[14:0], Gain_Change, RxDataPathQ rounded[14:0], Gain_Change}

The format of the enhanced SSI data for 15-bit and 22-bit mode is given in Figure 58.

DATA INTERFACE

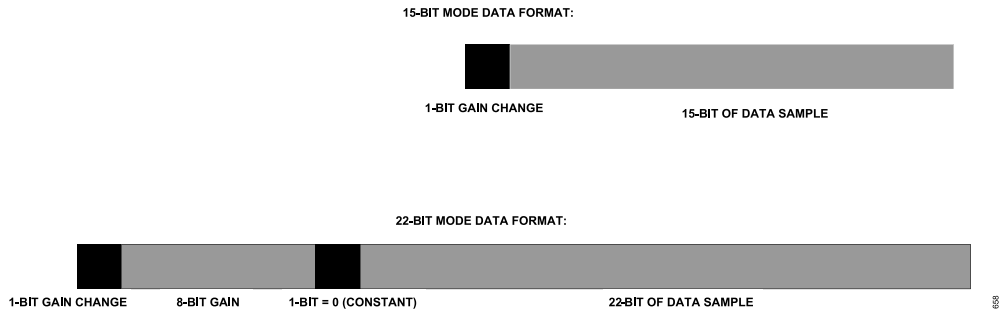


Figure 58. Enhanced SSI Data Format

Some other basic configuration modes, such as MSB/LSB first option, I or Q first option (for CMOS one-lane), and the Long/Short strobe option are similar to the previous SSI LVDS/CMOS 16-bit operations.

POWER SAVING FOR LSSI

In the time division duplex (TDD) mode, the LVDS SSI pads can be powered down/up dynamically based on the Tx_Enable and Rx_Enable level to save power. Three LSSI power-down modes are defined for different requirements (Table 27). The API **adi_adrv9001_Ssi_Power-Down_Set** is used to set the power-down mode for a specified channel.

Table 27. LSSI Power-Down Mode

LSSI Power-Down Mode	Description
ADI_ADRV9001_SSI_POWER_DOWN_DISABLED	All SSI PADS are powered up in PRIMED.
ADI_ADRV9001_SSI_POWER_DOWN_MEDIUM	RX_DCLK_OUT and TX_DCLK_OUT SSI pads are powered up, TX_DCLK_IN and all Tx/Rx STROBE and DATA SSI pads are powered down in PRIMED.
ADI_ADRV9001_SSI_POWER_DOWN_HIGH	All SSI pads are powered down in PRIMED.

SSI TIMING PARAMETERS

Figure 59 and Figure 60 show the receive SSI and transmit SSI timing diagrams.Table 28 and Table 29 show the timing specifications for the CMOS SSI and LVDS SSI.

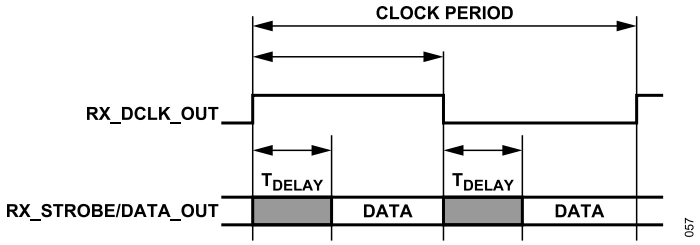


Figure 59. Receive SSI Timing Diagram

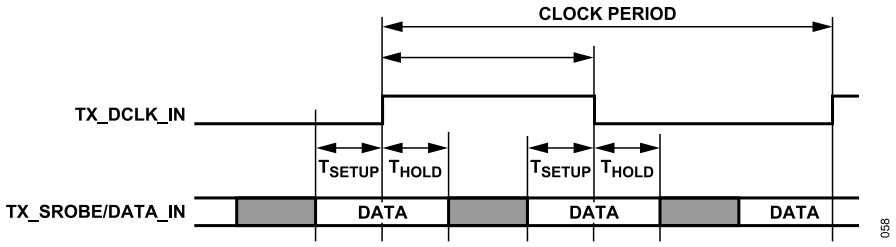


Figure 60. Transmit SSI Timing Diagram

Table 28. CMOS SSI Timing Specification

CMOS SSI	Timing	Description
CMOS Rx t_{DELAY} Maximum	4.5 ns	Clock to strobe/data delay

DATA INTERFACE

Table 28. CMOS SSI Timing Specification (Continued)

CMOS SSI	Timing	Description
CMOS Tx t_{SETUP} Minimum	2 ns	Strobe/data setup to clock
CMOS Tx t_{HOLD} Minimum	2 ns	Strobe/data hold after clock

Table 29. LVDS SSI Timing Specification

LVDS SSI	Timing	Description
Rx t_{DELAY} (Maximum)	200 ps	Clock to strobe/data delay
Tx t_{SETUP} (Minimum)	220 ps	Strobe/data setup to clock
Tx t_{HOLD} (Minimum)	390 ps	Strobe/data hold after clock

API PROGRAMMING

The ADRV9001 SSI configuration is performed in the chip initialization stage and based on the following data structure.

```
typedef struct adi_adrv9001_SsiConfig
{
adi_adrv9001_SsiType_e ssiType;
adi_adrv9001_SsiDataFormat_e ssiDataFormatSel;
adi_adrv9001_SsiNumLane_e numLaneSel;
adi_adrv9001_SsiStrobeType_e strobeType;
uint8_t lsbFirst;
uint8_t qFirst;
adi_adrv9001_SsiTxRefClockPin_e txRefClockPin;
bool lvdsIBitInversion;
bool lvdsQBitInversion;
bool lvdsStrobeBitInversion;
uint8_t lvdsUseLsbIn12bitMode;
bool lvdsRxClkInversionEn;
bool cmosDdrPosClkEn;
bool cmosClkInversionEn;
bool DdrEn;
bool rxMaskStrobeEn;
} adi_adrv9001_SsiConfig_t;
```

In the data structure, the previously mentioned SSI modes are defined for each Tx/Rx channel. Table 30 lists the SSI configuration parameters and some default values. Additionally, find the detailed data structure and enumerator description in the API Doxygen help file.

Table 30. SSI Configuration Parameters

Parameter	Type	Description	Note
ssiType	enum	Set SSI type	
ssiDataFormatSel	enum	Set SSI data format	
numLaneSel	enum	Set SSI number of lanes	
strobeType	enum	Set SSI strobe type	
lsbFirst	uint8_t	Set LSB first	Default '0', MSB first
qFirst	uint8_t	Set Q data first	Default '0', I data first
txRefClockPin	enum	Set TX SSI reference clock output (TX_DCLK_OUT) options	
lvdsIBitInversion	bool	Set LVDS SSI I bit differential pads polarity inversion	Default 'false'
lvdsQBitInversion	bool	Set LVDS SSI Q bit inversion	Default 'false," Rx SSI ignores this field, I/Q lanes share the configuration of "lvdsIBitInversion"
lvdsStrobeBitInversion	bool	Set LVDS SSI strobe bit inversion	Default 'false"
lvdsUseLsbIn12bitMode	uint8_t	Set LVDS 12-bit mode	Default '0', LVDS SSI uses 16-bit mode

DATA INTERFACE

Table 30. SSI Configuration Parameters (Continued)

Parameter	Type	Description	Note
lvdsRxClkInversionEn	bool	Set LVDS RX SSI clock inversion enable	Default 'false'
cmosDdrPosClkEn	bool	Set CMOS DDR positive clock enable	Default 'false'
cmosClkInversionEn	bool	Set CMOS DDR clock inversion enable	Default 'false'
DdrEn	bool	Set DDR mode enable	
rxMaskStrobeEn	bool	Set Rx Strobe Mask. Mask the Rx SSI Strobe when interface rate is multiple times of sample rate. See the Receive CSSI with Two, Four, and Eight Times Data Clock Rates section.	Default 'false'

Figure 48 shows the Rx CMOS SSI with the DDR clock in relation to the strobe/data. To ensure the BBIC gets the best setup/hold timing margin for RX CMOS DDR SSI, with Table 30 RX CMOS DDR relative default SSI configurations (cmosDdrPosClkEn=false, cmosClkInversionEn=false), follow the Rx CMOS SSI output clock/strobe/data phase timing diagram shown in Figure 61. ADI recommends this default RX CMOS SSI DDR configuration.

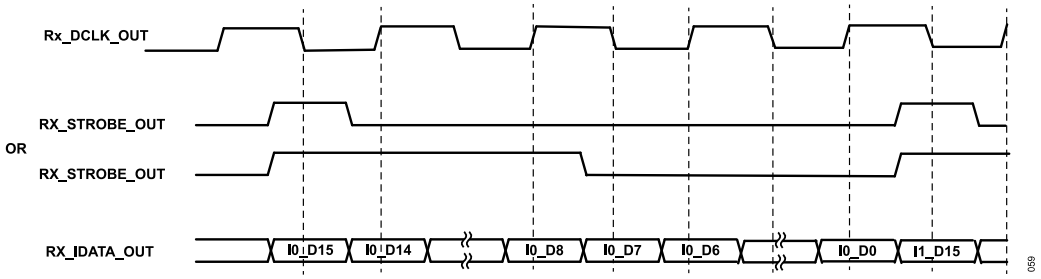


Figure 61. RX CMOS DDR SSI Default (Recommend) Output (cmosDdrPosClkEn=False, cmosClkInversionEn=False)

Also manually edit the CMOS DDR relative configurations based on the BBIC requirements. Figure 62, Figure 63, and Figure 64 show the corresponding Rx output Clock/Strobe/Data timing diagram with different CMOS DDR configurations.

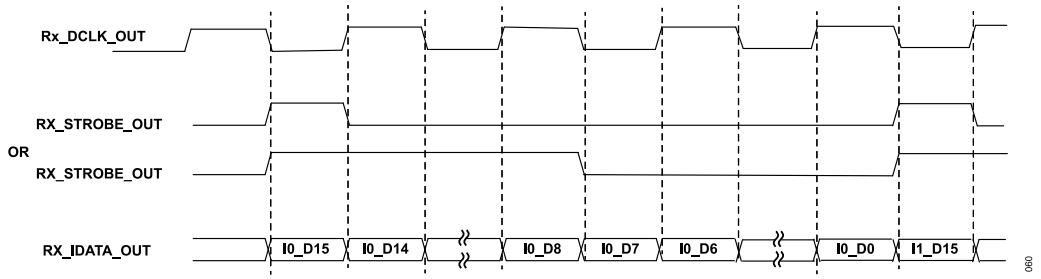


Figure 62. RX CMOS DDR SSI Output (cmosDdrPosClkEn=True, cmosClkInversionEn=False)

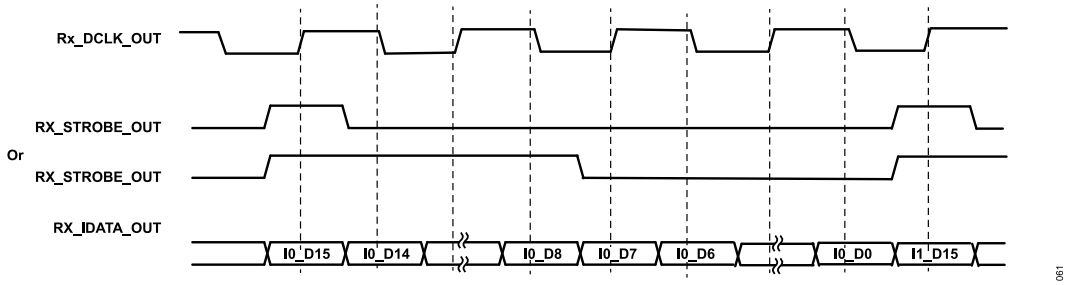


Figure 63. RX CMOS DDR SSI Output (cmosDdrPosClkEn=False, cmosClkInversionEn=True)

DATA INTERFACE

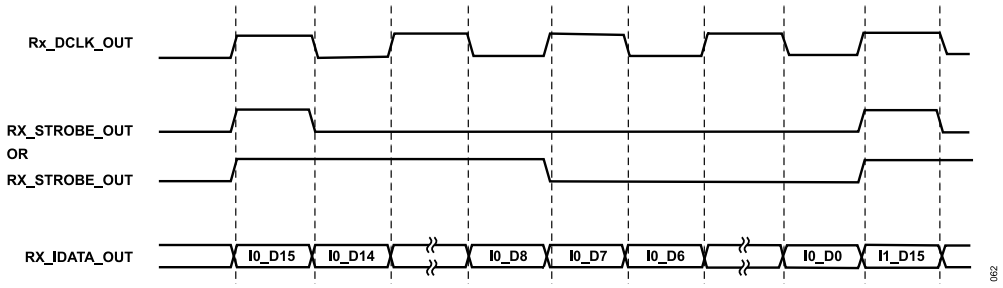


Figure 64. RX CMOS DDR SSI Output (cmosDdrPosClkEn=True, cmosClkInversionEn=True)

A set of API commands can set and inspect the SSI test/debug functions. Table 31 summarizes these.

Table 31. SSI Test/Debug API List

SSI Function Name	Description
adi_adrv9001_Ssi_Rx_TestMode_Configure	Configures the SSI test mode for the specified Rx channel.
adi_adrv9001_Ssi_Tx_TestMode_Configure	Configures the SSI test mode for the specified Tx channel.
adi_adrv9001_Ssi_Tx_TestMode_Status_Inspect	Inspects the SSI test mode status for the specified Tx channel.
adi_adrv9001_Ssi_Loopback_Set	Enables the Rx to Tx SSI loopback.
adi_adrv9001_Ssi_Delay_Configure	Programs the SSI delay configuration.
adi_adrv9001_Ssi_Delay_Inspect	Gets the SSI delay configuration from the ADRV9001 device.
adi_adrv9001_Ssi_PowerDown_Set	Sets the power-down mode for the specified channel and SSI type.

CSSI/LSSI TESTABILITY AND DEBUG

The ADRV9001 SSI has built-in test pattern generator and checker to quickly test and debug the SSI between the ADRV9001 and baseband processor. Figure 65 shows the ADRV9001 SSI testability and debug diagram with a baseband processor.

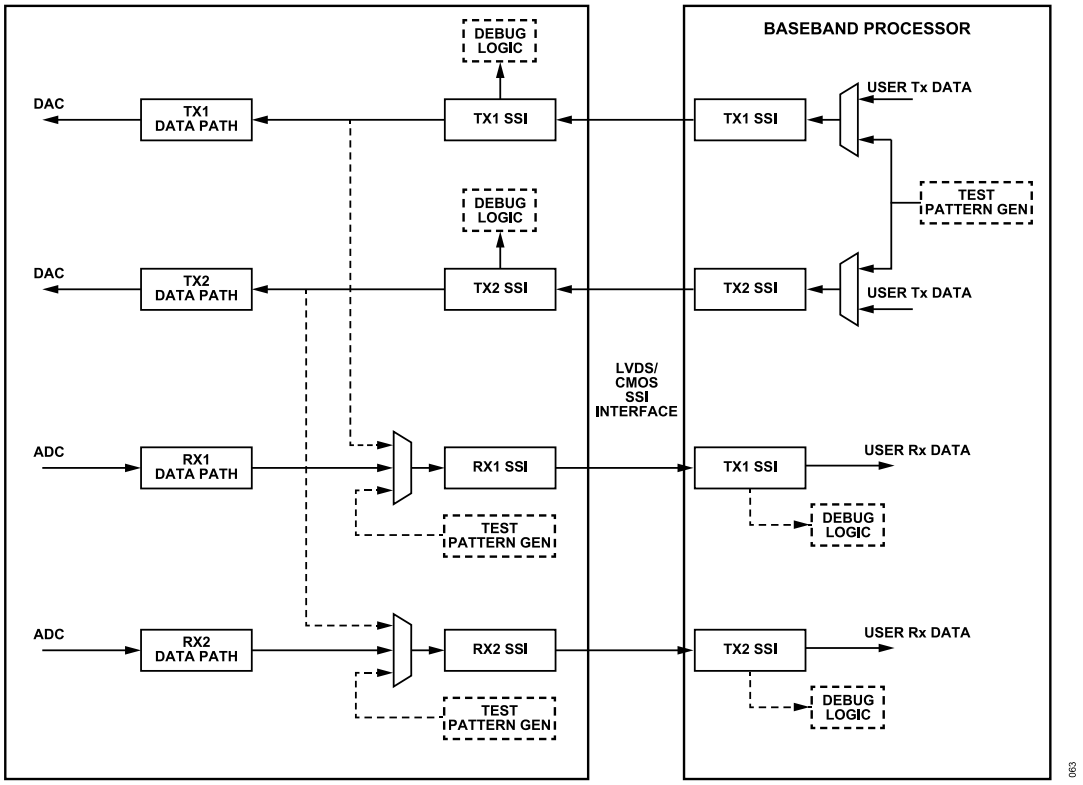


Figure 65. ADRV9001 SSI Testability and Debug Diagram

DATA INTERFACE

The ADRV9001 receive SSI can replace the receive channel data with a fixed pattern, ramp, or pseudorandom bit/binary sequence (PRBS) (LSSI only) pattern and send it to the baseband processor. Enable the receive debug function by calling **adi_adrv9001_Ssi_Rx_TestMode_Configure()**. Check the specified test pattern at the SSI output to test if the receive SSI from the ADRV9001 to BBIC functions correctly.

The data structure **adi_adrv9001_RxSsiTestModeCfg_t** enables and configures the RX SSI test pattern.

```
typedef struct adi_adrv9001_RxSsiTestModeCfg
{
adi_adrv9001_SsiTestModeData_e testData; /*!< Type of data to transmit over SSI */
uint32_t fixedDataPatternToTransmit; /*!< Value of Fixed pattern to transmit over interface. For vari▶
ous SSI data format:
CMOS: Pattern is truncated to bit3 - bit0 value is transmitted on RxSSI I and Q each nibble (where
applicable)
LVDS: Pattern is truncated to bit15 - bit0 value transmitted on RxSSI I and Q (where applicable) */
} adi_adrv9001_RxSsiTestModeCfg_t;
```

The enum **adi_adrv9001_SsiTestModeData_e** enables and chooses the specified test pattern and lists in Table 32.

Table 32. Definition of adi_adrv9001_SsiTestModeData_e

ENUM	Description
ADI_ADRV9001_SSI_TESTMODE_DATA_NORMAL	No test mode enabled
ADI_ADRV9001_SSI_TESTMODE_DATA_FIXED_PATTERN	Fixed pattern mode
ADI_ADRV9001_SSI_TESTMODE_DATA_RAMP_NIBBLE	Nibble ramp mode (CSSI only), I Q data is same and each 4-bit of the samples keep ramping (for example, sample0-0x0000, sampe1-0x1111, sample2-0x2222, …).
ADI_ADRV9001_SSI_TESTMODE_DATA_RAMP_16_BIT	16-bit ramp mode, I Q data is same, 16-bit of the samples keep ramping (for example, sample0-0x0000, sample1-0x0001, sample2-0x0002,…).
ADI_ADRV9001_SSI_TESTMODE_DATA_PRBS15	PRBS15 mode (LSSI only)
ADI_ADRV9001_SSI_TESTMODE_DATA_PRBS7	PRBS7 mode (LSSI only)

Enhanced Rx SSI 32-bit mode: the test pattern generation debug modes are similar to the existing 16-bit SSI debug operations. In the CMOS mode, for fixed pattern, the RX SSI 64-bit test pattern = {fixedDataPatternToTransmit [15:0], fixedDataPatternToTransmit [15:0], fixedDataPatternToTransmit [15:0], and fixedDataPatternToTransmit [15:0]}. For 16-bit ramp mode, the Rx SSI 64-bit test pattern = {RampPattern[15:0], RampPattern[15:0], RampPattern[15:0], and RampPattern[15:0]}.

Similarly, in the LVDS mode, the Rx SSI 32-bit fixed test pattern for I and Q is {fixedDataPatternToTransmit [15:0], fixedDataPatternToTransmit [15:0]}, and ramp patter for I and Q is {RampPattern[15:0], RampPattern[15:0]}. PRBS pattern is not supported in this enhanced Rx SSI 32-bit mode.

The ADRV9001 transmit SSI has a ramp and PRBS (LSSI only) pattern checker. Configure the ADRV9001 TX SSI test mode and transmit ramp or PRBS pattern through SSI to ADRV9001 to verify if the SSI is functioning correctly. Also transmit a fixed pattern and configure the ADRV9001 with the specified fixed pattern to verify the SSI functionality. Call API **adi_adrv9001_Ssi_Tx_TestMode_Configure** to enable and configure the test mode, and transmit the corresponding test patterns to the ADRV9001 through Tx SSI, then call **adi_adrv9001_Ssi_Tx_TestMode_Status_Inspect** to get the ADRV9001 TX SSI test mode status.

Similarly, data structure **adi_adrv9001_TxSsiTestModeCfg** enables and configures the ADRV9001 TX SSI test pattern checker. BBIC transmits relative test patterns and the format follows the description in Table 32. For the fixed pattern mode transmit, the BBIC puts bits 31:16 of fixedDataPatternToCheck on the TX SSI I data and bits 15:0 on Q data.

```
typedef struct adi_adrv9001_TxSsiTestModeCfg
{
adi_adrv9001_SsiTestModeData_e testData; /*!< Type of data to receive over SSI and check */
uint32_t fixedDataPatternToCheck; /*!< Value of Fixed pattern to check against pattern received over
interface */
} adi_adrv9001_TxSsiTestModeCfg_t;
```

DATA INTERFACE

The ADRV9001 transmit SSI data output can be looped back to receive the SSI data input using API **adi_adrv9001_Ssi_Loopback_Set**. When the transmit and receive SSI run at the same clock rate, use the pattern generator and checker to verify the functionality of the whole system SSI. Note that both the ADRV9001 TX and RX radio state must be in the “RF_ENABLED” state to ensure the TX/RX SSI is enabled when setting the SSI loopback test function.

As mentioned previously, the SSI clock, strobe, and data have programmable delays. Configure these delays using **adi_adrv9001_Ssi_Delay_Configure**. The unit for the SSI delay is step. This aids to meet the timing specifications described in the SSI Timing Parameters section.

MICROPROCESSOR AND SYSTEM CONTROL

The ADRV9001 supports quick configuration from idle states of operation and quick transition between receive and transmit states. Internal blocks called stream processors handle these transitions. The stream processor within the ADRV9001 device performs a series of configuration tasks upon an external request. Upon a request, the stream processor performs a series of actions defined in the image loaded into the ADRV9001 during the initialization process.

The stream processor, therefore, has streams (series of tasks) for:

- Tx1 Enable/Tx1 Disable
- Tx2 Enable/Tx2 Disable
- Rx1 Enable/Rx1 Disable
- Rx2 Enable/Rx2 Disable

Paths are enabled and disabled typically using pins. However, these paths can also be controlled over the SPI bus using an API command. The stream is not limited to path-enabling events and can react to other events such as a DGPIO input signal.

The ADRV9001 is flexible in its configuration, and, therefore, the stream is flexible. Similarly, as the initialization structures change with the profile, the stream processor image must change with the configuration. For example, the stream that enables Rx1 differs depending on the choice of either a narrowband or wideband profile. So, it is necessary to use a stream image for each configuration of the device. In this way, when the user saves configuration files (.c) using the ADRV9001 TES, a stream image is also saved automatically. This stream file is then used when using these configuration files.

Figure 66 describes the general ecosystem of the ADRV9001. On the right-hand side (data side), the ADRV9001 interfaces with the BBIC and on the left-hand side (antenna side), it interfaces with the RF components. The following section describes the control of the ADRV9001 datapaths.

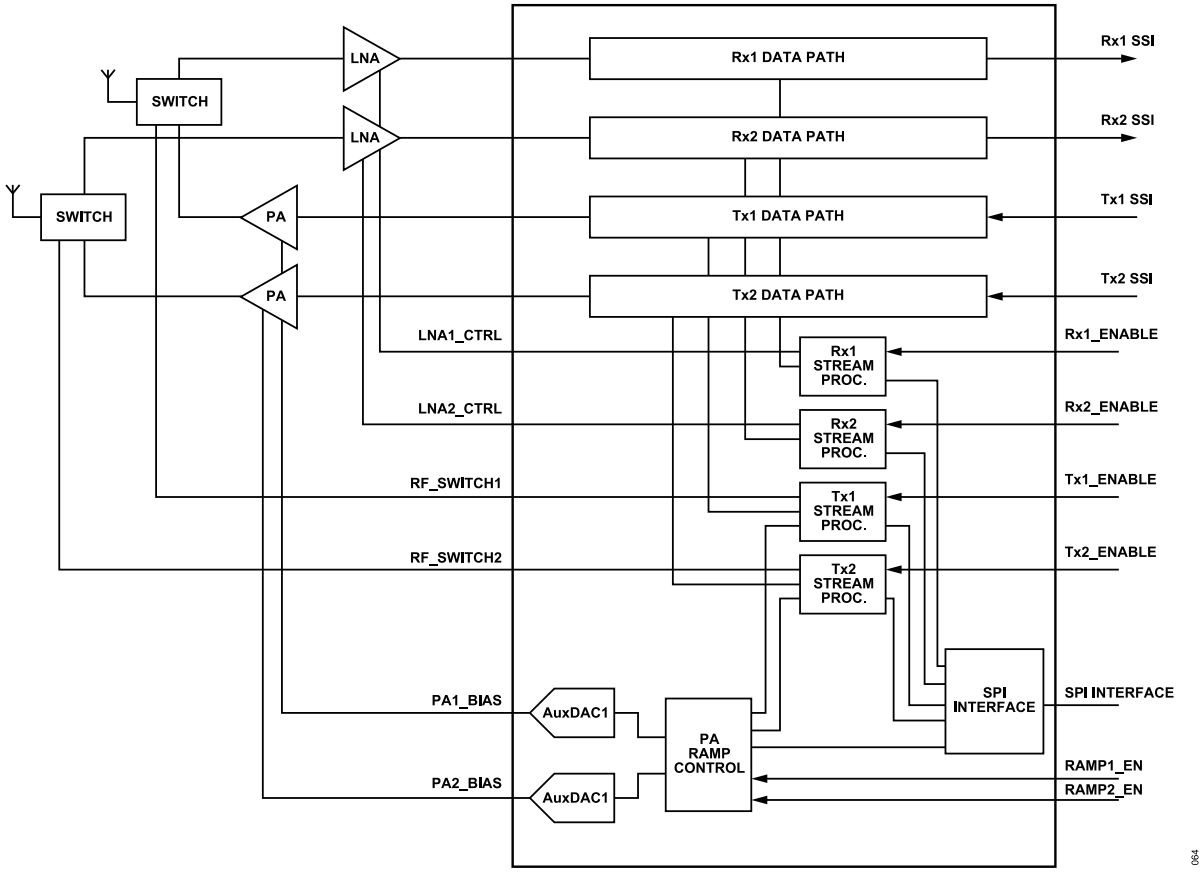


Figure 66. Datapath Control Signals

MICROPROCESSOR AND SYSTEM CONTROL

SYSTEM CONTROL

Control of the datapaths within the ADRV9001 is done through the API or ENABLE pin controls. For API control, this is reliant on the SPI communication bus and thus, for critical time alignment of powering on/off chains, pin control is recommended for TDD applications. Independently control each datapath with the following enable signals:

Table 33. Datapath Enable Signals

Enable Signal	Datapath
RX1_ENABLE	Rx1 datapath
RX2_ENABLE	Rx2 datapath
TX1_ENABLE	Tx1 datapath
TX2_ENABLE	Tx2 datapath

For ADRV9001 to receive and react to control signals, move it to the primed state. The primed state indicates that the system is ready for operation when the transmit and receive channels are enabled. After the channel is primed, and to start transmit or reception activities, further transition it from the primed state to the RF_ENABLED state, either in PIN mode by the enable signals listed in Table 33 or in SPI mode by API commands.

PIN Mode

1. Call **adi_adrv9001_Radio_ChannelEnableMode_Set( )** to set the PIN mode.
2. Toggle corresponding ENABLE pin to transition the channel between PRIMED state and RF_ENABLED state.

SPI Mode

1. Call **adi_adrv9001_Radio_ChannelEnableMode_Set( )** to set the SPI mode.
2. Call **adi_adrv9001_Radio_Channel_EnableRf( )** to transition the channel between PRIMED state.

After pin or SPI/API mode is executed, the ADRV9001 enables the requested channels. The channels remain active until further instruction through a pin command or SPI/API command.

TIMING PARAMETERS CONTROL

The ADRV9001 has integrated stream processors to handle various external and internal events. These events must be serviced in real time. These stream processors coupled with programmable delayed enable modules relieve the system firmware (running on integrated microprocessor) from managing all the critical events with a quick and parallel response to external and internal events. This configuration allows the 4-channels (Tx1, Tx2, Rx1, and Rx2) to operate independently from each other by using their own dedicated stream processor.

The ADRV9001 can support different applications, each with its own unique challenges. A set of programmable timing parameters for both transmitter and receiver meet the particular timing requirements in various TDD applications. It is crucial to understand the ADRV9001 timing parameters to ensure all TDD events take place at an accurate time order. In addition, configuring timing parameters in an optimal way, by taking advantage of the multiple power saving modes of the ADRV9001, improves the overall system power consumption performance significantly.

Timing Definition

Before explaining the typical values for each delay, this chapter attempts to visualize and explain each delay and how it pertains to the physical component. Figure 67 is a visual representation of the most pertinent delays in a system. Note that the length of each arrow does not represent the length of each delay in time. These are just representations of each delay in terms of the hardware.

MICROPROCESSOR AND SYSTEM CONTROL

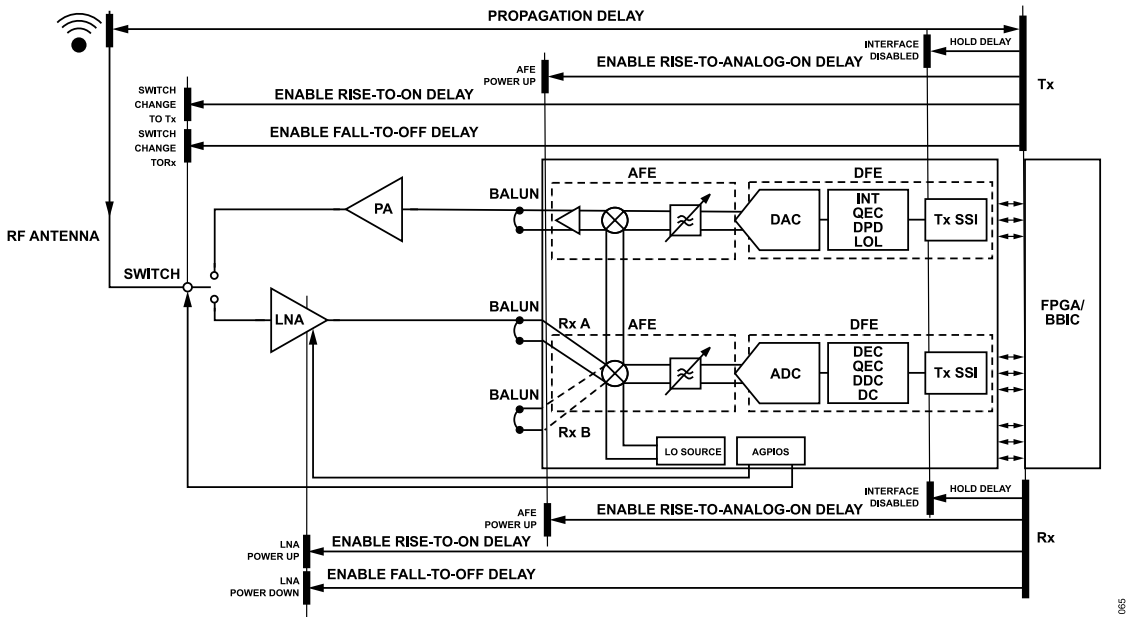


Figure 67. Visualization of Timing Parameters

Broadly, the delays in the system are described as follows:

- ***Enable Setup Delay*** is the time taken for the ADRV9001 to power up its analog front end (AFE). This may or may not include PLL tuning time based on the use case. For example, when the transmitter and receiver share the same IO but at different frequency, PLL tuning is needed at the frame boundary.
- ***Propagation Delay*** is the delay of data from the antenna to RF interface in either direction. Given that this delay encompasses external components, it is impossible for the ADRV9001 to determine what this delay is. The user must measure it. Naturally, this delay is also dependent on the setup and board layout. It does not have to be provided to the ADRV9001, but it can be used to derive other parameters required by the ADRV9001.
- ***Enable Rise-to-On Delay*** is the delay between the TX_ENABLE rising edge and receiver/transmitter switch changing to the transmitter channel. Conversely, it is also the delay between the RX_ENABLE rising edge and the external receiver LNA powering up. Align it with the desired time when the first symbol is on air. Typically, this delay is comparable in length with the propagation delay for transmitter channels. However, if the setup uses Guard Data to envelop the Frame on Air, account for this extra guard time. This delay does not apply to every setup. However, if the ADRV9001 is not controlling the antenna switch, this parameter is not needed except to determine other parameters.
- ***Enable Rise-to-Analog-On Delay*** is the delay between the TX_ENABLE / RX_ENABLE rising edge and analog power up beginning. This is a user-defined parameter to align the analog power up with the Frame on Air. Note that this parameter aids in saving power in setups with quite long propagation delays. If the transmitter propagation delay is long, delay the analog power up to save power or to keep the ADRV9001 internal transmitter AFE powered down during the receiver frame. If the propagation delay is small, set this to 0. If this parameter is greater than its maximum bound, delay the antenna switch/LNA power on time.
- ***Enable Guard Delay*** is the guard time at the beginning of the RF frame. This part of the Frame on Air does not contain useful data. Instead, it is used as a barrier between the end of the device setup and the start of the data transmission. Any distortions or noise applied to the guard data by the transmitter device do not affect error rates at the receiver. Not every application uses the guard data. However, for those that do, the Guard Delay only accounts for the Guard data at the beginning of the frame, not the end. The Guard data at the end of the frame is accounted for with the Hold delay. Guard Delay is reserved for future use, and should be set to 0 now.
- ***Enable Hold Delay*** is the delay between the RX_ENABLE falling edge and masking off datapath data sent over interface. Similarly, it is also the delay between the falling edge of TX_ENABLE and the transmitter interface being disabled.
- ***Enable Fall-to-Off Delay*** is the delay between the RX_ENABLE falling edge and the powering down of the ADRV9001 internal receiver AFE and external receiver LNA if the external receiver LNA control by the ADRV9001 is enabled. The ADRV9001 forces it to 0 currently, meaning the ADRV9001 internal receiver AFE and external LNA are disabled at the same time as the receiver is disabled. Similarly, it is also the delay between the TX_ENABLE falling edge and ADRV90001 transmitter AFE powering down and the antenna switch switching to Rx channel if the antenna switching feature is enabled. For the transmitter, set this delay equal to the internal path delay (discussed next) for all transmit data propagated to the front end to be transmitted out through the antenna.

- ***Internal Path Delay*** is the delay between the SSI port and RF port for either the transmitter or receiver signal chains. It does not include any external components, and the ADRV9001 internal calibration algorithm calibrates the internal path delay during the chip initialization stage. As part of the design of a custom setup, use the Internal Path Delay as an approximation, and further measure the entire Propagation Delay of the setup to ensure the accurate transmitter/receiver timing on air in TDD operations.

The following sections detail how each delay presents in the transmitter and receiver signal chains, respectively, as well as the design choices to make around them in different use cases.

Transmit Timing Definition

Transmit timing parameters define the events in order from the start of transmission at the ADRV9001 data port to the end when the transmit burst is sent through the antenna to the air.

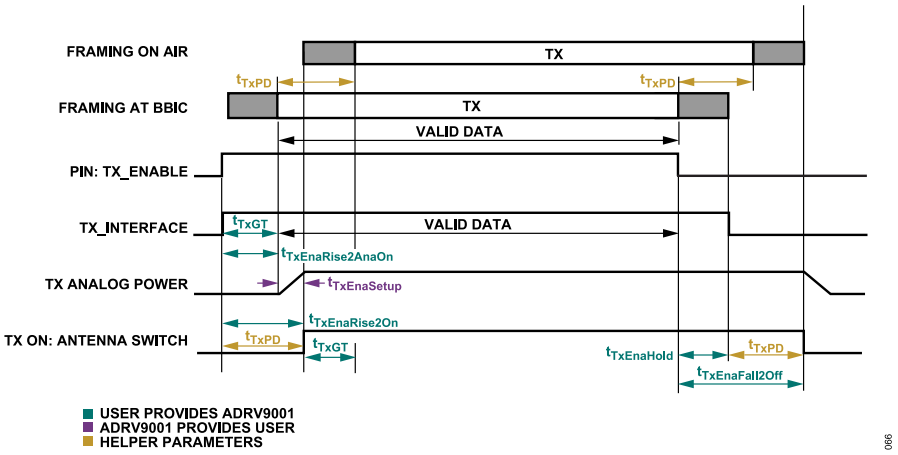


Figure 68. Transmitter Timing Parameters (t_{TxPD} > $t_{TxEnaSetup}$)

Figure 68 shows that a transmit burst consists of a series of valid transmit data with the option to pad guard data at the beginning and end of the valid data. Based on the configured timing parameters configured, decide if full or partial of the guard data must be transmitted to the air, and ensure the guard data usage is compliant with the standard requirement. Control the transmit enable pin to signal the start and end of a transmit burst at the data port. Based on the transmitter enable signal and a set of configured transmit timing parameters, the ADRV9001 further controls the transmitter interface, transmit internal analog components, as well as the antenna switch (if it is controlled by the ADRV9001 instead of the user) to make sure the transmit burst is on air at a deterministic time as desired.

Transmit timing parameters shown in Figure 68 can be categorized into three types: ADRV9001 parameter (ADRV9001 provides to the user), user parameter (user provides to the ADRV9001), and helper parameters (determined by the user, which are not needed to provide to the ADRV9001 but can be used to derive other required timing parameters). Table 34 further explains all these timing parameters. All bounds specified in Table 34 are suggestions for optimal operation. No hardware, or software restrictions prevent from setting out of bound values. The following sections specify the maximum programmable parameter value.

Table 34. Transmit Timing Parameters Description

Transmit Timing Parameters	Provided by	Bounds	Comments
enableSetupDelay ($t_{TxEnaSetup}$)	ADRV9001 Parameter	Min: N/A Max: N/A	No PLL retuning at frame boundary: 8 µs (analog power-up time) PLL tuning at frame boundary: 758 µs (analog power-up time + PLL tuning time) (The PLL tuning time 750 µs refers to when the internal LO is used and it represents the worst case. When external LO is used, calculate and use own PLL tuning time.)
propagationDelay (t_{TxPD})	Helper Parameter	Min: N/A Max: N/A	Measure this parameter. It is profile and board layout dependent. It does not have to provide to the ADRV9001. Use it to derive other parameters required by the ADRV9001.
enableRiseToOnDelay ($t_{TxEnaRise2On}$)	User Parameter	Min: 0 Typical: t_{TxPD} Max: t_{TxGT}+ t_{TxPD}	At minimum bound: antenna switch occurs $t_{TxEnaSetup}$ + some margin after transmit enable rising edge.

MICROPROCESSOR AND SYSTEM CONTROL

Table 34. Transmit Timing Parameters Description (Continued)

Transmit Timing Parameters	Provided by	Bounds	Comments
			At typical value: all symbols sent over the interface (including guard symbols) make it onto the air. At maximum bound: no guard symbols are transmitted over the air.
enableRiseToAnalogOnDelay ($t_{TxEnaRise2AnaOn}$)	User parameter	Min: 0 Max: $t_{TxEnaRise2On} - t_{TxEnaSetup}$	If the transmit propagation delay is long, delay the analog power up for power saving or to keep transmit analog powered down during receive frame. If the transmit propagation delay is small, set this to 0. If this parameter is greater than its maximum bound, delay the antenna switch time.
enableGuardDelay(t_{TxGT})	User parameter	Min: N/A Max: N/A	Reserved, not used.
enableHoldDelay ($t_{TxEnaHold}$)	User parameter	Min: 0 Max: None. Must be optimized to be minimal.	Tx_enable falling edge must ideally come as the last valid data is sent over the interface. Use this to disable the transmit algorithms, whose performance may be degraded if guard symbols are used. Keep the interface on even after Tx_enable falling edge to allow the transmission of user guard symbols.
enableFallToOffDelay ($t_{TxEnaFall2Off}$)	User parameter	Min: $t_{TxEnaHold}$ Max: None. Must be optimized to be minimal. (Recommended Max: $t_{TxEnaHold}$ + t_{TxPD} Note $t_{TxEnaHold}$ is forced to 0 currently.)	At minimum bound: antenna switch occurs soon after the transmit interface is disabled. It should always occur before powering down of the transmit analog. Not all symbols in the path make it on air. At maximum bound: antenna is switched away from the transmit channel just as the last user data has propagated to the antenna.

Design Strategies for Transmit Timing Parameters

Use Case 1: $t_{TXPD} > t_{TXENASETUP}$

Here, because the propagation delay is larger than the transmit analog setup delay, delay powering up the AFE while the data is propagating through the digital datapath, as shown in Figure 68. This saves power by reducing unnecessary AFE on time. For example, if the propagation delay is 2.5 ms, whereas the enableSetupDelay provided by the ADRV9001 is 8 µs, an AFE can be off to avoid burning power for the first 2.492 ms of the propagation time. Having the analog front end powered up early can also be a liability. For example, if the transmit propagation path delay is longer than the guard time between receive and transmit frames, the transmit enable rising edge may occur in the middle of a receive frame. In this case, keep the AFE of the transmitter channel powered down until the end of the receive frame. In such a case, set the enableRiseToAnalogOnDelay to some value less than or equal to:

propagationDelay – enableSetupDelay.

Set enableRiseToOnDelay equal to the propagationDelay or enableRiseToAnalogOnDelay + enableSetupDelay. The transmit enables rising edge must occur enableRiseToOnDelay before on air transmit begins. The transmit interface can be set high at the same time as transmit enables to start transmitting guard symbols.

When the frame ends, the transmit enable falling edge should ideally occur right as the last valid data that must be demodulated by a receiver sent over the interface. Hold the interface on for some time longer to send guard data across by setting enableHoldDelay to a value greater than zero. The parameter enableFallToOffDelay determines when the ADRV9001 transmit AFE is powered down after the transmit enables falling edge, and the antenna is switched away from the transmit channel if the antenna switch is enabled. Always set it to a value greater than or equal to enableHoldDelay. If both values are set equal, for example, both are set to 0, the interface turns off first, then analog powers down. To ensure all the data sent over the interface makes it onto the air, set enableFallToOffDelay greater than or equal to:

enableHoldDelay + propagationDelay.

If it is greater, then zeros are transmitted after all the data sent over the interface is propagated.

Use Case 2: $t_{TxPD} < t_{TxEnaSetup}$

In this case, as shown in Figure 69, the time taken for data to propagate from the digital interface to antenna is very small, i.e. , t_{TxPD} is smaller than the time to set up the AFE $t_{TxEnaSetup}$. In such a case, set enableRiseToAnalogOnDelay to 0, so that analog power up begins immediately after the TX_ENABLE rising edge. The parameter enableRiseToOnDelay can also be set to 0. Here, the antenna is switched to a transmit channel, as soon as analog power up completes. For a more deterministic delay between the transmit enable rising edge and antenna switch time, set enableRiseToOnDelay to a value greater than or equal to:

enableRiseToAnalogOnDelay + enableSetupDelay.

In such a case, after raising Tx_enable, send some guard data over the interface to ensure all valid transmit data is transmitted on air. Based on the length of the user guard time and transmit timing parameter configurations, only a part or none of the guard data is transmitted to the air.

When the frame ends, set enableFallToOffDelay similar to use case 1.

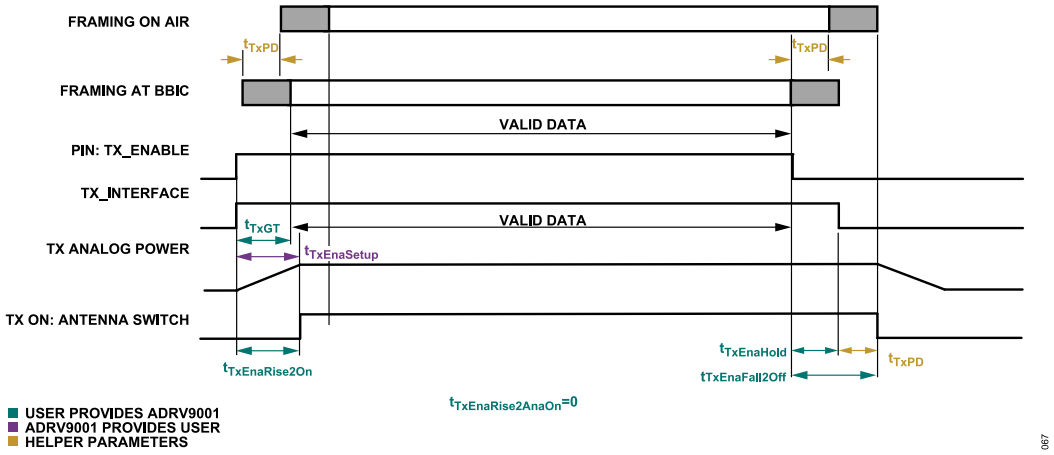


Figure 69. Transmit Timing Parameters ($t_{TxPD} < t_{TxEnaSetup}$)

Receive Timing Definition

Receive timing parameters define the events from the start of reception at the air to the end when the receive burst is sent through the ADRV9001 data port to the BBIC.

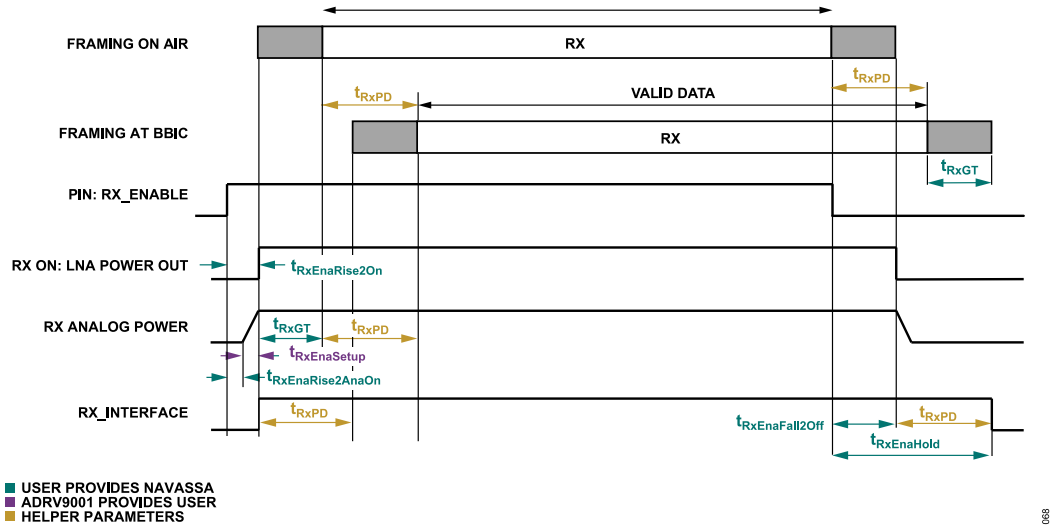


Figure 70. Receive Timing Parameters

Figure 70 shows that a receive burst has a series of valid receive data with the option to pad guard data at the beginning and end of the valid data. Similar to transmit, based on the configured timing parameters, receive full or partial guard data, and ensure the guard data usage is compliant with the standard requirement. Control the RX_ENABLE pin to signal the ADRV9001 the start and end of a receive burst at the air (note: RX_ENABLE must rise before the start of the receive burst at air to power up the AFE.). Based on the RX_ENABLE signal and a set of configured receive timing parameters, the ADRV9001 further controls receive analog components, receive interface, and the external LNA (if it is controlled by the ADRV9001 instead of the user) to make sure the received burst is sent to BBIC at the deterministic time as desired.

MICROPROCESSOR AND SYSTEM CONTROL

Similar to transmit timing parameters, as shown in Figure 68, receive timing parameters can be categorized into three types: ADRV9001 parameter (ADRV9001 provides to the user), user parameter (user provides to the ADRV9001), and helper parameters (determined by the user, which do not have to provide to the ADRV9001 but can be used to derive other required timing parameters).

All the parameters used in Figure 70 are explained further in Table 35. All bounds specified in Table 35 are suggestions for optimal operation. No hardware, or software restrictions prevent from setting out of bound values. The following sections specify the maximum programmable parameter value.

Table 35. Receive Timing Parameters Description

Delay	Provided By	Bounds	Comments
enableSetupDelay ($t_{RxEnaSetup}$)	ADRV9001 Parameter	Min: N/A Max: N/A	No PLL tuning at frame boundary: 8 µs (analog power-up time) PLL tuning at frame boundary: 758 µs (analog power-up time + PLL tuning time) (The PLL tuning time 750 µs refers to when internal LO is used and it represents the worst case. When external LO is used, calculate and use own PLL tuning time.)
propagationDelay(t_{RxPD})	Helper Parameter	Min: N/A Max: N/A	Meaure this parameter. It is profile and board layout dependent. It does not have to provide to the ADRV9001. However, use it to derive values for other parameters required by the ADRV9001.
enableRiseToAnalogOnDelay ($t_{RxEnaRise2AnaOn}$)	User Parameter	Min: 0 Max: duration of power up tasks in power savings or frequency hopping modes.	Only set to non-zero values if using power savings or frequency hopping. See the following sections to determine how to choose a non-zero value.
enableRiseToOnDelay ($t_{RxEnaRise2On}$)	User Parameter	Min: $t_{RxEnaRise2AnaOn}$ Typ: $t_{RxEnaRise2AnaOn}$ + $t_{RxEnaSetup}$ Max: None. Must be optimized to be minimal.	If set to $t_{RxEnaRise2AnaOn}$, the actual delay is $t_{RxEnaRise2AnaOn}$ + $t_{RxEnaSetup}$.
enableGuardDelay (t_{RxGT})	User Parameter	Min: N/A Max: N/A	Reserved
enableFallToOffDelay ($t_{RxEnaFall2Off}$)	User Parameter	Min: 0 Max: None. Must be optimized to be minimal.	Ideally, RX_ENABLE falling edge arrives when the last valid data is received over the air. By setting this value greater than 0, the ADRV9001 can continue receiving guard symbols, while signaling to certain algorithms or other systems that the valid data for the frame is already received.
enableHoldDelay ($t_{RxEnaHold}$)	User Parameter	Min: $t_{RxEnaFall2Off}$ Max: None. Must be optimized to be minimal. (Recommended Max: $t_{RxEnaFall2Off}$ + t_{RxPD} .)	The interface is disabled only after analog power down is completed. At minimum bound: Some of the data received at the antenna may not make it over the interface. At maximum bound: Digital datapath and receive SSI remain enabled until last received data is propagated to the interface.

Design Strategy for Receive Timing Parameters

As described, the ADRV9001 provides the enableSetupDelay, which is the time required to power up the receiver front end. By knowing this, set the RX_ENABLE pin high as at least enableRiseToOnDelay in advance, as shown in Figure 70. In regular TDD mode, i.e., no power savings or frequency hopping, set enableRiseToAnalogOnDelay always to 0, so that analog power-up begins immediately after the receive enable's rising edge (note that Figure 70 describes the receive timing parameters in a general case with enableRiseToAnalogOnDelay not equal to 0.) Set the parameter enableRiseToOnDelay also to 0. In this case, the LNA is powered up as soon as analog power up completes. For a more deterministic delay between the RX_ENABLE rising edge and LNA power up time, set enableRiseToOnDelay to a value greater than or equal to:

enableRiseToAnalogOnDelay + *enableSetupDelay*.

MICROPROCESSOR AND SYSTEM CONTROL

Once timing on air is established, raise RX_ENABLE some time before the start of the actual frame. As soon as the receiver analog power up completes, the digital interface turns on. However, if the path has a long propagation delay, the initial data coming off the interface is not the data received over the air.

When the frame ends, the users wish to continue receiving for a while. However, the ADRV9001 wishes to stop all the tracking algorithms to avoid any performance degradation. Achieve this by bringing the RX_ENABLE signal low as soon as the frame ends, but setting the enableFallToOffDelay equal to the time the user wishes to continue receiving data (Note that enableFallToOffDelay is forced to 0 currently by the ADRV9001.). This time should not be larger than the guard time before the next frame. The longer this value is, a delay in the next Rx_enable rising edge can occur. In cases where the receive path has a large propagation delay, turn off the receiver AFE to start a transmit frame but still leave the digital datapath and interface on so that data already received over the air may be sent over the interface. Use the enableHoldDelay parameter for this. Always set it at least to the enableFallToOffDelay. To receive all the data already received over the air, set it to:

enableFallToOffDelay + propagationDelay.

Guard/Hold Times Between Edges of TX_ENABLE and RX_ENABLE

Based upon the understanding of the transmit and receive timing parameters discussed separately in the previous sections, this section further discusses the minimum guard/hold time design between the rising and falling edges of TX_ENABLE and RX_ENABLE in a TDD system. There are six scenarios:

- Guard time between TX_ENABLE falling edge and RX_ENABLE rising edge
- Guard time between RX_ENABLE falling edge and TX_ENABLE rising edge
- Guard time between TX_ENABLE falling edge and TX_ENABLE rising edge
- Guard time between RX_ENABLE falling edge and RX_ENABLE rising edge
- Hold time between TX_ENABLE rising edge and TX_ENABLE falling edge
- Hold time between RX_ENABLE rising edge and RX_ENABLE falling edge

Always set the guard/hold timer greater than the minimum requirement. Note that no hardware or software restriction prevents from raising TX_ENABLE/RX_ENABLE at any time. The correct operation cannot be guaranteed if the rules described in the following sections are violated.

Guard Time Between TX_ENABLE Falling Edge and RX_ENABLE Rising Edge

The guard time between TX_ENABLE falling edge and RX_ENABLE rising edge makes sure the transmitter and the receiver AFE are not powered up simultaneously. As discussed in the previous sections, after TX_ENABLE falling edge, it takes $t_{TxEnaFall2Off}$ to power off the transmitter AFE. Therefore, the earliest time the receiver AFE can be powered up is $t_{TxEnaFall2Off}$ after the TX_ENABLE falling edge. Because it takes $t_{RxEnaRise2On}$ to power up the receiver AFE starting from the RX_ENABLE rising edge, the minimum guard time is $t_{TxEnaFall2Off} - t_{RxEnaRise2On}$ if $t_{TxEnaFall2Off}$ is greater than $t_{RxEnaRise2On}$. For $t_{TxEnaFall2Off}$ is less than $t_{RxEnaRise2On}$ (this can be possible when power saving modes are enabled as discussed in the following sections), RX_ENABLE rising edge can happen $t_{RxEnaRise2On} - t_{TxEnaFall2Off}$ before TX_ENABLE falling edge. Figure 71 describes both cases.

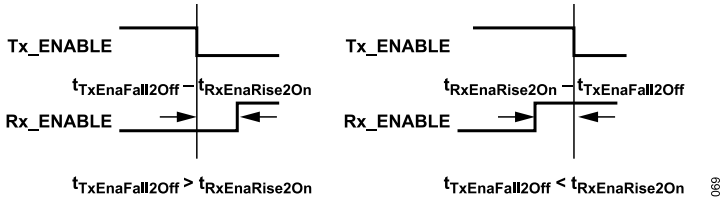


Figure 71. Minimum Guard Time Between TX_ENABLE Falling Edge and RX_ENABLE Rising Edge

Guard Time Between RX_ENABLE Falling Edge and TX_ENABLE Rising Edge

Similarly, the guard time between RX_ENABLE falling edge and TX_ENABLE rising edge makes sure the receiver and the transmitter AFE are not powered up simultaneously. As discussed in the previous sections, after RX_ENABLE falling edge, it takes $t_{RxEnaFall2Off}$ to power off the receiver AFE. Therefore, the earliest time transmitter AFE can be powered up is $t_{RxEnaFall2Off}$ after the RX_ENABLE falling edge. Because it takes $t_{TxEnaRise2On}$ to power up the transmitter AFE starting from the TX_ENABLE rising edge, the minimum guard time is $t_{RxEnaFall2Off}$

– $t_{TxEnaRise2On}$ if $t_{RxEnaFall2Off}$ is greater than $t_{TxEnaRise2On}$. For $t_{RxEnaFall2Off}$ is less than $t_{TxEnaRise2On}$, TX_ENABLE rising edge can happen $t_{TxEnaRise2On} - t_{RxEnaFall2Off}$ before TX_ENABLE falling edge. Figure 72 describes both cases.

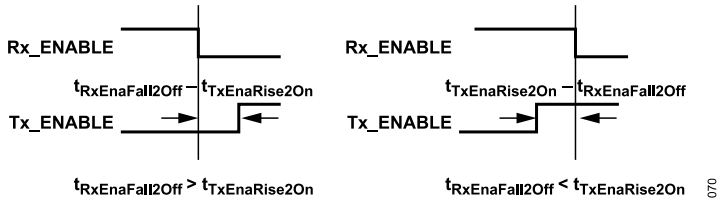


Figure 72. Minimum Guard Time Between RX_ENABLE Falling Edge and TX_ENABLE Rising Edge

Guard Time Between TX_ENABLE Falling Edge and TX_ENABLE Rising Edge

The guard time between TX_ENABLE falling edge and TX_ENABLE rising edge makes sure the interface is turned off at the end of the previous frame before it turns on again for the next frame. In addition, it must also make sure the AFE is powered off in the previous frame before powering up again in the new frame. Because it takes $t_{TxEnaHold}$ to turn off the transmit interface after the TX_ENABLE falling edge, the next TX_ENABLE rising edge must come after a delay of at least $t_{TxEnaHold}$. This ensures the interface is turned off at the end of the previous frame before it turns on again for the next frame. As it takes $t_{TxEnaFall2Off}$ to power down the transmitter AFE after the TX_ENABLE falling edge, the next TX_ENABLE rising edge must come after a delay of at least equal to $t_{TxEnaFall2Off} - t_{TxEnaRise2AnaOn}$. This ensures the AFE is powered off in the previous frame before powering up again in the new frame. If the timing parameters are set appropriately, these two conditions are almost identical. If they are not identical for some reason, set the guard time as the maximum of $t_{TxEnaHold}$ and $t_{TxEnaFall2Off} - t_{TxEnaRise2AnaOn}$. Figure 73 describes this scenario.

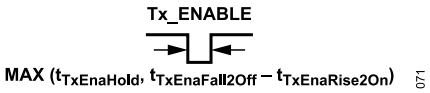


Figure 73. Minimum Guard Time Between TX_ENABLE Falling Edge and TX_ENABLE Rising Edge

Guard Time Between RX_ENABLE Falling Edge and RX_ENABLE Rising Edge

The guard time between the RX_ENABLE falling edge and RX_ENABLE rising edge makes sure the interface is turned off at the end of the previous frame before it turns on again for the next frame. Because it takes $t_{RxEnaHold}$ to turn off the receive interface after the RX_ENABLE falling edge, the next RX_ENABLE rising edge must come after a delay of at least $t_{RxEnaHold}$. This ensures the interface is turned off at the end of the previous frame before it turns on again for the next frame. Because the analog powers down before the interface, the AFE is guaranteed to power down before being powered up at the start of the next frame if this condition is met. Figure 74 describes this scenario.

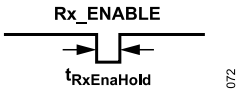


Figure 74. Minimum Guard Time Between RX_ENABLE Falling Edge and RX_ENABLE Rising Edge

Hold Time Between TX_ENABLE Rising Edge and TX_ENABLE Falling Edge

After a TX_ENABLE rising edge, its falling edge must come after a delay of at least $t_{TxEnaRise2AnaOn}$ or $t_{TxEnaRise2On}$ (if controlling the antenna switch). To actually transmit, the channel must be on for a duration longer than its propagation delay. Achieve this either by making sure that TX_ENABLE is high for longer than the propagation delay, or by ensuring the $t_{TxEnaHold}$ and $t_{TxEnaFall2Off}$ are longer than t_{TxPD}. Figure 75 describes this scenario.

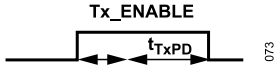


Figure 75. Minimum Hold Time Between TX_ENABLE Rising Edge and TX_ENABLE Falling Edge

MICROPROCESSOR AND SYSTEM CONTROL

Hold Time Between RX_ENABLE Rising Edge and RX_ENABLE Falling Edge

After a RX_ENABLE rising edge, its falling edge must come after a delay of at least $t_{RxEnaRise2AnaOn}$ or $t_{RxEnaRise2On}$ (if controlling LNA power). To actually receive data, the channel must be on for a duration longer than its propagation delay. Achieve this either by making sure that RX_ENABLE is high for longer than the propagation delay or by ensuring the $t_{RxEnaHold}$ is longer than t_{RxPD}. Figure 76 describes this scenario.

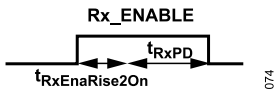


Figure 76. Minimum Hold Time Between RX_ENABLE Rising Edge and RX_ENABLE Falling Edge

Timing Parameters with Power Savings Modes

The ADRV9001 offers several channel power savings modes (Power Saving Mode 0, 1, and 2) that trade off better power savings with longer transition time to turn on and turn off a transmit or receive channel. For more details about power saving modes, see the Power Savings and Monitor Mode section. To take advantage of the power saving modes, set the timing parameters appropriately.

Note that the minimum guard time discussed previously does not consider the time taken to power down the transmit or receive analog by assuming it is insignificant. But it is highly recommended to allow extra time to make sure the analog power-up happens only after the analog power-down is fully completed. The analog power-down time is usually much less than the analog power-up time.

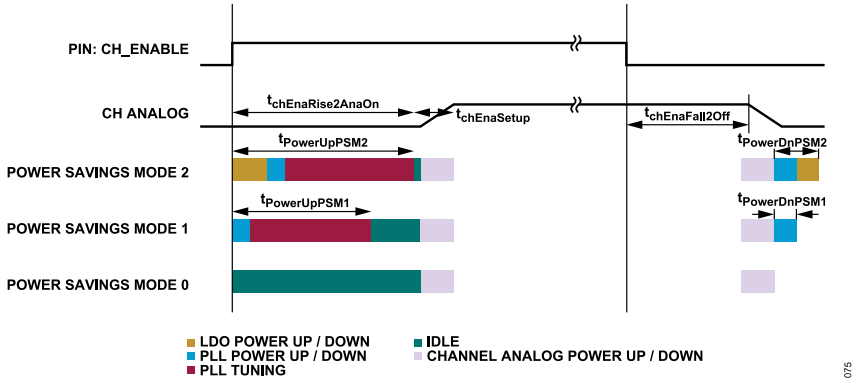


Figure 77. Channel Power-Up and Power-Down Sequences in Different Power Savings Modes

Figure 77 shows the sequence of events to power up or down a transmit or receive channel in the various channel power savings modes. In Power Savings Mode 1 and 2, the enableRiseToAnalogOnDelay is used to power up additional entities powered down at the end of the previous frame. (Note that in Power Savings Mode 1 and 2, PLL is powered down at the end of the previous frame. Therefore, when it is turned on at the start of the new frame, PLL tuning is required.) Thus, set the enableRiseToAnalogOnDelay long enough to allow the power up procedures to complete. If the additional power-up procedures in Power Savings Mode 2 take $t_{PowerUpPSM2}$ to complete, the ADRV9001 prevents the system from entering the Power Savings Mode 2, unless enableRiseToAnalogOnDelay is set greater than $t_{PowerUpPSM2}$. The same is true for Power Savings Mode 1. The ADRV9001 prevents the system from entering Power Savings Mode 1, unless enableRiseToAnalogOnDelay is set greater than $t_{PowerUpPSM1}$. In Power Savings Mode 0, which is the default mode, there are no additional power-up procedures. Thus, there are no additional restrictions on enableRiseToAnalogOnDelay other than those already specified in earlier sections.

If switching dynamically between several power savings modes, set the enableRiseToAnalogOnDelay to satisfy the restrictions of the highest power savings mode. Figure 77 shows that there is a longer idle time when switching to a lower power savings mode. The parameter enableRiseToAnalogOnDelay cannot be changed dynamically. Thus, the timing of the TX_ENABLE/RX_ENABLE rising edge relative to the on air time must also remain the same even when dynamically switching between different power savings modes.

In certain use cases, when the transmit and receive are using the same LO, but at different frequencies, if the transition time between the transmit and receive frames are always long enough, PLL tuning at the start of the frame. This is not related to any specific power saving mode and PLL tuning happens even in Power Saving Mode 0. Figure 78 shows the timing diagram.

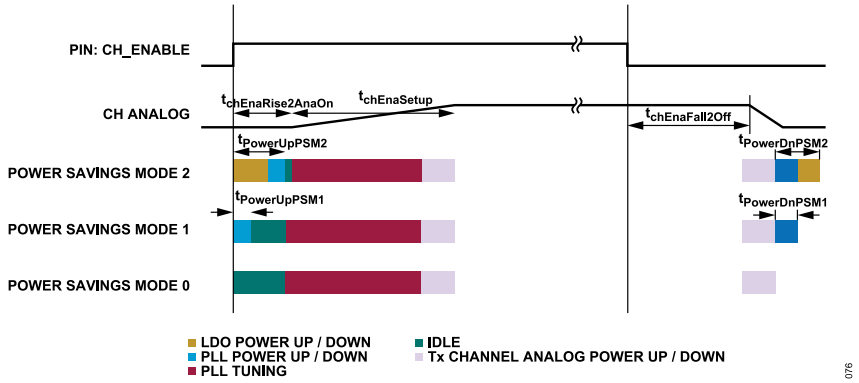


Figure 78. Channel Power-Up and Power-Down Sequence in Different Power Savings Modes (PLL Re-Tune at Frame Boundary Case)

In this case, in all power saving modes, the PLL tuning is performed during enableSetupDelay instead of enableRiseToAnalogOnDelay. Therefore, enableSetupDelay is much longer as it must allow time to tune the PLL. This means that the additional power-up duration $t_{PowerUpPSM}$ is much shorter, and thus higher power savings can be achieved while setting the enableRiseToAnalogOnDelay to a much smaller value.

All the previously-mentioned descriptions are for internal LO scenarios. If the ADRV9001 is configured with the external LO mode, take the responsibility to configure or retune the external PLLs. The ADRV9001 channel power-up and power-down sequence in different power saving modes are same as in Figure 77. Make sure the external LOs are ready before the enableRiseToAnalogOnDelay ends.

Impact of Power Savings on Timing Parameter Selection

As explained in the previous section, certain power savings modes cannot be entered if the enableRiseToAnalogOnDelay for that channel is not greater than the duration of the additional power-up procedures needed in that mode.

For the transmit channels, if the propagation delay is quite large, the enableRiseToAnalogOnDelay chosen is already larger than the longest power-up procedure duration, i.e. , $t_{PowerUpPSM2}$. In this case, there is no impact on the selection of the timing parameters.

For the receive channels, or transmit channels with short propagation delays, choose the enableRiseToAnalogOnDelay larger than $t_{PowerUpPSM1}$ to enter Power Savings Mode 1 and larger than $t_{PowerUpPSM2}$ to enter Power Savings Mode 2 and higher. The enableRiseToOnDelay, if it is being used, must also increase as it must always be larger than enableRiseToAnalogOnDelay. However, none of the other timing parameters are affected by the power savings mode.

At the end of the frame, the power-down procedures take some small but finite time. For the receiver channels with large propagation delay, this has no impact because the digital datapath might be on for a long time after the analog has powered down.

For the transmit channels or receive channels with short propagation delays, the minimum period between the channel enables falling edge and the next rising edge must be enableHoldDelay plus the additional time needed for the extra power down procedures ($t_{PowerUpPSM1}$, $t_{PowerUpPSM2}$). This prevents PLL or LDO from beginning power up in the new frame even before it has finished powering down in the old one.

Hardware and Software Restrictions for Timing Parameters

As previously mentioned, the bounds provided for each of these timing parameters and the guard times between rising and falling edges of the receiver and transmitter enable signals are only guidelines. There are almost no hardware or software restrictions preventing from setting these parameters anyway, including harmful or useless ways. There are a few restrictions, however, which are outlined as follows:

- All provided timing parameters must be within the range of 0 ms to 91 ms. These bounds are specified, assuming the delay generation blocks run at 184.32 MHz (system clock). If operating at a different frequency, the maximum bound scales accordingly. For example, if using a 160 MHz clock, the maximum delay is 91 ms/184.32 × 160 = 79 ms.
- For all channels, the enableRiseToOnDelay must be greater than or equal to the enableRiseToAnalogOnDelay, provided the enableRise-ToOnDelay parameter is being used, i.e., the ADRV9001 is controlling antenna switch and/or LNA power.
- For the transmitter channels, the enableHoldDelay must be less than or equal to the enableFallToOffDelay.
- For the receiver channels, the enableFallToOffDelay must be less than or equal to the enableHoldDelay.
- For a specific channel, Power Savings Mode 2 or higher is disallowed when the enableRiseToAnalogOnDelay is less than $t_{PowerUpPSM2}$.

- For a specific channel, Power Savings Mode 1 or higher is disallowed when the enableRiseToAnalogOnDelay is less than $t_{PowerUpPSM1}$.

API Programming and Default Values for Timing Parameters

There is a set of API commands to configure the timing parameters. Because the timing parameters are related to the channel power saving mode, set the channel power saving mode first before configuring the timing parameters. The API command **adi_adrv9001_powerSavingAndMonitorMode_ChannelPowerSaving_Configure( )** sets the channel power saving mode for a specified channel when the channel is in the standby or calibrated state. After that, use the API command **adi_adrv9001_Radio_ChannelEnablementDelays_Configure( )** to configure the timing parameters for the selected channel. The following data structure holds all the ADRV9001 required timing parameters:

typedef struct adi_adrv9001_ChannelEnablementDelays

```
{
uint32_t riseToOnDelay;         /* Delay from rising edge until antenna switch (Tx) or LNA (Rx) is
powered up */
uint32_t riseToAnalogOnDelay;   /* Delay from rising  edge until Tx/Rx analog power up procedure
commences */
uint32_t fallToOffDelay;        /* Delay from falling edge until antenna switch (Tx) or LNA (Rx) is
powered down */
uint32_t guardDelay;            /* Reserved for future use*/
uint32_t holdDelay;             /* Delay from falling edge until the Tx/Rx interface is disabled */
} adi_adrv9001_ChannelEnablementDelays_t
```

Note that the ADRV9001 reserves guardDelay for future use and forces it to 0 for both transmit and receive channels. In addition, for the transmit channel, reserve holdDelay for future use and force it to 0. For the receive channel, reserve fallToOffDelay for future use and force it to 0. Call the API command **adi_adrv9001_Radio_ChannelEnablementDelays_Configure( )** when the channel is in the standby or calibrated state.

To set all those timing parameters properly, gather prior knowledge about the ADRV9001 timing parameters (ADRV9001 provides to the user) as well as helping parameters such as the transmit and receive propagation delay. The prior timing parameters include enableSetupDelay, propagationDelay, and maximum intended power savings mode, $t_{PowerUpPSM1}$, and $t_{PowerUpPSM2}$.

Table 36 summarizes all these timing parameters for both transmit and receive. Note that all timing parameters specified in units of time assume a system clock frequency of 184.32 MHz. If using a different system clock frequency, adjust it by:

scaleFactor = 184.32 (MHz)/*system clock Frequency*

Table 36. Prior Tx/Rx Timing Parameters

	No PLL Retuning at Frame Boundary (Use Case in Figure 77)	PLL Retuning at Frame Boundary (Use Case in Figure 78)
enableSetupDelay	Analog Power-Up*scaleFactor	PLL Tuning + Analog Power-Up *scaleFactor
propagationDelay	From user's own measurement	Same as No PLL tuning case
$t_{PowerUpPSM1}$	PLL Tuning + PLL Power-Up *scaleFactor	PLL Power-Up *scaleFactor
$t_{PowerUpPSM2}$	PLL Tuning + LDO Tuning + PLL Power-Up *scaleFactor	LDO Tuning + PLL Power-Up *scaleFactor

The system clock frequency depends on the profile, and the corresponding value is found under the **TDD Enablement Delays** tab in TES. In addition, TES also displays the timing parameters provided by the ADRV9001 to determine the prior transmit/receive timing parameters, as described in Figure 79, which shows the picture of TES where those timing parameters and the system clock for the user-selected profile are located.

MICROPROCESSOR AND SYSTEM CONTROL

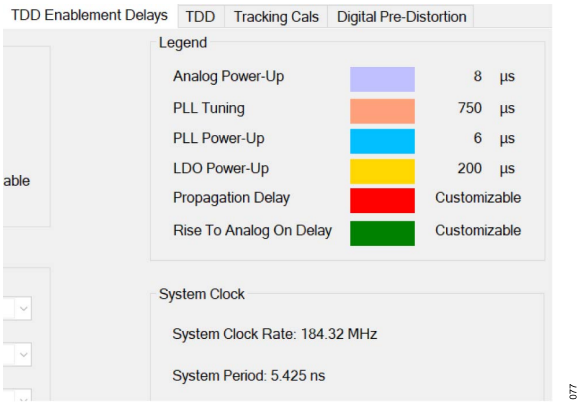


Figure 79. ADRV9001 Provided Timing Parameters and the System Clock for the Selected Profile in TES

Based on the information provided in Figure 79, further configure the ADRV9001 required timing parameters.

Default Timing Parameters for Transmit Channels

Figure 80 shows the ADRV9001 transmitter's required timing parameters and their minimum, maximum, and default values, and Table 37 summarizes some recommendations.

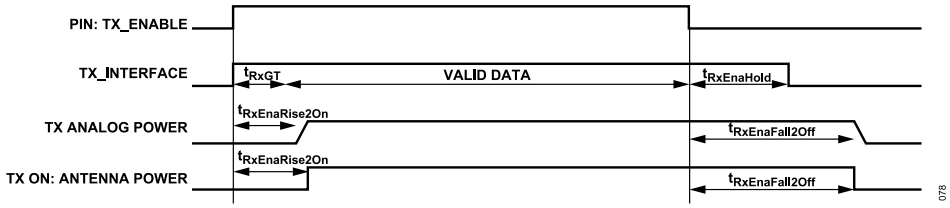


Figure 80. Transmit Timing Parameters

Table 37. ADRV9001 User Provided Transmit Timing Parameters

Timing Parameter	Min Value	Max Value	Comment
enableRiseToAnalogOnDelay ($t_{TxEnaRise2AnaOn}$)	Max of the following values: 0, propagationDelay – enableSetupDelay, $t_{PowerUpPSM}$ (for the maximum intended power savings mode)	91 ms/scaleFactor	Default = min
enableRiseToOnDelay ($t_{TxEnaRise2On}$)	enableRiseToAnalogOnDelay + enableSetupDelay	91 ms/scaleFactor	Default = min Not needed if not controlling LNA power.
enableGuardDelay (t_{TxGT}) (Not Used Currently)	0	91 ms/scaleFactor	Default = min Can increase to non-zero if performance degradation is observed, and the channel is transmitting for some time before the start of the actual frame (forced to 0 currently by the ADRV9001).
enableHoldDelay ($t_{TxEnaHold}$) (not used currently)	0	91 ms/scaleFactor	Default = min Can increase if performance degradation is observed, and the channel is transmitting for some time after the end of the actual frame (forced to 0 currently by the ADRV9001).
enableFallToOffDelay ($t_{TxEnaFall2Off}$)	enableHoldDelay	enableHoldDelay + propagationDelay	Default = max Can decrease if not all data sent through interface must be transmitted.

MICROPROCESSOR AND SYSTEM CONTROL

Default Timing Parameters for Receiver Channels

Figure 81 shows the ADRV9001 receiver's required timing parameters and their minimum, maximum, and default values. Table 38 summarizes some recommendations.

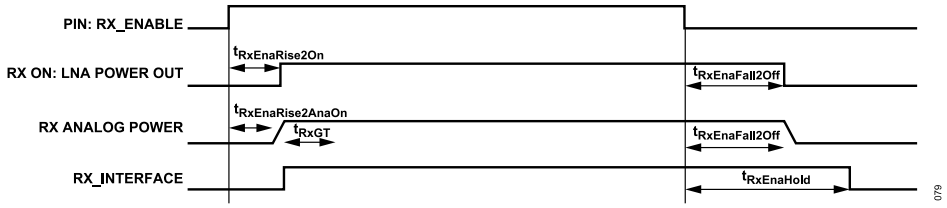


Figure 81. Receiver Timing Parameters

Table 38. User Provided Receiver Timing Parameters

Timing Parameter	Min Value	Max Value	Comment
enableRiseToAnalogOnDelay ($t_{RxEnaRise2AnaOn}$)	Max of the following values: 0, $t_{PowerUpPSM}$ (for the maximum intended power savings mode)	91 ms/scaleFactor	Default = min
enableRiseToOnDelay ($t_{RxEnaRise2On}$)	enableRiseToAnalogOnDelay + enableSetupDelay	91 ms/scaleFactor	Default = min Not needed if not controlling LNA power.
enableGuardDelay (t_{RxGT}) (not used currently)	0	91 ms/scaleFactor	Default = min Can increase if performance degradation is observed, and the channel is receiving for some time before the start of the actual frame (forced to 0 currently by the ADRV9001).
enableFallToOffDelay ($t_{RxEnaFall2Off}$) (not used currently)	0	91 ms/scaleFactor	Default = min Can increase if performance degradation is observed, and the channel is receiving for some time after the end of the actual frame. Can still be used if not controlling LNA power- down (forced to 0 currently by the ADRV9001).
enableHoldDelay ($t_{RxEnaHold}$)	enableFallToOffDelay	enableFallToOffDelay + propagationDelay	Default = max Can decrease if not all data received on air must be sent over the interface.

When the ADRV9001 calculates the default values, it uses the transmit/receive propagation delay internally characterized for different profiles. Measure the propagation delay for the entire system to determine all the required timing parameters accurately. This includes a power amplifier, LNA, filters, and anything else that might be between the chip and antenna. During the measurement, set all the timing parameters as the default values for simplification, and it is also important to use the same profile and configurations as the actual application being deployed.

An example of measuring the receive propagation delay: A typical system level measurement can be done with an RF switch controllable from the BBIC. Switch off the switch and turn on the receiving signal and switch on the switch and start recording data from the interface. This signal can be a pattern from a signal generator with an external sync. Then the BBIC can use the external sync to start recording and obtain a very accurate measurement.

After calculating all the timing parameters required by the ADRV9001, configure them through TES, as shown in Figure 82.

MICROPROCESSOR AND SYSTEM CONTROL

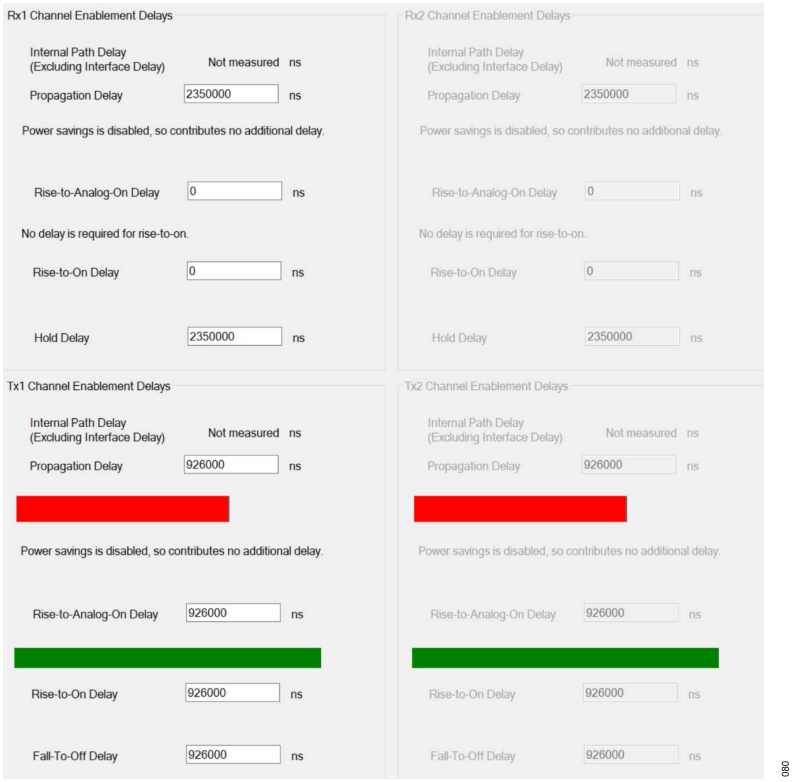


Figure 82. Timing Parameters Configuration in TES

Figure 82 shows that only relevant channels are enabled to configure the timing parameters. Enter all the values in ns. The propagation delay is a helper parameter, which is not needed by the ADRV9001. It helps to set the other timing parameters for the ADRV9001.

As previously mentioned, use the API command **adi_adrv9001_Radio_ChannelEnablementDelays_Configure( )** to set the timing parameters. For more details, refer to the API Doxygen document.

Pin Control Mode Timing Measurement

The ADRV9001 receives transmitter/receiver enable control signals and related timing parameters to perform the transmitter/receiver RF On/Off operation in the pin control mode. As previously mentioned, the enableSetupDelay (EnaSetup) is the time taken for the ADRV9001 to power up its AFE, and the ADRV9001 also takes less time to finish the operations to power down its AFE at the falling edge of the transmitter/receiver enable signals. It is advised that the programmable “EnaRise2AnaOn” and “EnaFall2Off” delay the ADRV9001 analog components power on and off, respectively. The enableSetupDelay varies with different ADRV9001 internal processor clock rate (System Clock). Enable the ADRV9001 internal stream status to GPIO debug function to accurately measure the transmitter/receiver enable control signal and when they are actually taken effective by the ADRV9001 internal stream processor.

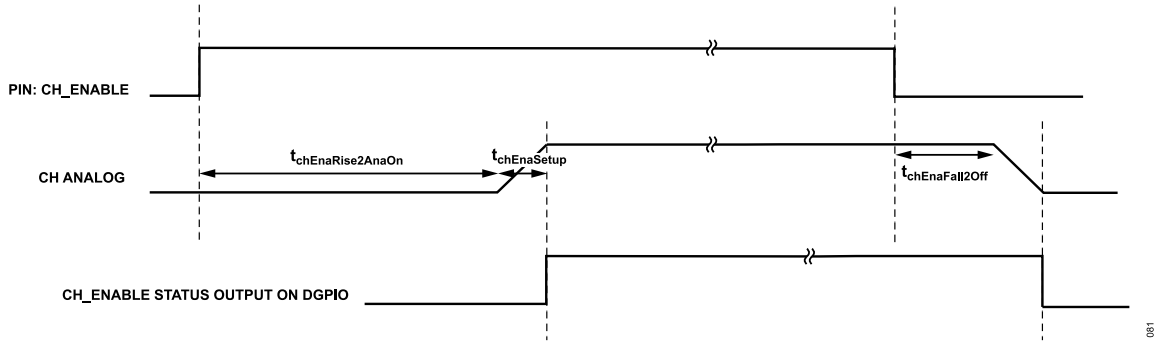


Figure 83. Channel Enable Control Status Output on DGPIO

MICROPROCESSOR AND SYSTEM CONTROL

Figure 83 shows the channel enable signal and its corresponding active output timing on DGPIO. The DGPIO high indicates the start of the channel AFE fully on up to the end of the channel AFE fully off. If the EnaRise2AnaOn is set to 0, the time from CH_ENABLE rising edge to DGPIO rising edge is the “enableSetupDelay.” Similarly, if set EnaFall2Off to 0, the time from CH_ENABLE falling edge to DGPIO falling edge is the time taken for the ADRV9001 to fully power down the AFE.

Enable this stream status to GPIO debug function by API **adi_adrv9001_Stream_C0_Gpio_Debug_Set()** in the chip “Standby” or “Calibrated” state. The RX1_ENABLE output status is mapped to DPGIO_0, TX1_ENABLE output status to DGPIO_1, RX2_ENABLE output status to DGPIO_2 and TX2_ENABLE output status to DGPIO_3. There are no options to configure these DGPIOs mapping, and once the debug function is enabled, the DGPIO_0~DPGIO_3 is not available for other functions, no matter how many channels are configured in the profile. Analog Devices recommends this debug function to measure the transmitter/receiver ENABLE timing for a given profile, and disable this function to release the DGPIOs usage once measuring is done.

API EXECUTION TIMING

The ADRV9001 APIs have several categories in how they interact between the BBIC and the ADRV9001. Most APIs communicate with the ADRV9001 through the mailbox, where the ADRV9001 ARM processor schedules tasks. These mailbox type APIs are usually not time critical and can take tens of microseconds up to several milliseconds, depending on the real-time operating system (RTOS) task status. There are also time-sensitive APIs that use faster methods of communication, either with higher priority in the mailbox or through software interrupt. Some APIs do not communicate with the ARM processor and instead directly read or write to SPI registers on the ADRV9001. Table 39 further details these API categories. All APIs have the category type listed in their source code. For the timing of API execution, see the Measurement Method section.

Table 39. API Timing Categories

API Category	Description
Direct	Direct ADRV9001 register access: API calls that bypass the ADRV9001 processor and directly write or read SPI registers.
Mailbox	ADRV9001 mailbox commands: API calls that communicate through normal mailbox commands to the ADRV9001 on-board processor.
Priority Mailbox	High-priority ADRV9001 mailbox commands: API calls that communicate through high-priority mailbox commands to the ADRV9001 on-board processor.
Fast Messaging	Fast-messaging: API calls that communicate through fast message to the ADRV9001 on-board processor for more deterministic timing.
Interrupt	Software interrupt: API calls that communicate through software interrupt to the ADRV9001 on-board processor.

CLOCK GENERATION

In the ADRV9001, the CLKGEN generates all clocks for the converters and main digital. CLKGEN receives from two clocks, a high performance (HP) clock PLL and low power (LP) clock PLL. The high performance clock PLL has a programmable frequency range of 7.2 GHz to 8.8 GHz. The low power clock PLL can generate a programmable frequency range of 3.3 GHz to 5 GHz. CLKGEN also has the clocks divided and retimed with reset pulses from the clock PLLs. Figure 85 shows that the CLKPLL can be programmed to provide arbitrary sample rate.

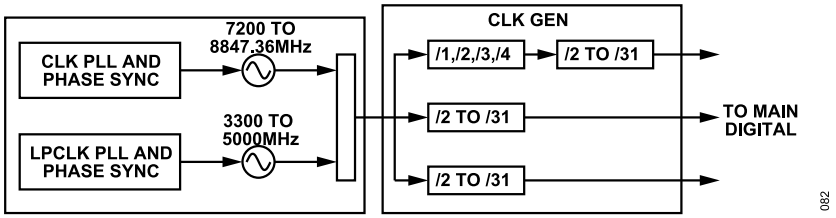


Figure 84. ADRV9001 Clock Generation

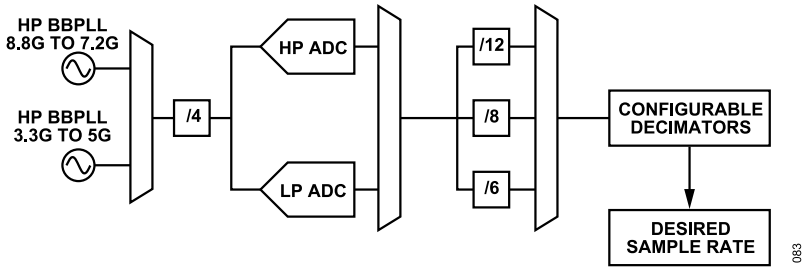


Figure 85. CLKPLL Can Be Programmed to Provide Arbitrary Sample Rate

LOW POWER CLOCK PHASE-LOCK LOOP (LP CLKPLL)

By default, LP CLKPLL works at a fixed frequency 4423.68 MHz. However, it can be set to a tuning operation range. The operating frequency range of LP CLKPLL is from 3.3 GHz to 5 GHz. Provide the final sampling frequency only at the interface, and the final frequency of LP CLKPLL is determined internally, all before the chip is programmed. A profile is generated based on the provided sampling frequency.

Note that LP CLKPLL uses less power than HP CLKPLL but has a smaller tuning range, which cannot be used for some profiles. Table 40 lists the supported data sample rates with different standards by LP CLKPLL. Table 41 lists the supported data lane rate by LP CLKPLL. Note that the LTE 40 MHz at 16-bit is not supported by the LP CLKPLL.

Table 40. Sample Rate Supported by LP CLKPLL

Standard	Sample Rate
DMR I/Q	2.40E + 04
TETRA	1.44E + 05
TETRA	2.88E + 05
LTE 1.5	1.92E + 06
LTE 3	3.84E + 06
LTE 5	7.68E + 06
LTE 10	1.54E + 07
LTE 15	2.30E + 07
LTE 20	3.07E + 07
LTE 40 at12 bits	6.14E + 07

Table 41. Supported Data Lane Rate by LP CLKPLL

Standard	Serialization Factor Per Data Lane	Data Lane Rate
DMR/P25 Direct Modulation	2	9.60E +03
P25 Direct Modulation	2	1.20E +04
FM Direct Modulation	16	1.28E +05
DMR I/Q	32	7.68E + 05
	8	1.92E + 05

CLOCK GENERATION

Table 41. Supported Data Lane Rate by LP CLKPLL (Continued)

Standard	Serialization Factor Per Data Lane	Data Lane Rate
	16	3.84E + 05
FM Direct Modulation	16	1.54E + 06
TETRA	32	4.61E + 06
	8	1.15E + 06
	16	2.30E + 06
TETRA	32	9.22E + 06
	8	2.30E + 06
	16	4.61E + 06
LTE 1.5	32	6.14E + 07
	8	1.54E + 07
	16	3.07E + 07
LTE 3	8	3.07E + 07
	16	6.14E + 07
LTE 5	8	6.14E + 07
	16	1.23E + 08
LTE 10	16	2.46E + 08
LTE 15	16	3.69E + 08
LTE 20	16	4.92E + 08
LTE 40 at 12 bits	12	7.37E +08

ARBITRARY SAMPLE RATE

With a programmable frequency range of both HP CLK PLL and LP CLK PLL (as mentioned previously), the ADRV9001 supports the arbitrary sample rate (ASR) mode, which provides great flexibility to configure the desired sample rates in the applications. The ASR mode supports an almost continuous range of rates up to 61.44 MHz, with a list of dead zones mainly due to the limitations in the decimation/interpolation rate supported in the datapath. Table 42 and Table 43 summarize the current dead zone frequency ranges:

Table 42. Narrowband Dead Zones

Dead Zone	Lower Bound (kSPS)	Upper Bound (kSPS)	CMOS SDR 1 Lane		CMOS SDR 4 Lane		CMOS DDR 4 Lane		LVDS DDR	
			Custom	Custom Extended	Custom	Custom Extended	Custom	Custom Extended	Custom	Custom Extended
1	195.4	208.3	x	x	x	x	x	x	x	x
2	522	554	x	✓	✓	✓	✓	✓	✓	✓
3	781.3	833.3	x	✓	x	✓	x	✓	x	✓

Table 43. Wideband Dead Zones

Dead Zone	Lower Bound (MSPS)	Upper Bound (MSPS)	CMOS SDR 1 Lane		CMOS SDR 4 Lane		CMOS DDR 4 Lane		LVDS DDR	
			Custom	Custom Extended	Custom	Custom Extended	Custom	Custom Extended	Custom	Custom Extended
4	1.05	1.1	x	x	x	x	x	x	x	x
5	1.1	1.18	x	x	✓	✓	✓	✓	✓	✓
6	1.18	1.2	x	✓	✓	✓	✓	✓	✓	✓
7	1.6	1.65	x	✓	x	✓	x	✓	x	✓
8	2.1	2.2	x	x	x	x	x	x	x	x
9	2.2	2.35	x	x	✓	✓	✓	✓	✓	✓
10	2.35	2.4	x	✓	✓	✓	✓	✓	✓	✓
11	3.15	3.3			x	✓	x	✓	x	✓
12	4.2	4.4			x	x	x	x	x	x
13	4.4	4.7			x	x	✓	✓	✓	✓

CLOCK GENERATION

Table 43. Wideband Dead Zones (Continued)

Dead Zone	Lower Bound (MSPS)	Upper Bound (MSPS)	CMOS SDR 1 Lane		CMOS SDR 4 Lane		CMOS DDR 4 Lane		LVDS DDR	
			Custom	Custom Extended	Custom	Custom Extended	Custom	Custom Extended	Custom	Custom Extended
14	4.7	4.95			x	✓	✓	✓	✓	✓
15	6.3	6.65			x	✓	x	✓	x	✓
16	8.35	8.8			x	x	x	x	x	x
17	8.8	9.95			x	x	x	x	✓	✓
18	12.55	13.3					x	✓	x	✓
19	16.7	18.75					x	x	x	x
20	18.75	19.95					x	✓	x	✓
21	25.1	26.6							x	✓
22	33.4	37.5							x	x
23	37.5	39.9							x	✓
24	50.1	53.3							x	✓

In Table 42 and Table 43 , an “x” means the sample rate is not supported and a “✓” means the sample rate is supported. For each dead zone, the sample rates that cannot be configured are between the lower and upper bounds with the lower and upper bound frequencies excluded, which means the dead zone = (lower bound, upper bound). If a sample rate is not in the range of any of the 24 dead zones, then the sample rate is supported.

As described in the Data Interface section, there are several modes of operation of the SSI. Each mode of operation has different dead zones. For example, the sample rate of 2.3 MSPS falls into the dead zone for CMOS SDR 1 Lane mode but not for CMOS SDR 4 Lane, CMOS DDR 4 Lane, or LVDS DDR modes. If the sample rate of 2.3 MSPS is desired, it is still configurable under those modes.

The “Custom” and “Custom Extended” modes refer to the options to configure the sample rate in TES. The custom mode allows to set an arbitrary sample rate. The custom extended mode also allows to set an arbitrary sample rate with a wider range of supported sample rates by allowing the system clock to run as slow as 150 MHz. This imposes certain constraints on the system. If AGC is enabled, there should be an analog RF band-limiting filter before the Rx input with a bandwidth equal to half the system clock frequency.

The dead zones were determined by sweeping over the full range of sample rates to determine where the dead zones occur. For narrowband sample rates, a sweep was done over the range of 24 kSPS to 1 MSPS at 2 kSPS increments with an RF bandwidth of 1/3 of the sample rate. For wideband sample rates, a sweep was done over the range of 1 MSPS to 61.44 MSPS in 0.1 MSPS steps with an RF bandwidth of ½ of the sample rate.

Due to the complexity of the datapath, there is no freedom to select own CLK PLL clock frequencies and datapath configurations. Provide the desired sample rate only, and the ADRV9001 TES determines the appropriate ADRV9001 profile.

Note: **The above table provides a theoretical guideline of the dead zones. Double check the validity of the sample rates using the TES. The results from the TES are considered final**.

MULTICHIP SYNCHRONIZATION

INTRODUCTION

MCS is needed when an application requires deterministic latency among datapaths within one ADRV9001 device or multiple ADRV9001 devices. For example, in MIMO applications, MCS is the solution to align the data in time for multiple channels. Some applications not only require the delay to be deterministic but also require the phase to be the same. The ADRV9001 supports PLL phase synchronization in addition to the deterministic delay.

THEORY OF OPERATION

Figure 86 illustrates the synchronization among multiple ADRV9001 devices using the MCS pin. The external clock chip (for example, AD9528) generates the MCS signal using the device clock DEV_CLK, and each ADRV9001 device captures it using the negative or positive edge of DEV_CLK to meet setup and hold time with good margins. Each ADRV9001 device uses this sampled MCS to synchronize all internally generated clocks, which makes them align among all devices' internal clocks. Set the ADRV9001 MCS input port to the LVDS or CMOS mode.

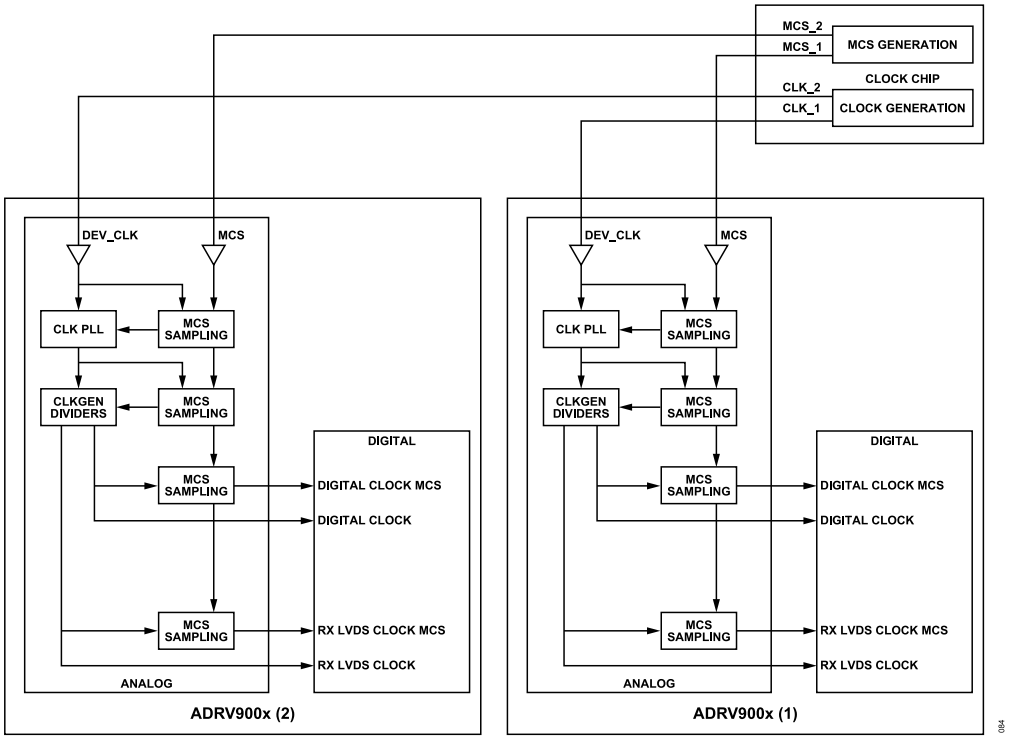


Figure 86. Multichip Synchronization System Diagram

Sampled MCS

The ADRV9001 samples the MCS pulse signal internally by the DEV_CLK signal rising or falling edge. Figure 87 shows the example of an MCS pulse sampled by the rising edge of the DEV_CLK. This process guarantees that the sampled MCS signal, which is used to synchronize all ADRV9001 devices, is time aligned.

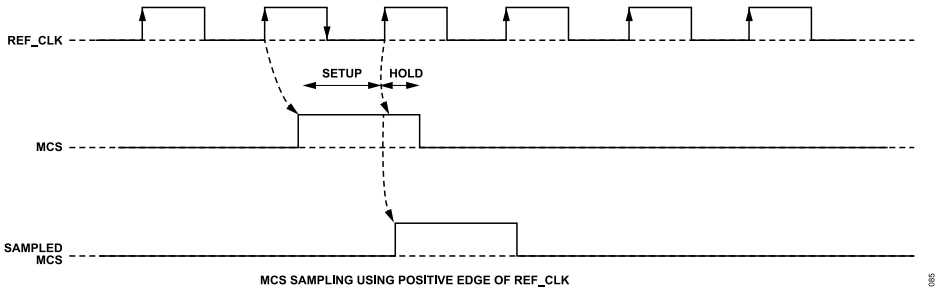


Figure 87. Sample MCS Signal at Rising Edge of DEV_CLK

MULTICHIP SYNCHRONIZATION

An external clock module is required to generate the DEV_CLK and MCS pulses for multiple ADRV9001 devices. Each ADRV9001 receives a DEV_CLK and an MCS signal. The MCS signals should arrive at all ADRV9001 devices within one DEV_CLK cycle as they must be sampled by the DEV_CLK, as mentioned in Figure 87. So, it is recommended that the layout has equal-length traces between the external clock module and each ADRV9001 device. Carefully tune the external clock module so that the pulses arrive at all ADRV9001 devices within one clock cycle time.

The minimum MCS setup time requirement is 0.62 ns for the LVDS mode and 1 ns for the CMOS mode. The minimum hold time requirement for the LVDS mode is 0 ns and 3 ns for the CMOS mode.

MCS Pulses

Figure 88 shows the MCS signal that must be received by the ADRV9001. There are six pulses. The first four pulses are for the analog clock divider synchronization, and the last two are for the digital clock divider and SSI synchronization. Together, they synchronize all internal components of the ADRV9001.

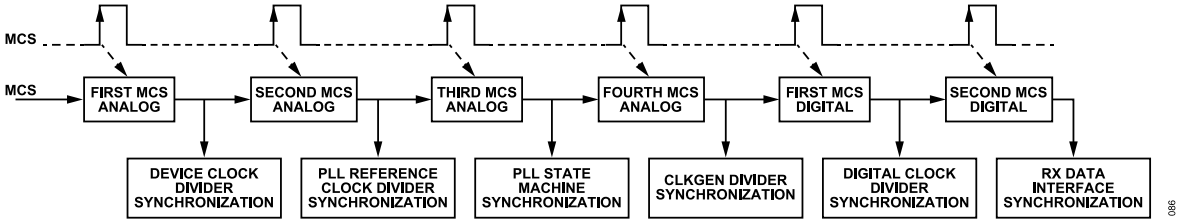


Figure 88. MCS Pulses for Analog and Digital Synchronization

Pulse Width and Delay

Table 44 shows the minimum pulse width of each MCS pulse, as well as the wait time required after each pulse. Use this reference to design MCS pulse generation.

Table 44. Minimum Time Requirement for MCS Pulse Width and Wait Time

Pulse No.	Pulse Width (No. of Reference Clock Cycles)	Wait Time After the Pulse Tn (µs)
1	≥2	>1
2	≥2	>1
3	≥2	>1
4	≥2	>100
5	≥2	>100
6	≥2	>1

MCS and Phase Synchronization

Enabling MCS guarantees the delay between the RF ports to the SSI (or reverse direction) to be deterministic, across all channels on one or multiple ADRV9001 devices.

Additionally, the ADRV9001 also provides phase synchronization for the PLLs across multiple devices. Enable this option so that the data is synchronized in time, and the phases of the PLLs are also synchronized.

Note: If choosing the MCSMODE_ENABLED mode, which does not guarantee phase synchronization, the process is done only once after the device is initialized and at the calibrated state for the first time. This means after all MCS pulses are sent, and all ADRV9001 components are synchronized, no more pulses are needed. The MCS is complete and no longer needs to run again unless the chip is reset.

If the MCSMODE_ENABLED_WITH_RFPLL_PHASE mode is selected MCS is done at the initialization stage like the MCSMODE_ENABLED mode, and once complete, it guarantees phase synchronization and no longer requires MCS pulses. Whenever PLL changes, perform the phase sync again to ensure the phase among all channels is synchronized, which is done automatically by the ADRV9001 device without any user interaction.

MULTICHIP SYNCHRONIZATION

To select one of the modes, use this structure:

```
typedef enum adi_adrv9001_McsMode {
ADI_ADRV9001_MCSMODE_DISABLED = 0, /*!< Multi Chip Synchronization disabled */
ADI_ADRV9001_MCSMODE_ENABLED, /*!< Multi Chip Synchronization enabled */
ADI_ADRV9001_MCSMODE_ENABLED_WITH_RFPLL_PHASE /*!< Multi Chip Synchronization enabled with RFPLL phase
*/
} adi_adrv9001_McsMode_e;
```

Frequency Hopping

For frequency hopping, choose one option from the Enum to enable MCS to have the deterministic delay or to add additional phase synchronization. For the first option, consider only the PLL settling time as there is no additional phase synchronization required. For the second option, an additional time is consumed, and this depends on the reference clock speed and the LO frequencies used for frequency hopping.

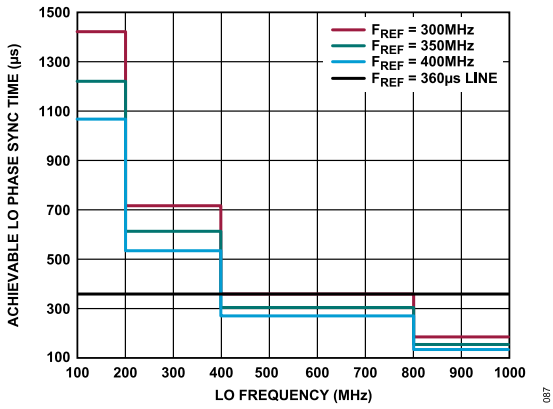


Figure 89. PLL Phase Synchronization Time vs. LO Frequency and Fref

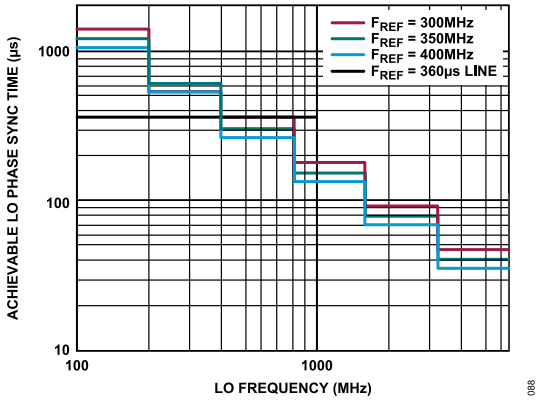


Figure 90. Theoretical Phase Sync Time up to 6GHz

Figure 89 shows the phase synchronization timing when it is required. Note: The higher the LO frequency and reference clock speed, the lower the time it requires for phase synchronization. Consider this additional timing when designing frequency hopping with phase synchronization.

Figure 90 shows the theoretical phase synchronization timing up to 6 GHz. The actual time varies but should be close to the theoretical limit.

For frequency hopping, if phase synchronization is not needed, select the MCS only mode. In this case, do not care about the phase synchronization time as it is disabled. There is no additional timing consideration because all the MCS is done at the initial stage, not at the hop stage.

MULTICHIP SYNCHRONIZATION

To have the phase synchronized, select the MCS and phase sync mode. In this case, consider phase synchronization as additional timing. As frequency hops the PLL phase changes, it takes the additional time to do phase synchronization, as mentioned. This happens at the hop stage, not at the initial stage.

MCS SUBSTATES (INTERNAL MCS STATE TRANSITION)

MCS Ready Substate

- Definition: ADRV9001 clock is switched from the CLK PLL to reference clock. MCS is initialized. Ready to receive MCS pulses.
- MCS cannot be enabled with power saving mode. So, if the MCS command is sent with power saving mode > 0, ADRV9001 returns error.
- All ADRV9001 chips can enter **MCS Ready** asynchronously. This means BBIC should wait for the ready status notified by all the ADRV9001 chips before issuing MCS pulses.

MCS Transition Substate

- Definition: ADRV9001 is in MCS pulse 1-6 transition but not finished.
- BBIC or clock chip sends MCS pulses to all the ADRV9001 chips synchronously. BBIC monitors the MCS status and restarts MCS pulses, if needed, to all the ADRV9001 chips.
- Internally, ADRV9001 keeps monitoring the MCS status. After detecting the fifth MCS pulse, switch the reference clock to the clock PLL.

MCS Done Substate

- Definition: MCS procedure is done. ADRV9001 is ready to move to the primed state.
- After receiving the sixth MCS, the substate is changed to **MCS Done**. ADRV9001 waits for the Primed command or MCS command again to rerun MCS.
- If MCS is enabled, MCS substates are kept in the **MCS Done** status after the MCS procedure is finished regardless of the channel state, unless the chip is reinitialized.

MCS PROCEDURE AND STATUS CHECK

Before issuing MCS pulses, the ADRV9001 must be in the Calibrated state. Note this is the first Calibrated state after performing initial calibrations. If there is a change to other states after the first calibrated state and then a change back to the calibrated state, MCS does not work. Two APIs, **adi_adrv9001_Mcs_Status_Get()** and **adi_adrv9001_Mcs_SwStatus_Get()**, allow the BBIC to monitor the MCS synchronization status during the entire MCS procedure. The first API verifies the synchronization status of the analog and digital subsystems of the device after issuing one or all of the MCS pulses. The second API verifies the current state of adrv9001 software after issuing one or all of the MCS pulses.

The MCS procedure and status check are described as follows:

1: Validate initial states before MCS

Validate the radio state is in the calibrated state and the device system state is not in the MCS but in normal mode (**ADI_ADRV9001_ARM_SYSTEM_NORMALMODE**) by calling **adi_adrv9001_Radio_State_Get()** API.

2: Entering MCS Ready substate

Move all the ADRV9001 channels of interest to the **MCS Ready** substate by calling **adi_adrv9001_Radio_ToMcsReady()** API. This function transitions all the ADRV9001 channels from the Calibrated state to **MCS Ready** substate. This is necessary before issuing all MCS pulses. Use **adi_adrv9001_Radio_State_Get()** to get the system state, which should be at **ADRV9001_ARM_SYSTEM_MCS** and the MCS state, which should be at **ADRV9001_ARMMCSSTATES_READY**. Use **adi_adrv9001_Mcs_SwStatus_Get()** API to make sure the MCS software state is at **ADRV9001_MCSSWSTATUS_READY**.

3: Configure and send six MCS pulses

The BBIC or clock chip configures MCS with at least six pulses as inputs to the ADRV9001 MCS pin. Table 44 specifies the timing for these pulses. This is a BBIC-specific function implemented by the user. **adi_fpga9001_Mcs_Configure()** and **adi_fpga9001_Mcs_Start()** are examples in the SDK provided for the evaluation system field programmable gate array (FPGA) board. Specify the pulse width and wait time

MULTICHIP SYNCHRONIZATION

and make sure they satisfy the minimum requirement. While MCS is running, the ADRV9001 should be in the MCS transition substate. Table 45 shows the detailed expected states after sending each pulse. Utilize this table to debug any problems that occur during the MCS procedure.

Table 45. Expected States after Each MCS Pulse

	adi_adrv9001_Radio_State_Get()		adi_adrv9001_Mcs_SwStatus_Get()	adi_adrv9001_Mcs_Status_Get()
	System State	MCS State		
After MCS PULSE 1	ARM_SYSTEM_MCS	ARMMCSSTATES_READY	MCSSWSTATUS_READY	N/A
After MCS PULSE 2	ARM_SYSTEM_MCS	ARMMCSSTATES_TRANSITION	MCSSWSTATUS_PULSE2_RECEIVED	CLK PLL (HP or LP) Reference clock divider sync completed. RF PLL (rf1 and/or rf2) reference clock divider sync completed.
After MCS PULSE 3	ARM_SYSTEM_MCS	ARMMCSSTATES_TRANSITION	MCSSWSTATUS_PULSE3_RECEIVED	CLK PLL (HP or LP) SDM clock divider sync completed. RF PLL (rf1 and/or rf2) SDM clock divider sync completed.
After MCS PULSE 4	ARM_SYSTEM_MCS	ARMMCSSTATES_TRANSITION	MCSSWSTATUS_PULSE4_RECEIVED	CLK PLL (HP or LP) digital clocks sync completed and clock gen divider sync completed. RF PLL (rf1 and/or rf2) Digital clocks sync completed and clock gen divider sync completed.
After MCS PULSE 5	ARM_SYSTEM_MCS	ARMMCSSTATES_TRANSITION	MCSSWSTATUS_DEVICE_SWITCHED_TO_HSCLK	CLK PLL (HP or LP) sync completed. RF PLL (rf1 and/or rf2) sync completed. LVDS first sync completed (if in LVDS mode). First digital sync completed.
After MCS PULSE 6	ARM_SYSTEM_MCS	ARMMCSSTATES_DONE	MCSSWSTATUS_DEVICE_SWITCHED_TO_HSCLK	LVDS second sync completed (if in LVDS mode). Second digital sync completed.

Note that the first and second digital sync and the LVDS first and second sync complete to become true after the first pulse, which is ignored. They are reset to false after the fourth pulse. The states after the fourth pulse reflect the correct expected states.

4: MCS done

After all six pulses are received, MCS is finished. At this point, ADRV9001 is in the **MCS Done** substate. Start to prime the channels to prepare for normal operations.

SAMPLE DELAY AND READ DELAY

The ADRV9001 also supports the scenario where the data coming from/going to the SSI for multiple channels have different delays. This is typically more important on the transmit side, where the data coming to the SSI has different delays. On the receive side, the baseband processor manipulates the delay.

To mitigate this delay difference, the ADRV9001 provides the measurement from the last MCS pulse edge to the transmitter strobe for different channels. This measurement is effective for all channels associated to the MCS, as the transmitter strobe of each channel can have a different delay relative to the MCS pulse signal. The API function that achieves this is **adi_adrv9001_Mcs_TxMcsToStrobeSampleLatency_Get()**, which takes the channel number and provides the latency, in samples, from the MCS to the transmitter strobe for the specified channel.

Once the latency is measured for all channels, refer to this as MCS_to_Strobe latency. Next, calculate the Sample Delay and Read Delay separately, and set the delays into the ADRV9001 to achieve the same transmitter latency for all channels among all the ADRV9001 chips.

Sample Delay: It helps delay the ADRV9001 internal FIFO read point so that if MCS_to_Strobe is too big, then it delays the FIFO so that FIFO saves less irrelevant information.

MULTICHIP SYNCHRONIZATION

Read Delay: It helps delay FIFO read time. Once information is fed in the FIFO, this value guarantees the FIFO has samples for all channels before reading.

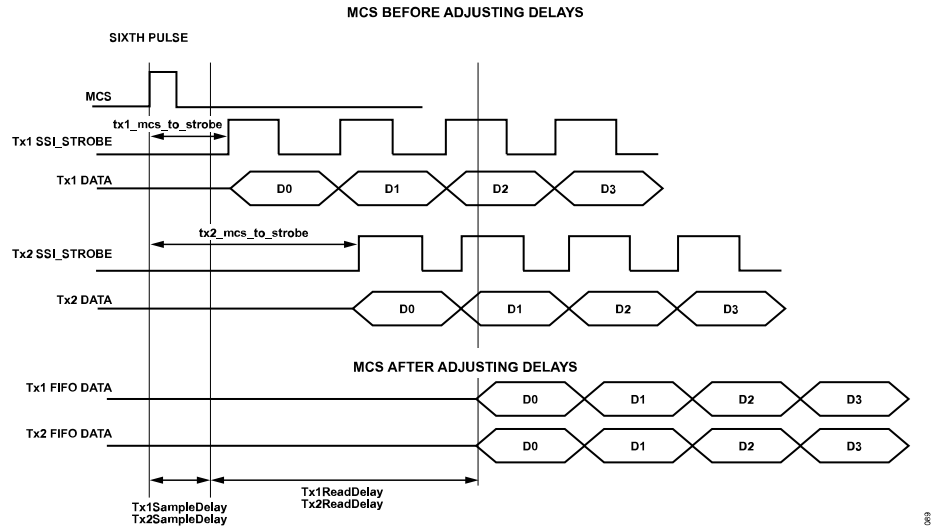


Figure 91. Tx MCS to Strobe Timing Diagram

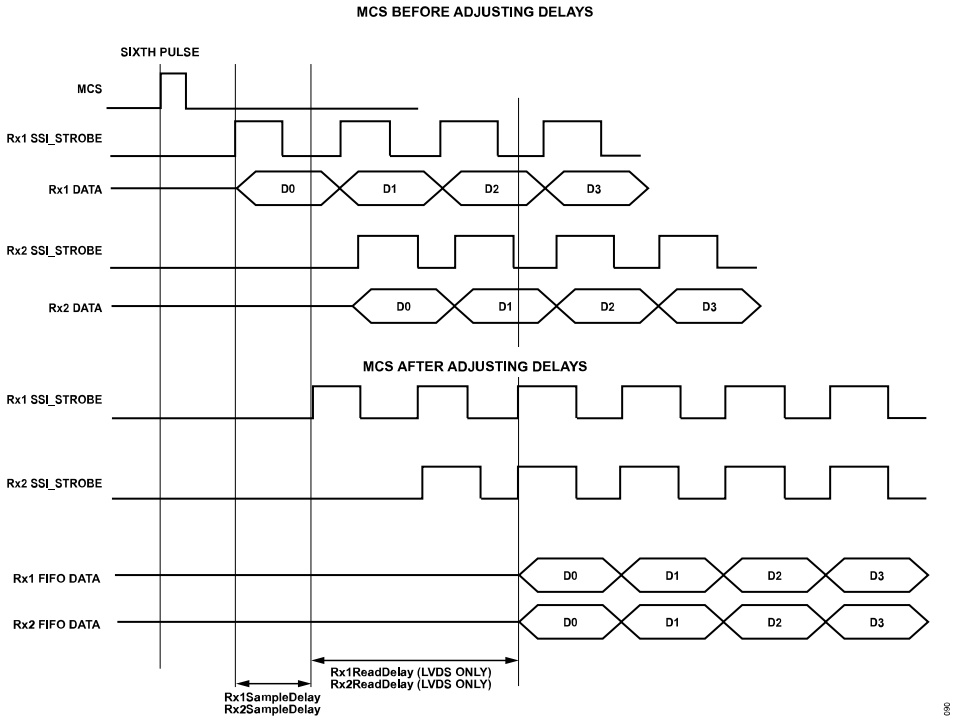


Figure 92. Rx MCS to Strobe Timing Diagram

Calculate them in the following way:

Transmitter:

- Each channel is independent from other channels
  - sampleDelay = MCS_to_Strobe latency – 1
  - readDelay = 4
- Syncing multiple channels: (for example, TX1 and TX2, or multiple devices)

MULTICHIP SYNCHRONIZATION

- sampleDelay = Min (all MCS_to_Strobe latency) – 1
- readDelay = Max (all MCS_to_Strobe latency) - Min (all MCS_to_Strobe latency) + 4

Receiver:

- Each channel is independent from other channels.
  - sampleDelay = UserDefined
  - readDelay (LVDS Only) = UserDefined (minimum of 1 to compensate for clock timing in LVDS from analog, imposed by API)
- Syncing multiple channels: (for example, RX1 and RX2, or multiple devices)
  - sampleDelay = UserDefined
  - readDelay (LVDS Only) = UserDefined (minimum of 1 to compensate for clock timing in LVDS from analog, imposed by API)

Once sample delay and read delay are calculated, set them by API **adi_adrv9001_Mcs_ChannelMcsDelay_Set()**.

Note that the MCS_to_Strobe delay difference among the different channels should be within two samples.

Note on the receiver side, the functions to calculate sample delay and read delay are user defined. Use the default values provided in the software. Analog Devices also allows an adjustment to use the above method. Fundamentally, this is controlled from the baseband processor, and there is full control over how data is timed. In contrast, on the transmitter side, the ADRV9001 does not control how the data is passed in. Therefore, it tries to adjust the different delays and aligns the data in time.

PHASE SYNCHRONIZATION

The following plots are trying to show the effect of the phase synchronization function. The test is done with two receiver signals coming from the same signal source. The signal uses continuous waves (CW) tones. The reference clock frequency is set to 38.4 MHz. ADI uses the LTE61.44 profile for this test. The test is run from 100 MHz to 3GHz with a 100 MHz step, using internal LOs. Analog Devices also used five different ADRV9001 chips for the test.

Figure 93 shows the effect of phase synchronization for the receiving signals. The signals are captured at the interface to calculate phase error.

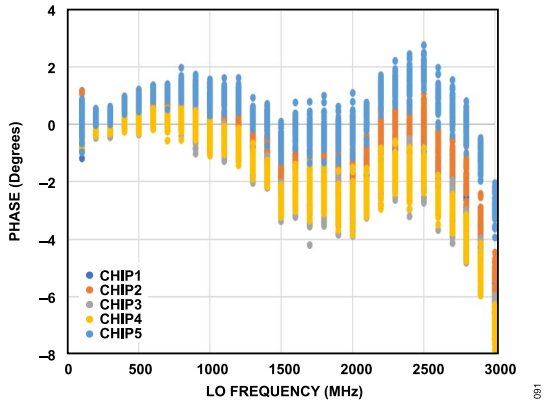


Figure 93. Phase Error After Synchronization Between Two Receivers

Figure 94 shows the phase error after synchronization of the two LOs, and these are internal register reads but should be accurate.

MULTICHIP SYNCHRONIZATION

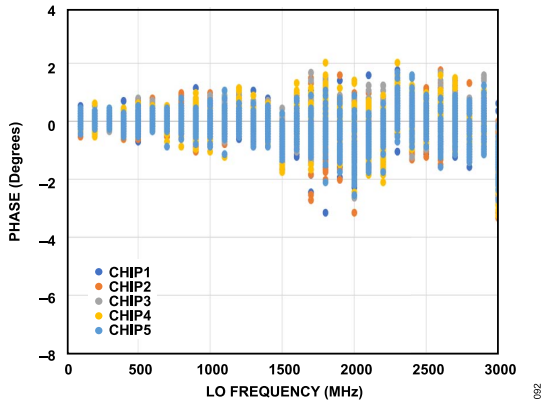


Figure 94. Phase Error After Synchronization Between Two LOs

SYNTHESIZER CONFIGURATION AND LO OPERATION

The ADRV9001 family devices employ four PLL synthesizers: clock, RF (×2), and auxiliary. Each PLL is based on a fractional-N architecture and consists of a common block made up of a reference clock divider, phase frequency detector, charge pump, loop filter, feedback divider, digital control block, and either a 1 or 4 core voltage-controlled oscillator (VCO). The VCO has a tuning range of 6.5 GHz to 13 GHz. Each PLL drives its own LO generator: RF LOGEN, aux LOGEN, and CLKGEN. The output of the LOGEN block is a divided version of the VCO frequency. No external components are required to cover the entire frequency range of the device. The reference frequency for the PLL is scaled from the reference clock applied to the device. Figure 95 shows the synthesizer interconnection and clock/LO distribution block diagram.

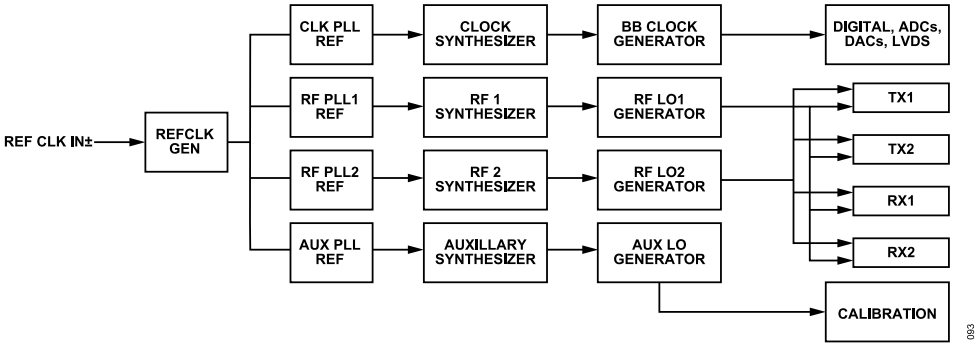


Figure 95. Synthesizer Interconnection and Clock/LO Distribution Block Diagram

Each receiver channel is used as an ORx for transmitter channels, as shown in Figure 96.

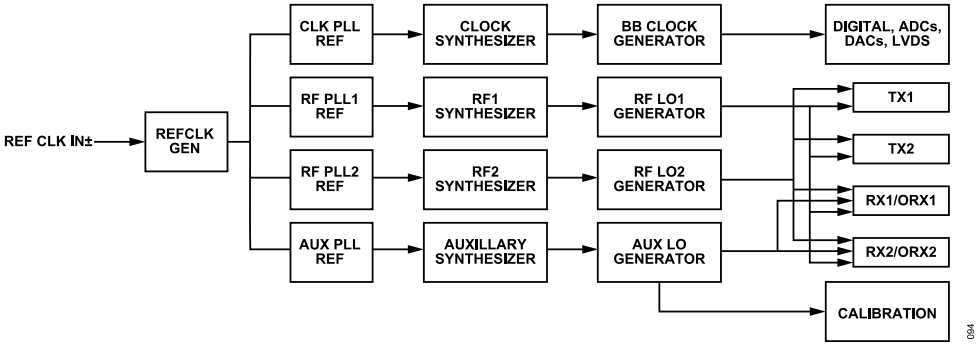


Figure 96. Synthesizer Interconnection and Clock/LO Distribution Diagram (Receiver Channels Configured as Observation Receivers for Transmitter Channels)

CLOCK SYNTHESIZER

The clock synthesizer generates all the clocking signals necessary to run the device. The synthesizer uses a single core VCO block. The reference frequency for the clock PLL is scaled from the device clock by the reference clock generator. Reconfiguration of the clock synthesizer is typically not necessary after initialization. The most direct approach to configure is to follow the recommended programming sequence and use the provided API functions to set the clock synthesizer to the desired mode of operation. The clock generation block of the clock synthesizer provides clock signals for the high speed digital clock, receiver ADC sample and interface clocks, transmitter DAC sample and interface clocks, and LVDS interface clocks.

There are two types of clock synthesizers for the digital clock generation. For more details, see the Clock Generation section.

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 four core VCO block, which provides a 6 dB phase noise improvement over the single core VCO. As with the other synthesizers in the device, the reference for RF PLL 1 and RF PLL 2 are sourced from the reference generation block of the device. The RF PLLs are also fractional-N architectures with a programmable modulus. The default modulus of 8,388,593 is programmed to provide an exact frequency on at least a 5 kHz raster using certain reference clocks, which are integer multiples of 38.40 MHz. The RF LO frequency is derived by dividing down the VCO output in the LOGEN block. The tunable range of the RF LO is 30 MHz to 6000 MHz.

A switching network is implemented in the device to provide flexibility in LO assignment for the two RF LO sources. Figure 97 and Figure 98 show the switching network.

SYNTHESIZER CONFIGURATION AND LO OPERATION

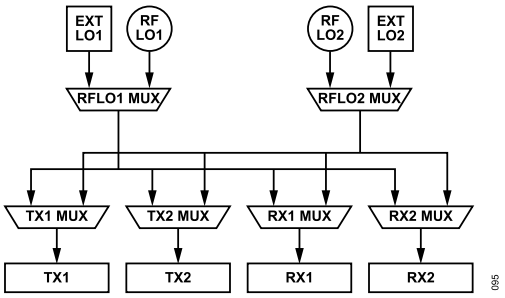


Figure 97. LO Switching Network

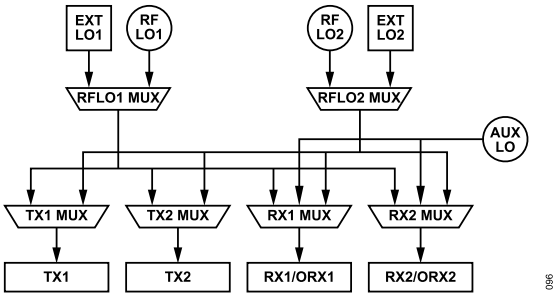


Figure 98. LO Switching Network (Receiver Channels Configured as Observation Receivers for Transmitter Channels)

Note that depending on the application, select the best phase noise or best-power saving options for better optimization.**This changes the frequency the internal VCO operates at to optimize for phase noise or power consumption.** The option of best phase noise only works for LO frequency under 1 GHz. The RF LO power consumption has three configurable options **(Low, Medium High)**, which balance the power consumption and RF LO phase noise. This option changes some of the analog DC biases to reduce the VCO voltage swing, and thus the power consumption. High power means low LO phase noise, and vice versa.

The ADRV9001 provides two modes of retuning the PLL: the normal mode and fast mode. In terms of phase noise performance, there is no difference between the fast and normal modes. However, the normal mode tracks temperature over time while the fast mode does not. Therefore, the fast mode takes less time to complete retuning.

RF Synthesizer Frequency Accuracy

Although configuration of the RF synthesizer is valid to 1 Hz accuracy, the frequency might vary slightly. The actual carrier frequency can be calculated from this based on the equation as follows:

$$Rf\_synth\_freq = \text{round}(f\_carrier/f\_step,max) \times f\_step,max \quad (1)$$

The frequency step size of the RF PLLs is given by the following equation:

$$f\_step,max = DEV\_CLK/((2\text{^}23-15) \times RfLogenDivider) \quad (2)$$

where *RfLogenDivider* can be retrieved from the **adi_adrv9001_Radio_RfLogenDivider_Get()** API. The *RfLogenDivider* is based on the VCO frequency and the LO optimization (power consumption/phase noise).

Example Frequency Accuracy Calculation

The desired carrier frequency is 4 GHz and the device clock is 307.2 MHz. LO optimization is set to 'Best Power Saving'.

From the **adi_adrv9001_Radio_RfLogenDivider_Get()** API, the RfLogenDivider was found to be 2.

The maximum frequency step can then be calculated as: 307.2 MHz/((2^23−15) × 2) = 18.31057962 Hz.

The actual carrier frequency can then be calculated as: round(4 GHz/18.31..Hz) × 18.31...Hz = 4.000 000 006 GHz i.e. 6 Hz offset from the desired carrier frequency.

SYNTHESIZER CONFIGURATION AND LO OPERATION

AUXILIARY SYNTHESIZER

An integrated auxiliary synthesizer generates the signals necessary to calibrate the device. This synthesizer uses a single core VCO. The reference frequency for the auxiliary synthesizer is scaled from the device clock through the reference clock generation system. The output signal is connected to a switching network and injected into the various circuits to calibrate filter bandwidth corners or into the receive signal chain as an offset LO. Calibrations are executed during the initialization sequence at start-up. There should be no signal present at the receiver/observation receiver input during tone calibration time. Calibrations are fully autonomous. During the calibration, the auxiliary synthesizer is controlled solely by the internal ARM microprocessor and does not require any user interactions. The auxiliary LO signal is also available as an LO source for the observation receiver mixers.

EXTERNAL LO

The device is provisioned with two external LO ports. These ports are available as a pair of balls and are configured to be the input for external LO signals.

The external LO can receive a signal between 60 MHz and 12 GHz through a matched differential impedance of 100 Ω and delivers a programmable signal between 30 MHz and 6 GHz as the LO for transmitters and receivers in the device. Maintain the amplitude between ±6 dBm. For more information, see the External LO Port Impedance Matching Network section..

Single-ended external LO in is also supported. The matched single-ended impedance is 50 Ω. On-chip duty cycle correction circuit can correct limited range of external LO duty cycle error if it is not 50%.

Enable external LO with the 1× divider for the carrier frequency range from 500 MHz to 1 GHz.

Single-Ended vs. Differential External LO

Note that the ADRV9001 evaluation board only supports differential external LO. However, there is no restriction to use single-ended LO in the end system. Change clocks.ext1LoType and clocks.ext2LoType from 0 to 1. The following Enum contains this.

```
enum adi_adrv9001_ExtLoType {
ADI_ADRV9001_EXT_LO_DIFFERENTIAL = 0,
ADI_ADRV9001_EXT_LO_SINGLE_ENDED = 1}
```

Figure 99 shows the RF LO generation diagram.

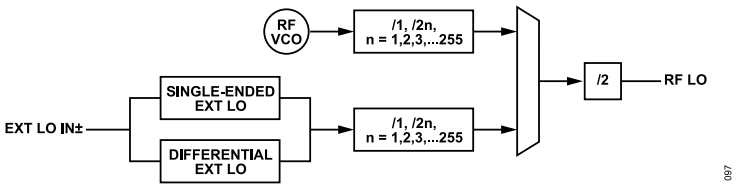


Figure 99. RF LO Generation Diagram

RF PLL LOOP FILTER RECOMMENDATIONS

For optimal phase noise and error vector magnitude (EVM) performance, the ADRV9001 firmware has a look-up table of RF PLL loop filter bandwidth settings. The ADRV9001 automatically selects the best RF PLL loop filter configuration based on the LO frequency, DEV_CLK frequency, and PLL bandwidth. Alternatively, program own RF PLL loop filter bandwidth by following the instructions in the Loop Filter Configuration section.

PLL PHASE NOISE

Figure 100 shows the typical PLL phase noise contributors. For low offset frequencies, the reference clock dominates the phase noise, and for high offset frequencies, the VCO noise dominates the phase noise. Optimize the phase noise by:

- Providing better reference clock source.
- Providing higher reference clock frequency (PFD).
- Adjusting the loop filter bandwidth to trade-off between the close-in band and far-out band noise.

SYNTHESIZER CONFIGURATION AND LO OPERATION

When changing the loop filter bandwidth, the typical consideration is the wider the bandwidth, the better the close-in band noise, but the worse the far-out band noise. Trade off between the two to find the optimal setting for the specific application. Note the highest phase frequency detector (PFD) frequency the ADRV9001 supports is 307.2MHz.

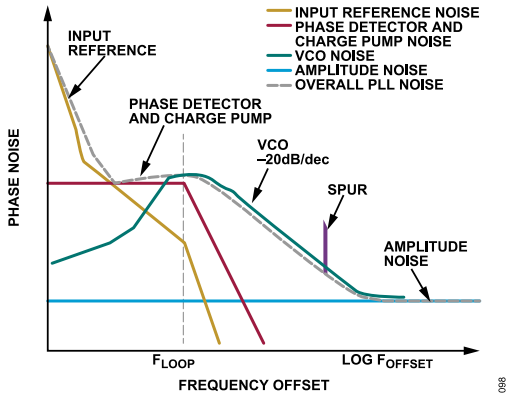


Figure 100. PLL Phase Noise Contributors

Figure 101 shows that the PLL phase noise is highly dependent on the PFD frequency (REF_CLK). A higher PFD frequency achieves a better phase noise.

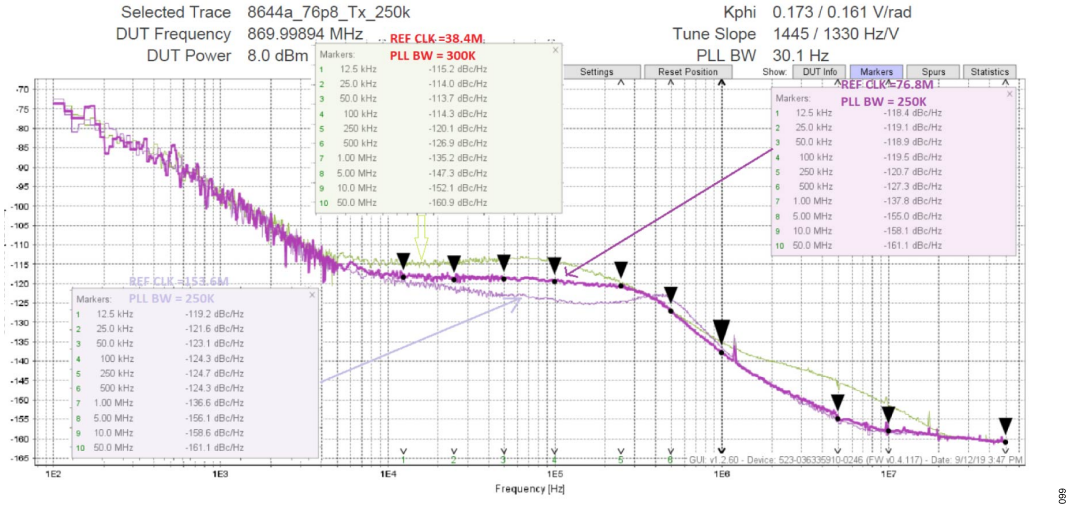


Figure 101. Effect of PFD (REF_CLK) Frequency on PLL Phase Noise

Figure 102 shows where a trade off can be made between the close-in phase noise and far-end phase noise by adjusting the loop filter bandwidth. As mentioned, the higher the loop filter bandwidth, the better the close-in noise but with the scarification of the far-end noise, and vice versa.

SYNTHESIZER CONFIGURATION AND LO OPERATION

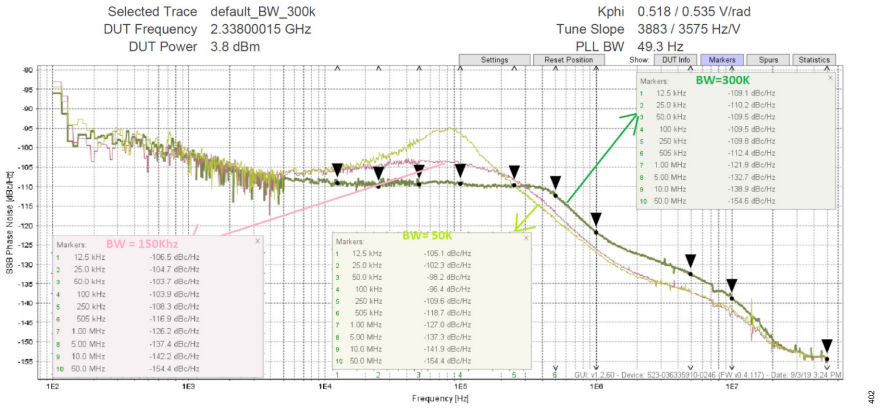


Figure 102. Effect of Loop Filter Bandwidth on PLL Phase Noise

API OPERATION

Data Structure and Enumerations

Table 46. Data Structures Related to LO Operation

Data Structure	Description
adi_adrv9001_Device_t	Data structure to hold ADRV9001 device instance settings.
adi_adrv9001_Carrier_t	Carrier structure for carrier configuration.
adi_common_Port_e	Enumeration of port types.
adi_common_ChannelNumber_e	Enumeration of channel numbers.
adi_adrv9001_LoGenOptimization_e	Enumeration of LO optimization.
adi_adrv9001_PllCalibration_e	Enumeration for PLL calibration mode.
adi_adrv9001_Carrier_t	Carrier frequency configuration
adi_adrv9001_Pll_e	Enumeration of PLL selections.
adi_adrv9001_PllLoopFilterCfg_t	Data structure to hold synthesizer PLL loop filter settings.

API Commands

The Doxygen document, with the SDK package, provides more detailed information, including parameters and return values.

Table 47. API Commands Related to LO Configuration Settings

API Function	Description
adi_adrv9001_Radio_Channels_EnableRf()	Enable or disable RF channel (transition the specified channel between the RF_ENABLED and PRIMED states) (this function only works if the channel is in the SPI mode).
adi_adrv9001_Radio_Channel_Prime()	Prime the specified channel (transition the specified channel between the CALIBRATED and PRIMED states).
adi_adrv9001_Radio_PllStatus_Get()	Checks if the PLLs are locked.
adi_adrv9001_Radio_PllLoopFilter_Set()	Configures the loop filter for the specified PLL.
adi_adrv9001_Radio_PllLoopFilter_Get()	Gets the loop filter configuration for the specified PLL.
adi_adrv9001_Radio_Carrier_Configure()	Sets the carrier configuration for the given channel.
adi_adrv9001_Radio_Carrier_Inspect()	Inspects the carrier configuration.

Local Oscillator (LO) Change Procedure

To set the LO frequency to a particular channel:

1. Verify that the internal ARM microprocessor is initialized.
2. If the device is in the RF_ENABLED state, first set it to the PRIMED state.
   a. If in the SPI mode, do this by calling **adi_adrv9001_Radio_Channels_EnableRf()**.

SYNTHESIZER CONFIGURATION AND LO OPERATION

   b. If in the PIN mode, do this by moving TX_ENABLE or RX_ENABLE to the low state.
3. Once in the PRIMED state, move the channel state to the calibrated state by calling **adi_adrv9001_Radio_Channel_ToCalibrated()**.
4. Once the device is in the calibrated state, set the LO frequency by calling **adi_adrv9001_Radio_Carrier_Configure()**. Note that it is recommended to run a minimum set of calibrations after changing the LO frequency by a significant step as described in the Configure the Initial Calibrations Through TES section.
5. Lastly, bring the device back to the RF_ENABLED state. First bring it back to PRIMED, then to RF_ENABLED, calling the same functions above but in reverse order.

For more details, see the API Initialization Sequence section about the radio states mentioned in this section.

There is no specific control over the CLK_PLL. The CLK_PLL is configured at initialization.

Note that when changing the LO, make sure all channels that utilize the LO are configured. For example, if Tx1 and Rx1 are using LO1, and there is a need to change LO1, configure both Tx1 and Rx1. If all channels (Tx1, Tx2, Rx1, and Rx2) are using the same LO, and there is a need to change the LO, configure all channels (Tx1, Rx2, Rx1, and Rx2).

Loop Filter Configuration

The default loop filter bandwidth for RF PLL is 300 kHz, as discussed in the PLL Phase Noise section. Change the PLL BW into the ADRV9001 chip initialization state based on the specified DEV_CLK_IN frequency and LO phase noise requirements. The ADRV9001 TES has the RF PLL loop filter bandwidth configuration interface to tune and achieve the best phase noise in the system.

FREQUENCY HOPPING

Before delving into the frequency hopping feature details, read the Multichip Synchronization and Timing Parameters Control sections of this document.

In the ADRV9001, FH is mainly designed for TDD applications. It allows to quickly switch radio signals among different frequency channels at different times. Figure 103 shows the high-level concept of FH.

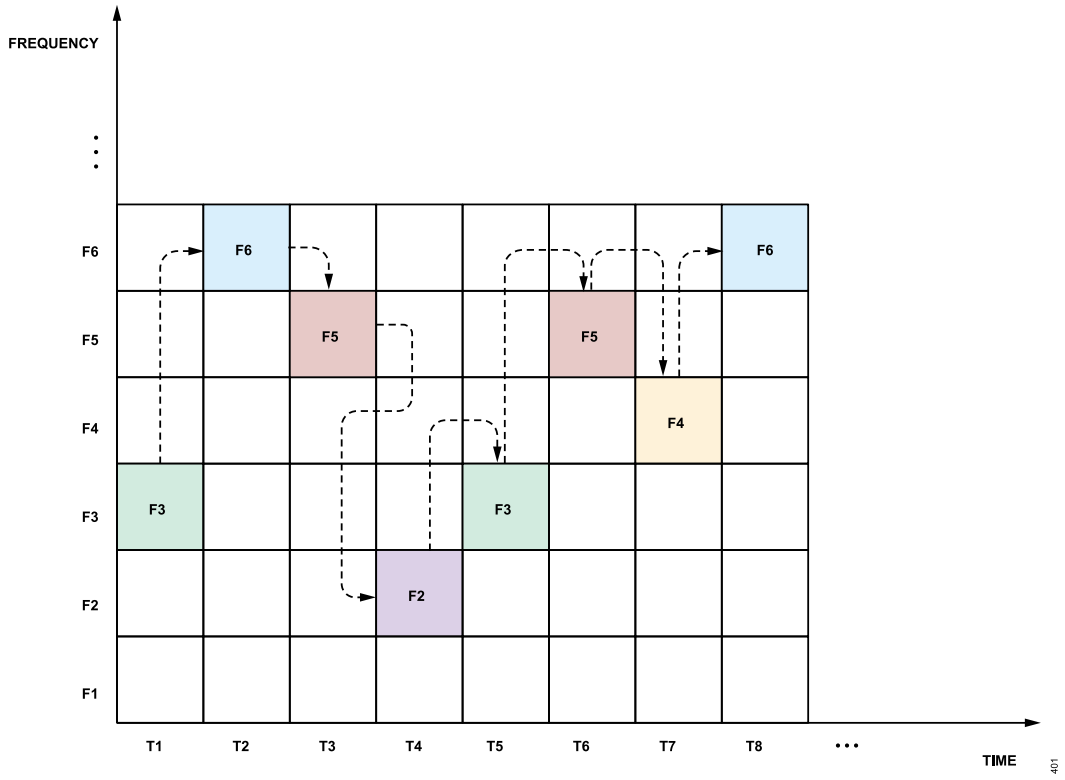


Figure 103. High-level Concept of Frequency Hopping

Figure 103 shows that different frequencies are used at different time slots. FH provides the benefit of high resistance to narrowband interference and strong capability in combating malicious interception and jamming. In addition, FH signals can easily share the bandwidth with other conventional communications due to the minimal mutual interference, resulting in high spectrum efficiency. With an increased hop rate and a larger set of frequency sub-bands, the advantages of FH become more prominent, which makes it an attractive solution for many different applications.

The ADRV9001 provides advanced FH capability through its unique and flexible PLL design, i.e., instead of having one dedicated PLL for the receive datapath and one for the transmit datapath as many other transceivers, the device employs two RF PLLs, and both PLLs can optionally source any receiver or transmitter, or both, or neither. This flexibility is essential for the ADRV9001 to support FH in various TDD applications such as single-channel and dual-channel operations including transmit-only mode (1T/2T), receive-only mode (1R/2R), and transmit and receive mode (1T1R/2T2R). Both channel diversity and channel multiplexing are supported for the dual-channel operations.

FH capability is achieved by retuning the PLLs before switching to the new frequency. There are different FH modes based on the method of using the two PLLs. One method is to ping pong between the two PLLs, which means while one PLL is being used for on-air signal transmission or reception on one frequency, the other PLL is retuned to prepare for the next frequency. This makes FH very fast. Another method is to use only one PLL for a pair of Tx/Rx channel for FH. When one hopping frame ends, the same PLL retunes to the new frequency. Therefore, it is possible to hop two pairs of channels (Rx1/Tx1 and Rx2/Tx2) separately by using two different PLLs.

KEY SIGNALS

To understand FH operations in the ADRV9001, it is crucial to first understand several key signals the BBIC needs to set up such as “HOP”, “Tx Setup”, and “Rx Setup” signals.

FREQUENCY HOPPING

Figure 104 shows a typical frequency hopping timing diagram with the HOP, Tx Setup, and Rx Setup signals. Besides these, it also shows the hopping frequency selected through a hopping table defined by the BBIC and the timing when hopping frames are on air. The following sections discusses them in more detail.

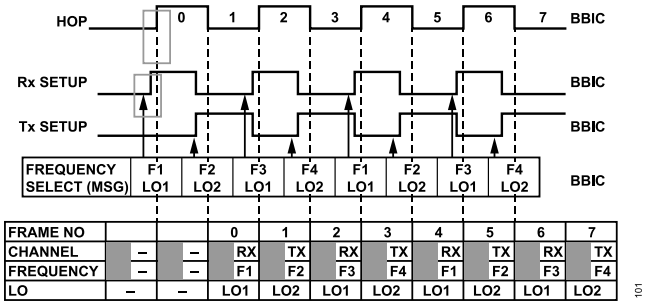


Figure 104. Typical Timing Diagram for Frequency Hopping

Hop Signal and Hop Frame

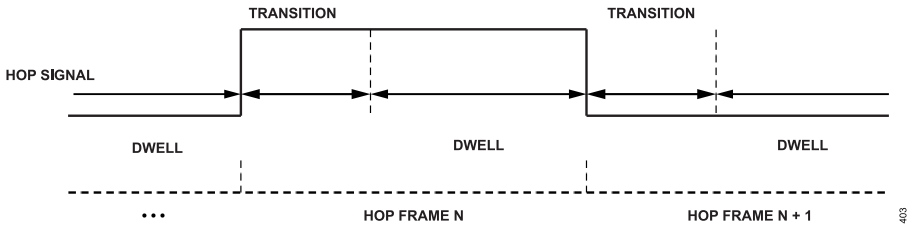


Figure 105. Hop Signal and Hop Frame

In FH, hop frame is the period during which a channel is enabled. The hop frame duration is defined by the time between the rising and falling edge of the hop signal that the BBIC defines. During each hop frame, data can be operated on a new carrier frequency with either Rx or Tx. Figure 105 shows that a hop frame consists of a transition and a dwell time period.

The **transition** period is, at a minimum, the time required to set up the Rx or Tx channel and switch to a new operating frequency. The **dwell** period is the "on air" time, where a channel is on transmit or receive operation. The dwell period can be any length of time to operate on a frequency. The start and end of the dwell time should align with the signal transmitted or received on air for a transmit or receive frame, respectively. The hop signal is triggered by a DGPIO pin, which can be assigned by the BBIC from any of the available DGPIO pins. Each edge of the hop signal marks both the possible start and end of a hop frame. Figure 104 shows the transition time is represented by the gray time period and Tx/Rx frames are on air during the dwell time within each hop frame.

Channel Setup Signal

The channel enable pins (Tx/Rx_ENABLE pins used in non-FH operation) are repurposed in FH to signal if an upcoming hop frame is operating on an Rx or Tx channel. These pins are redefined as "Tx/Rx Setup"; however, they are the same dedicated channel enable hardware pins that are used to enable an Tx or Rx channel in non-FH mode. Figure 104 shows that a rising edge of the "Tx_Setup" or "Rx_Setup" signal before the hop signal edge indicates if the coming frame is a Tx or Rx frame. For example, the first "Rx_Setup" rising edge before the first hop signal edge indicates that hop frame 0 is a Rx frame. The first "Tx_Setup" rising edge before the second hop signal edge indicates that hop frame 1 is a Tx frame. With the Tx_Setup and Rx _Setup signals, as shown in Figure 104, Rx and Tx frames alternate.

MODES OF OPERATION

The ADRV9001 provides four FH modes to achieve various framing requirements, as shown in Table 48.

Table 48. Frequency Hopping Modes of Operation

Mode	Transition Time	Total Frame Duration (Transition + Dwell)	PLLs	Max PLL Retune Time Allowed	PLL Cal Mode[1]	Channel
PLL Mux with hop table preprocess	< channel setup + PLL retune	>= 13 µs	2 PLLs	2 transitions + 1 dwell	Fast/Normal	Single (1T/1R /1T1R)

FREQUENCY HOPPING

Table 48. Frequency Hopping Modes of Operation (Continued)

Mode	Transition Time	Total Frame Duration (Transition + Dwell)	PLLs	Max PLL Retune Time Allowed	PLL Cal Mode[1]	Channel
PLL Mux with hop table real-time process	< channel setup + PLL retune	>= 25 µs	2 PLLs	2 transitions + 1 dwell	Fast/Normal	Single/Dual (1T/1R/1T1R or 2T/ 2R/2T2R diversity)
PLL Retune with hop table real-time process	> channel setup + PLL retune	Transition time >= 48 µs and total frame duration > transition time	1 PLL	1 transition	Fast/Normal	Single/Dual (1T/1R/1T1R or 2T/ 2R/2T2R diversity)
PLL Retune with dual hop and hop table real-time process	> channel setup + PLL retune	Transition time >= 48 µs and total frame duration > transition time	1 PLL	1 transition	Fast/Normal	Dual (2T/2R/2T2R multiplexing)

[1] ADRV9002, ADRV9003, ADRV9004, ADRV9005 support both Fast/Normal calibration modes. ADRV9006 only supports Normal cal mode.

Fundamentally, the ADRV9001 defines two modes of PLL usage: PLL Mux and PLL Retune. The PLL Mux mode utilizes two PLLs. While one PLL is in operation, the other one is tuning. The two PLLs are swapped during the transition time. This mode is suitable for use cases where the transition time is not sufficiently long to complete channel setup and PLL tuning. By using two PLLs, the PLL being retuned can have a maximum tuning time of one frame length plus one transition time or equivalently two transition times plus one dwell time. On the other hand, the PLL Retune mode is suitable for use cases where the transition time is sufficiently long that once a PLL finishes its operation, it can start tuning and the channel setup and retuning can be completed within the transition time.

The ADRV9001 provides two modes of retuning the PLL: the normal mode and fast mode, as described in the RF Synthesizer section. For FH, typically, the hop frames are short. Therefore, the fast Mode is usually sufficient.

The following timing diagram describes the PLL Mux mode.

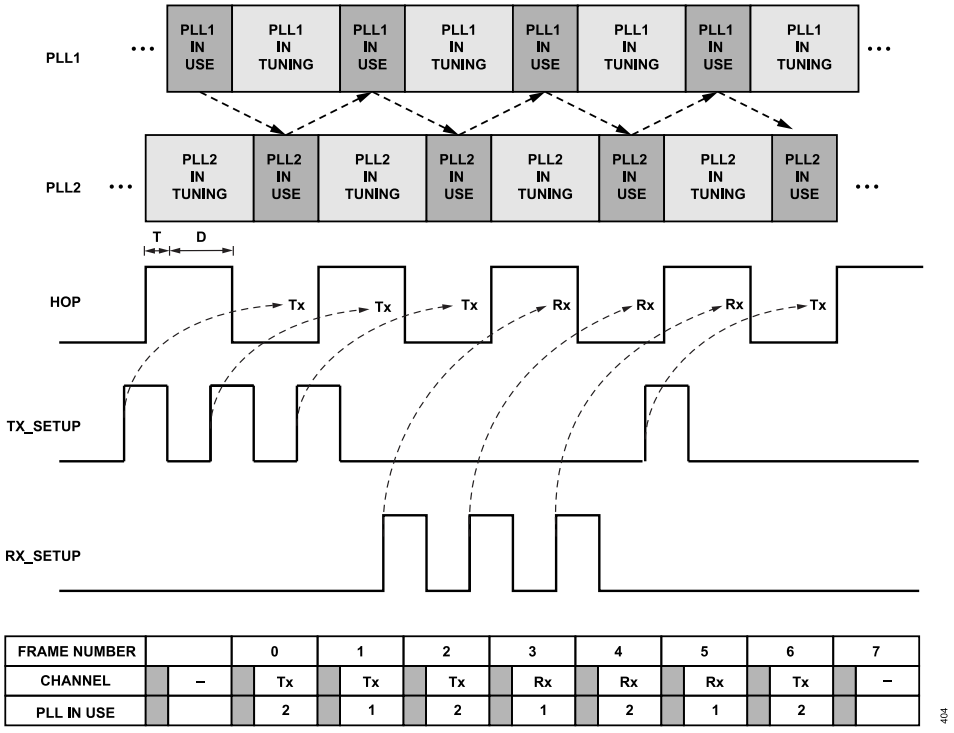


Figure 106. PLL Mux Timing Diagram Example

Figure 106 shows that each hop frame consists of a transition time period and a dwell time period, as mentioned earlier. While one PLL is used during the dwell time, the other PLL starts tuning from the beginning of the transition time of the same hop frame. It can continue the tuning until the end of the transition period of the next hop frame. Then, the two PLLs are swapped during the transition time.

FREQUENCY HOPPING

Note in the PLL Mux mode, the rising edge of Tx_Setup or Rx_Setup before a HOP signal edge (the start of hop frame n or end of hop frame n-1) indicates if the hop frame n+1 is operated on Tx or Rx. There is one frame delay in the PLL Mux mode. In another word, to operate Tx on hop frame n+1, the rising edge should appear at hop frame n-1.

The following timing diagram describes the PLL Retune mode.

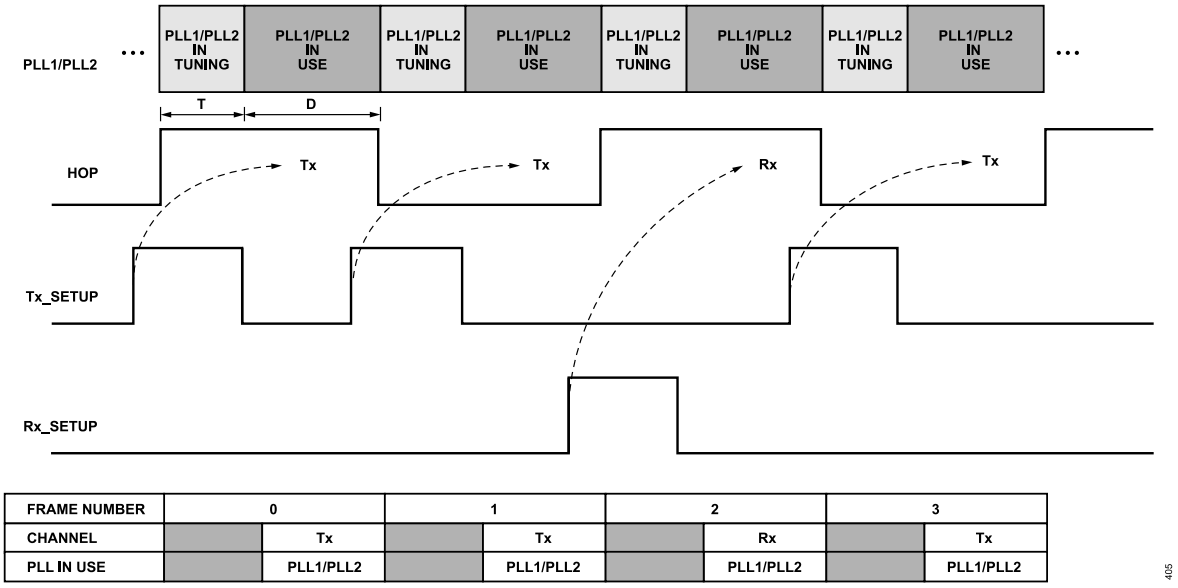


Figure 107. PLL Retune Timing Diagram Example

Figure 107 shows that, with the PLL Retune mode, the same PLL is used (either PLL1 or PLL2). Once the hop frame ends, it starts to retune to the new frequency. Another major difference from the PLL Mux mode is that the rising edge of Tx_Setup or Rx_Setup before a HOP signal edge (the start of hop frame n or end of hop frame n-1) indicates if the hop frame n is operated on Tx or Rx instead of n+1. There is no one frame delay in the PLL Retune mode. In another word, to operate Tx on hop frame n+1, the rising edge should appear at hop frame n.

PLL retune with dual hop is similar to PLL retune mode described previously except in this mode, two hop signals are used. Hop signal 1 will always control channels associated with LO1 and hop signal 2 will always control channels associated with channel 2. Assign any channel (Tx1/Rx1/Tx2/Rx2) to any LO (LO1/LO2). The timings of hop 1 and hop 2 is described in the Dual Hop Timing section. Using dual hop mode could be viewed as having two independent TDD transceivers.

PLL lock time depends on the PLL reference frequency, which is discussed later.

The four hopping modes, as shown in Table 48, also specify the method of loading a hopping table, which is discussed in the following sections.

FREQUENCY HOPPING TABLE

Hopping Table Definition and Entries

All modes of FH require the use of a frequency hopping table. A frequency hopping table is a list of frequencies and other operational parameters for each hop frame.

The ADRV9001 supports the loading of two tables (table A and B), each with a minimum length of 1, and a maximum length of **64 entries** (a total of 128 hop entries/frequencies, if two tables are used).

An entry in the frequency hopping table is defined as in Table 49.

Table 49. Hop Table Entry

Parameter	Descriptions
hopFrequencyHz	Operating frequency, in Hz.
rx1OffsetFrequencyHz	The intermediate frequency, if a frame is Rx1.
rx2OffsetFrequencyHz	The intermediate frequency, if a frame is Rx2.

FREQUENCY HOPPING

Table 49. Hop Table Entry (Continued)

Parameter	Descriptions
rx1GainIndex	Starting gain index, if frame is Rx1.
rx2GainIndex	Starting gain index, if frame is Rx2.
Tx1Attenuation_mdB	Starting attenuation level, in mdB, if frame is Tx1.
Tx2Attenuation_mdB	Starting attenuation level, in mdB, if frame is Tx2.

The channel(s) enabled for each hop frame are not known at initialization. Therefore, the hopping table does not present this information. During FH operation, the BBIC employs the channel setup signals to inform the ADRV9001 on the channel to enable, as described earlier. Therefore, each entry in the hop table contains parameters for both Rx and Tx, and it only utilizes the relevant parameters based on the channel setup signals, depending on whether an upcoming hopping frame is operating on Rx or Tx. The irrelevant parameters are ignored.

Note: The TES provides some examples of hopping tables. The ranges of the fields should be the same as in the non-FH mode.

Frequency Hopping Table Modes

The ADRV9001 supports the utilization of the hopping tables in different ways. This section details the available methods to index, switch, and update hopping tables.

Hopping Table Indexing Mode: Automatic Increment vs. Pin Automatic Increment

The standard mode of operation in the ADRV9001 is the automatic increment mode. In this mode, FH begins at index 0, and continues to iterate over the table until the number of table entries is reached. Once reached, it wraps back to index 0. This is also called the “Loop” mode when one hopping table is used.

Figure 108 shows an example of automatic increment of table index with three entries in the hopping table with the PLL Retune mode.

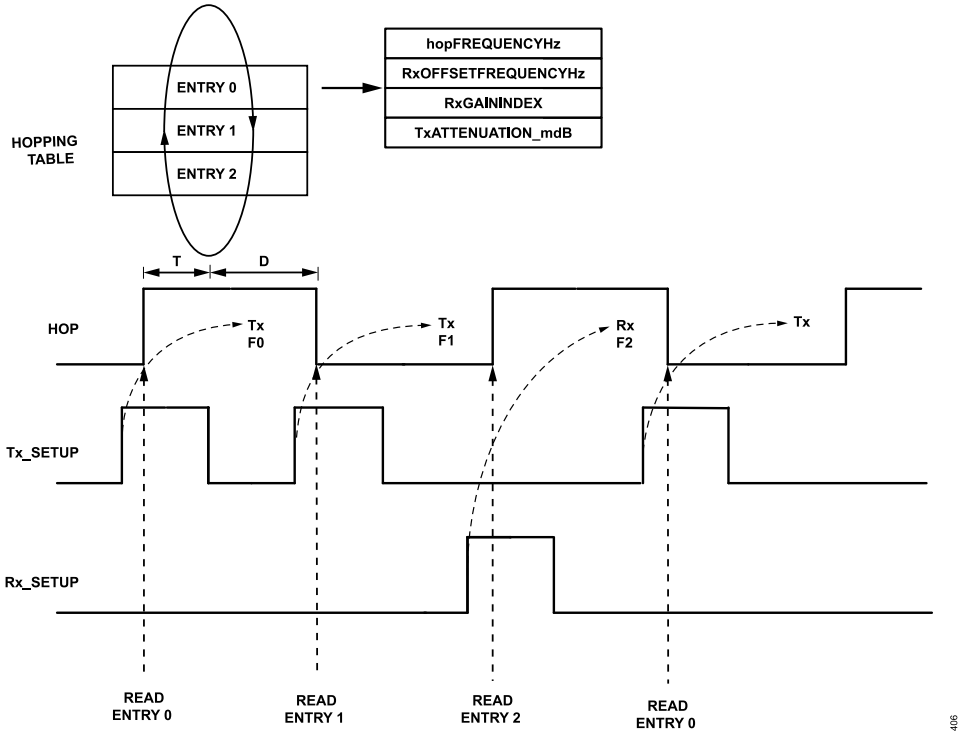


Figure 108. Automatic Increment Table Indexing Example with PLL Retune Mode

Figure 108 shows that the ADRV9001 reads the entries at the hop edges (both rising and falling edges). Therefore, make sure the desired table is loaded and ready to read at that time. More details about the method of loading a hopping table are discussed later.

FREQUENCY HOPPING

Index by Pin

Rather than automatically incrementing through a hopping table, use DGPIO pins to index any valid entry in the hopping table.

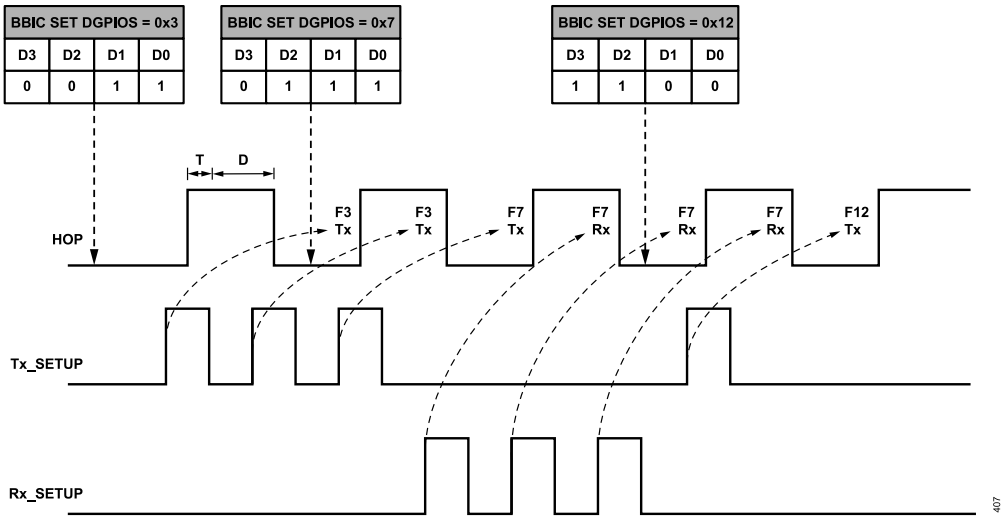


Figure 109. DGPIO Table Indexing with PLL Mux Mode

At initialization, assign up to 6 DPGIO pins to provide 64 possible indices to the current hopping table. During operation, set the DGPIO pins **prior to the upcoming hop edge**. At each hop edge, the ADRV9001 samples the DGPIO pins to understand the hopping table entry to use.

Figure 109 shows an example of table indexing using four DGPIO pins with the PLL Mux mode. At the hop edge, the ADRV9001 samples the DPGIO pins and one PLL retunes to the new frequency starting from the same hop edge.

The following restrictions apply to this mode:

- Each DGPIO pin represents a bit in the index, and the ADRV9001 samples each assigned pin to form a full index to the table.
- The ADRV9001 expects that the lower DPGIO number is the LSB of the index.
- Assign adjacent DGPIO pins (for example, cannot assign DGPIO-0 to Bit 0, and DPGIO-11 to Bit 1, and so on).
- The ADRV9001 samples the DGPIO pins at every hop edge. It uses any index if the sampled index is deemed valid (meaning, provide a valid index to the frequency hopping table). Therefore, ensure to load the correct table entries and assign the appropriate number of GPIO pins to avoid having an index that is out of boundary.

Hopping Table Switching Method: Automatic Ping Pong vs. Manual Switch

As mentioned earlier, the ADRV9001 allows to define up to two hopping tables. In any FH mode, the ADRV9001 allows to switch between these two hopping tables, if both are defined. Use the following rules to switch tables:

- When a table is switched, the index to the new table is set to 0 by the ADRV9001.
- The new table is read at the next hop signal edge.

Automatic Table Ping Pong Switch

The automatic ping pong mode works like the regular automatic increment mode. However, once the end of one table is reached, the ADRV9001 automatically switches to the other table. In this way, continuously increment through, at most, 128 table entries/hop frequencies.

This mode does not require any external signal. The ADRV9001 automatically switches between the two tables.

Manual Switch with Automatic Increment

Switch between the two hopping tables at any time manually using an API or a DGPIO pin.

Configure the DGPIO pin to switch the hopping table at initialization. Besides using DGPIO, issue an API to switch between the two tables.

FREQUENCY HOPPING

In the DGPIO mode, when the hopping table select pin is low, table A is set as the active table, and when the hopping table select pin is set high, table B is set as the active table. The falling or rising edge of HOP Table Select signals the switch to the A or B table, respectively.

Figure 110 shows that the HOP Table Select pin is sampled at the hop edge. Set the HOP Table Select pin before the upcoming hop edge. The ADRV9001 resets the hopping table index to zero upon switching. In other words, when switching between tables, the new table always starts at entry 0. By using the Automatic Increment indexing method, the table index is iterated over, as discussed earlier, and as shown in Figure 110.

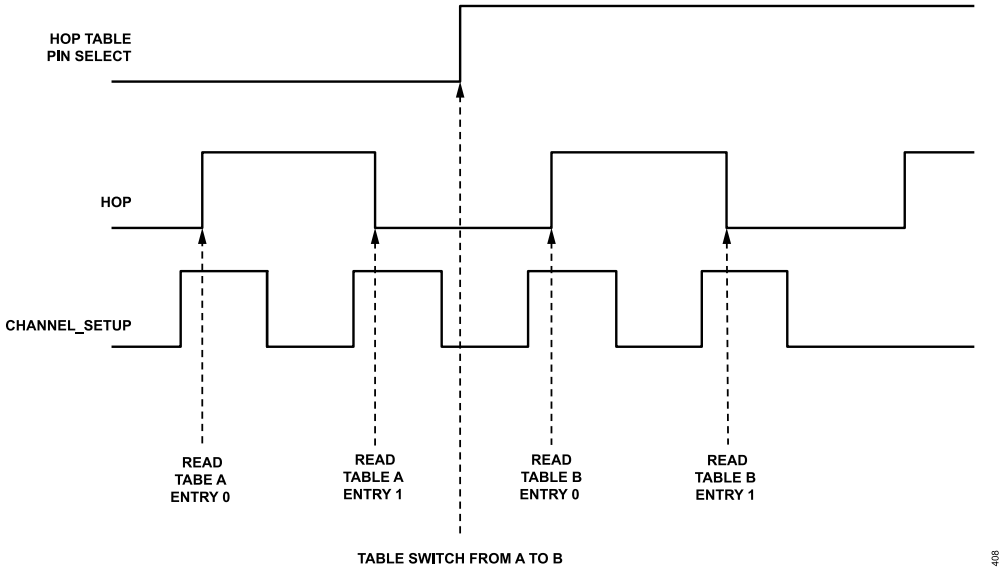


Figure 110. Manual Table Switch with Automatic Increment

Manual Switch with Index by Pin

With manual switch, user can also index the current hopping table by using a DGPIO pin. Note that the ADRV9001 cannot automatically perform ping pong switching between tables when indexing the table by the pin mode.

Frequency Hopping Table Real-Time Process

Pre-process refers to the hopping tables being sent to the ADRV9001 and processed before the FH operation begins. In such a case, user should fully know the hopping table content at the initialization and it cannot be changed on the fly.

Real-time Process refers to the hopping table not being processed at initialization but at the hopping stage. At each hop, the next entry in the current table is read and processed. This allows the update of the hopping tables on the fly. Note this mode also supports loading the table at the initialization, if it is known.

The following examples show two typical uses cases with the real-time process.

Example 1: Load New Frequencies with Automatic Ping Pong

This example, as shown in Figure 111, demonstrates how new frequencies are loaded on the fly in the automatic ping pong mode with the PLL Mux mode. Suppose the automatic ping pong mode is already configured and table A has a single entry loaded before FH operation, a new table with a single entry is loaded for each hopping frame.

FREQUENCY HOPPING

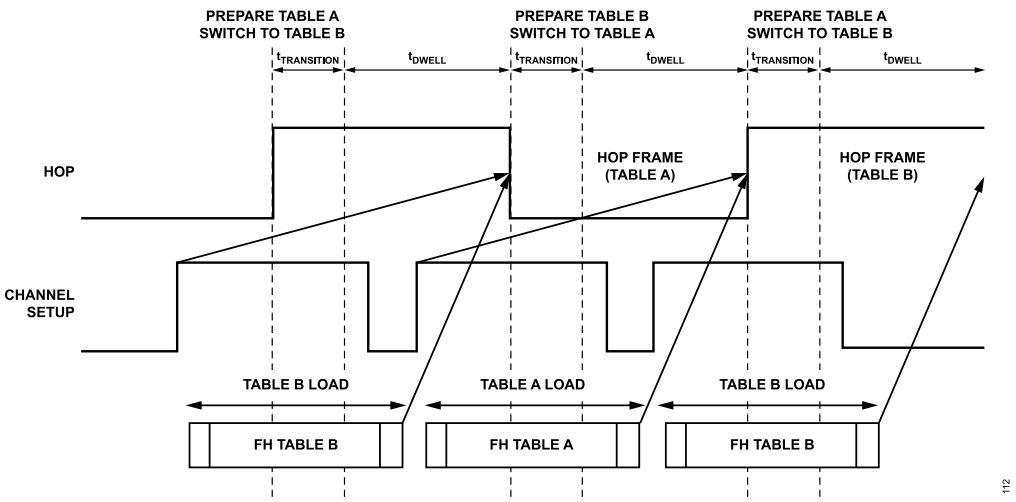


Figure 111. Load New Frequencies with Automatic Ping Pong with PLL Mux Mode

- At the first hop signal rising edge, the ADRV9001 reads the hop table entry from table A (the single entry) and prepares that frequency for the next hopping frame. One PLL starts to retune to this new frequency at the same hop edge. Around the same time, start to load table B (using **adi_adrv9001_fh_HopTable_Static_Configure()**) and make sure the loading of table B completes before the hop signal edge when the table B is read (the first falling edge). Note at the first hop signal rising edge, the ADRV9001 switches to table B, which currently has no frequency information as the loading might not have completed. Therefore, the ADRV9A001 does not read from table B yet at this point. It reads from table B at the upcoming hop signal edge (the first falling edge).
- At the first falling edge of the hop signal, the ADRV9001 reads the new entry from table B and switches to table A. This hop frame (from the first falling edge to the second rising edge) uses the table A entry.
- At the next hop edge (the second rising edge) the BBIC should complete loading a new frequency entry into table A. This frame (from second rising edge to second falling edge) uses the table B entry. And the process is repeated.

By following this process, the user can provide a new frequency on the fly to the ADRV9001 at each HOP edge.

Note that there is no need to load just a single entry. With the automatic ping pong mode, user can load from 1 to 64 entries. However, ensure the frame timing length is sufficient to load the table before the automatic switch. **adi_adrv9001_fh_HopTable_Static_Configure()** should always be used for loading FH tables. **adi_adrv9001_fh_HopTable_Dynamic_Configure()** is only intended to be used on the ADRV9001 evaluation platform.

Example 2: Loading a Larger Set of Frequencies with Manual Table Switch

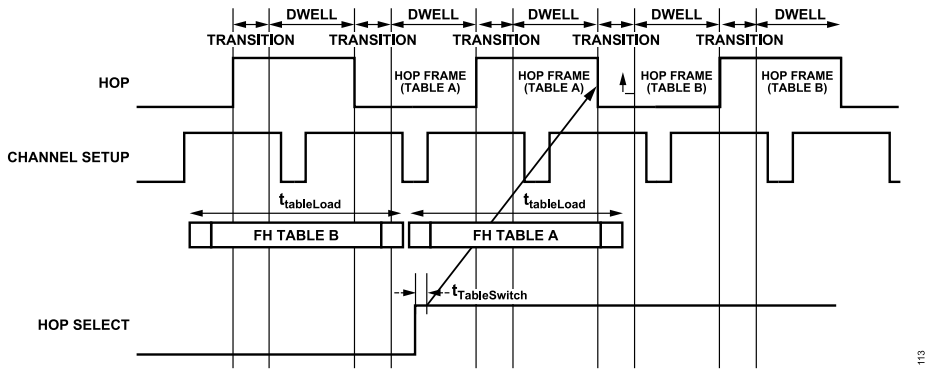


Figure 112. Loading a Larger Set of Frequencies with Manual Table Switch with PLL Mux Mode

This example, as shown in Figure 112, demonstrates how to load a larger set of frequencies on the fly, and then use a DGPIO pin to switch to the new table. This is like the first example. However it requires an additional Hop Table Select signal to switch to the new table. Configure the table indexing mode to automatic increment or index by pin.

FREQUENCY HOPPING

Load a larger frequency hopping table, which is used in multiple hop frames. Once the table is loaded into the memory, set the Hop Select DGPIO pin, allowing the ADRV9001 to start reading from the second table at the next Hop edge.

- Ensure the Hop Table Select Pin is set before the appropriate hop edge.
- Use the Hop Table Select table pin to force the switch to the second table at any time. The ADRV9001 does not require the switch to happen after completion on the first table.
- This example is also applicable to the automatic ping pong mode, as long as the second hop table is loaded before the completion of the first table.

Note: **adi_adrv9001_fh_HopTable_Static_Configure()** should always be used for loading FH tables. **adi_adrv9001_fh_HopTable_Dynamic_Configure()** is only intended to be used on the ADRV9001 evaluation platform.

Dual Hopping Tables

In dual hop mode, tables A and B are subdivided into A1, A2 and B1, B2. LO1 will always switch between A1 and B1, while LO2 switches between A2 and B2. Create and set a table as normal using **adi_adrv9001_fh_HopTable_Static_Configure()**. The table will be divided equally into two parts where the first half is assigned to A1 and the second half is assigned to A2 (or B1/B2).

SELECTING THE CHANNEL AND PROFILE

As mentioned earlier, the ADRV9001 supports single-channel and dual-channel operations. For dual-channel operations, it supports both channel diversity and channel multiplexing modes. Table 50 summarizes all the channel use cases.

Table 50. Channel Selection

Channel Use Case	Profile Propagation Delay Requirements	Number of Channels	Selectable Mode
TRx	Propagation delay must be less than the duration of a single hop frame.	1T1R	**All modes**
		2T2R	**PLL Mux** with hop table real-time process **PLL Retune** with hop table real-time process (diversity) **PLL Retune** with hop table real-time process (multiplexing)
Receiver	Propagation delay can be greater than the duration of a hop frame.	1R	**All modes**
		2R	**PLL Mux** with hop table real-time process **PLL Retune** with hop table real-time process (diversity) **PLL Retune** with hop table real-time process (multiplexing)
Transmitter	Propagation delay can be greater than the duration of a hop frame.	1T	**All modes**
		2T	**PLL Mux** with hop table real-time process **PLL Retune** with hop table real-time process (diversity) **PLL Retune** with hop table real-time process (multiplexing)

FREQUENCY HOPPING OPERATION RANGES

Operation Ranges

- The frequency range is from 30 MHz to 6 GHz.
- The full Rx gain range available in the non-FH mode is supported in all the FH modes.
- The full Tx attenuation range available in the non-FH mode is supported in all the FH modes.

FREQUENCY HOPPING CALIBRATIONS

When FH is enabled, the ADRV9001 calibrates over a range of frequencies. Define the minimum and maximum frequency when configuring the FH operation. If a hopping table is not loaded at the initialization and no frequency range is provided, the ADRV9001 calibrates over 42 discrete regions, from 30 MHz to 6 GHz, to allow to operate with any frequency within this range. However, to reduce the initial calibration time, reduce the number of regions by setting the minimum and maximum operating frequency properly in the FH configuration.

FREQUENCY HOPPING

The ADRV9001 stores the calibration results for the frequency regions. During operation, when it reads the upcoming hopping frequency from the table, it maps it to the appropriate region and applies the corresponding algorithm coefficients.

FREQUENCY HOPPING TIMING

This section shows the timing information for different FH use cases. Read the Timing Parameters Control section in this document because many timing parameters are explained there in detail.

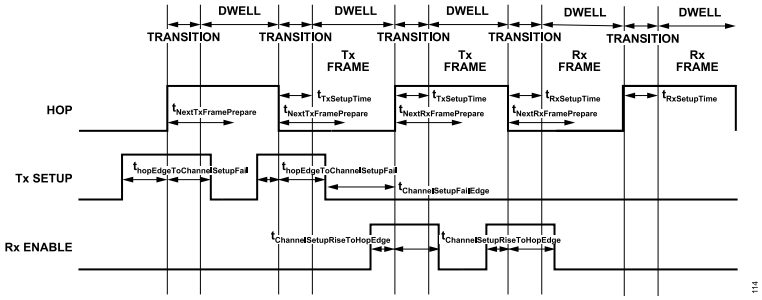


Figure 113. Frequency Hopping Minimum Timing

Table 51. Frequency Hopping Timing Parameters Minimum Timing

Timing Parameter	Description	Time (µs at 184.32 MHz)
$t_{NextTxFramePrepare}$	Time required by the internal controller to prepare the next transmitter frame.	Min: 13 µs
$t_{NextRxFramePrepare}$	Time required by the internal controller to prepare the next receiver frame.	Min: 13 µs
$t_{TxSetupTime}$ (case 1)	Time taken from hop edge to transmitter power up if no LO retune is required and $t_{txRiseToAnaOn}$ is zero.	Min: 6.7 µs
$t_{TxSetupTime}$ (case 2)	Time for a consecutive transmitter frame. Performs transmitter attenuation ramp and LO muxing. The transmitter analog is not fully powered up or down and digital remains on.	Min: 4 µs
$t_{RxSetupTime}$ (case 1)	Time taken from hop edge to receiver power up if no LO retune is required and $t_{rxRiseToAnaOn}$ is zero.	Min: 6.7 µs
$t_{RxSetupTime}$ (case 2)	Time taken for a consecutive receiver frame.	Min: 2 µs
$t_{hopEdgeToChannelSetupFall}$	Minimum time between the hop edge and when the channel setup can go low. This restriction only applies for Tx.	0.42 µs
$t_{ChannelSetupRiseToHopEdge}$	Minimum time required between the channel setup rising edge and hop edge.	At least >3 µs
$t_{ChannelSetupFallToHopEdge}$	Minimum time required between the channel setup falling edge and hop edge. This restriction only applies to Tx.	0.42 µs

Transmitter Timing

For the PLL Mux mode, the Tx setup falling edge starts the Tx interface, as shown in Figure 114. But for the PLL Retune mode, the Tx setup rising edge starts the Tx interface, as shown in Figure 115.

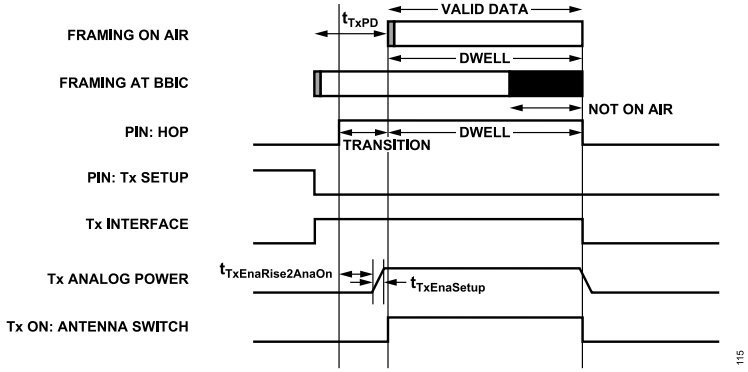


Figure 114. Frequency Hopping Typical Tx Timing for PLL Mux Mode

FREQUENCY HOPPING

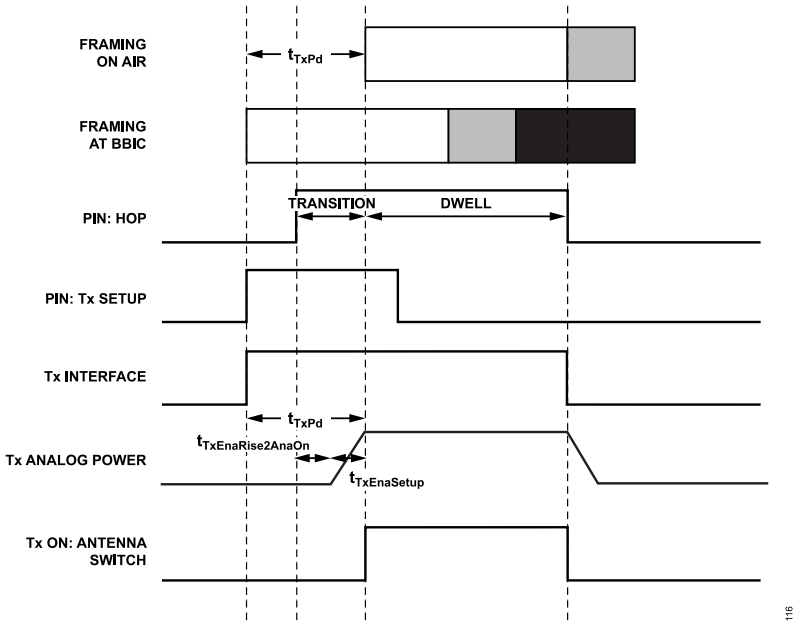


Figure 115. Frequency Hopping Typical Tx Timing for PLL Retune Mode

Table 52 summarizes the Tx timing parameters.

Table 52. Tx Timing Parameters

Delay Parameter	Descriptions	Bounds	Notes
enableSetupDelay ($t_{txEnaSetup}$)	Time taken for the ADRV9001 to power up the analog front end. This time may or may not include the PLL retuning time.	N/A	No PLL retuning @ hop edge: ~5 µs PLL retuning @ frame boundary: ~PLL_retune_time
propagationDelay	Delay from ADRV9001 digital interface to antenna.	N/A	This time is not used for any delays in the ADRV9001. Enable the interface and begin data transmission ample time prior to the end of the transition time to account for the propagation delay.
enableRiseToOnDelay	Delay between hop edge and antenna switching to transmitter channel.	Min: 0 Max: Transition time	If the ADRV9001 is not controlling the antenna switch, this parameter is not needed except to determine other parameters. Typically, the enableRiseToAnalogOnDelay + enableSetupDelay is the transition time, or transition time + guard time.
enableRiseToAnalogOnDelay ($t_{TxEnaRiseToAnaOn}$)	Delays the power up of the AFE relative to the hop edge.	Min: 0 Max: enableRiseToOnDelay - enableSetupDelay	Use this parameter to delay the transmitter analog power up if the transition time is greater than the analog power up time. Also use this to delay the analog power up depending on propagation delay requirements.
enableHoldDelay	Not used in frequency hopping. Delay from hop edge and transmitter interface being disabled.	NA	Hop edge indicates the end of the frame on air. The datapath and interface are not kept on beyond this point.
enableFallToOffDelay	Delay between the hop edge and powering down the analog and transmitter antenna switch. Not used in frequency hopping.	NA	Hop edge indicates the end of frame on air. The analog and transmitter antenna switch can be powered down immediately.

Receiver Timing

Figure 116 shows the typical Rx timing for the PLL Mux mode. For the PLL Retune mode, the timing diagram is similar, as the Rx Setup rising or falling edge does not control the Rx interface. Instead, the timing of Rx interface is relative to the hop signal edge. For the PLL Mux mode, it

FREQUENCY HOPPING

is relative to the hop signal edge after the Rx Setup falling edge. But, for the PLL Retune mode, it is relative to the hop signal edge after the Rx Setup rising edge.

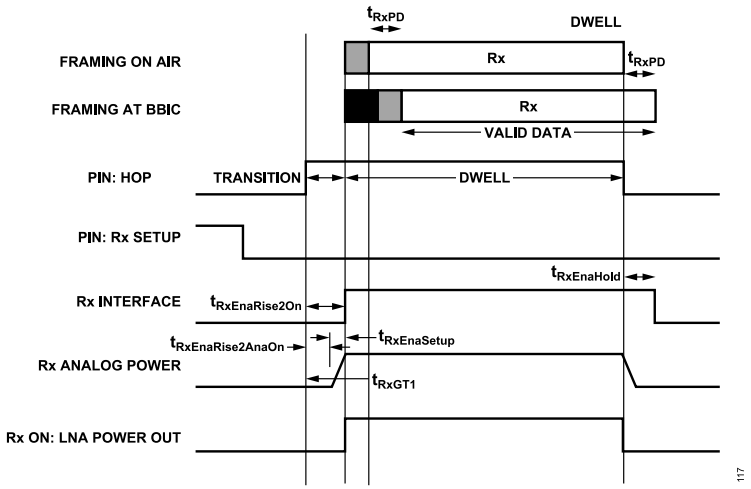


Figure 116. Frequency Hopping Typical Rx Timing for PLL Mux Mode

Table 53. Rx Timing Parameters

Delay Parameter	Descriptions	Bounds	Notes
enableSetupDelay ($t_{rxEnaSetup}$)	Time taken for the ADRV9001 to power up the analog. This time may or may not include the PLL retuning time.	N/A	No PLL retuning @ hop edge: ~7 µs PLL retuning @ frame boundary: ~ PLL_retune_time
propagationDelay	Delay from antenna to the Rx interface.	N/A	This parameter is profile and board layout dependent. Not necessary to configure the ADRV9001, but can be necessary to derive the other timing parameters
enableRiseToOnDelay	Delay between the hop edge and LNA power-up. If the ADRV9001 is not controlling the LNA power-up, this variable is not needed.	Min: enableRiseToOnDelay + enableSetupDelay Max: -	
analogGuardTime	Minimum time between the hop edge and analog power-up to prevent Rx and Tx FE being powered up at the same time.	N/A	~0.15 µs
enableRiseToAnalogOnDelay ($t_{RxEnaRiseToAnaOn}$)	Delays the power up of the AFE relative to the hop edge.	Min: analogGuardTime Max: -	The minimum time of this delay is the analogGuardTime.
enableGuardDelay (t_{RxGT1})	Delay between the hop edge and first valid data received over the air.	Min: enableRiseToOnDelay + enableSetupDelay Max: 0	Not used currently but can be used to delay starting the tracking algorithms until the first valid data is received over the air.
enableHoldDelay ($t_{RxEnaHold}$)	Delay between the hop edge and disabling of the Rx interface.	propagationDelay	At the end of an Rx frame, or a sequence of Rx frames, the interface can be left on, even into the next frame, to allow propagation delay.

TRx Timing

Figure 117 shows the typical TRx timing for the PLL Mux mode.

FREQUENCY HOPPING

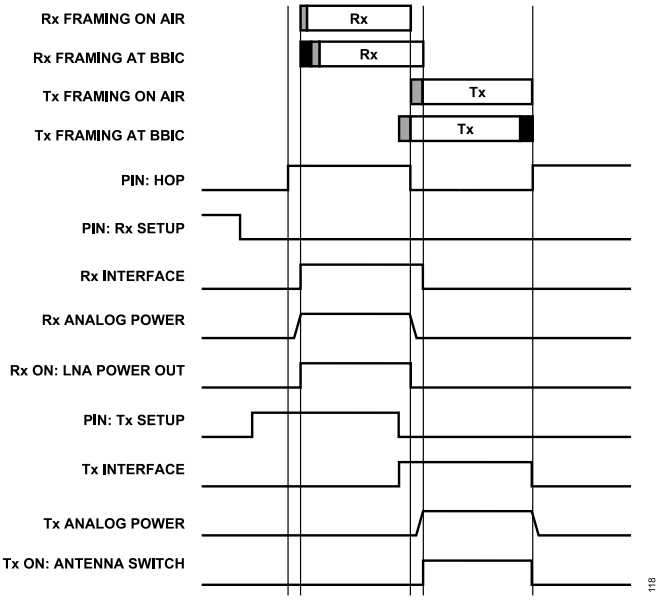


Figure 117. Frequency Hopping Typical TRx Timing for PLL Mux Mode

For the TRx operation, because a hop edge can mark both the start and end of an receiver or transmitter frame, the ADRV9001 guarantees that the receiver and transmitter front ends are not powered up at the same time.

To achieve this, the ADRV9001 enforces a minimum setting for "RxRiseToAnaOn", specified by "analogGuardTime", to ensure that the receiver front end is powered up after the transmitter front end is powered down.

No minimum setting for "TxRiseToAnaOn" is required. This is because the receiver front end is always powered down before the transmitter front end power up routine starts, and no extra delay is required.

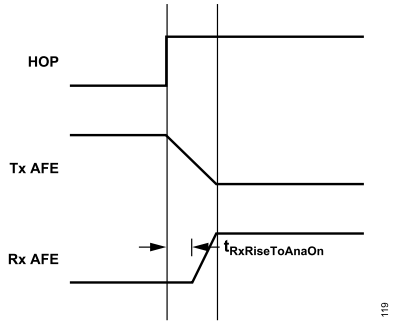


Figure 118. Requirement of RxRiseToAnaOn When Switching from Tx to Rx Hopping Frame

Dual Hop Timing

As per Table 48, 'PLL Retune with dual hop and hop table real-time process' (dual-hop mode) is identical to 'PLL Retune with hop table real-time process' (single-hop PLL retune mode) with the addition of a second independent hop channel.

The frame timing of dual-hop mode is identical to Figure 107 and the channel enablement timings are identical to the 'PLL Retune Mode' timings given in the Frequency Hopping Timing section for each hop channel.

The hop channels in dual-hop mode can be considered fully independent of each other so each hop-channel can have different frame timings, and use either Tx or Rx independent of each other. The two hop channels only have one dependency on each other which is that they can not be enabled at the same time i.e. Hop 1 and Hop 2 can not be triggered in sync.

FREQUENCY HOPPING

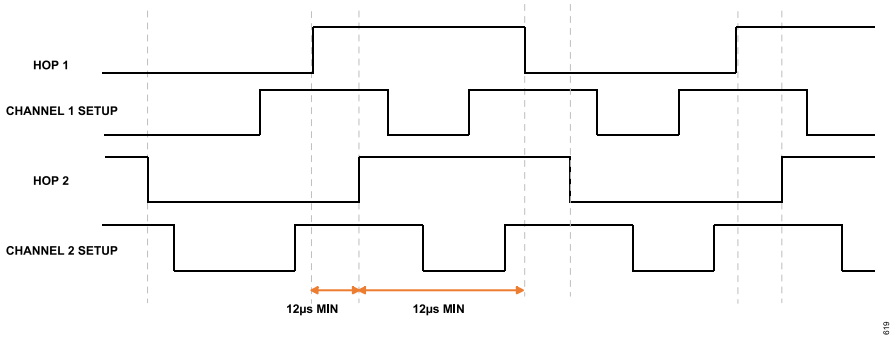


Figure 119. Dual-Hop Timing

Figure 119 shows the relationship between Hop 1 and Hop 2 signals in a dual-hop mode of operation. Hop 1 and Hop 2 are triggered independently but must not be triggered at the same time. There must be a minimum of 12 µs offset between Hop 1 and Hop 2 edges. The timing of the channel setup signals is the same as the 'PLL Retune Mode'.

PLL Retune

In the PLL Retune mode, the time to the start of the Rx/Tx analog front end being powered up is the maximum of: $t_{pllRetune} + t_{FHProcessing}$, and $t_{chRiseToAnaOn}$, where $t_{FHProcessing}$ = 20 µs. $t_{FHProcessing}$ = 20 µs in standard profile for 184.32 MHz system clock. The system clock is variable per profile between ~150 MHz and ~200 MHz. Retrieve the system clock for a desired profile from the 'TDD Enablement Delay' tab in TES. $t_{FHProcessing}$ could be measured for a particular system clock by comparing the transition times between two profiles and subtracting the PLL retune time using the measurement described in the PLL Retune Measurement section. For example when system clock = 152 MHz, $t_{FHProcessing}$ = 26.7 µs, and when system clock = 192 MHz, $t_{FHProcessing}$ = 19.3 µs.

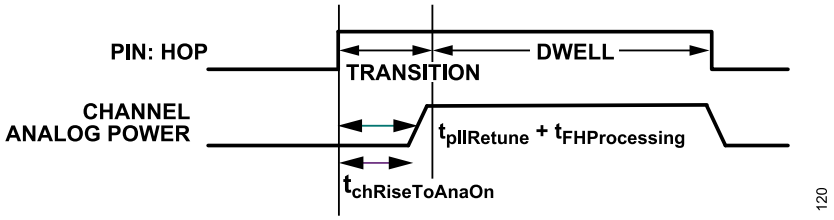


Figure 120. PLL Retune

Table 54. PLL Retune Time (PLL Calibration Mode = Fast)

	PLL Loop BW = 1200 kHz		PLL Loop BW = 300 kHz	
Device Clock (MHz)	Average (µs)	Std. (µs)	Average (µs)	Std. (µs)
30	92.6	6.1	101.3	4.3
38.4	73.2	5.8	83.5	4.4
50	56.7	5.3	68.4	5
100	34.6	3.3	36.3	2.6
150	26	3	36.3	2.6
200	21.3	2.4	31.5	2.2
245.76	18.7	1.2	27	3
300	16.1	1.1	25.5	2.1

Table 55. PLL Retune Time (PLL Calibration Mode = Normal)

	PLL Loop BW = 1200 kHz		PLL Loop BW = 300 kHz	
Device Clock (MHz)	Average (µs)	Std. (µs)	Average (µs)	Std. (µs)
30	380.3	41.5	382.4	51.5
38.4	347.4	36.7	354	21
50	327.7	45.1	325.5	30.4
100	291.2	51.7	298	38.6

FREQUENCY HOPPING

Table 55. PLL Retune Time (PLL Calibration Mode = Normal) (Continued)

	PLL Loop BW = 1200 kHz		PLL Loop BW = 300 kHz	
Device Clock (MHz)	Average (µs)	Std. (µs)	Average (µs)	Std. (µs)
150	275.7	55.2	293.5	38.7
200	256	37.3	274.4	30.1
245.76	252.1	38	269.1	35.8
300	267.2	62.1	265.4	53.6

The table above shows the measured PLL retune time, where 'std.' means the standard deviation. Note the required retune time reduces with the increase of the device clock frequency (PFD). Both 'Fast Calibration' and 'Normal Calibration' modes are provided. 'Fast Calibration' mode reduces the retune time but has limited temperature tracking. 'Normal Calibration' mode increases the retune time but has full temperature tracking. 'Normal Calibration' mode should only be used in cases of a long frame time where temperature might change significantly. The PLL loop bandwidth is another configurable parameter of the PLL. A loop BW of 1200 k was found to give optimal retune timing for fast calibration mode.

Note: the PLL retune time is defined as the time from the start of PLL retuning until the time when the PLL settles within 500 Hz of the settled frequency.

PLL Retune Measurement

The PLL retune time can be measured by observing the PLL settling time between two consecutive Tx frames. A summary of the measurement is given in this section. An illustration of two consecutive Tx frames in Figure 121 shows the relationship between the 'Hop' and 'Tx Setup' signals as well as the 'Tx On Air' for a FH LO retune case. Note that $t_{transition}$ is the sum of $t_{FHProcess}$ and $t_{pllRetune}$.

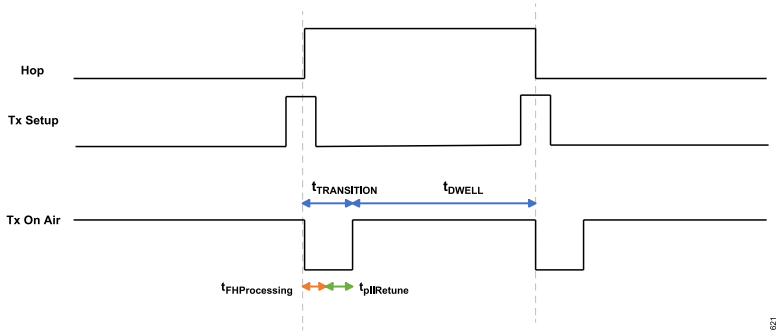


Figure 121. Measuring PLL Retune Time Timing Diagram

The goal of the measurement will be to measure the transition time as shown in Figure 121. The general method is as follows:

1. Two consecutive Tx frames are used. The full frame time should be at least $t_{pllRetune}$ + $t_{FHProcessing}$ (based on Table 54) and $t_{chRiseToAnaOn}$ should be set to 0. For example, for a 38.4 MHz device clock, 300 kHz loop BW, and using fast calibration mode, the minimum frame time would be 83.5 µs + 20 µs = 103.5 µs. Some margin should be added to clearly see the 'on air' time. Ensure to align Tx DMAs correctly or simply have DMAs always on. Otherwise, the Tx datapath could be on but no Tx data is propagating through. See the Frequency Hopping Timing section for further details on the timing parameters.
2. Configure the FH with 'LO retune mode'. Start the FH operation and ensure that the operation works as expected. 'LO retune mode' is used to view the transition time clearly. All other modes will have the same $t_{pllRetune}$.
3. Connect the Tx output to a spectrum analyzer with a time overview mode and frequency settling mode. Trigger the capture using the hop edge for best results. Ensure that the frame timing is as expected using the time overview mode.
4. Enter the frequency settling mode. The retune time can be measured by observing the time it takes to settle the PLL/Tx output. Set the frequency tolerance to +/− 500 Hz to match the above measured results or set desired limits. Measure the transition time by observing the 'settling time' between two frames. The settling time is defined as the time between the Tx output leaving the defined limits until it re-enters the defined limits. This is equivalent to the transition time. $t_{pllRetune}$ can then be calculated as $t_{transition}$ (measured) − $t_{FHProcess}$ (20 µs).

Example: the following capture was taken of a FH transition time for the following criteria: 38.4 MHz device clock, 300 kHz loop BW, fast calibration mode. A FH table was constructed with two entries where the frequencies were similar (1 Hz apart) and the Tx attenuation was

FREQUENCY HOPPING

changed by 10 dB. This FH table was used to easily see the transition between two Tx frames. An ADRV9001 evaluation board and a Tektronix RSA306B spectrum analyzer was used in this measurement. The transition time was measured to be 103.93 µs. From this, the $t_{pllRetune}$ was calculated to be 83.93 µs (103.93 µs – 20 µs) which closely matches the results given in Table 54 (83.5 µs).

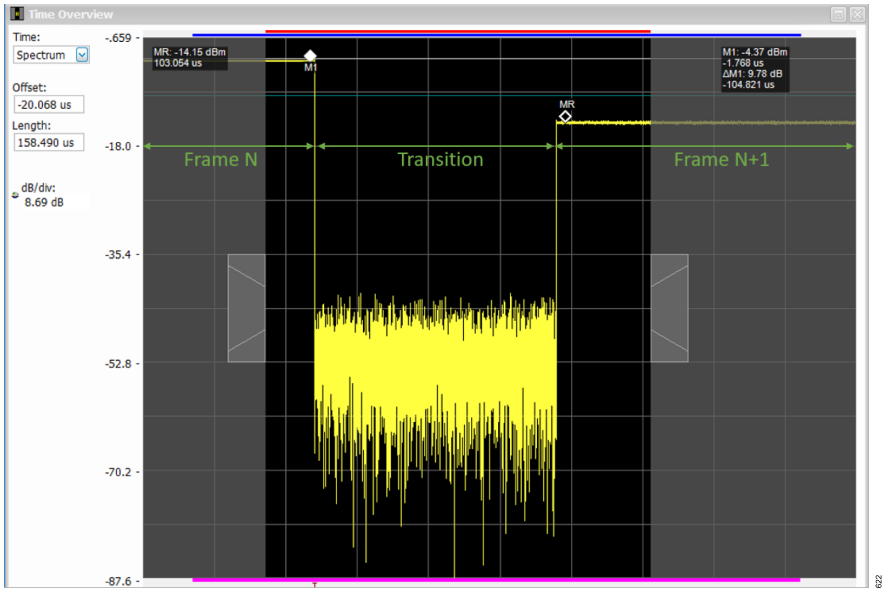


Figure 122. PLL Retune Time—Time Overview

The time overview mode shows the end of frame N, the transition time, and start of frame N+1. Frame N power level is higher than frame N+1 as the Tx attenuation was defined as higher in the FH table.

Note: the markers don't accurately measure the transition time. These are only helpful to see the relationship between the two views.

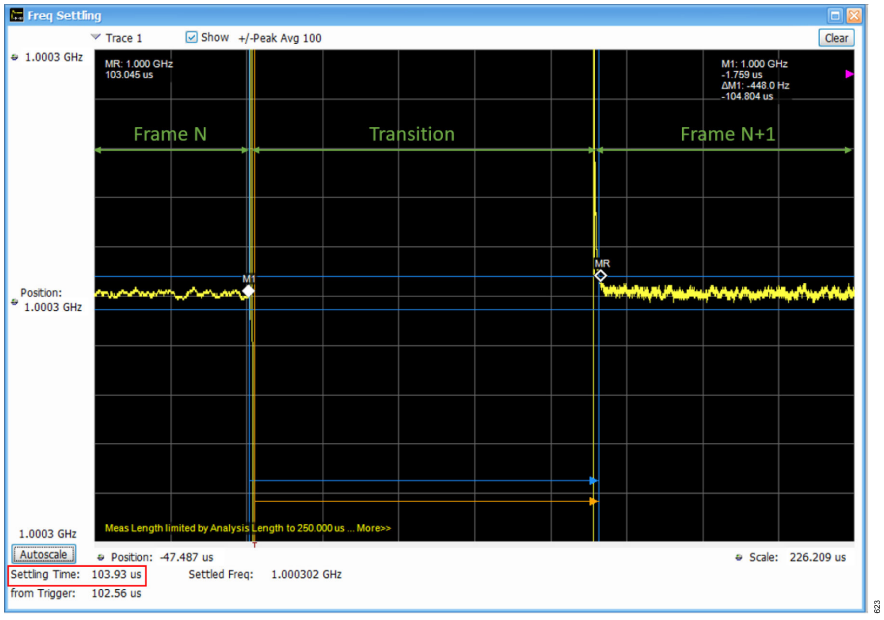


Figure 123. PLL Retune Time—Frequency Settling

In the frequency settling view, the same frames can be seen. In this view, the time taken to for the frequency to settle can be measured. The blue horizontal lines indicate the +/− 500Hz frequency limits. The settling time can be read directly as 103.93 µs.

FREQUENCY HOPPING

Minimum FH Frame Timing

As per Table 48 and Table 54, the minimum frame timing can be achieved by using:

- PLL Mux with hop table preprocess mode
- 300 MHz device clock (Max PFD is actually 307.2 MHz which could produce marginally faster retune time)
- 1200 k loop filter bandwidth.

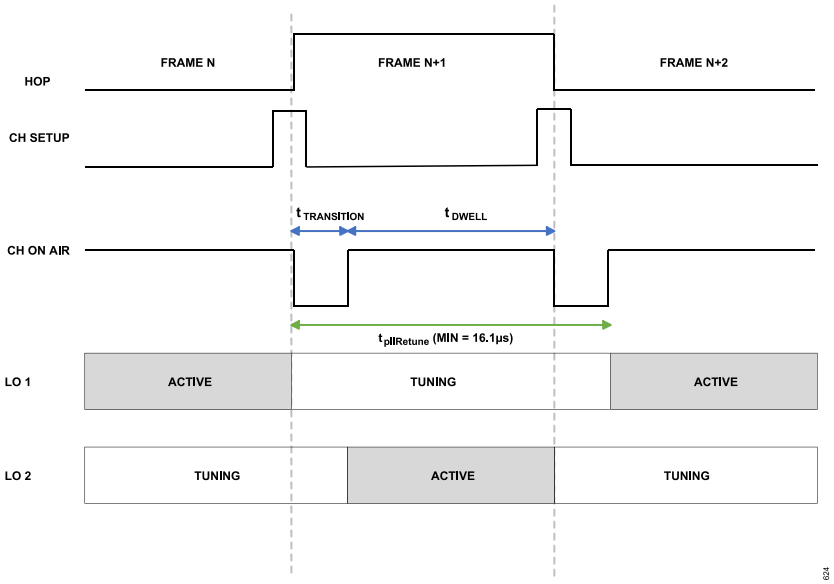


Figure 124. Minimum Frame Timing Strategy

The above figure shows a strategy to allow for the minimum frame timing. As detailed previously, the minimum PLL retune time is 16.1 µs as per Table 54. As mux mode uses two PLLs that are switched between, only one PLL needs to be active at a time. Furthermore, the PLL only needs to be tuned for one dwell time, as shown. The remainder of the time, both PLLs can be retuning for upcoming frames. In practice, this means that a PLL can take 1 dwell time and 2 transition times to retune. The time needed for transition and dwell timing is user defined. Using this strategy, the minimum frame time can be reached.

Note: in 'mux with hop table preprocess' mode, $t_{FHProcessing}$ is not part of the frame time as this processing is done at the initialization stage.

ADDITIONAL FREQUENCY HOPPING OPERATIONS

Receiver Only with Long Propagation Delay

The ADRV9001 supports the scenario when Rx propagation delay is greater than the duration of a hopping frame. This, however, is only supported with the Rx-only modes. The Tx-only mode is described in the next section.

To achieve this, set the timing parameter, enableHoldDelay, to the propagation delay to keep the Rx datapath and interface on after the hopping frame, or a series of hopping frames, has ended on air. Figure 125 shows an example timing diagram with the PLL Mux mode.

FREQUENCY HOPPING

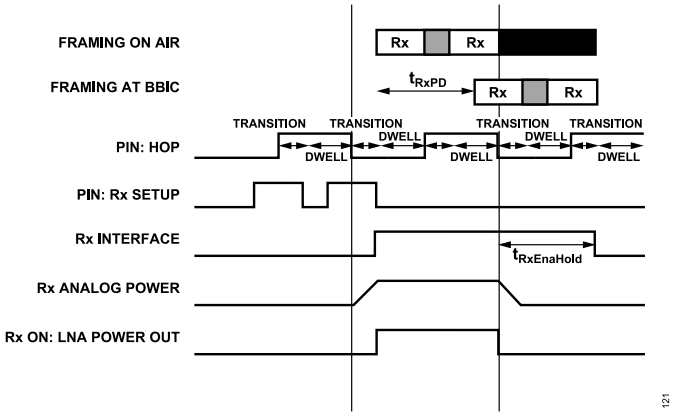


Figure 125. Rx Only with Long Propagation Delay with PLL Mux Mode

Transmitter Only with Long Propagation Delay

The ADRV9001 also supports the transmitter only case, where the propagation delay is greater than the duration of a hop frame.

To achieve this, specify a parameter as a part of the FH configuration to delay the powering up of the transmitter analog in terms of hop frames. Then time the enabling of the interface, along with using the "txRiseToAnalogOn" timing parameters, to fine-tune the delay.

The delay parameter specifies the delay in terms of hop frames after the first transmitter setup rising edge is received. By design, the ADRV9001 enforces a minimum delay for both the transmitter and receiver of 1 in the PLL Mux mode. However, for transmitter only, this delay can be greater than 1. If set to 0, the ADRV9001 defaults the delay to 1.

After the first transmitter setup rising edge, then continuously send a pulse train of transmitter setup signals until the hop edge, in which the transmitter frame begins on air. After this, maintain any number of consecutive transmitter frames, as long as the transmitter setup is continuously toggled. After the pulse train of transmitter setup stops (meaning a hop edge without a preceding transmitter setup rising edge), the transmitter channel is powered down at the next hop edge. A subsequent transmitter setup begins the process again.

This example shows a four-frame delay, with the transmitter frame starting on air at the fifth frame after the first transmitter setup rising edge.

To account for the transition period between the consecutive frames, pad the valid data with guard symbols. These are marked by the gray boxes, as shown in Figure 126. Also, pad the data to keep the transition and dwell times consistent. This is because the transition required for the first transmitter frame is greater than for consecutive frames.

There is also the option to program the "riseToAnalogOn" timing parameter to 0, indicating the analog power-up can be powered up immediately after the hop edge. Then, time the data to the antenna based on when the transmitter setup rising edge is dropped and start transmitting data.

For the PLL Mux mode, the transmitter setup falling edge marks the beginning of the interface. "txAnalogPowerOnFrameDelay" starts to decrease from the original value when the hop signal first samples a high of the transmitter setup signal until it is decreased to 0. When reaching 0, it waits another frame before the transmitter analog is powered on due to the 1 frame delay in the PLL Mux mode. The value remains 0 until the transmitter setup is sampled with low, and the value then sets back to the original value.

FREQUENCY HOPPING

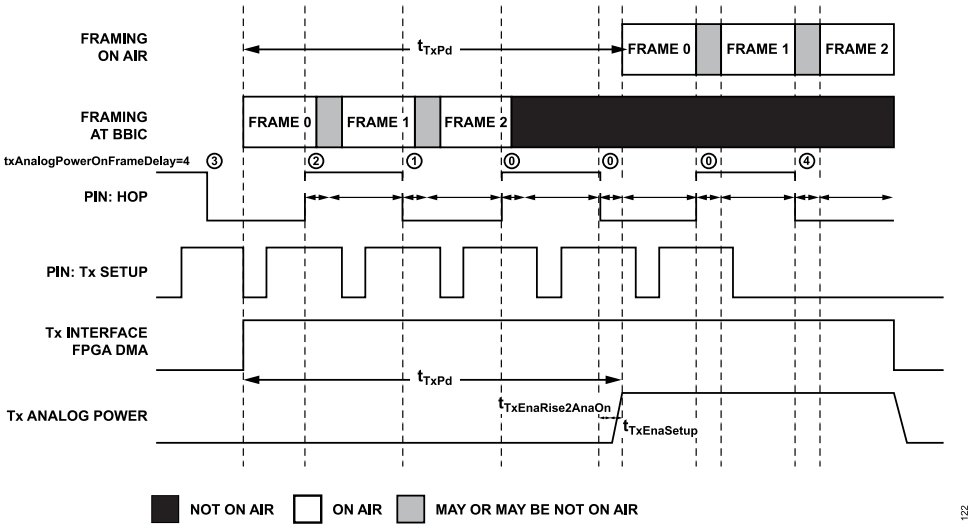


Figure 126. Tx Only with Long Propagation Delay for PLL Mux Mode

For the PLL Retune mode, the difference is the transmitter setup rising edge, which now marks the beginning of the interface. "txAnalogPower-OnFrameDelay" starts to decrease when the hop signal first samples a High of transmitter setup signal, until it is decreased to 0. When reaching 0, the transmitter analog is powered on as there is no 1 frame delay in the PLL Retune mode. The value remains 0 until the transmitter is sampled with low, and the value is then set back to the original value.

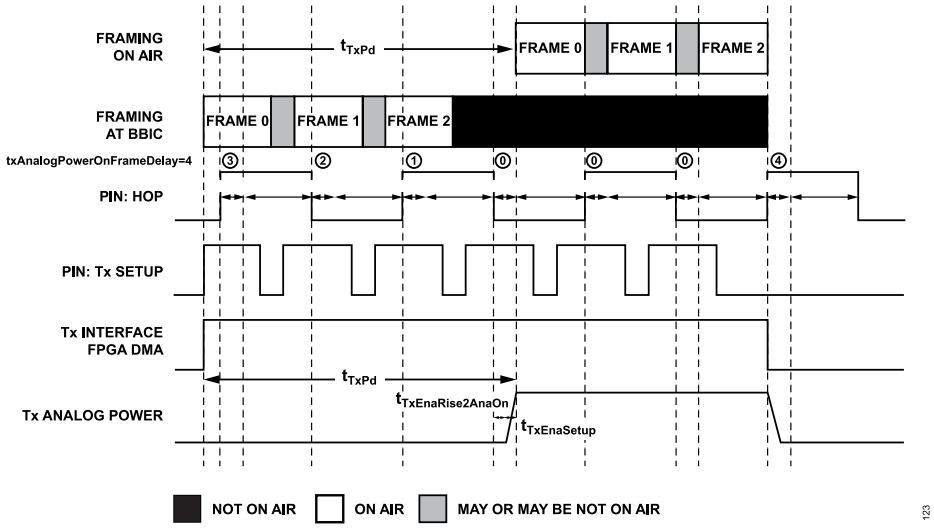


Figure 127. Tx Only with Long Propagation Delay for PLL Retune Mode

Transmitter Only with Short Propagation Delay

There is a restriction to how long before the hop edge the transmitter setup falling edge must come. For profiles with very short propagation delays, this means the interface is on longer than required for valid data to reach the analog front end.

Figure 128 shows one option to pad the data with zeros or a known pattern and start transmitting as soon as the interface is enabled. This way, although the interface is on earlier, the transmitted data is a known pattern when it reaches the analog front end.

FREQUENCY HOPPING

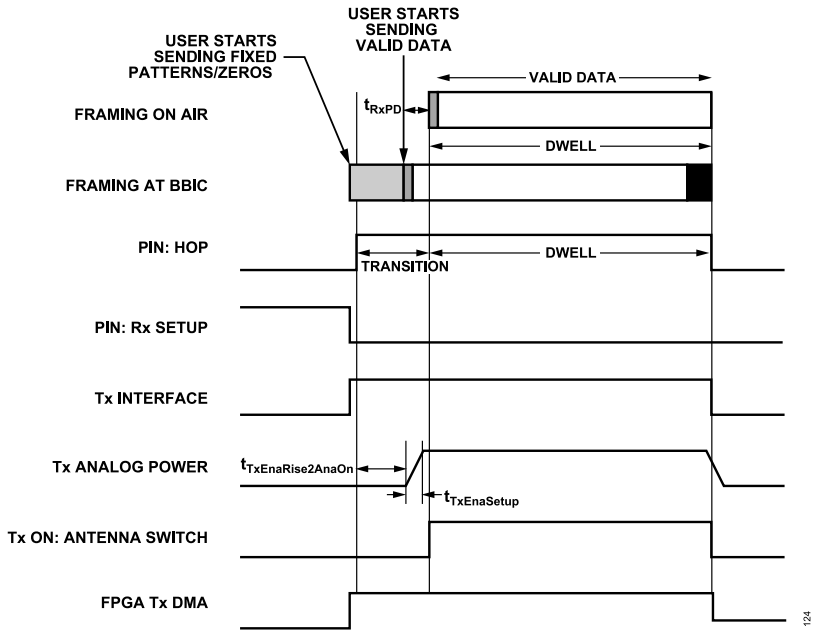


Figure 128. Padded Data Method for Tx Only Hopping with Short Propagation Delay

Figure 129 shows an alternative to hold off from the transmission until $t_{propagationDelay}$ prior to the analog front end is enabled. Here, the FPGA transmitter DMA involves to delay the transmitting data until the analog front end is ready.

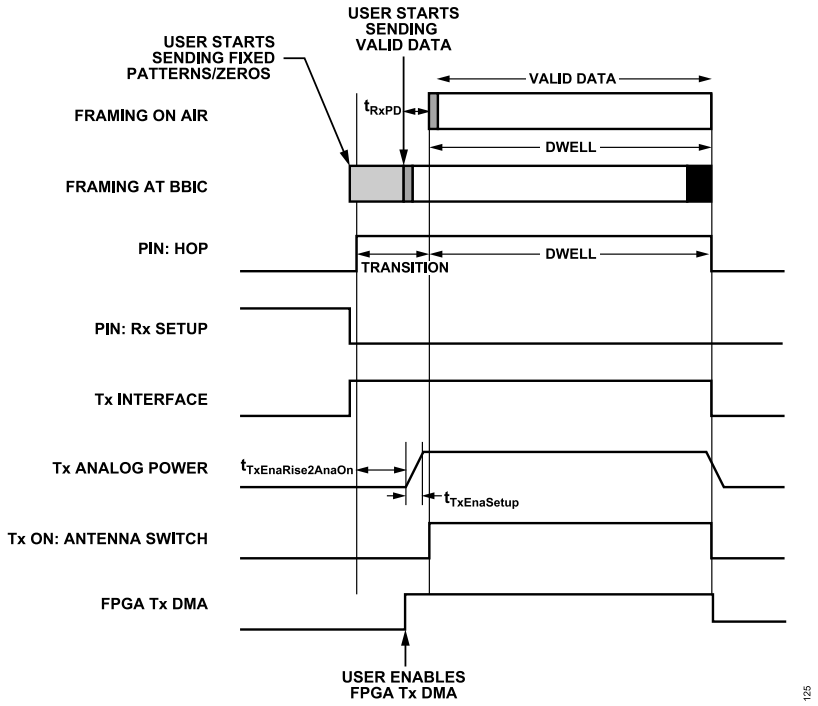


Figure 129. FPGA Delay Method for Tx Only Hopping with Short Propagation Delay

ORx Operation

The ADRV9001 supports ORx operation in the FH modes. ORx in FH works like the normal TDD mode. During the actual Tx frame, set the ORx enable signal (assign to a DGPIO pin) high to enable ORx. Before the end of the Tx frame, set ORx low to disable ORx. Note this is required regardless of whether the next frame is Tx or Rx. Take the ORx setup and bring down time into consideration, as well as the

FREQUENCY HOPPING

propagation delay of the profile. Table 56 shows the required time to set up and bring down the ORx and Table 57 shows the ORx timing restrictions.

Table 56. ORx Timing Parameters Time Required

Timing Parameter	Time Required (µs)	
	Narrowband	Wideband
$t_{OrxRiseToOn}$	8	9
$t_{OrxFallToOff}$	6	5

Table 57. ORx Timing Parameters Timing Restrictions

Timing Parameter	Timing Restriction
$t_{HopEdgeToORxRise}$	Must be greater than the transition time, or $t_{TxRiseToOn}$.
$t_{ORxFallToHopEdge}$	Allow enough time for the ORx disable to complete.

When using ORx in frequency hopping, consider the following parameters to determine the frame duration:

- ORx enable rising edge time ($t_{OrxRiseToOn}$)
- ORx enable falling edge time ($t_{OrxFallToOff}$)
- Propagation delay ($t_{propagationDelay}$)

The total time the ORx is enabled should be at least ($t_{OrxRiseToOn}$ + $t_{propagationDelay}$), otherwise no valid data is received.

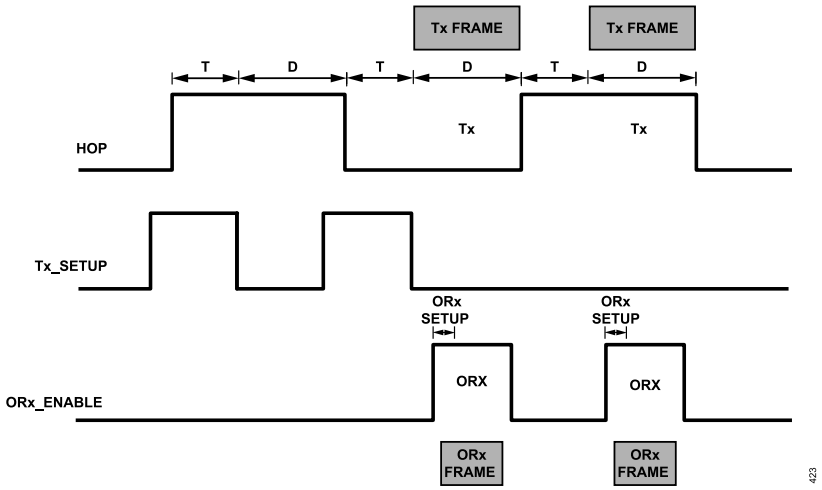


Figure 130. ORx Timing with Tx Setup with PLL Mux Mode

DIVERSITY MODE

The ADRV9001 supports the diversity mode in FH. However, there might be more timing constraints to consider.

Basically, the ADRV9001 requires increased time to prepare for an upcoming frame if two channels are enabled, this increased time must be taken into consideration.

To enable two channels for diversity, the operation is like a 1T1R use case. However, control the setup signal for both channels. Prior to the first hop edge, set channel 1 setup rising edge and channel 2 setup rising edge high. There is no restriction on when these signals come relative to each other. However, the last one set should be 1 µs prior to the hop edge.

FREQUENCY HOPPING

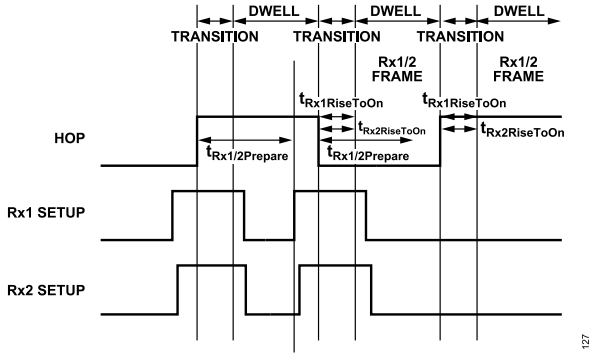


Figure 131. Diversity Timing Example, Rx1/Rx2

FREQUENCY HOPPING WITH RX/ORX GAIN CONTROL

In FH operation, configure the receiver gain control as either manual gain control or automatic gain control (AGC).

Configure the AGC as in the non-FH mode and set it to either reset at the start of each Rx frame to a starting gain index, as specified in the hopping table, or to continue from the previous receiver frame.

In the ADRV9001, the AGC configuration has the field "*resetOnRxOn*". For AGC to continue from the previous Rx frame, set *resetOnRxOn* to FALSE. For AGC to reset to a starting gain index each frame, set *resetOnRxOn* to TRUE.

In FH, if the AGC is configured and the *resetOnRxOn* field is true, then the starting Rx gain for each frame is taken from the hopping table instead of the *resetOnRxonGainIndex* field in the AGC configuration.

With manual gain control, the ADRV9001 programs the manual gain index for each frame based on the gain information in the hopping table or through the DGPIOs.

The ADRV9001 allows up to 3 DGPIO pins to select the Rx gain or Tx attenuation for each hopping frame from a Rx gain or Tx attenuation table with a maximum of eight entries. The GPIO pins formulate an index to the defined Rx gain or Tx attenuation table and the GPIO pins are sampled at the hop edge. Then, the corresponding Rx gain index or Tx attenuation values are applied to the corresponding upcoming hopping frames.

ORx Gain Control

The ORx gain operates in manual gain control. In this mode, set the desired starting gain index before enabling ORx. Update the gain throughout the ORx frame by calling adi_adrv9001_ORx_Gain_Set() API.

SPECIAL FREQUENCY HOPPING OPERATIONS

Port-Based Frequency Hopping

As noted previously, all four channel ports (Rx1, Rx2, Tx1, Tx2) can be used in FH. However, these ports are not indicated in the hopping tables. Instead, the channel enable pins are repurposed as channel setup pins to select the channels to hop upon. In the dual hop LO Retune mode, there are also two hop signals to indicate when new frames start, one for each LO.

Each port is also assigned a controlling LO, which is fully configurable in certain FH modes. For example, LO1 controls channel 1 ports (Rx1/Tx1) and LO2 controls channel 2 ports (Rx2/Tx2), or LO1 controls both receive ports (Rx1/Rx2) and LO2 controls both transmit ports (Tx1/Tx2), or LO1 controls any other combination such as Rx1 and LO2 controls Rx2, Tx1 and Tx2.

Configure the LO assignment for each in the adi_adrv9001_FhCfg_t structure. In this structure, there are fields for the Rx ports (rxPortHopSignals) and the Tx ports (txPortHopSignals). In these fields, each port can be assigned a "HOP" signal, where hop signal 1 is for LO1 and hop signal 2 is for LO2.

NCO-Only Frequency Hopping

The ADRV9001 provides a numerically-controlled oscillator (NCO)-only mode, which is only used for Rx FH. By enabling this feature, it is possible to define a non-zero rx1OffsetFrequencyHz or rx2OffsetFrequencyHz in the hopping table and those frequencies are applied to the

FREQUENCY HOPPING

NCO only. It adds an offset to the current LO frequency. Note that IF operation is supported in the regular Rx FH mode. The IF frequencies are also defined using rx1OffsetFrequencyHz and rx2OffsetFrequencyHz. The difference is that in the regular mode, both the LO and NCO frequencies are changed, while in the NCO-only mode, only the NCO frequencies are changed but LO frequencies are not affected. This is the fundamental difference between the regular Rx FH mode and NCO-only Rx FH mode.

The NCO-only mode changes the NCO frequency at the beginning of each hop frame based on the hopping table entry it is using. Also, the NCO-only mode can be utilized to perform frequency changes or "hops" anytime within a hop frame by using a GPIO pin. To use this feature, assign a GPIO to trigger the NCO change. Update the desired Rx frequency offset using **adi_adrv9001_fh_RxOffsetFrequency_Set()** API at any point, but it does not take effect until the NCO change pin is toggled (both rising and falling edges). When the GPIO pin is toggled, the offset frequency set through the API is used and the original offset frequency defined in the hopping table is ignored. At the start of the next frame, the NCO frequency is first obtained from the hopping table until the NCO change pin is toggled.

Figure 132 shows a timing diagram for an NCO-only example. This timing diagram examines the actual RF frequency of the Rx1 port. For simplicity, all frames are for Rx1. This figure shows two frames: "n" and "n+1". The first point to note here is that the carrier frequency for each frame is taken from the FH table and does not change during the frame. The dynamic elements of the diagram are the Rx1 offset frequency, the NCO 1 change pin, and the Rx1 output frequency. Throughout the frames, the BBIC loads new values for the Rx1 offset frequency. The first offset frequency is retrieved from the hopping table. It adds an offset to the Rx1 output frequency, which does not change until the NCO 1 pin is toggled. When the NCO 1 change pin is toggled, the offset frequency is updated based on the value set through the API, and thus the Rx1 output frequency is shifted correspondingly.

Table 58. Simplified Hop Table for NCO-Only Hop Example

Index	Carrier Frequency (MHz)	Rx1 Offset Frequency (kHz)
n	500	500
n+1	600	500

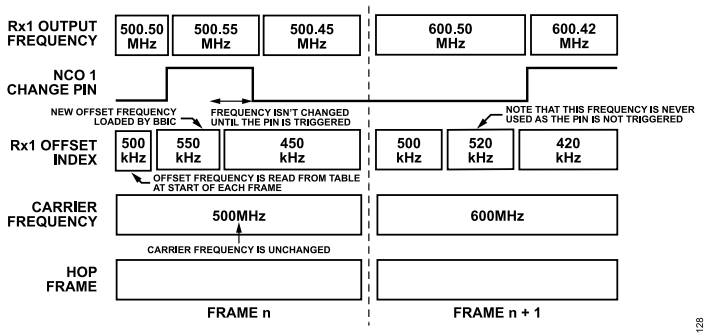


Figure 132. NCO-Only Hop Timing Diagram Example

Frequency Hopping with PFIR Switching

The frequency hopping with PFIR switching allows the changing of the programmable FIR filter at each hop frame. Each channel (Rx1, Tx1, Rx2, Tx2) has two PFIRs associated with it: primary and secondary PFIRs. At each hop, the PFIRs will be automatically switched between. Figure 133 shows the operation of the PFIR switching.

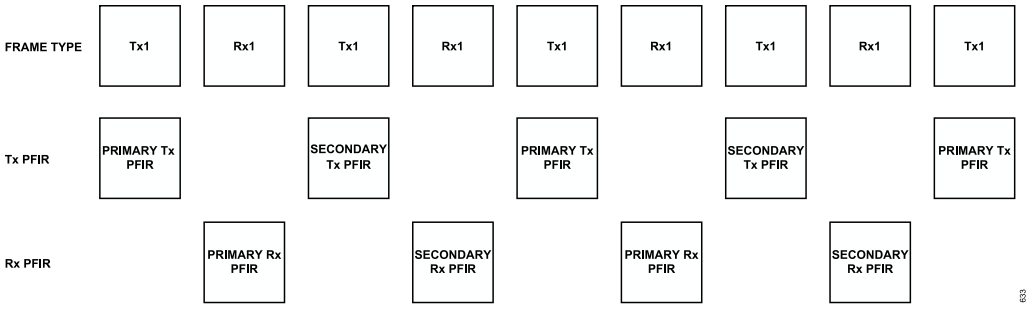


Figure 133. FH with PFIR Switching Operation

FREQUENCY HOPPING

For each channel, the PFIR is switched automatically between the primary and secondary PFIR on each frame of that channel e.g. For Tx1, the primary Tx PFIR is used for the first Tx1 frame and switched to the secondary PFIR for the second Tx1 frame. The PFIRs are switched automatically so no extra control signal is needed.

It is also possible to upload a new PFIR dynamically as long as it is not in use e.g. uploading a new PFIR to the primary Tx1 PFIR while the secondary Tx1 PFIR is in use. In this way, a new PFIR can be used at each hop allowing an arbitrary number of PFIRs to be employed.

Due to frame timing constraints, this feature only operates in the LO Retuning mode so cannot be used in either muxing mode.

PFIR Switching in TES

In the configuration section of the frequency hopping pane in TES, there is an option to select 'LO Retuning/Dynamic PFIR' in the 'Hop Mode' dropdown.

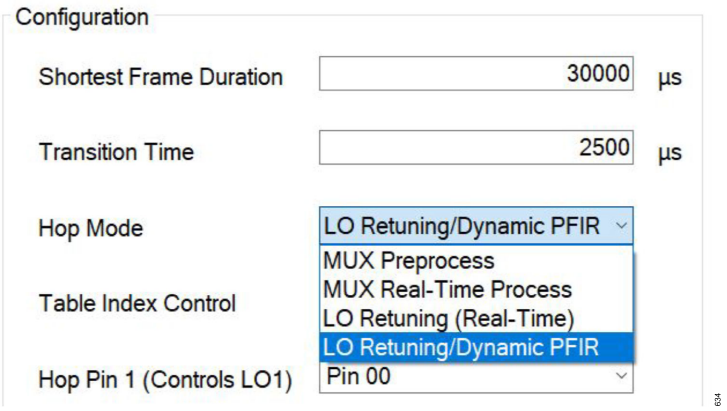


Figure 134. TES Configuration 1

Depending on the mode selected, there will be a section enabled to upload each PFIR (note that it is valid to upload identical PFIRs to more than one bank if desired).

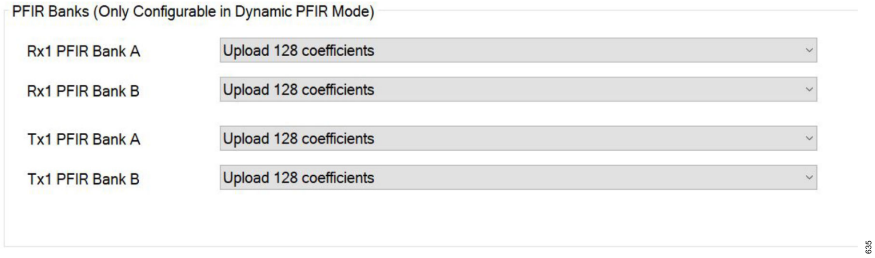


Figure 135. TES Configuration 2

Set up the rest of your FH configuration as desired. Now, when hopping, it can be seen that the PFIRs are switched automatically.

FREQUENCY HOPPING

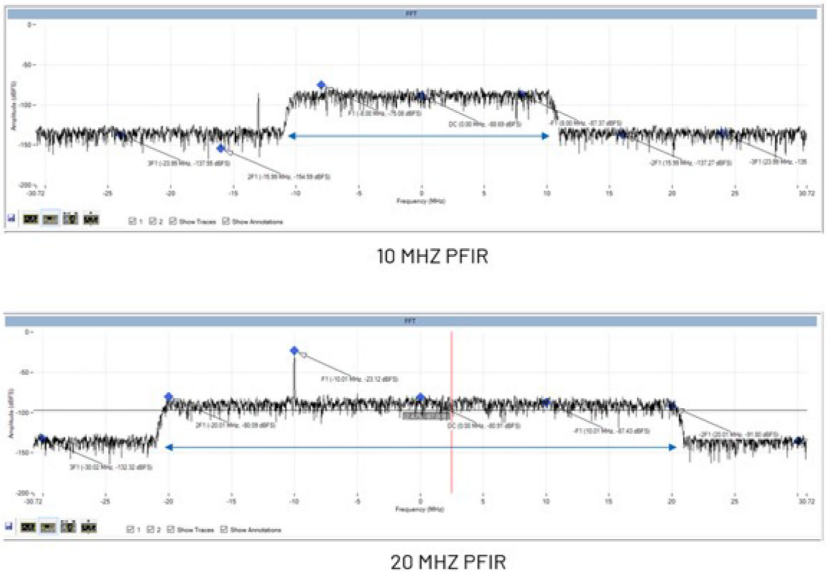


Figure 136. Rx1 PFIR Switching

INTEGRATION WITH OTHER ADVANCED FEATURES

Frequency Hopping with MCS

The Frequency Hopping section describes frequency hopping (FH) with multichip synchronization (MCS).

Frequency Hopping with DPD

Frequency Regions

The ADRV9001 supports FH to work with digital predistortion (DPD). The ADRV9001 divides frequencies into 8 frequency regions. Specify the start and end frequencies of 7 regions. There is one last region (8th region) to "catch all", which captures the remaining range of the frequencies. For example, for 6 regions, specify 5 regions and the rest are in the 8th region.

DPD calculates its coefficients based on the specified regions. This means when transmitting on frequency within a certain region, DPD operates only for that region. This includes capturing the data, calculating the coefficients, and lastly, updating the coefficients in the actuator.

To define frequency regions, provide the start and end frequencies in **adi_adrv9001_DpdFhRegion**.

```
typedef struct adi_adrv9001_DpdFhRegions
{
uint64_t startFrequency_Hz; //!< Carrier frequency greater than or equal to this is included in the
region
uint64_t endFrequency_Hz; //!< Carrier frequency less than this is included in the region
} adi_adrv9001_DpdFhRegions_t
```

Then configure the DPD frequency hopping regions in **adi_adrv9001_dpd_fh_regions_Configure()**, which takes an array of the frequency regions.

```
int32_t adi_adrv9001_dpd_fh_regions_Configure(adi_adrv9001_Device_t *adrv9001,
adi_common_ChannelNumber_e channel,
adi_adrv9001_DpdFhRegions_t dpdFhRegions[],
uint32_t size);
```

FREQUENCY HOPPING

FREQUENCY HOPPING API PROGRAMMING

Table 59. Data Structures and Enums Related to Frequency Hopping

Data Structure/Enum	Description
adi_adrv9001_FhHopFrame_t	Settings for HOP frame information.
adi_adrv9001_FhGainSetupByPinCfg_t	Frequency hopping gain setup by pin config. Up to 8 receiver gains and 8 transmitter attenuation levels are loaded to the ARM memory and indexed during frequency hopping operation by up to three GPIO pins. This structure provides the configuration data for GPIO assignment and gain/attenuation table loading. Load any valid receiver gain index or transmitter attenuation level in the tables.
adi_adrv9001_FhhopTableSelectCfg_t	Hop Table Select configurations
adi_adrv9001_FhCfg_t	Contains ADRV9001 Frequency Hopping data types
adi_adrv9001_FhFrameIndex_e	Enumeration of hop frame indices.
adi_adrv9001_FhHopSignal	Enumeration of hop signals.
adi_adrv9001_FhHopTable_e	Enumeration of frequency hopping table IDs.
adi_adrv9001_FhHopTableSelectMode_e	Enumeration of frequency hopping table select modes.
adi_adrv9001_FhMode_e	Enumeration of frequency hopping modes.
adi_adrv9001_TableIndexCtrl_e	Enumeration of frequency hopping table indexing control mode

Table 60. API Commands Related to Frequency Hopping

Function Name	Description
adi_adrv9001_fh_Configure()	Configures Frequency Hopping settings.
adi_adrv9001_fh_Configuration_Inspect()	Inspects/reads the parameters in the frequency hopping configuration data structure..
adi_adrv9001_fh_FrameInfo_Inspect()	Gets hop frame information for specified index. The ADRV9001 maintains the state for two frequency hopping frames: current frame and upcoming frame (frame at next hop edge). This command fetches the hop frame information, as specified by adi_adrv9001_FhHopFrame_t, from any of these states. Receiver Offset frequency information is only valid for receiver Channel(s) in the RF_ENABLED state.
adi_adrv9001_fh_Hop()	Triggers the hop signal by either SPI or mailbox. This API has no effect if it is called before any channel, enabled for frequency hopping, is in the PRIMED state.
adi_adrv9001_fh_HopTable_BytesPerTable_Get()	Gets number of SPI packed bytes per dynamic FH table load.
adi_adrv9001_fh_HopTable_Dynamic_Configure()	**This API is only used for ADRV9001 evaluation system. Do not use in own system.**
adi_adrv9001_fh_HopTable_Get()	Gets the hop table currently in use.
adi_adrv9001_fh_HopTable_Inspect()	Inspects frequency hopping table in ARM memory. This function reads back a frequency hopping table from the ARM memory and loads it into hopTable. The frequency hopping table to read back is specified by a tableId.
adi_adrv9001_fh_HopTable_Set()	Sets which hop table to use. Invokes the selection of FH_HOP_TABLE_A or FH_HOP_TABLE_B.
adi_adrv9001_fh_HopTable_Static_Configure()	Loads a frequency hopping table into the ARM memory. A frequency hopping table is made up by an array of hop frame information defined by adi_adrv9001_FhHopFrame_t. **Use this API for both static and dynamic table loading in own system.**
adi_adrv9001_fh_RxOffsetFrequency_Set()	Writes the receiver offset frequency to update NCO with subsequent GPIO Pulse.

Note: The preceding table shows that the API **adi_adrv9001_fh_HopTable_Dynamic_Configure()** is only used to dynamically load the hopping table in the ADRV9001 evaluation system due to the limitation of the FPGA. In the user systems **adi_adrv9001_fh_HopTable_Static_Configure()** is the correct API to load the hopping table either statically at the initialization time or dynamically during hopping. To learn the API use in the different modes of FH operations, use the TES “Driver Debugger” feature or auto generated code.

TRANSMITTER SIGNAL CHAIN

The ADRV9001 device integrates dual direct-conversion (Zero-IF) transmitters. It supports both the time division duplex (TDD) and frequency division duplex (FDD) modes, and is capable of transmitting both narrowband (NB) and wideband (WB) signals. It supports a wide range of applications, such as DMR, P25, and TETRA, as examples of NB standards, and LTE as an example of WB standards.

In general, each transmitter consists of an independent I and Q signal path with separate digital filters, digital-to-analog converters (DAC), analog transmit low-pass filters (LPF), and upconversion mixers. After mixers, an analog attenuator is employed to control the transmitter output signal power.

Data from a baseband processor is input to the transmitter signal path through SSI. The serial data is converted to the parallel format through the deframer and then the data is processed by interpolation filters. There are several signal conditioning functions, such as transmitter gain control, power amplifier protection, digital predistortion (DPD), transmitter quadrature error correction (QEC), and transmitter LO leakage (LOL) handling before the data is passed on to the DACs. The DAC outputs are filtered by LPF, upconverted to RF through the mixer, and attenuated through the analog attenuator to prepare for RF transmission. The ADRV9001 device also supports frequency modulation (FM)/frequency shift keying (FSK) for some NB applications.

Figure 137 shows the high level block diagram of the transmitter signal path.

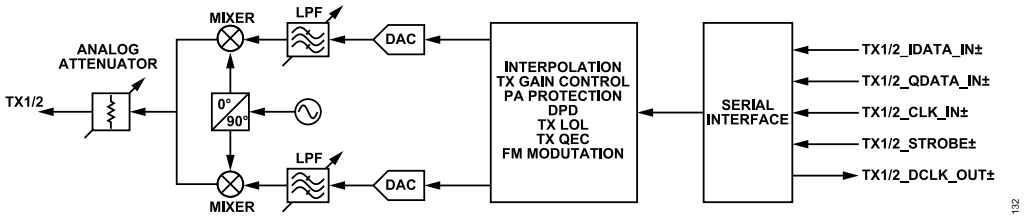


Figure 137. High Level Block Diagram of TX Signal Chain

DATA INTERFACE

The transmitter data interface supports several different interface rates and configurations. It has five differential pairs , i.e., 10 wires. The interface is operated single-ended in the CMOS synchronous serial interface (CSSI) mode and differential in the LVDS synchronous serial interface (LSSI) mode.

As mentioned earlier, the ADRV9001 supports many NB and WB standards. Depending on the selected standard and the specific symbol rate chosen through the API profile, the interface clock rate can vary significantly. Note that CSSI is a slow-speed interface and cannot cover this entire frequency range. For more details, see the Data Interface section.

DATAPATH

Figure 138 shows the high level datapath composed of an analog front end (AFE) and a digital front end (DFE).

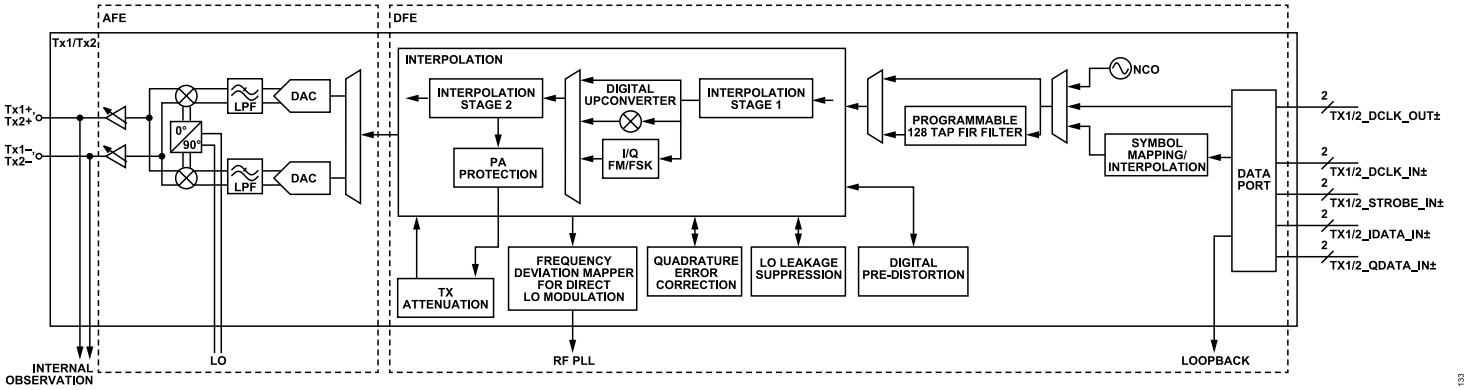


Figure 138. Tx Datapath Block Diagram

In the DFE subsystem, the SSI passes data to the transmitter preprocessing blocks, including a symbol mapping/interpolation and a 128-tap programmable FIR filter (PFIR). The symbol mapping/interpolation is used for interpolation and/or symbol mapping necessary for certain NB standards. Note that if it is configured as an interpolator, ensure proper filtering of the interpolation images in the PFIR. The PFIR is followed

TRANSMITTER SIGNAL CHAIN

by two interpolation stages through a flexible combination of interpolation filters. The interpolation ratios and filters are controlled by the profiles. By design, the interpolation images are rejected by more than 110 dB. Between the two interpolation stages, there is an optional FM/FSK modulator called IQ FM/FSK and a digital upconverter (DUC), which can both be bypassed. Finally, for IQ data, the signal is interpolated to the DAC sample rate.

Figure 138 shows that the DFE subsystem also includes various signal conditioning algorithms, such as LOL suppression and quadrature error correction (QEC). Besides these, it also provides power amplifier protection and transmitter attenuation control blocks.

The output of the DFE is sent through the DAC in the AFE subsystem. The DAC standard clock rate can be programmed to be 184.32, 368.64, or 552.96 MHz, which is set by the profile. (Note that other sample rates can also be supported when an arbitrary sample rate is employed.) Then, the DAC output is filtered by the LPF and input to the upconversion mixer.

Figure 138 shows that the ADRV9001 device also supports another method of FM/FSK modulation called Direct FM/FSK modulation. In this mode, the DUC, IQ FM/FSK, the interpolation stage 2, power amplifier protection, and transmitter attenuation blocks (digital part) are all bypassed. RFPLL is used to generate a constant envelope phase-modulated signal by modulating the sigma-delta modulator (SDM) with the data stream. In Direct FM/FSK, both the DAC and LPF can be powered down.

The following subsections discuss the major transmitter functionalities.

DIGITAL FRONT END (DFE)

Programmable FIR Filter (PFIR)

The PFIR has 128 taps with 24-bit coefficients. There are two FIR filters (PFIR_I and PFIR_Q), as shown in Figure 139. It is configured to operate in parallel, one for I data and one for Q data in digital IQ modulation applications such as long-term evolution (LTE). It can also be configured to use PFIR_I or PFIR_Q only or to operate both filters sequentially for FM/FSK applications. Optionally, use this PFIR for applications by loading a set of PFIR coefficients.

Before this PFIR, there is also an option to perform interpolation with the interpolation factor of 1 and 2. In such a case, design the PFIR for filtering after interpolation. For unity gain, calculate the PFIR coefficients scaling factor as 2 ^ 23 * Interpolation_factor/(sum of the filter coefficients). If fewer than 128 coefficients are specified, append the zeros.

A PFIR's cut off bandwidth can be no less than 1/5th of the RF Channel bandwidth e.g. if the RF channel bandwidth is set to 20 MHz, then the minimum PFIR cut off frequency supported is 4 MHz.

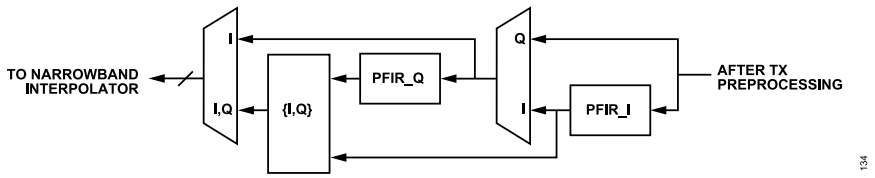


Figure 139. Programmable FIR Filters

Transmit Attenuation Control

Transmit Gain Table

The transmitter attenuation block controls the transmitter output power. A transmitter gain table with 960 entries is loaded into the ADRV9001's memory during initialization. Currently, only the first 840 entries are used. Each entry equals a 0.05 dB gain step. Therefore, there is a total gain range of 42 dB. The total transmitter attenuation is distributed into two portions, an analog attenuation portion and a digital attenuation portion. In the analog attenuation, there is a digitally controlled step attenuator (DSA) with 64 unit steps on a linear scale. The gaps between the analog gain steps are interpolated by a 12-bit digital multiplier to 0.05 dB steps. The maximum analog attenuation is 36 dB, and the maximum digital attenuation is 6 dB. Note in the direct FM/FSK mode, the maximum transmitter attenuation is 12 dB with 0.5 dB step size.

Table 61 shows the first five rows of the transmitter gain table.

TRANSMITTER SIGNAL CHAIN

Table 61. Sample Rows from the Tx Gain Table

Transmitter Attenuation Index	Total Transmitter Attenuation (dB)	Analog Attenuation Control Word[5:0]	Analog Attenuation (dB)	Digital Attenuation (dB)	Digital Attenuation (Linear)	Digital Attenuation Control Word[11:0]
0	0	0	0.00	0.00	1.00	4095
1	0.05	0	0.00	0.05	0.9943	4072
2	0.10	0	0.00	0.10	0.9886	4049
3	0.15	1	0.14	0.01	1.00	4090
4	0.20	1	0.14	0.06	0.9928	4066
…	…	…	…	…	…	…

Table 61 shows that the first column is the transmitter attenuation index. The second column shows the total transmitter attenuation in dB for the corresponding index. Note that the attenuation step size for the adjacent index is 0.05dB. The third column is the control word to calculate the analog attenuation (dB) shown in the fourth column. The equation for this calculation is:

Analog Attenuation (dB) = 20 × $\log_{10}$(1 − *Analog Attenuation Control Word*/64)

The 5th and 6th columns show the required digital attenuation in dB and linear domain respectively to achieve the total attenuation in the second column. The last column stands for the digital attenuation control word to calculate the linear digital attenuation in the sixth column based on the following algorithm:

If Digital Attenuation Control Word = 4095

Digital Attenuation = 4096/2^{12}

else,

Digital Attenuation = *Digital Attenuation Control Word*/2^{12}

Note only the third and the seventh columns are actually stored in memory. Other columns shown in Table 61 are only for explanation.

Read and write the table through the following API command, "**adi_adrv9001_Tx_AttenuationTable_Read()**" and "**adi_adrv9001_Tx_AttenuationTable_Write()**". For more details about API functions, see the API Doxygen document.

Note that the table content is defined by the data structure "adi_adrv9001_TxAttenTableRow_t" as the following:

```
typedef struct adi_adrv9001_TxAttenTableRow{
uint16_t txAttenMult;
uint8_t  txAttenHp;
uint8_t  Reserve;} adi_adrv9001_TxAttenTableRow_t
```

where:

"txAttenMult" denotes the "Digital Attenuation Control Word" (in the range of 0 to 4095).

"txAttenHp" denotes the "Analog Attenuation Control Word" (in the range of 0 to 63).

Table 61 shows these.

Transmitter Attenuation Mode

There are four modes to control the transmitter attenuation block: bypass, SPI, GPIO, and closed-loop gain control (CLGC) modes. The API command "**adi_adrv9001_Tx_AttenuationMode_Set()**" sets the transmitter attenuation mode.

Besides that, another API, "**adi_adrv9001_Tx_Attenuation_Configure()**" sets more configurations for the transmitter attenuation block, such as the transmitter attenuation step size.

The enum "adi_adrv9001_TxAttenuationControlMode_e" defines four transmitter attenuation modes.

```
typedef enum adi_adrv9001_TxAttenuationControlMode
{
ADI_ADRV9001_TX_ATTENUATION_CONTROL_MODE_BYPASS = 0,
```

TRANSMITTER SIGNAL CHAIN

```
ADI_ADRV9001_TX_ATTENUATION_CONTROL_MODE_SPI = 1,
ADI_ADRV9001_TX_ATTENUATION_CONTROL_MODE_PIN = 3,
ADI_ADRV9001_TX_ATTENUATION_CONTROL_MODE_CLGC = 4,
} adi_adrv9001_TxAttenuationControlMode_e
```

The following subsections discuss the first three modes. For the CLGC mode, see the Closed Loop Gain Control (CLGC) section.

Bypass Mode

Select the bypass mode when the transmitter attenuation mode is set as "**ADI_ADRV9001_TX_ATTENUATION_CONTROL_MODE_BYPASS**". In this mode, the transmitter attenuation functionality is not used, which means 0 dB total transmitter attenuation.

SPI Mode

Select the serial peripheral interface (SPI) mode when the transmitter attenuation mode is set as "**ADI_ADRV9001_TX_ATTENUATION_CONTROL_MODE_SPI**". In this mode, set the transmitter attenuation value through the API command "**adi_adrv9001_Tx_Attenuation_Set()**".

The SPI mode consists of two options: the TDD ramp and constant-step size modes. The TDD ramp mode is designed for power ramping in TDD systems. The constant-step size mode allows to control an exact constant gain step size to reach the targeted attenuation level.

TDD Ramp Mode

The time division duplex (TDD) ramp mode is designed for use in TDD systems. Program an "On power" and "Off power" for the next time slot. Control the ramp rate through a step size (tdd_ramp_step_size) and wait duration (tdd_ramp_wait_duration) between the steps. Initiate the ramp up or down through API commands. Note that these user interactions are currently not supported. Figure 140 depicts a typical TDD ramp.

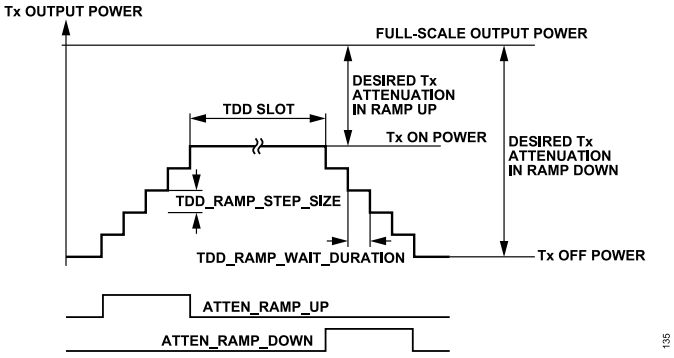


Figure 140. TDD Ramp Profile

Constant-Step Size Mode

In the constant-step size mode, the transmitter attenuation controller ramps to the new transmitter attenuation value immediately after it is set through an API command. Again, control the slope through a step size (const_step_mode_step_size) and wait duration (const_step_mode_wait_duration) between the steps. Figure 141 shows a typical output power transient for this mode.

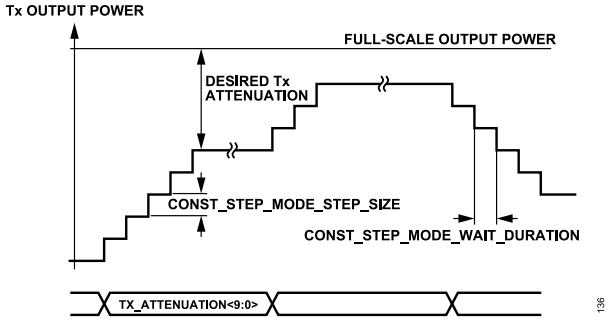


Figure 141. Constant Step Mode

TRANSMITTER SIGNAL CHAIN

GPIO Mode

The general purpose input/output (GPIO) mode is another method to control the transmitter attenuation block. In this mode, two GPIO pins are used to increment or decrement the current attenuation value. An API command "**adi_adrv9001_Tx_Attenuation_PinControl_Configure()**" configures the GPIO pins and sets the step size. Figure 142 shows a typical output power transient.

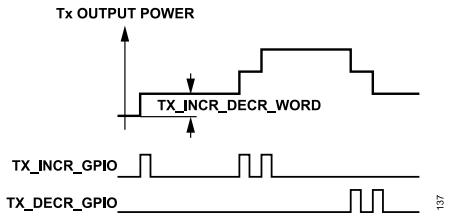


Figure 142. GPIO Increment/Decrement Mode

Power Amplifier Protection

In the transmitter signal chain, two power amplifier protection mechanisms protect the power amplifier from excessive peak or average power excursions. Note that the current release does not fully support these features.

Power Monitor

The power monitor is one of those power amplifier protection methods, and it uses the transmitter attenuation block to adjust the power by continuously monitoring the output power of the Tx datapath.

Through API commands, enable power monitor and set configuration parameters such as average and peak power thresholds. The average power is accumulated over a specified integration time, and an error flag is asserted if it exceeds the threshold. In addition, the instantaneous or peak power is detected, and the error flag is asserted if a specified number of peaks is exceeded. Read the power amplifier error flag through an API command. The power delivered to the power amplifier is automatically reduced if the error flag is asserted. In the scenario depicted in Figure 143, the error flag is asserted after two power peaks are detected. The power amplifier power is automatically ramped down to maximum attenuation. Note that the average power also exceeds its threshold, but not for long enough.

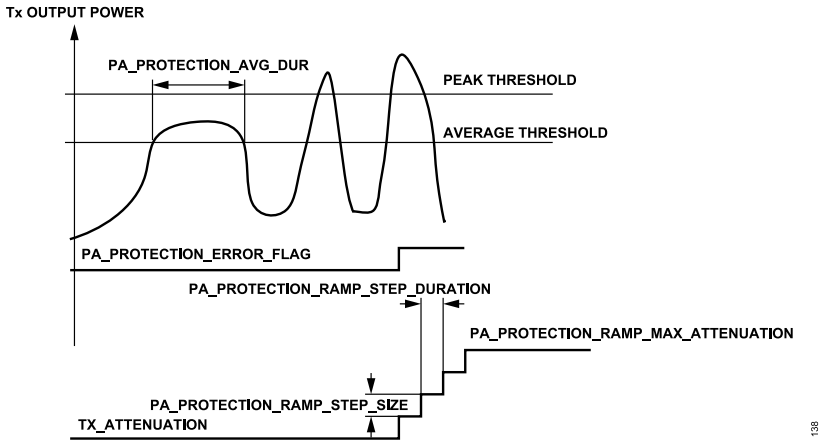


Figure 143. Power Amplifier Monitor

Slew Rate Limiter (SRL)

The slew rate limiter is the other method for power amplifier protection. It essentially limits the rate of change of a waveform by continuously monitoring the difference between the input and output of the block and limiting the amount of output that can change during one cycle. As a result, sudden changes in the input spread to the output over several cycles or symbols. Through API commands, control this slew limit as a fraction of full-scale, which can be varied from very strong slew limiting to no slew limiting at all. For example, if the slew limit is set to 10% full-scale, a full-scale step input to the step limiter results in a ramp, which spreads over the next 10 clock cycles. Figure 144 shows the basic block diagram of the implemented slew rate limiter. The figure shows that the output sample is looped back to subtract from the input sample to decide the slew rate. Based on the slew limit selection, a proper scaling factor is applied to reduce the slew rate to the desired level.

TRANSMITTER SIGNAL CHAIN

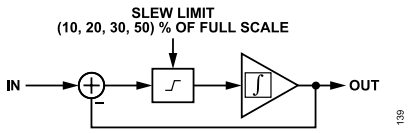


Figure 144. Basic Block Diagram of the Slew Rate Limiter

Transmitter QEC

In the analog circuitry of a direct-conversion transmitter, there are three major non-idealities: gain variation between the I and Q datapath, phase imbalance (non-90 degrees between LO driving I and Q mixers), and differences in the LPF such as group-delay variations. Without properly correcting them, the output spectrum of the transmitter can significantly degrade due to the undesired images.

Transmitter QEC is designed to estimate and correct the non-idealities through initial and tracking calibrations. The initial calibration is performed by generating a tone through the NCO and inserting it into the transmitter datapath. Note that this tone is visible at the transmitter output; therefore, ensure the antenna is isolated from the transmitter (power amplifier is off) during initial transmit calibration. Internally in the device, the output from the transmit upconverter is looped to the ORx through the internal loopback (ILB) path. The ORx output and the transmitted tone estimate the mismatches. The tables are generated to record the initial calibration results, which can be further refined through tracking calibrations on-the-fly. During signal transmission, the mismatch estimations are applied in the transmitter datapath so that the non-idealities are compensated. For more details, see the Transmitter/Receiver/Observation Receiver Signal Chain Calibrations section.

Transmitter LOL

In the transmitter, any coupling of the LO to the RF output or the baseband DC offset generates an undesired tone at the LO frequency. Without proper correction, it can cause a negative impact on the system performance.

Transmitter LOL is designed in the transmit signal chain to handle this problem. Similarly, it estimates the DC offset through initial and tracking calibrations, and then applies the estimation in the baseband to cancel the undesired tone. For more details, see the Transmitter/Receiver/Observation Receiver Signal Chain Calibrations section.

Digital Predistortion (DPD)

DPD is an optional feature available on the ADRV9001 device to achieve higher power amplifier efficiency, while still meeting the error vector magnitude (EVM) and adjacent channel leakage ratio (ACLR) requirements in the transmitter signal chain in compliance with standard requirements. DPD works on the principle of predistorting the transmitter data to cancel distortion caused by power amplifier compression. It uses the tracking calibration to capture the transmitted data samples and the data samples looped back through ORx to estimate the distortion parameters. By applying the estimations in real time, the transmitted signal is predistorted to compensate for the power amplifier nonlinearity.

For more details, see the Digital Predistortion (DPD) section.

Transmitter NCO Internal Signal Source

The ADRV9001 has an internal quadrature numerically-controlled oscillator (NCO). It serves two major purposes. First, it generates the calibration tones for the initial calibrations, such as the transmitter QEC. Second, it generates test tones through an API command to disable the data port interface and simplify the design for specific use cases or testing. In all cases, as shown in Figure 138, the transmitter preprocessor takes the input data from the NCO instead of the data port interface.

The internal signal source cannot be used with frequency hopping or TDD because these modes of operation require strict timing control by the FPGA or BBIC. For example, based on the TDD timing parameters, the FPGA sets Tx_enable to high and aligns the time of sending the Tx data at the Tx_enable rising edge, which cannot be achieved by the NCO.

Transmitter Frequency Offset Correction

The ADRV9001 provides user capability to correct small deviations in the transmit LO frequency through an API command. Through this API, the user can provide the desired frequency offset in Hz, and change the frequency immediately or update it at the start of the next available frame.

TRANSMITTER SIGNAL CHAIN

FM/FSK Modulation

The ADRV9001 provides a frequency modulation (FM)/frequency shift keying (FSK) for standards that use constant-envelope frequency modulation schemes, such as digital mobile radio (DMR), Analog FM, P25 Phase 1, and Phase 2. It also has the option to perform symbol mapping and interpolation operations on the transmit data received from a baseband processor for FM/FSK modulation. This capability provides more flexibility when preparing the data for transmission. The ADRV9001 also has an option to allow user send either pre-mapped and pre-interpolated transmit data by enabling this functionality, or send post-mapped and post-interpolated data by bypassing this functionality. For example, for the DMR standard, which uses 4.8 ksps symbol data, the baseband processor sends the symbol data directly to the ADRV9001 and lets it map the symbol data and then interpolate the data to generate frequency deviation data. Note that this functionality is currently not enabled in the datapath.

Currently, to use the FM/FSK modulation capability of the ADRV9001, perform symbol mapping, interpolating, and pulse shaping filtering in the baseband processor to generate frequency deviation data before sending it to the ADRV9001. Two different options for FM/FSK modulation are deployed in the ADRV9001: Direct FM/FSK and IQ FM/FSK, as shown in Figure 138. The following subsections briefly discuss these.

Direct FM/FSK

Implement the frequency modulation by modulating the transmitter RF PLL directly in the Direct FM/FSK option. Figure 145 shows the transmitter datapath with the Direct FM/FSK.

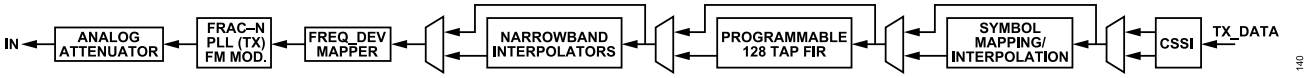


Figure 145. Direct FM/FSK Datapath Block Diagram

Figure 145 shows that the baseband processor TX_DATA can optionally go through the symbol mapping/interpolation and programmable FIR, and after interpolation and frequency deviation mapping, the Frac-N PLL implements the FM/FSK modulation at the desired RF output frequency. Finally, the PLL output is attenuated before feeding to the transmitter RF interface. The programmable 128-tap FIR works as the pulse shaping filter. In this scenario, optionally load the filter coefficients according to the standard requirement through API commands. In Direct FM/FSK modulation, the DAC and LPF can both be powered down, and some digital blocks such as the common interpolators, power amplifier protection, and transmitter attenuation control can all be bypassed. Therefore, it can save power.

IQ FM/FSK

Implement IQ FM/FSK modulation by modulating the digital NCO, as shown in Figure 138. The modulated IQ data goes through the interpolator, DAC, LPF, and then is upconverted to RF frequency by the mixer. The processing stages before the digital FM/FSK modulator are like the Direct FM/FSK option, which also includes optional symbol mapping/interpolation and pulse shaping functions. The selection between the direct FM/FSK and IQ FM/FSK is determined by profile.

ANALOG FRONT END (AFE)

Digital-to-Analog Converter (DAC)

The ADRV9001 integrates a 16-bit DAC, which can be operated at a standard rate of 184.32 MHz, 368.64MHz, or 552.96 MHz (note: with an arbitrary sample rate, DAC can operate at other different rates as well.) The selected profile sets the sampling rate. The DAC is auto-tuned to remove mismatches in the DAC units, which improves the linearity of the DAC. Boost the nominal full-scale current of the DAC by 3 dB through the API command "**adi_ADRV9001_Tx_OutputPowerBoost_Set()**". The increased signal swing throughout the entire analog signal chain results in better AM noise performance. By default, the 3 dB boost is disabled.

Low Pass Filter (LPF)

The analog LPF is used to attenuate the sampling images of the DAC. The frequency response has a second-order Butterworth shape. The corner frequency is auto-tuned to compensate for process and temperature variations. The API profile sets the operating corner frequency. The ADRV9001 allows to configure LPF at three different power consumption levels to achieve a system power saving target.

TRANSMITTER SIGNAL CHAIN

Upconversion Mixer

The upconversion mixer translates the baseband signal to RF. It is an IQ modulator, which receives a quadrature baseband and LO signal. Due to the image rejection property of IQ modulators, it produces an output only on one side of the LO, i.e., the image is rejected. The LO leakage and quadrature errors of the mixer are calibrated at initialization, and continually tracked by the transmitter LOL and QEC, as discussed.

RF Attenuator

Following the mixer is a digitally controlled step attenuator with 64 linear gain steps. This results in a total gain range of 42 dB. Note that the analog gain steps are not linear-in-dB. However, as pointed out in the previous section, the analog gain steps are interpolated by a digital multiplier to achieve 0.05 dB gain steps. Note in the Direct FM/FSK mode, the total gain is 12 dB with 0.5 dB step size.

TRANSMIT DATA CHAIN API PROGRAMMING

There are a set of transmitter data chain APIs for interaction with the ADRV9001 device transmit datapath. Some of them are discussed in the previous sections. The following table summarizes the list of available API functions with a brief description for each. For more up-to-date information and detailed descriptions, refer to the API Doxygen document.

Table 62. A List of Tx Data Chain APIs

Receiver Gain API Function Name	Description
adi_adrv9001_Tx_Attenuation_Configure	Configures the transmitter attenuation for the specified channel.
adi_adrv9001_Tx_Attenuation_Inspect	Inspects the transmitter attenuation for the specified channel.
adi_adrv9001_Tx_AttenuationMode_Set	Sets the attenuation control mode.
adi_adrv9001_Tx_AttenuationMode_Get	Gets the attenuation control mode.
adi_adrv9001_Tx_Attenuation_Set	Sets the transmitter attenuation for the specified channel.
adi_adrv9001_Tx_Attenuation_Get	Gets the transmitter attenuation for the specified channel.
adi_adrv9001_Tx_OutputPowerBoost_Set	Enables or disables the transmitter output power boost.
adi_adrv9001_Tx_OutputPowerBoost_Get	Gets the current transmitter output power boost enable status.
adi_adrv9001_Tx_AttenuationTable_Write	Writes the attenuation table for the specified transmitter channels.
adi_adrv9001_Tx_AttenuationTable_Read	Reads the attenuation table for the specified transmitter channels.
adi_adrv9001_Tx_InternalToneGeneration_Configure	Sets the transmitter NCO internal tone frequency for the specified transmitter channel.
adi_adrv9001_Tx_InternalToneGeneration_Inspect	Gets the transmitter NCO internal tone frequency for the specified transmitter channel.
adi_adrv9001_Tx_PaProtection_Configure	Configures power amplifier protection for the specified transmitter channel.
adi_adrv9001_Tx_PaProtection_Inspect	Inspects power amplifier protection for the specified transmitter channel.
adi_adrv9001_Tx_SlewRateLimiter_Configure	Configures the slew rate limiter for the specified transmitter channel.
adi_adrv9001_Tx_SlewRateLimiter_Inspect	Inspects the slew rate limiter for the specified transmitter channel.
adi_adrv9001_Tx_PaRamp_Configure	Configures the power amplifier ramp for the specified transmitter channel.
adi_adrv9001_Tx_PaRamp_Inspect	Inspects the power amplifier ramp for the specified transmitter channel.
adi_adrv9001_Tx_Attenuation_PinControl_Configure	Configures the transmitter attenuation through GPIO pins for the specified transmitter channel..
adi_adrv9001_Tx_Attenuation_PinControl_Inspect	Inspects the transmitter attenuation through GPIO pins for the specified transmitter channel.
adi_adrv9001_Tx_FrequencyCorrection_Set	Sets the NCO frequency to correct for small deviations in transmitter LO frequency.
adi_adrv9001_Tx_DataPath_Loopback_Set	Sets the receiver datapath to transmitter datapath loopback.

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

The ADRV9001 offers dual receive channels. With a minimum number of external components, each receive channel builds a complete RF-to-bits signal chain, which serves as RF front end for a wide range of applications. It supports both the time division duplex (TDD) and frequency division duplex (FDD) modes, and receives both narrowband (NB) and wideband (WB) signals up to 40 MHz. NB applications include DMR, P25, and TETRA, while WB applications are geared towards LTE transmissions. For example, the ADRV9001 supports standard sample rates of 24 kHz (typically for FM waveforms), 144 kHz and 288 kHz (typically for TETRA signals), and 1.92 MHz, 3.84 MHz, 7.68 MHz, 15.36 MHz, 30.72 MHz, and 61.44 MHz (typically for LTE signals). Besides these standard rates, the ADRV9001 also supports an almost continuous range of sample rates between 24 kHz and 61.44 MHz. Some sample rates are not supported due to internal clocking constraints.

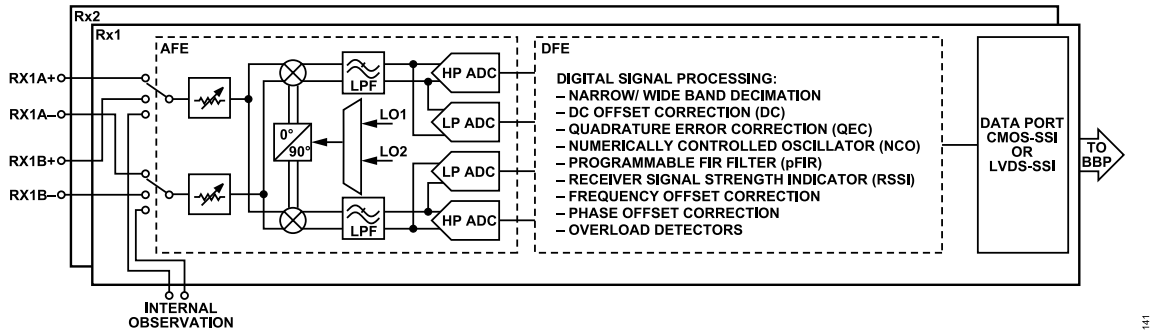


Figure 146. Top Level Structure of ADRV9001 Dual Receiver

Figure 146 describes the top-level structure of the ADRV9001 receivers. Figure 146 shows that each receiver path Rx1 or Rx2 contains two major subsystems, the analog front end (AFE) and digital front end (DFE). The AFE subsystem contains four major components: programmable front end attenuator, matched I and Q mixer, low-pass filter (LPF), and analog-to-digital converter (ADC). The attenuators control the signal gain to avoid overloading the datapath when a strong signal interferes. It is followed by the mixers to down convert the received signals for digitization. The output current of the mixers is further converted to voltage and filtered by the LPFs before passing to the ADCs. The ADRV9001 provides two pairs of ADCs, a pair of high performance (HP) ADCs to achieve high linearity performance and a pair of low power (LP) ADCs with slightly less linearity performance but significant lower power consumption. This design allows a flexible trade-off between power consumption and linearity performance.

The DFE subsystem contains a series of digital signal processing components such as sample rate decimation (DEC), DC offset correction (DC), quadrature error correction (QEC), digital down conversion (DDC) with numerically controlled oscillator (NCO), a programmable 128-tap FIR filter (PFIR), receiver signal strength indicator (RSSI), frequency offset correction (FOC), phase offset correction (POC), and overload detectors. The DEC decimates the ADC sample rate to the desired output sample rate. The DC, QEC, PFIR, FOC, and POC condition the digital signals at different stages of the datapath for optimal performance.

The overload detectors are used for gain control in the datapath. The RSSI provides signal power measurement to control the bit-width of the output signal. In addition, it detects the presence of a signal in a desired frequency band. At the end of the signal chain, through CMOS-SSI or LVDS-SSI data port, the output signal is delivered to the based band processor for further processing.

The ADRV9001 supports an RF LO range from 30 MHz to 6 GHz. The RF LOs are generated through two internal phase lock loops (PLL) or applied externally to the part. The digital subsystem contains an optional digital mixer driven by a programmable NCO. The receiver LO can offset from the frequency of the desired channel and then make use of the digital mixer to down convert the signal to baseband before being processed by the baseband processor. There are several advantages to offset the receiver LO from the frequency of the desired channel: impairments that exist around the receiver LO, such as LO-leakage, can be avoided. The effect of flicker noise from the baseband circuits can be mitigated because the received signal is offset from DC in the analog signal path. Also, image rejection can be improved if the receiver LO is offset sufficiently far from the desired channel, such that the image frequency lies in the attenuation region of the external RF filter. The IF operation works with both NB and WB applications. Typically, when the receiver is operating in the NB mode, the sensitivity requirements for these applications demand very low noise performance. Therefore, the intermediate frequency (IF) approach is preferred. When the receiver is operating on a WB signal, it uses direct down conversion or zero IF (ZIF) (although IF approach is also available for WB signal). In this mode, the DDC is bypassed.

Figure 147 describes the simplified transmit and receive signal path between the antenna and the ADRV9001 device. The components between the antenna and the ADRV9001 device are external components. In the transmit path, typically, the output signal from the device goes through a variable gain amplifier (VGA), a low-pass filter (LPF), and a power amplifier before transmitting through antenna. In the receive path, typically, the RF signal receiving from the antenna goes through a low noise amplifier (LNA) and a band-pass filter (BPF) before sending to the

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

device. The duplexer is to support both the FDD and TDD modes, which stands for a frequency duplexer in the FDD mode and an RF switch in the TDD mode.

Figure 147 shows that each receiver, besides acting as a primary data channel for receiving RF signals, also serves as an observation channel, which receives loopback signals from the transmitter. There are three loopback paths: internal loopback (ILB), external loopback type 1 (ELB1), and external loopback type 2 (ELB2), as shown in Figure 147. When users are in full control of the loopback channel for running their own algorithms, the receiver is renamed as the observation receiver (ORx). In such a case, use either ELB1 or ELB2 depending on the algorithm requirements. For example, if running an external DPD, use the ELB2 path.

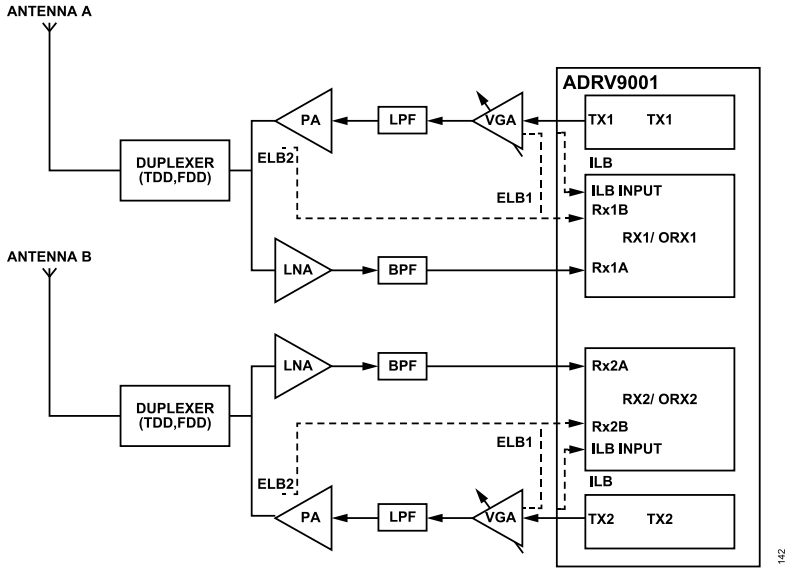


Figure 147. ADRV9001 Rx/ORx/Loopback Diagram

The ADRV9001 also uses the three loopback paths internally for two major purposes: transmitter initial calibrations and transmitter tracking calibrations, including the integrated digital predistortion (DPD) operation. Transmitter initial calibrations configure the device properly based on the system configurations during the initialization time. It can be done through either ILB, ELB1, or ELB2. The major advantage of using ELB1 compared to ILB is to observe common-mode voltage. In addition, during the transmitter initial calibrations procedure, test tones are generated and present at the transmitter output. Therefore, ensure an appropriate level of isolation from the ADRV9001 transmitter output to the antenna to ensure that test tones are not transmitted by the system. Achieve this isolation by disabling the power amplifier during transmitter initial calibration in ELB1. For ELB2, the calibration signal might be transmitted out through the antenna. Although the power level of calibration signal is set as low as possible, make sure this does not cause any problem when using this option. For more details, see the Transmitter/Receiver/Observation Receiver Signal Chain Calibrations section.

The transmitter tracking calibrations tweak the system on the fly during its normal operations for optimal performance. Similarly, it uses ILB, ELB1, or ELB2. ILB and ELB1 are used when DPD is not required, while ELB2 is used when DPD is required, in which the transmit signal is looped back to the receiver after power amplifier. Note that ELB1 and ELB2 share the same receiver input. So, they cannot be used simultaneously.

No matter whether it is used by the user or internally by the device, the observation channel shares the same datapath as the regular receiver. Therefore, the observation can only be performed when no regular reception is required at the same time. This is the case for transmitter initial calibrations and be aware that receiver might not be idle, as it works as the observation channel during initialization internally by the device. Different from transmitter initial calibrations, transmitter tracking calibrations are performed on the fly. So, they have to share time with the regular receiver operations. For example, in a TDD system, when the transmitter is transmitting, the receiver should not be receiving. Therefore, it can be used for transmitter tracking calibrations. For a system where the receiver is fully occupied all the time to receive RF signals, such as a 2Tx2Rx FDD system, it is not possible to perform transmitter tracking calibrations, including DPD operations. However, in a 1Tx1Rx FDD system, because one receiver is always idle, it can be used as an observation channel. For example, if Transmitter 1 is transmitting, then Receiver 1 can be used for observation by receiving loop back signals from Transmitter 1 and Receiver 2 can be used as the main receive path. The observation receiver must be on the same side of the transmitter it observes. So, observation channel 1 is always for Transmitter 1 and observation channel 2 is always for Transmitter 2. When in control of the observation channel, configure the ORx based on requirements

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

such as the ORx gain. When the ADRV9001 device is in control of the observation channel, it must configure the observation channel properly without any user intervention.

Figure 147 shows that each receiver has three inputs. One is the ILB input dedicated for receiving ILB signal. The others are Rx1A/Rx2A and Rx1B/Rx2B inputs. One is configured to receive RF signals and the other to receive ELB signals. The ADRV9001 has an option to select the main receive port (either port A or port B) during initialization. For example, if port B is selected as the main receive port, then port A is used to receive ELB signals. Furthermore, the ADRV9001 provides dynamic port switching capability to switch the receive ports (either port A or port B) on the fly. This is mainly designed to accommodate applications that have limitations on operating frequency range of the RF Balun. In this case, multiple Baluns can be used to cover the entire frequency range (30 MHz to 6 GHz) supported by the ADRV9001. To support seamless switch between multiple Baluns that cover different frequency ranges, the dynamic port switching feature allows the use of both port A and B as main data receiver channels while switching between the two ports at run-time based on the carrier frequency ranges defined for each port. Note that once this feature is enabled, it applies to both receive channels, which means there is no support for switching port A/B on Rx1 but using fixed port on Rx2. In addition, ELB cannot be enabled in this case.

Due to the support of a wide range of applications, the user interaction with the receiver signal chain is mainly done through configuration profiles. Based on the channel profile, which includes key parameters such as bandwidth, sample rate, and AGC settings, initial calibration is performed in the device to set up the receive chain properly. When DPD is performed internally in the device, the switch between receiver and observation channel is fully determined without user interaction. When the DPD is performed externally by the baseband processor, the baseband processor owns the entire ORx channel. Make sure that there is no conflict between the DPD operations and the transmitter tracking calibrations in the device.

In the ADRV9001, a specialized “Monitor mode” allows the device to autonomously poll a region of the spectrum for the presence of a signal, while in a low power state. In this mode, the chip continuously cycles through sleep-detect-sleep states controlled by an internal state machine. Save power by ensuring the sleep duty cycle is greater than the “detect” duty cycle. In the “sleep” state, the chip is in a minimal power consumption configuration, where few functions are enabled. After a predetermined period, the chip enters the “detect” state. In this state, the chip enables a receiver and performs a signal detect over a bandwidth and at a receiver LO frequency determined by the user. If a signal is detected, the “Monitor Mode” state machine exits its cycle and normal signal reception resumes. If no signal is detected, the chip resumes its sleep-detect-sleep cycle. The sleep-detect duty cycle and signal detection method are user-programmable, and are set before enabling “Monitor Mode.” For more details, see the Power Saving and Monitor Mode section.

The ADRV9001 provides various levels of power control. Power scaling is performed on individual analog signal path blocks to trade off power and performance. In addition, enabling and disabling various blocks in the TDD receive and transmit frames to reduce power are customized, at the expense of receive/transmit, or transmit/receive turnaround time. For more details, see the Power Saving and Monitor Mode section.

The following sections provide topical information on:

- Receive Data Chain: Describes how the analog and digital components are used at the different stages of the receiver chain to convert RF signals to bits at the desired sample rate for further baseband processing.
- Analog Front-End Components : Discusses each major AFE component and its functionality.
- Digital Front End Components: Discusses each major DFE component and its functionality.
- Receive Data Chain API Programming: Outlines the API programming capabilities of the receiver data chain for user interactions.

RECEIVE DATA CHAIN

The ADRV9001 supports both NB and WB applications in a common design. Figure 148 describes the block diagram of the entire receiver data chain, which is composed of AFE and DFE. The AFE includes a front-end attenuator, which controls the received RF signal level, mixer for RF to baseband (or IF) down conversion, low-pass filter, and a pair of HP and LP ADCs. The LPF has a programmable -3dB bandwidth from about 6.7 MHz to 40 MHz, depending on the profile. Its configuration and filter characteristics are automatically tuned internally to achieve optimal performance for different applications. In principle, the AFE design is based on the WB architecture with a very high dynamic range to absorb both the desired signal and interference without distortion. Therefore, in such a design, very little channelization, or blocker filtering is needed through LPF because the HP and LP ADC can simultaneously absorb weak signals and large blockers. The blocker suppression and channelization are then achieved efficiently in the digital signal path. After ADC, the digital output signal is further processed through multiple stages in DFE.

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

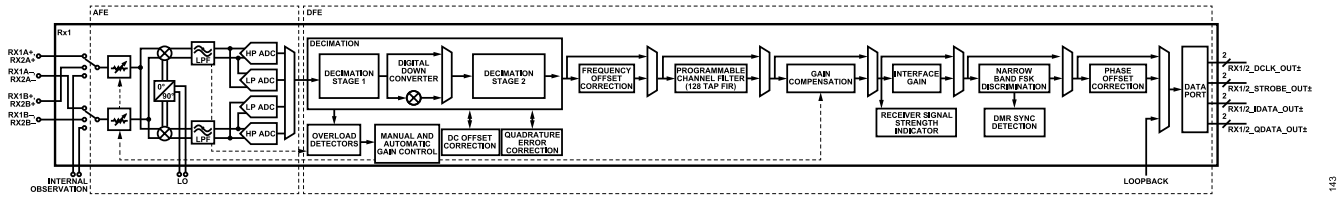


Figure 148. Rx Signal Chain Block Diagram

The ADRV9001 supports three standard ADC clocks: ADC-H clock 2211.84 MHz, ADC-M clock 1474.56 MHz, and ADC-L clock 1105.92 MHz for both HP ADC and LP ADC (note that the ADC clock varies with arbitrary sample rate.). In the DFE subsystem, the ADC output signals at three different sample rates go through two decimation stages, as shown in Figure 148, to convert to the desired sample rate by using a flexible combination of decimators. Between the two decimation stages, there is an optional DDC, which is employed in the applications, and which adopts IF reception scheme.

At different decimation stages, several signal conditioning algorithms are performed, which are overload detection for gain control, DC offset correction (DC), and QEC, as shown in Figure 148. The overload detection result is used by the AGC or manual gain control (MGC) algorithms to properly control both analog and digital gain through a receiver gain table. The analog gain is applied at the front-end attenuator to avoid overload/underload situations. The digital gain is applied at the gain compensation block in the receiver datapath and it has two major functionalities: one is to correct the small step size inaccuracy of the front end analog gain and the other is to compensate for the front end gain change completely so that it is transparent to users. The different receiver gain tables are loaded for either correction or compensation based on the user's configuration during the device initialization. In the ADRV9001, there is a sophisticated gain control mechanism (AGC/MGC). For more details, see the Receiver Gain Control section. The DC and QEC are used to correct the DC offset and quadrature error, so that the signal distortion is minimized to achieve an optimal performance before sending data to the baseband processor. To achieve the best performance for different applications, QEC algorithm is designed differently for WB and NB applications.

After decimation stage two, the ADRV9001 provides an option to correct the small carrier frequency offset through API commands, followed by a 128-tap programmable PFIR as a channel selection filter.

After PFIR, besides applying the digital gain as discussed, apply an interface gain optionally using the signal strength measurement from the RSSI. Apply the interface gain through a "Slicer" by properly shifting the signal. When the signal is large, it is used to avoid saturating the data port due to a limited bit-width, and when the signal is small, it is used to avoid losing sensitivity. An API command **adi_adrv9001_Rx_Rssi_Read()** reads the signal strength measurement. Apply the interface gain automatically in the device or manually through API commands. This is beneficial when there is saturation in the baseband processor. For more details, see the Receiver Gain Control section.

The RSSI is also used as the signal detector in the receiver Monitor Mode. In NB applications, at the end of the datapath, the device provides an option to discriminate the FSK frequency shift and, in addition, detect the DMR sync patterns, which is critical for the receiver Monitor Mode. Note that there is phase offset correction capability at the end of the receiver datapath to ensure the signal fidelity. Finally, the output signal is sent through the CMOS-SSI/LVDS-SSI data port to the baseband processor for further processing.

ANALOG FRONT-END COMPONENTS

Analog Front Attenuator

The analog front attenuator is a PI resistive network that provides a constant 100 Ω differential input impedance with the passive mixer. The gain control functionality controls it in the receive datapath to adjust the signal gain to avoid overloading the datapath through overload detectors. When there is a strong interferer, the gain is decreased and when the interferer disappears, the gain is increased, so that the desired signal level is adjusted back to the proper level.

The attenuator has 256 gain settings providing a receiver attenuation range from 0 to 20 × log(1/256) = -48 dB. Typically, only a subset of this range is used. In the ADRV9001, the current range of the attenuation is from 0 to -34 dB. Calculate the gain of the attenuator by the following equation:

$$ATTEN_{dB} = 20 \times \log10 \left[\frac{256 - fe\_gain\_cw[7:0]}{256}\right]$$

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

The fe_gain_cw[7:0] is an 8-bit control word defined in the receiver gain table. Based on the information from the signal detectors, the gain control algorithm finds the index of this gain table so that the corresponding gain control word at this index is used to calculate the gain at the front-end attenuator. For more details, see the Receiver Gain Control section.

Mixer

The passive down converting mixer follows the RF attenuator. It down converts the RF signal to IF or baseband. The passive mixer uses a non-overlapping ¼ duty cycle local oscillator generated by the four phases 50% duty cycle LO. The duty cycle distortion (DCD) circuit controls the non-overlapping time. the DCD is implemented by delaying the rising edge of the 50% duty cycle LO.

Low Pass Filter (LPF)

In the receiver data chain, the LPF sits between the mixer and the ADC as a receiver baseband filter and supports a baseband 3dB bandwidth of about 6.7 MHz to 40 MHz. It also converts the baseband signal current to voltage. The capacitor arrays are implemented to program the various cut-off frequencies based on the system requirements. In addition, along with other AFE components, it provides a static gain of about 20 dB, which is independent of the gain control functionality through the receiver data chain.

The LPF can be configured in the transimpedance amplifier (TIA) mode with the single pole or in bi-quad (BIQ) mode with two complex poles in the transfer function, which are denoted as first order LPF and second order LPF, respectively. The second order mode LPF has additional out of band attenuation compared to the first order mode. Figure 149 and Figure 150 show the first and second order LPF simulated frequency response at different f1db configurations. The second order mode LPF consumes about twice the power than the first order mode, and also the in-band noise is around 2.5 dB worse than the first order mode. This is the tradeoff for the out-of-band rejection capability and power consumption/in-band noise. The users rely on their application to choose either the first or second LPF order mode.

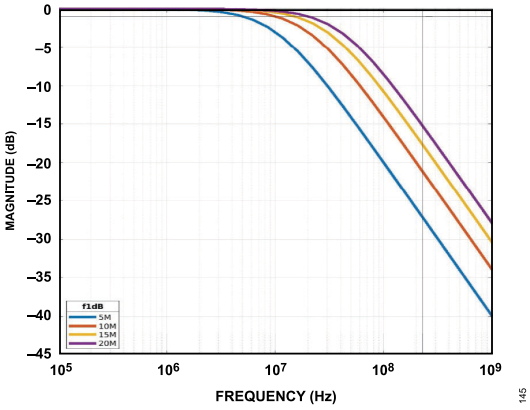


Figure 149. First Order RX LPF Frequency Response at Different f1db Configuration

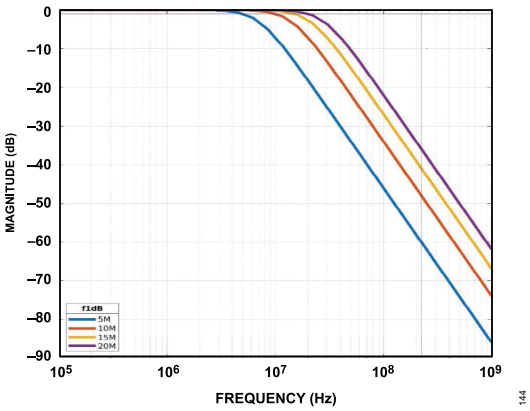


Figure 150. Second Order RX LPF Frequency Response at Different f1db Configuration

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

The LPF is calibrated during device initialization to ensure a consistent frequency corner across all devices. The 3dB bandwidth is set within the device data structure and is profile dependent. The user could optionally tune the 1 dB/3 dB corner frequency of the LPF based on the application. The ADRV9001 also allows to configure LPF at three different power consumption levels to save system power.

The LPF's primary purpose in the Rx channel is as an anti-aliasing filter prior to the ADC. Inside this LPF block there is a magnitude compensating (mag-comp) filter to sharpen the corner-frequency response of the LPF. This mag-comp filter is calculated depending on the cut-off frequency chosen and is optimized for both the first order and second order filters. As a result, in wideband use-cases (e.g. 40 MHz bandwidth) the first order filter can cause some degradation at the edges of the band when the cut-off frequency is below 7 MHz. It is recommended in these particular use cases to use a minimum of 7 MHz if selecting the first order filter option, but ideally the LPF would not be set this low for such a high BW application.

Analog-to-Digital Converter (ADC)

As mentioned earlier, the ADRV9001 provides a pair of HP ADCs and LP ADCs to achieve a flexible tradeoff between power consumption and linearity performance. The HP ADC is based on the continuous time delta sigma (CTDS) architecture and is 5 bits wide. The LP ADC is based on the voltage-controlled oscillator (VCO) architecture and is 16 bits wide. Each type of ADC is capable of accepting the same input voltage, but the output bus width is different due to the different modulator orders and presence of linearity correction in the LP ADC.

The HP and LP ADCs provide a similar level of noise and dynamic range (full scale to thermal noise) performance. Therefore, the noise figure (NF) performance is similar at the input. Even with slight NF difference at the device input, the difference at antenna input is smaller as a result of the LNA gain in the front end. The major difference between the HP and LP ADC is the linearity performance and power consumption. The intermodulation distortion (IMD) performance of HP ADC is slightly better than LP ADC, at the expense of higher power consumption. Refer to the data sheet for detailed information.

Given the high dynamic range of both the HP and LP ADC, very little channelization or blocker filtering occur in the analog signal chain as the HP ADC can simultaneously absorb weak signals and large blockers. The blocker suppression and channelization are then achieved efficiently in the digital signal path.

Therefore, the HP ADC provides the maximum interferer tolerance and performance. The LP ADC provides the best power consumption performance under a slightly relaxed interferer condition. Based on the application, select between HP and LP ADC for linearity and power consumption performance tradeoff. In addition, dynamically switch HP ADC and LP ADC on the fly through API commands **adi_adrv9001_Rx_AdcSwitchEnable_Set( )** and **adi_adrv9001_Rx_AdcSwitch_Configure( )**. The first API function enables the ADC switching feature, and it is called at the STANDBY state before initial calibrations. When the dynamic ADC switch is enabled, both the HP ADC and LP ADC initial calibrations are performed. The second API configures the ADC switching functionality for a specified receiver channel to operate in different modes. It is called at the CALIBRATED state after performing initial calibrations.

When the receiver Monitor Mode is enabled, the device might switch between the HP ADC and LP ADC to reduce power consumption. Additional algorithms are employed in the ADRV9001 to compensate for the gain and delay differences while operating with different type of ADCs, so that any internal switch is transparent to users.

DIGITAL FRONT END COMPONENTS

Decimation (DEC)

In receiver data chain, a series of decimators (organized into two different decimation stages) convert the ADC sample rate to a desired sample rate in both the NB and WB modes. Figure 151 shows how to achieve the standard sampling rates for different standards through a flexible combination of decimators in the data chain. For simplicity, any other non-DEC blocks are skipped in the diagram.

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

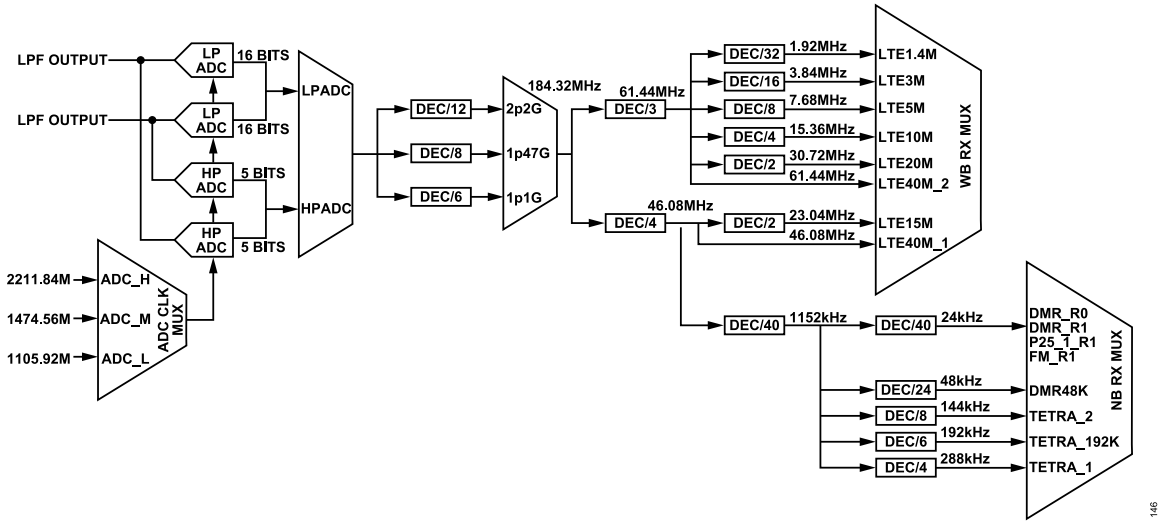


Figure 151. Decimation Schemes in Receiver Data Chain to Support Various Standards

Figure 151 shows that in the NB and WB mode, three different ADC output sample rates are first decimated to a common rate of 184.32 MHz in standard use cases. Then, it is further converted to two different rates. One is 61.44 MHz for WB mode only and the other is 46.08 MHz for both the NB and WB modes. All LTE standard modes are considered WB and the desired sample rate is further generated from both 61.44 MHz and 46.08 MHz through a decimation rate of 2 to 32. The DMR, FM, P25, and Tetra are NB modes, and the desired sample rate is further generated from 46.08 MHz with a decimation rate of 160 to 1920.

For each decimator shown in Figure 151, there is a combination of lower rate decimators. For example, DEC/40 is implemented as a cascade of DEC/10 and 2 DEC/2 decimators. In addition, the different decimation rates are achieved by strategically enabling and disabling some lower rate decimators. For example, in WB mode, with an initial sample rate of 61.44 MHz, if all lower rate decimators are used, it can achieve a decimation rate of 32. If two of the DEC/2 are disabled, a decimation rate of 8 can be achieved. All the decimation filters are carefully designed to satisfy the system performance requirements.

With the arbitrary sample rate, the user could get an almost continuous range of sample rates from 24 kHz to 61.44 MHz, except for some “dead zones” due to internal clocking constraints. Achieve this by adjusting the internal CLK PLL frequency as well as a flexible arrangement of decimators.

DC Offset

The ADRV9001 receiver supports both IF and ZIF down conversions. The source of the DC offset is mainly from the receiver LO leakage caused by the finite isolation between the LO and RF ports of a mixer, which is typical for silicon-based ICs. It generates a high DC component at the center of the desired signal band, especially for ZIF operation. Through the datapath, the induced DC offset is amplified and reduces the ADC dynamic range significantly. In addition to receiver LO leakage, the device mismatch in LPF and ADC also contributes to the DC offset problem. Without properly correcting the DC offset, it can cause a negative impact on the system performance.

In the ADRV9001, a two-step approach estimates and corrects the DC offset. The first step comprises a DC estimation step in the digital domain and a correction procedure in the analog domain, which is called RFDC. The second step is an all-digital DC offset estimation and correction technique that estimates and corrects for any residual DC offset after the first step, which is called BBDC. BBDC is basically a notch filter, and controls the width of the notch. The default value is 1/(2048*16), but can be changed for a wider or narrower notch.

The ADRV9001 also provides an option to enable an LO interference detection (LOID) algorithm. It optimizes the RFDC performance in the presence of a blocker around DC. If a blocker is detected, then the LOID signals the RFDC to freeze the data correction updates, as performing updates in the presence of a blocker degrades the performance of RFDC. If a blocker is not detected, then the LOID feature signals the RFDC to resume its operation. The LO interference detection threshold is configurable. By default, it is set as -61 dBFS. When the LO interference is above the threshold, RFDC is disabled, and when the interference is below the threshold, RFDC is re-enabled.

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

Quadrature Error Correction (QEC)

In an ideal analog mixer, the in-phase (I) and quadrature-phase (Q) sinusoidal signals are orthogonal. In addition, the I and Q paths of LPF and ADC should have identical frequency responses. However, in reality, IQ imbalance always exists in the mixer, LPF, and ADC, resulting in quadrature errors. Without properly handling, it seriously degrades reception performance. For IF reception, the respective image mixes partially onto the desired signal during the IF down conversion. In direct conversion reception, IQ-imbalance leads to a distortion of the IQ-signals themselves within the respective desired baseband channel.

In general, quadrature error can be classified as frequency independent error (FIE) and frequency dependent error (FDE). FIE is mainly caused by the mixer I/Q sinusoid mismatch in both gain and phase, while FDE is mainly caused by the inconsistent filter responses.

Because the ADRV9001 supports both the NB and WB modes, NBQEC and WBQEC algorithms are developed accordingly to handle quadrature errors in these two modes effectively. NBQEC employs a time-domain adaptive algorithm to estimate both the gain and phase mismatch. Then, estimations are applied to correct the distorted input signal in real-time before passing to DDC. WBQEC designs a correction filter to cancel the effect of the mismatch filter by modeling the quadrature error generation as a mismatch filtering process. The correction filter parameters are obtained through the initial calibration by injecting RF tones into the mixer at selected frequencies and then on-the-fly adjustment by processing the receiver data in real-time.

Digital Down Converter (DDC)

DDC is only used when IF reception is employed. By using a programmable NCO configurable from 45 kHz to 21 MHz, it further converts the IF signal to the baseband.

Frequency Offset Correction

In a communication system, the transmitter transmits a desired signal at RF over the air. As the clock reference at the transmitter or the receiver is independent of each other, this can result in the RF carrier frequency offset between the transmitter and the receiver. This frequency difference is called the carrier frequency offset (CFO). In the receiver data chain, there is a frequency offset correction block to further correct small carrier frequency offsets in both the NB and WB modes through an API command. The BBIC must estimate and provide the correction value. The correction can occur immediately or relative to the receiver frame boundary. Another programmable NCO is employed with a configurable frequency -12 kHz to +12 kHz and frequency tuning word (FTW) 32 bits wide.

The API command **adi_adrv9001_Rx_FrequencyCorrection_Set( )** corrects small deviations in the receiver LO frequency. Provide the frequency deviation value at Hz and specify if the correction should take place immediately or at the start of the next available frame. Note that the device employs the digital NCO in the datapath to correct the frequency deviation instead of the RF PLL retuning.

Programmable FIR Filter (PFIR)

The PFIR is an optional 128-tap programmable FIR used in both the NB and WB modes. The four sets of customized FIR profiles are stored at the initialization phase. One of the four stored FIR profiles is switched to and loaded on the fly under the control of the baseband processor.

The PFIR is loaded with a customized low-pass filter profile to reduce the adjacent channel interference, which helps in better channel selectivity. For more details, see the Receiver Demodulator section.

A PFIR's cut off bandwidth can be no less than 1/5th of the RF Channel bandwidth e.g. if the RF channel bandwidth is set to 20 MHz, then the minimum PFIR cut off frequency supported is 4 MHz.

Receiver Signal Strength Indicator (RSSI)

RSSI measures the received signal power over a period of time, which is employed to calculate the interface gain to avoid saturating the data port. In addition, in the Monitor Mode, it performs signal detection in WB applications and works together with the FSK Discrimination block to detect signals in NB applications.

An API command **adi_adrv9001_Rx_Rssi_Read( )** retrieves the measured signal level. The API function reads back the RSSI status for the given receiver channel and is called any time after the device is fully initialized. The following data structure retrieves the power measurement in both the milli-dBFS and linear format.

```
typedef struct adi_adrv9001_RxRssiStatus{
uint16_t power_mdB;             /* Power in milli dB */
```

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

```
/* Linear power is calculated by this formula: linear power = (mantissa * 2^-15) * 2^-exponent */
uint16_t linearPower_mantissa;   /* Mantissa of Linear Power */
uint16_t linearPower_exponent;   /* Exponent of Linear Power */
} adi_adrv9001_RxRssiStatus_t
```

To accommodate different applications, the ADRV9001 provides some flexibility in configuring the RSSI calculation. There is the option to configure the duration of each RSSI calculation and the number of individual RSSI calculations used in moving the average to generate a more smooth RSSI result.

Manual RSSI

Manual RSSI is an alternative method of reading the RSSI (received signal strength indicator). The legacy RSSI API uses a mailbox command to read the RSSI, which normally takes around 1 ms to complete. Manual RSSI instead directly reads the ADRV9001 registers where the latest RSSI value is stored (direct register access). It has two modes of operation, high speed and high precision, and one user-configurable parameter, the RSSI Interval.

The RSSI in the ADRV9001 works by loading a buffer with a set number of samples (determined by the RSSI interval), and performing the calculation. When the calculation is complete, a new set of samples is loaded into the buffer. There is no overlap between the buffer in one calculation and the buffer in the next—each calculation uses an entirely new set of samples.

The RSSI value calculated by the RSSI block in the ADRV9001 is stored across three bytes: two bytes for the mantissa, and one byte for the exponent. Those values must be combined with the current slicerIn and slicerOffset values to get an accurate value of RSSI.

It is critical to note that while the RSSI will always return a value, this value may be out-of-date depending on the settings used. This is due to the difference between read time and measurement time. These two times are combined into a single reaction time, which is discussed in the High Precision and High Speed sections.

The manual RSSI API comes in two parts: configuration (**adi_adrv9001_Rx_Rssi_Manual_Configure()**), and readback (**adi_adrv9001_Rx_Rssi_Manual_Status_Get()**), and the readback API returns a struct of type **adi_adrv9001_ManualRssiReadStatus_t**.

RSSI Interval

The RSSI Interval determines how many samples are used for each RSSI calculation. This offers a trade-off between speed and accuracy—the more samples are used, the more accurate the RSSI value, but the longer it takes to calculate.

The RSSI runs at the same sample rate as the SSI, so the time taken for the RSSI value to update is the product of the RSSI Interval and the sample period.

High Precision

The high-precision mode is used to read the full value of RSSI stored in the registers of the ADRV9001. It starts by stopping the RSSI block, reading the registers, then restarting the RSSI block. This is done to ensure that the RSSI value is not updated while the registers are being read, which could lead to incorrect values if the exponent of one RSSI calculation is combined with the mantissa of another.

The registers read by this mode are the three RSSI registers (two bytes for the mantissa, and one exponent), and the slicer registers (slicerIn, and slicerOffset). The expected reaction time for this mode is the sum of the RSSI update period (RSSI interval × sample period), the SPI transaction time (the time taken to read all the registers over SPI), and the calculation time. The formula for the calculation is as follows:

$$RSSI\_linear = mantissa \times (2^{\wedge} - 15) \times (2^{\wedge} (-exponent + slicerIn \times (18 - 2 \times (3 - slicerOffset)))) \quad (3)$$

$$RSSI\_dBFS = 10\log10(RSSI\_linear) \quad (4)$$

High Speed

The high-speed mode is used to read to read an approximate value of RSSI stored in the registers of the ADRV9001. It does so by reading only the exponent register. Since only one register is being read, there is no need to stop the RSSI block for the duration of the read.

Due to the floating-point nature of how the linear RSSI value is stored, using only the exponent the high-speed mode returns readings in increments of 3 dB. The expected reaction time for this mode is the sum of the RSSI update period (RSSI interval × sample period), the SPI transaction time (the time taken to read one register over SPI), and the calculation time. The formula for calculating the RSSI using the high-speed mode is as follows:

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

$$RSSI\_dBFS = 3 \times (-(exponent) + slicerIn \times (18 - 2 \times (3 - slicerOffset))) \quad (5)$$

Note, that since this method does not read the slicerIn and slicerOffset registers. It requires that the values for those variables stored in the struct are correct at the time of calculation. Depending on the profile, these values can change at run-time. They are updated in the **adi_adrv9001_ManualRssiReadStatus_t** struct when calling either **adi_adrv9001_Rx_Rssi_Manual_Status_Get()** in the high-precision mode, or **adi_adrv9001_Rx_Rssi_Manual_Configure()**.

Interface Gain

Due to the bit-width limitation of the data port, an interface gain is applied by shifting the signal properly so as not to clip the output upon saturation. It increases the signal level for small signals to avoid losing sensitivity when a quantizer is used to limit the receiver output data bit-width. The interface gain is automatically adjusted internally inside the device by utilizing the RSSI measurement or through API commands. Optionally, retrieve the signal level measured by the RSSI through an API command to control the interface gain. For more details, see the Receiver Gain Control section.

Phase Offset Correction

In both the NB and WB modes, there is a phase offset correction block to adjust the sampling phase offset on IQ data or frequency deviation data. It samples the incoming received signal again by reconstructing intermediate samples between every two input samples according to the phase parameter configured through an API command. Currently, it is only programmable by the device.

NB FSK Discrimination

In NB applications, the ADRV9001 device demodulates and detects FSK/FM signals. This block has two operation modes; detecting and detected modes. The detecting mode is only used when the Monitor Mode is enabled. It detects the FSK/FM signals. As mentioned earlier, the signal detection can be accomplished by RSSI only. However, this block is used in the NB mode for more accurate signal detection. After the FSK/FM signal is detected, this block operates in the detected mode. Some components in the datapath are reconfigured to operate differently from the detecting mode. If no FSK/FM signal is detected, the transmitter/receiver moves to the sleep mode.

The DMR and FM radio usually have about 90% idle time. Therefore, both the RF front end and the baseband processor can go to sleep to save power. As a traditional solution, both the baseband processor and transmitter/receiver must power up to do the carrier detection, and the transmitter/receiver only passes through the data. With the equipped capability of the ADRV9001, it detects the DMR and FM signal independent of the baseband processor during its idle state so that the baseband processor can sleep through the entire idle time to extend the battery life. For more details, see the Receiver Demodulator section.

RECEIVE DATA CHAIN API PROGRAMMING

There is a set of receiver data chain APIs to interact with the ADRV9001 device receive datapath. Some of them are briefly discussed in the previous sections. This set of APIs is classified into three categories: Receiver Gain, Interface Gain, and Miscellaneous APIs, as shown in Table 63, Table 64, and Table 65, respectively. Each table summarizes the list of API functions with a brief description of each. For more details, see the ADRV9001 Device API Doxygen document.

Table 63. A List of Rx Gain APIs

Receiver Gain API Function Name	Description
adi_adrv9001_Rx_GainTable_Write	Programs the gain table settings for receiver channels.
adi_adrv9001_Rx_GainTable_Read	Reads the gain table entries for requested receiver channels .
adi_adrv9001_Rx_Gain_Set	Sets the Manual Gain Index for the given receiver channels.
adi_adrv9001_Rx_Gain_Get	Reads the receiver Gain Index for the requested receiver channels.
adi_adrv9001_Rx_GainIndex_Gpio_Configure	Configures GPIO pins to route the ADRV9001 Rx1 and Rx2 gain indices.

Table 64. A List of Interface Gain APIs

Receiver Interface Gain API Function Name	Description
adi_adrv9001_Rx_InterfaceGain_Configure	Sets the receiver interface gain control configuration parameters for the given receiver channel.
adi_adrv9001_Rx_InterfaceGain_Set	Sets the receiver interface gain for the given receiver channel.
adi_adrv9001_Rx_InterfaceGain_Inspect	Gets the receiver interface gain control configuration parameters for the given receiver channel.

RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN

Table 64. A List of Interface Gain APIs (Continued)

Receiver Interface Gain API Function Name	Description
adi_adrv9001_Rx_InterfaceGain_Get	Gets the receiver interface gain for the given receiver channel.
adi_adrv9001_Rx_DecimatedPower_Get	Gets the decimated power at configurable locations for the specified channel.

Table 65. A List of Rx Miscellaneous APIs

Receiver Miscellaneous API Function Name	Description
adi_adrv9001_Rx_Rssi_Read	Reads the received signal power measurement in both the linear and dB format.
adi_adrv9001_Rx_FrequencyCorrection_Set	Corrects small deviations in the receiver LO frequency offset.
adi_adrv9001_Rx_AdcSwitchEnable_Set	Sets the readiness of the dynamic switch between Low Power and High Performance ADCs.
adi_adrv9001_Rx_AdcSwitchEnable_Get	Gets the readiness of the dynamic switch between Low Power and High Performance ADCs.
adi_adrv9001_Rx_AdcSwitch_Configure	Configures ADC dynamic switch settings for the specified channel.
adi_adrv9001_Rx_AdcSwitch_Inspect	Inspects the current ADC dynamic switch settings for the specified channel.
adi_adrv9001_Rx_AdcType_Get	Gets the current ADC type for the specified channel.
adi_adrv9001_Rx_PortSwitch_Configure	Configures receiver port switching.
adi_adrv9001_Rx_PortSwitch_Inspect	Inspects the current receiver port switching settings.
adi_adrv9001_Rx_ExternalLna_Configure	Configures the external LNA for the desired receiver channel.
adi_adrv9001_Rx_ExternalLna_DigitalGainDelay_Set	Sets LNA digital gain delay for the desired receiver channel.
adi_adrv9001_Rx_ExternalLna_DigitalGainDelay_Get	Gets LNA digital gain delay for the desired receiver channel.
adi_adrv9001_Rx_Loid_Configure	Configures LOID settings.
adi_adrv9001_Rx_Loid_Inspect	Inspects LOID settings.

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

In the ADRV9001, to achieve optimal performance, an ARM performs calibrations, which are classified into two categories: initial calibrations performed at initialization before the device is operational and tracking calibrations performed regularly while the device is operational.

Initial calibrations are considered a part of the device initialization, which moves the device from a "STANDBY" state to a "CALIBRATED" state to prepare for transmit/receive operations. Tracking calibrations are performed regularly on-the-fly while the device is operational to track the changes such as attenuation/gain, temperature, and so on. The ADRV9001 includes two transmitters and two receivers. For each receiver, besides acting as a primary data channel for receiving RF signals, it also serves as an observation channel, which receives transmit signals through loopback paths. The observation channel can be controlled fully by the user or internally controlled by the device for some transmitter initial and tracking calibrations. Note that for some systems such as FDD 2T2R, the transmitter tracking calibrations requiring loopback paths are not available. For more details, see the ADRV9001 Example Use Cases section.

Most initial calibrations use internally generated tones or wideband signals for calibration, which requires users to satisfy external system requirements. Later sections discuss this topic in more detail. Different from initial calibrations, tracking calibrations use real-time traffic data for calibration. Therefore, tracking calibrations are transparent and do not require any user intervention. The ADRV9001 ARM schedules and performs both initial and tracking calibrations.

INITIAL CALIBRATIONS

There are three types of initial calibrations:

- System (non-channel related) initial calibrations
  - Initial calibrations for RF PLLs to calibrate the RF PLL for very fast frequency hopping mode
  - Aux PLL initial calibration
- Transmitter initial calibrations
  - Quadrature error correction (QEC)
  - Local oscillator (LO) leakage
  - Loop back path delay (LB PD)
  - Duty cycle correction (DCC)
  - Baseband analog filter (BBAF)
  - Baseband analog filter-group delay (BBAF GD)
  - Attenuation delay (ATTEN DELAY)
  - Digital-to-analog converter (DAC)
  - Path delay
- Receiver initial calibrations
  - High power ADC resistor/capacitor (HP ADC RC) (reserved, run automatically in analog initialize)
  - High power ADC flash offset (HP ADC flash)
  - High power ADC DAC (HP ADC DAC) (reserved, not used in ADRV9001)
  - Duty cycle correction (DCC)
  - Low power ADC (LP ADC)
  - TIA cutoff frequency (TIA Cutoff)
  - Transimpedance amplifier group delay
  - Wideband quadrature error
  - Frequency independent quadrature error
  - Internal loop back LOD (ILB LOD) (reserved, not used in ADRV9001)
  - DC offset (RF DC)
  - Gain path delay
  - DMR path delay

Note that the receiver initial calibrations must also be performed on loopback paths to prepare for transmitter initial and tracking calibrations.

To successfully perform all the initial calibrations, configure the ADRV9001 device properly. It is fully controlled by the ADRV9001 ARM. Therefore, no user interaction is required. However, besides the internal configurations, there are requirements for the external system as well. For example, during some transmitter initial calibrations, tones are generated and present at the transmitter output. Therefore, ensure

an appropriate level of isolation from the ADRV9001 transmitter output to the antenna to make sure the test tones are not transmitted by the system. Disable the power amplifier during the transmitter initial calibration to achieve this isolation.

Initial Calibrations API Programming

The ADRV9001 ARM in the device schedules/performs initial calibrations to optimize performance before device operation. The top-level API function **adi_adrv9001_cals_InitCals_Run( )** is used to perform initial calibrations.

The initial calibration is based on the initial calibration configuration defined by the following data structure.

```
typedef struct adi_adrv9001_InitCals
{
    adi_adrv9001_InitCalibrations_e sysInitCalMask;
    adi_adrv9001_InitCalibration_e chanInitCalMask[ADI_ADRV9001_MAX_RX_ONLY];
    adi_adrv9001_InitCalMode_e calMode;
    bool force;
} adi_adrv9001_InitCals_t
```

In this structure, sysInitCalMask is the initial calibration mask for system calibrations, chanInitCalMask[] is an array containing a calibration bit mask for channel related initial calibrations, (chanInitCalMask[0] is the mask for Rx1/Tx1 channels, chanInitCalMask[1] is the mask for Rx2/Tx2 channels), calMode specifies the mode to run the desired initial calibration algorithms, and force is a flag, which forces all enabled calibrations to rerun when it is set to be true.

The following enumerator type defines all the initial calibrations.

```
typedef enum adi_adrv9001_InitCalibrations
{
    ADI_ADRV9001_INIT_CAL_TX_QEC=0x00000001,
    ADI_ADRV9001_INIT_CAL_TX_LO_LEAKAGE=0x00000002,
    ADI_ADRV9001_INIT_CAL_TX_LB_PD=0x00000004,
    ADI_ADRV9001_INIT_CAL_TX_DCC=0x00000008,
    ADI_ADRV9001_INIT_CAL_TX_BBAF=0x00000010,
    ADI_ADRV9001_INIT_CAL_TX_BBAF_GD=0x00000020,
    ADI_ADRV9001_INIT_CAL_TX_ATTEN_DELAY=0x00000040,
    ADI_ADRV9001_INIT_CAL_TX_DAC=0x00000080,
    ADI_ADRV9001_INIT_CAL_TX_PATH_DELAY=0x00000100,
    ADI_ADRV9001_INIT_CAL_RX_HPADC_RC=0x00000200,
    ADI_ADRV9001_INIT_CAL_RX_HPADC_FLASH=0x00000400,
    ADI_ADRV9001_INIT_CAL_RX_HPADC_DAC=0x00000800,
    ADI_ADRV9001_INIT_CAL_RX_DCC=0x00001000,
    ADI_ADRV9001_INIT_CAL_RX_LPADC=0x00002000,
    ADI_ADRV9001_INIT_CAL_RX_TIA_CUTOFF=0x00004000,
    ADI_ADRV9001_INIT_CAL_RX_GROUP_DELAY=0x00008000,
    ADI_ADRV9001_INIT_CAL_RX_QEC_TCAL=0x00010000,
    ADI_ADRV9001_INIT_CAL_RX_QEC_FIC=0x00020000,
    ADI_ADRV9001_INIT_CAL_RX_QEC_ILB_LO_DELAY=0x00040000,
    ADI_ADRV9001_INIT_CAL_RX_RF_DC_OFFSET=0x00080000,
    ADI_ADRV9001_INIT_LO_RETUNE=0x000B902B,
    ADI_ADRV9001_INIT_CAL_RX_GAIN_PATH_DELAY=0x00100000,
    ADI_ADRV9001_INIT_CAL_RX_DMR_PATH_DELAY=0x00200000,
    ADI_ADRV9001_INIT_CAL_PLL=0x00400000,
    ADI_ADRV9001_INIT_CAL_AUX_PLL=0x00800000,
    ADI_ADRV9001_INIT_CAL_TX_ALL = 0x000001FF,
    ADI_ADRV9001_INIT_CAL_RX_ALL = 0x001FFE00,
    ADI_ADRV9001_INIT_CAL_RX_TX_ALL = 0x001FFFFF,
```

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

```
    ADI_ADRV9001_INIT_CAL_SYSTEM_ALL = 0x00C00000,
} adi_adrv9001_InitCalibrations_e
```

The following enumerator type defines the operating modes for initial calibrations.

```
typedef enum adi_adrv9001_InitCalMode
{
    ADI_ADRV9001_INIT_CAL_MODE_ALL,
    ADI_ADRV9001_INIT_CAL_MODE_SYSTEM_AND_RX,
    ADI_ADRV9001_INIT_CAL_MODE_LOOPBACK_AND_TX,
    ADI_ADRV9001_INIT_CAL_MODE_ELB_ONLY
}adi_adrv9001_InitCalMode_e;
```

ADI_ADRV9001_INIT_CAL_MODE_ALL runs all the selected initial calibrations, including both receiver (non-loopback and loopback paths) and transmitter initial calibrations. **ADI_ADRV9001_INIT_CAL_MODE_SYSTEM_AND_RX** runs the system and selected receiver initial calibrations (non-loopback paths), and **ADI_ADRV9001_INIT_CAL_MODE_LOOPBACK_AND_TX** runs the selected receiver calibrations on loopback paths (both internal and external loopback paths) and the selected transmitter initial calibrations. When using an external LO for both receiver and transmitter and when receiver LO and transmitter LO are at different frequencies, it takes time to change the LO frequency. Therefore, instead of running all the initial calibrations (select mode 0), first set the receiver LO and run the system and receiver initial calibrations (non-loopback path) (select mode 1), and then change to transmitter LO and run the receiver initial calibrations (loopback path) and transmitter initial calibrations (select mode 2). **ADI_ADRV9001_INIT_CAL_MODE_ELB_ONLY** runs all the initial calibrations on external loopback paths only. Do not explicitly use this mode. It is used when the user calls the **adi_adrv9001_cals_ExternalPathDelay_Run()** API command to get the external loopback path delay, which is used as an input to **adi_adrv9001_cals_ExternalPathDelay_Set()** for characterization.

Table 66 describes the mask bit assignment for initial calibrations in **adi_adrv9001_InitCalibrations_e**. It also explains the functionality of each initial calibration. Note that it is possible to select a different mask for Channel 1 (Tx1/Rx1) and Channel 2 (Tx2/Rx2).

Table 66. Initial Calibration Mask Bit Assignments

Bits	Corresponding Enum	Calibration	Description
D0	ADI_ADRV9001_INIT_CAL_TX_QEC	Transmitter QEC Initial Calibration	Performs an initial QEC calibration for frequency independent errors for the transmitter path. It estimates the gain and phase mismatch and applies the gain mismatch in the digital domain. Currently, it uses the transmitter path and an ILB path. If the transmitted data is quadrature modulated, perform this initial calibration, but it is not used if the transmitter is set to the direct modulation (DM) mode.
D1	ADI_ADRV9001_INIT_CAL_TX_LO_LEAKAGE	Transmit LOL Initial Calibration	Performs an initial LOL calibration. It estimates the LOL and applies the cancellation in the digital domain. Currently, it uses the transmitter path and an ILB path. If transmitted data is quadrature modulated, perform this initial calibration, but it is not used if the transmitter is set to the direct modulation (DM) mode.
D2	ADI_ADRV9001_INIT_CAL_TX_LB_PD	Transmit Loop Back Path Delay Calibration	Calibrates the internal transmitter loop back path delay (ILB PD). This information is required to calibrate QEC and LOL.
D3	ADI_ADRV9001_INIT_CAL_TX_DCC	Transmitter DCC Initial Calibration	Corrects the 50% duty cycle for external LO when the divisor is 2.
D4	ADI_ADRV9001_INIT_CAL_TX_BBAF	Transmitter BBAF Initial Calibration	Tunes the low-pass corner frequency and the pass-band flatness of the transmitter baseband analog filter.
D5	ADI_ADRV9001_INIT_CAL_TX_BBAF_GD	Transmitter BBAF-GD Initial Calibration	Estimates and corrects the filter group delay to remove frequency dependent quadrature error between the I and Q channels in each transmitter.
D6	ADI_ADRV9001_INIT_CAL_TX_ATTEN_DELAY	Transmitter ATTD Initial Calibration	Estimates the delay between the transmitter analog attenuation and digital attenuation. The delay is the same for all dynamic datapath

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

Table 66. Initial Calibration Mask Bit Assignments (Continued)

Bits	Corresponding Enum	Calibration	Description
			profiles, gain indices and frequency regions, and so on. Perform the calibration only on a single channel.
D7	ADI_ADRV9001_INIT_CAL_TX_DAC	Tx DAC Initial Calibration	Calibrates the DAC for the required profile bandwidth.
D8	ADI_ADRV9001_INIT_CAL_TX_PATH_DELAY	Transmitter Path Delay Initial Calibration	Estimates the delay between the transmitter input and Tx output.
D9	ADI_ADRV9001_INIT_CAL_RX_HPADC_RC	Receiver HP ADC RC Initial Calibration	Determines how much the unit R and C vary from the ideal and then tunes the HP ADC's programmable Rs and Cs to obtain the desired values. Without this calibration, the HP ADC's noise performance is negatively impacted. The HP ADC becomes unstable. It is not used when only LP ADC is used.
D10	ADI_ADRV9001_INIT_CAL_RX_HPADC_FLASH	Receiver HP ADC Flash Offset Initial Calibration	Optimizes the HP ADC output noise by correcting comparator offsets in the backend flash. It is not used when only LP ADC is used.
D11	ADI_ADRV9001_INIT_CAL_RX_HPADC_DAC	Receiver HP ADC DAC Initial Calibration	Corrects for element mismatch HP ADC current. It is not used when only LP ADC is used. **This is disabled by default – uncalibrated performance sufficient.**
D12	ADI_ADRV9001_INIT_CAL_RX_DCC	Receiver HP ADC Stability Initial Calibration	Corrects the 50% duty cycle for external LO when the divisor is 2.
D13	ADI_ADRV9001_INIT_CAL_RX_LPADC	Receiver LP ADC Initial Calibration	Calibrates the LP ADC. The major purpose of using the LP ADC instead of the HP ADC is to reduce power consumption. It is not used when only HP ADC is used.
D14	ADI_ADRV9001_INIT_CAL_RX_TIA_CUTOFF	Receiver TIA Cutoff Initial Calibration	Tunes the 3 dB cut-off frequency of the TIA filter.
D15	ADI_ADRV9001_INIT_CAL_RX_GROUP_DELAY	Receiver TIA Fine Initial Calibration	Compensates the mismatch in 3 dB cut-off frequency between I and Q path. Corrects quadrature error in the analog domain, which simplifies the correction in the digital domain.
D16	ADI_ADRV9001_INIT_CAL_RX_QEC_TCAL	Receiver Tone Initial Calibration	Performs an initial QEC calibration in wideband systems for frequency dependent quadrature errors. Also used for narrowband systems to improve receiver QEC performance, especially when the LO frequency is greater than 3.2 GHz.
D17	ADI_ADRV9001_INIT_CAL_RX_QEC_FIC	Receiver Frequency Independent Error Initial Calibration	Performs an initial QEC calibration for frequency independent errors for both narrowband and wideband systems.
D18	ADI_ADRV9001_INIT_CAL_RX_QEC_ILB_LO_DELAY	Receiver ILB LO Delay Initial Calibration	Adjusts the analog delay between the inphase and quadrature LO components at the mixer on the internal loopback path. This is not enabled currently.
D19	ADI_ADRV9001_INIT_CAL_RX_RF_DC_OFFSET	Receiver RFDC Offset Initial Calibration	Mitigates the RFDC offset added due to LO self-mixing.
D20	ADI_ADRV9001_INIT_CAL_RX_GAIN_PATH_DELAY	Receiver Gain Path Delay Initial Calibration	Calculates the path delay between the receiver analog and digital attenuation blocks. This delay is then used to offset the onset of receiver analog and digital attenuations relative to each other to compensate for the path delay between these blocks. It is independent of gain index and frequency region. Perform the calibration only on a single channel.
D21	ADI_ADRV9001_INIT_CAL_RX_DMR_PATH_DELAY	Receiver DMR Path Delay Initial Calibration	Measures the delay from the internal FIFO to sync detection time needed in the DMR and P25 monitor modes.

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

Table 66. Initial Calibration Mask Bit Assignments (Continued)

Bits	Corresponding Enum	Calibration	Description
D22	ADI_ADRV9001_INIT_CAL_PLL	PLL Initial Calibration	Performs VCO frequency calibration, VCO real-time temperature/aging calibration, and charge pump calibration to make RF PLL ready for operation. It is only used for very fast frequency hopping mode. **This is not enabled currently.**
D23	ADI_ADRV9001_INIT_CAL_AUX_PLL	AUX PLL Initial Calibration	Performs VCO frequency calibration, VCO real-time temperature/aging calibration, and charge pump calibration to make aux PLL ready for operation. **This is not enabled currently.**

The ADRV9001 ARM proceeds through the calibrations in the required sequential order. The system's initial calibrations are performed first, followed by receiver initial calibrations, and then transmitter initial calibrations. Table 67 and Table 68 show the receiver initial calibration order, and the transmitter initial calibration order, respectively.

Table 67. Rx Initial Calibration Order

Order	Receiver Initial Calibrations
1	RX_HPADC_RC
2	RX_HPADC_FLASH
3	RX_HPADC_DAC
4	RX_LPADC
5	RX_TIA_CUTOFF
6	RX_DCC
7	RX_GROUP_DELAY
8	RX_RF_DC_OFFSET
9	RX_GAIN_PATH_DELAY
10	RX_QEC_ILB_LO_DELAY
11	RX_QEC_TCAL
12	RX_QEC_FIC
13	RX_DMR_PATH_DELAY

Table 68. Tx Initial Calibration Order

Order	Transmitter Initial Calibrations
1	TX_DCC
2	TX_BBAF
3	TX_DAC
4	TX_LB_PD
5	TX_LO_LEAKAGE
6	TX_QEC
7	TX_BBAF_GD
8	TX_ATTEN_DELAY
9	TX_PATH_DELAY

The calibration order is mostly determined by the algorithm dependency. It is important to wait for these calibrations to complete successfully before continuing with the other steps of initialization for the device.

Note: Table 66 provides a full list of initialization calibrations for the device.

System Considerations for Transmitter Initial Calibrations

In this section, high level block diagrams show the device configurations and external system requirements for some transmitter initial calibrations. In all the diagrams, grayed-out lines and blocks are not active in the calibration. Note that as the ADRV9001 ARM performs each of the calibrations, it configures the ADRV9001 device per the following diagrams, with respect to enabling/disabling paths, and so on. No user input is required in this regard. However, it is important that external system conditions are met, such as having the power amplifier off for all calibrations except for some initialization calibrations utilizing ELB2.

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

Among nine transmitter initial calibrations, except for TX_DAC, all other eight calibrations require to insert the tone/wideband signal into the transmitter datapath from an internal signal generator. Therefore, the internal microprocessor disables the data port to avoid interference. Some calibration algorithms, such as TX_LB_PD, TX_QEC, TX_LOL, TX_DCC, and TX_ATTEN_DELAY further require the use of the observation datapath through ILB to receive the transmitted signal so that a joint analysis is performed by observing the relationship between the transmitted and received signals.

Transmitter Initial Calibrations Utilizing Internal Signal Generation Without Loopback

Figure 152 shows a high level block diagram of the system configurations for transmitter initial calibrations that require inserting signals into the transmitter datapath without using a loopback path.

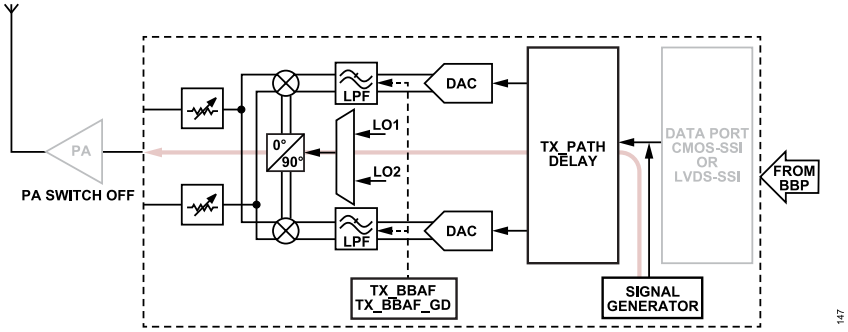


Figure 152. Transmitter Initial Calibration System Configuration with Signal Generation Without Loop Back

Figure 152 shows how an internal signal generator inserts calibration signals (red) into the transmitter datapath. The data port is disabled during initial calibrations to avoid interference. TX_PATH_DELAY is performed in the digital domain, whereas TX_BBAF and TX_BBAF_GD are performed in the analog domain. All use the signal generator to insert tones for calibration. During all these calibrations, test signals are inserted into the transmitter datapath and appear at the transmitter output. So, it is important to switch off the power amplifier connected to the device output, which also prevents signals from the antenna reaching the transmitter during calibrations. A 50 Ω termination is also needed to prevent tone signals bouncing back from the power amplifier input and reaching the device transmitter output, thereby confusing internal calibrations. The following paragraph summarizes the external system requirements.

External system requirements: For transmitter initial calibrations utilizing internal signal generation without loopback, power off the power amplifier in the transmitter path during these calibrations. When the power amplifier is disabled, the load at the transmitter output should be 50 Ω.

Transmitter Initial Calibrations Utilizing Internal Signal Generation and ILB

Figure 153 shows a high level block diagram of system configurations for transmitter initial calibrations utilizing internal signal generation and internal loop back (ILB) path.

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

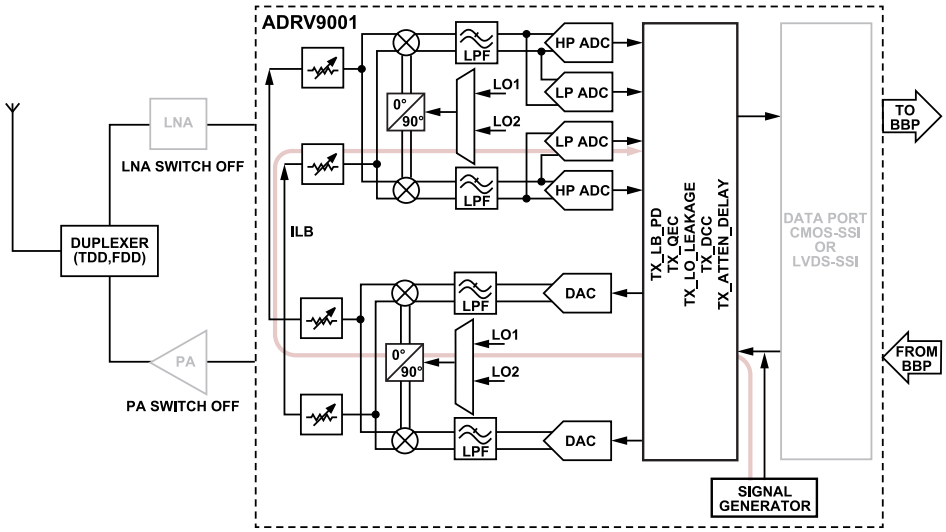


Figure 153. Tx Initial Calibration System Configuration with Signal Generation and Internal Loop Back

Figure 153 shows how TX_LB_PD, TX_QEC, TX_LO_LEAKAGE, TX_DCC, and TX_ATTEN_DELAY use the ILB for calibrations. TX_LO_LEAKAGE and TX_QEC calculate the initial correction parameters. TX_LB_PD provides a measurement of the loop back path delay for the TX_LO_LEAKAGE and TX_QEC algorithms. Both TX_LO_LEAKAGE and TX_QEC calibrations sweep through a series of attenuation values and create a table of initial calibration values. Then, during operation and upon application of a new transmitter attenuation setting, the corresponding QEC and LO_LEAKAGE correction values are applied to the transmitter channel by the ADRV9001 ARM. TX_DCC estimates the duty cycle error in the digital domain, but applies the correction in the analog domain. TX_ATTEN_DELAY measures the delay between the transmitter digital attenuation block and the transmitter analog attenuation block, and it uses the ILB to observe and estimate the delay.

Similarly, during all these calibrations, the power amplifier connected to the device output is switched off, and 50 Ω termination is needed. In addition, it is important to switch off the LNA (or RF switch if no LNA is present externally) external to the receiver datapath to avoid interference from the RF port into the receiver input used for data traffic. The following paragraph summarizes the external system requirement.

External system requirement: For transmitter initial calibrations using the ILB, power off the power amplifier in the transmitter path during these calibrations. When the power amplifier is disabled, the load at the transmitter output should be 50 Ω. Also switch off the LNA (or RF switch if no LNA is present externally) for the loopback path to avoid receiving signals from the RF port.

Initial Transmitter Calibration Utilizing Internal Signal Generation and ELB

It is also possible to perform some transmitter initial calibrations using external loop back (ELB). The Receiver/Observation Receiver Signal Chain section shows how using the ELB1 for initial calibrations provides the advantage of observing common-mode voltage. The use of ELB1 has the same external system requirements as when using ILB. Figure 154 shows the high level block diagram of initial transmit calibrations using internal signal generation and ELB1. Note that the ELB1 Initial calibration is a reserved function and the ADRV9001 does not support it.

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

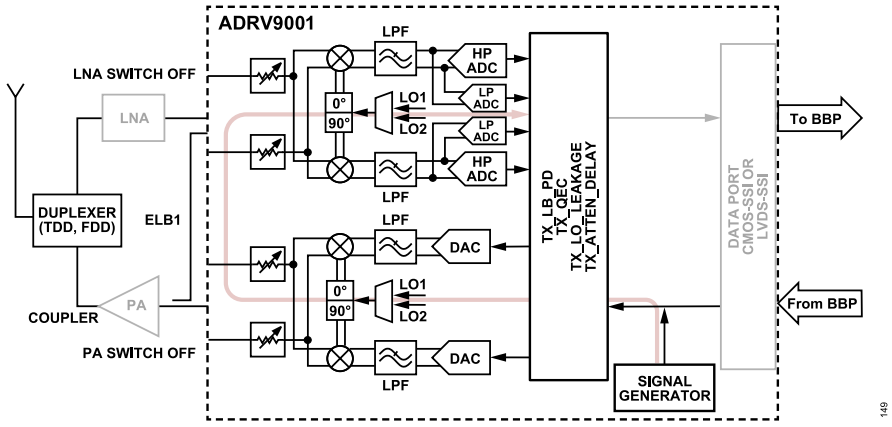


Figure 154. Transmitter Initial Calibration System Configuration with Signal Generation and External Loopback Type 1

For TX_LO_LEAKAGE, another option is to use ELB2. Figure 155 shows the high level block diagram of system configurations for TX_LO_LEAKAGE initial calibrations using ELB2 (note that the TX_LB_PD initial calibration using ELB2 is required for TX_LO_LEAKAGE). Note that the TX_LO_LEAKGE initial calibration using ELB2 is a reserved function and the ADRV9001 does not support it.

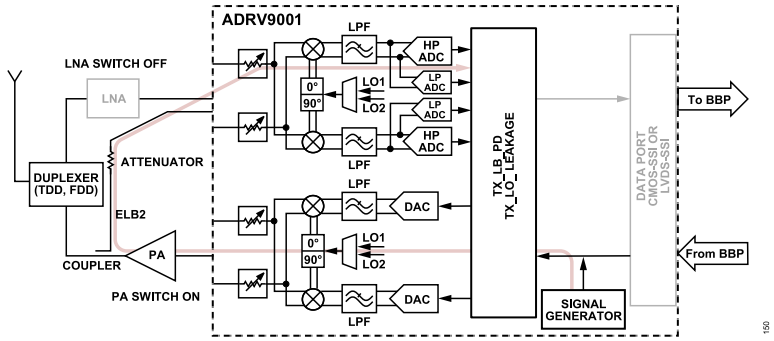


Figure 155. Tx Initial Calibration System Configuration with Signal Generation and External Loopback Type 2

When the ELB2 feedback path is used, enable the power amplifier to make a full external loop between the transmitter outputs and the observation channel inputs. The advantage of this calibration is to obtain a good estimate (gain/phase) of the external loop channel conditions before operation. Figure 155 shows the device configuration. Note that the calibration signal might be transmitted through the antenna. Although the power level of the calibration signal is set as low as possible, make sure it does not cause any problem when using this option.

It is important to choose a suitable attenuator between the power amplifier output and the observation channel input. This is to prevent the transmitted data from saturating the observation channel input. The following paragraph summarizes the external system requirements.

External system requirements: Choose a suitable attenuator between the power amplifier output and the observation channel input to prevent transmitted data from saturating the observation channel input. Switch off the LNA (or RF switch if no LNA is present externally) for the loopback path to avoid receiving signals from the RF port.

System Considerations for Receiver Initial Calibrations

This section also has high level block diagrams that show the device configurations and external system requirements for receiver initial calibrations. In all the diagrams, grayed-out lines and blocks are not active in the calibration. Note that the ADRV9001 ARM performs each of the calibrations. It configures the ADRV9001 device per the diagrams, with respect to enabling/disabling paths, and so on. No user input is required in this regard. However, it is important to meet the external system conditions, such as terminating the receiver input properly for receiver initialization calibrations.

Among 13 different receiver initial calibrations, RX_HPADC_RC, RX_HPADC_FLASH, RX_HPADC_DAC (reserved not enabled), and RX_LPADC calibrations are performed in the analog domain, and the corrections are applied to HP ADC or LP ADC (based on which one is used), while all other receiver initial calibrations are performed in the digital domain. For RX_QEC_FIC, RX_QEC_TCAL, RX_GAIN_PATH_DELAY, and RX_DMA_PATH_DELAY, the calibration results are applied in the digital domain for correction. For RX_DCC, RX_RF_DC_OFFSET,

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

RX_TIA_CUTOFF, RX_GROUP_DELAY, and RX_QEC_ILB_LO_DELAY, the calibration results are applied in the analog domain for correction. The Figure 156 shows the high level block diagram of system configurations for receiver initial calibrations. Note that different calibrations perform at different locations in the receiver datapath, which is simplified in Figure 156.

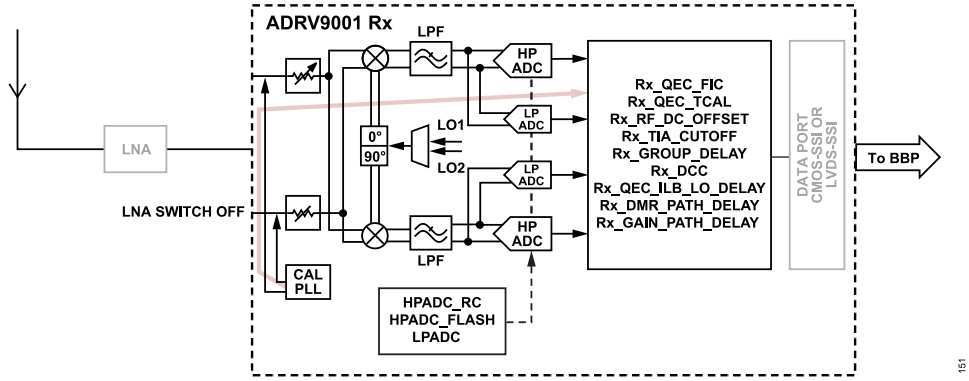


Figure 156. Receiver Initial Calibration System Configuration

During the receive initial calibration, as shown in Figure 156, the data port is disabled to avoid sending data to the baseband processor. The ADRV9001 ARM controls this without user interaction. Except for the RX_RF_DC_OFFSET calibration, all other digital domain calibration algorithms must be injected with calibration tones generated by the calibration PLL and these must be injected internally at the receiver input. For example, the RX_QEC_TCAL calibration routine sweeps a number of internally generated test tones across the desired frequency band, and then measures quadrature performance and calculates correction coefficients. Therefore, during the receive calibration, any incoming signals from the RF port must not be received, as these can interfere with the calibration tones. To ensure this, it is important to isolate the device receiver input port from the incoming signals by disabling the LNA (or by switching off the external RF switch if no LNA is present externally). This also prevents the calibration tones from reaching the antenna through the RF coupling. A 50 Ω termination is needed to prevent the tone signals bouncing back from an external LNA output and reaching the receiver input, confusing the internal calibrations. The following paragraph summarizes the external system requirements.

External system requirements: For optimal performance, lower calibration duration during receiver initial calibrations, and the device receiver input port requirements are isolated from incoming signals. For many receiver calibrations, the calibration tones appear on the receiver pins. Therefore, prevent it from reaching the antenna through the receiver port by terminating it properly. This also prevents the calibration tones from reaching the antenna through the RF coupling. A 50 Ω termination is needed to prevent tone signals bouncing back from an external LNA output and reaching the receiver input, confusing the internal calibrations.

Configure the Initial Calibrations Through TES

To achieve optimal performance, enable all initial calibrations. However, the TES allows to disable some initial calibrations, mainly for debugging purposes. Table 69 briefly compares all initial calibrations in terms of “User Override Capability”, “Run at Boot”, “Signal Used by Calibration”, “External Termination Needed”, and “Dependency”. A minimum set of initial calibrations must be rerun after “LO changes equal to or more than a certain range (for example, 100 MHz) ”. This set of initial calibrations is defined by the bit mask “ADI_ADRV9001_INIT_LO_RETUNE = 0x000B902B” in “adi_ADRV9001_InitCalibrations_e” enumerator type, as suggested by the Table 69. Note that "ADI_ADRV9001_INIT_LO_RETUNE" will run both Tx and Rx minimum calibrations. In a case where either Tx or Rx is disabled in a channel, this cannot be used. Instead, use the minimum calibrations as per Table 69.

Note that the full suite of initial calibrations can also be rerun to achieve optimal performance. Figure 157 and Figure 158 demonstrate the transmitter LO leakage performance and transmitter image rejection performance under different initial calibrations. In this experiment, the LO is swept from 100 MHz to 2.9 GHz at the 100 MHz step size. The blue line stands for the performance if the full suite of initial calibrations are rerun at each LO change. The red line stands for the performance of running the full suite of initial calibrations and then the minimum set of initial calibrations, alternatively (full initial calibrations for LO = 100 MHz, 300 MHz, 500 MHz,… and minimum initial calibrations for LO = 200 MHz, 400 MHz, 600 MHz,…). The gray line stands for the performance of running the full suite of initial calibrations and then no initial calibrations, alternatively (full initial calibrations for LO = 100 MHz, 300 MHz, 500 MHz,… and no initial calibrations for LO = 200 MHz, 400 MHz, 600 MHz,…). It is clear that not rerunning any initial calibrations after LO change = 100 MHz causes a significant performance penalty. However, the optimal performance (achieved by rerunning the full suite of initial calibrations) is mostly retained if rerunning the minimum set of initial calibrations.

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

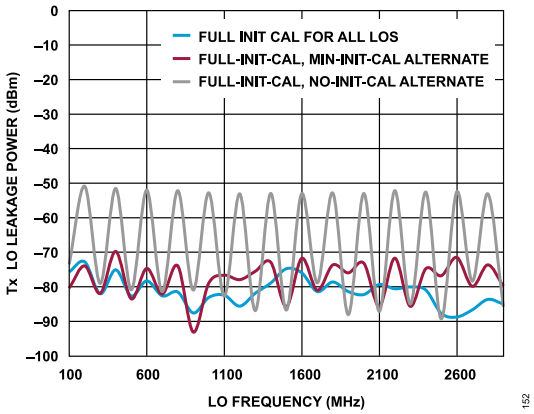


Figure 157. Tx LO Leakage Performance When LO Change = 100MHz with Full Init Cals, Min Init Cals and No Init Cals

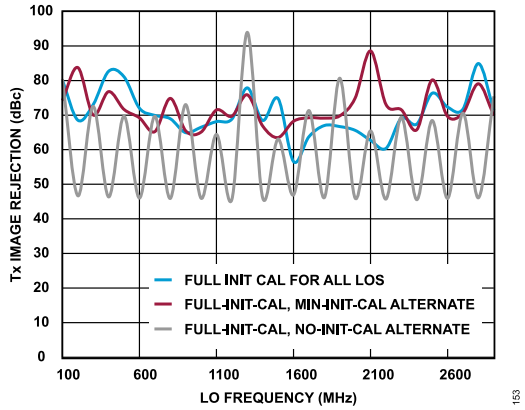


Figure 158. Tx Image Rejection Performance When LO Change = 100MHz with Full Init Cals, Min Init Cals and No Init Cals

Table 69. Initial Calibration Comparison Summary

Bits	Enum	User Override Capability	Run at Boot	Rerun after LO Change >100 MHz	Signal Used by Calibration (Tones, Wide-band, None)	External Termination Needed	Dependent on Which Init Cals is Run First
D0	TX_QEC	Yes	Yes	Yes	Tone	Yes	TX_DAC, TX_BBAF, all Receiver Init Cals on ILB, TX_LB_PD
D1	TX_LO_ LEAKAGE	Yes	Yes	Yes	Wideband	Yes	TX_DAC, TX_BBAF, all Receiver Init Cals on ILB and ELB, TX_LB_PD
D2	TX_LB_PD	Yes	Yes	No	Wideband	Yes	TX_DAC, TX_BBAF, RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC, RX_TIA_CUTOFF, RX_RF_DC_OFFSET
D3	TX_DDC	No	Yes	Yes	Tone	Yes	None
D4	TX_BBAF	No	Yes	No	Tone	Yes	None
D5	TX_BBAF_GD	No	Yes	Yes	Tone	Yes	TX_BBAF, TX_QEC
D6	TX_ATTEN_ DELAY	No	Yes	No	Tone	Yes	TX_DAC, RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC
D7	TX_DAC	No	Yes	No	None	No	None

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

Table 69. Initial Calibration Comparison Summary (Continued)

Bits	Enum	User Override Capability	Run at Boot	Rerun after LO Change >100 MHz	Signal Used by Calibration (Tones, Wide-band, None)	External Termination Needed	Dependent on Which Init Cals is Run First
D8	TX_PATH_DELAY	Yes	Yes	No	Tone	Yes	TX_ATTEN_DELAY
D9	RX_HPADC_RC	No	Yes	No	None	No	None
D10	RX_HPADC_ FLASH	No	Yes	No	None	No	None
D11	RX_HPADC_DAC	Not enabled	Not enabled	Not enabled	Not enabled	Not enabled	Not enabled
D12	RX_DDC	No	Yes	Yes	Tone	Yes	RX_TIA_CUTOFF
D13	RX_LPADC	No	Yes	No	None	No	None
D14	RX_TIA_CUTOFF	No	Yes	No	Tone	Yes	RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC
D15	RX_GROUP_DELAY	No	Yes	Yes	Tone	Yes	RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC, RX_TIA_CUTOFF
D16	RX_QEC_TCAL	No	Yes	Yes	Tone	Yes	RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC, RX_RF_DC_OFF-SET, RX_TIA_CUTOFF, RX_GROUP_DELAY
D17	RX_QEC_FIC	Yes	Yes	Yes	Tone	Yes	RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC, RX_RF_DC_OFF-SET, RX_TIA_CUTOFF, RX_GROUP_DELAY
D18	RX_QEC_ILB_LO_DELAY	No	Yes	No	Tone	Yes	RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC
D19	RX_RF_DC_ OFFSET	Yes	Yes	Yes	None	No	RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC
D20	RX_GAIN_PATH_ DELAY	No	Yes	No	Tone	Yes	RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC
D21	RX_DMR_PATH_DELAY	No	Yes	No	None	No	RX_HPADC_RC, RX_HPADC_FLASH, RX_LPADC
D22	PLL	Not enabled	Not enabled	No	Not enabled	Not enabled	Not enabled
D23	AUXPLL	Not enabled	Not Enabled	No	Not enabled	Not enabled	Not enabled

For the optional initial calibrations, the TES enables or disables these, as shown in Figure 159 . Note that in the current release, the configurable transmitter initial calibrations are LO Leakage (TX_LO_LEAKAGE), Loop Path Delay (TX_LB_PD), QEC (TX_QEC), and Duty Cycle Correction (external LO only) (TX_DDC). When TX_LO_LEAKAGE or TX_QEC is enabled, TX_LB_PD must be enabled too. When “transmitter Direct FM/FSK” mode is selected in DMR or AnalogFM profiles, TX_LO_LEAKAGE, TX_QEC, and TX_LB_PD calibrations are not used. These options are disabled in TES. TX_DDC is only applicable when external LO is used for the transmitter. The configurable receiver initial calibrations are QEC frequency independent (RX_QEC_FIC), RFDC (RX_RF_DC_OFFSET), and Duty Cycle Correction (external LO only) (RX_DDC). Similarly, RX_DDC is only applicable when external LO is used for receiver.

The initial calibrations are performed on clicking **Program** in TES. It takes some time to complete all the calibrations . When it is successful, the TES shows “Programmed”. If not, it shows “Programming Failed”. When “Programming Failed” happens, first check if all the external system requirements are satisfied. For example, no Rx input should present. If it is not the root cause, as the next step, enable/disable the optional

initial calibrations to see if the problem is related to some of the enabled calibrations. Similarly, for performance issues during the test, use the optional initial calibrations as a preliminary debug method.

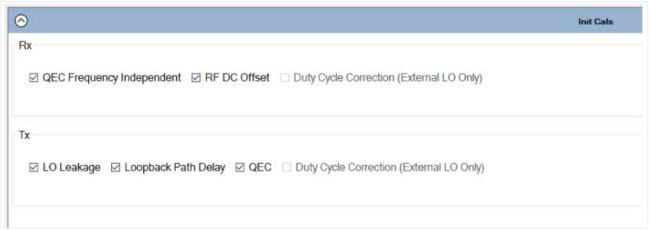


Figure 159. Initial Tx/Rx Calibration Configuration Through TES

TRACKING CALIBRATIONS

There are 12 different types of tracking calibrations, which are classified into transmitter and receiver/observation receiver tracking calibrations.

Transmitter Tracking Calibrations

- Quadrature error correction (QEC) tracking calibration
- Local oscillator leakage (LOL) tracking calibration
- Loop back path delay (LB PD) tracking calibration (reserved, not supported in the ADRV9001)
- Digital predistortion and closed loop gain control (DPD-CLGC) tracking calibration

Receiver/Observation Receiver Tracking Calibrations

- Harmonic distortion (second order) (HD2) tracking calibration
- Receiver quadrature error correction wideband poly (receiver QEC WBPLOY) tracking calibration
- Observation receiver quadrature error correction wideband poly (ORx QEC WBPLOY) tracking calibration
- Baseband DC offset (BBDC) tracking calibration
- RF DC (RFDC) tracking calibration
- Quadrature error correction FIC (QEC FIC) tracking calibration
- Automatic gain control (AGC) tracking calibration
- RSSI tracking calibration

The ADRV9001 internal microprocessor fully controls all the tracking calibrations. Therefore, there is no need for user interactions.

Tracking Calibrations API Programming

The ADRV9001 internal microprocessor in the device schedules/performs tracking calibrations to optimize the performance during its operation. The top-level API function **adi_ADRV9001_cals_Tracking_Set( )** performs the tracking calibrations.

The tracking calibrations are performed based on the tracking calibration configuration defined by the following data structure.

```
typedef struct adi_adrv9001_TrackingCals
{
    adi_adrv9001_TrackingCalibrations_e chanTrackingCalMask[ADI_ADRV9001_MAX_RX_ONLY];
} adi_adrv9001_TrackingCals_t
```

In this structure, chanTrackingCalMask[] is an array containing calibration bit mask for channel related tracking calibrations, (chanTrackingCalMask[0] is the mask for Rx1/Tx1 channels, and chanTrackingCalMask[1] is the mask for Rx2/Tx2 channels). The following enumerator type defines all the initial calibrations.

```
typedef enum adi_adrv9001_TrackingCalibrations
{
    ADI_ADRV9001_TRACKING_CAL_TX_QEC = 0x00000001,
    ADI_ADRV9001_TRACKING_CAL_TX_LO_LEAKAGE = 0x00000002,
```

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

```
    ADI_ADRV9001_TRACKING_CAL_TX_LB_PD = 0x00000004,
    ADI_ADRV9001_TRACKING_CAL_TX_DPD_CLGC = 0x00000010,
    /* Bit 6-7: Not used (Reserved for future purpose) */
    ADI_ADRV9001_TRACKING_CAL_RX_HD2 = 0x00000100,
    ADI_ADRV9001_TRACKING_CAL_RX_QEC_WBPOLY = 0x00000200,
    /* Bit 10-11: Not used (Reserved for future purpose) */
    ADI_ADRV9001_TRACKING_CAL_ORX_QEC_WBPOLY = 0x00001000,
    /* Bit 13-18: Not used (Reserved for future purpose) */
    ADI_ADRV9001_TRACKING_CAL_RX_BBDC = 0x00080000,
    ADI_ADRV9001_TRACKING_CAL_RX_RFDC = 0x00100000,
    ADI_ADRV9001_TRACKING_CAL_RX_QEC_FIC = 0x00200000,
    ADI_ADRV9001_TRACKING_CAL_RX_RX_GAIN_CONTROL_DETECTORS = 0x00400000,
    ADI_ADRV9001_TRACKING_CAL_RX_RSSI = 0x00800000
    /* Bit 24-31: Not used */
} adi_adrv9001_TrackingCalibrations_e
```

Table 70 describes the mask bit assignment for tracking calibrations in the "**adi_ADRV9001_TrackingCalibrations_e**". It also explains the functionality of each tracking calibration. Note that it is possible to select different masks for Channel 1 (Tx1/Rx1) and Channel 2 (Tx2/Rx2).

Table 70. Tracking Calibration Mask Bit Assignments

Bits	Corresponding Enum	Calibration	Description
D0	ADI_ADRV9001_TRACKING_CAL_TX_QEC	Tx QEC Tracking Calibration	Performs tracking QEC calibration for frequency independent errors for the Tx path. It estimates the gain and phase mismatch on-the-fly using the real-time traffic data and applies the gain mismatch in the digital domain. Similar to the initial calibration, currently it uses the Tx path and an ILB path. If transmitted data is quadrature modulated, this tracking calibration is performed, but it is not used if the data modulation is direct modulation (DM).
D1	ADI_ADRV9001_TRACKING_CAL_TX_LO_LEAKAGE	Tx LOL Tracking Calibration	Performs tracking LOL calibration. It estimates the LOL on-the-fly and applies the cancellation in the digital domain. It uses the Tx path and a loopback path (external loopback path preferred, if available). If transmitted data is quadrature modulated, this calibration is performed, but it is not used if the data modulation is direct modulation (DM).
D2	ADI_ADRV9001_TRACKING_CAL_TX_LB_PD	Tx Loop Back Path Delay Tracking Calibration	Tracks the Tx loop back path delay (can be for either ILB or ELB) on-the-fly. This information is required for QEC, LOL, and DPD tracking calibrations. Currently, this tracking calibration is not available. QEC, LOL, and DPD tracking calibration use the delay measurement obtained from the Tx loopback path delay initial calibration.
D4	ADI_ADRV9001_TRACKING_CAL_TX_DPD_CLGC	Tx DPD and CLGC Tracking Calibration	Predistorts the transmit signal in real-time to compensate for the power amplifier nonlinearity to achieve higher power efficiency and also compensate for the gain variation in the power amplifier. For more details, see the Digital Predistortion (DPD)section.
D6	Reserved		
D7	Reserved		
D8	ADI_ADRV9001_TRACKING_CAL_RX_HD2	Rx HD2 Tracking Calibration	Corrects the Rx second order harmonic distortion.
D9	ADI_ADRV9001_TRACKING_CAL_RX_QEC_WBPOLY	Rx QEC WB PLOY Tracking Calibration	Corrects the Rx frequency dependent quadrature error for WB applications.
D10	Reserved		
D11	Reserved		

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

Table 70. Tracking Calibration Mask Bit Assignments (Continued)

Bits	Corresponding Enum	Calibration	Description
D12	ADI_ADRV9001_TRACKING_CAL_ORX_QEC_WBPOLY	ORx QEC WB Tracking Calibration	Corrects ORx frequency dependent quadrature error for WB applications.
D13	Reserved		
D14	Reserved		
D15	Reserved		
D16	Reserved		
D17	Reserved		
D18	Reserved		
D19	ADI_ADRV9001_TRACKING_CAL_RX_BBDC	Rx BBDC Tracking Calibration	Mitigates the Rx DC offset at the baseband on-the-fly.
D20	ADI_ADRV9001_TRACKING_CAL_RX_RFDC	Rx RFDC Tracking Calibration	Mitigates the Rx DC offset at the RF (analog) on-the-fly.
D21	ADI_ADRV9001_TRACKING_CAL_RX_QEC_FIC	Rx QEC FIC Tracking Calibration	Corrects Rx frequency independent quadrature error for both NB and WB applications.
D22	ADI_ADRV9001_TRACKING_CAL_RX_GAIN_CONTROL_DETECTORS	Rx Gain Control Detectors Tracking Calibration	Enables/disables gain control detectors on-the-fly. By disabling it, AGC stops responding to changes in the input signal and instead holds its last gain index.
D23	ADI_ADRV9001_TRACKING_CAL_RX_RSSI	Rx RSSI Tracking Calibration	Enables/disables Rx signal strength measurement on-the-fly.

External System Requirements for Tracking Calibrations

Different from initial calibrations, tracking calibrations are performed on-the-fly with real-time traffic data. Therefore, it is mostly transparent and fully controlled by the internal microprocessor. The external system requirements are as follows:

- Make sure the external paths are available when running some of the tracking calibrations on external loopback paths.
- When using an external loopback path after a power amplifier (ELB2), choose a suitable attenuator between the power amplifier output and the observation channel input to prevent the transmitter output data from saturating the observation channel input.
- When an external DPD is employed in the system, it shares time with the other transmitter tracking calibrations to avoid conflicts.

For tracking calibrations, the TES enables or disables these calibrations, as shown in Figure 160. Note that in the current release, the configurable transmitter tracking calibrations are digital predistortion (TX_DPD_CLGC), LO leakage (TX_LO_LEAKAGE), and QEC (TX_QEC). When "Tx Direct FM/FSK" mode is selected in the DMR or AnalogFM profiles, TX_LO_LEAKAGE and TX_QEC calibrations are not applicable. These options are disabled in TES. The configurable receiver tracking calibrations are gain control detectors (RX_GAIN_CONTROL_DETECTORS), baseband DC offset (RX_BBDC), (second order) harmonic distortion (RX_HD2), frequency independent QEC (RX_QEC_FIC), and frequency dependent QEC for WB (RX_QEC_WBPOLY).

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

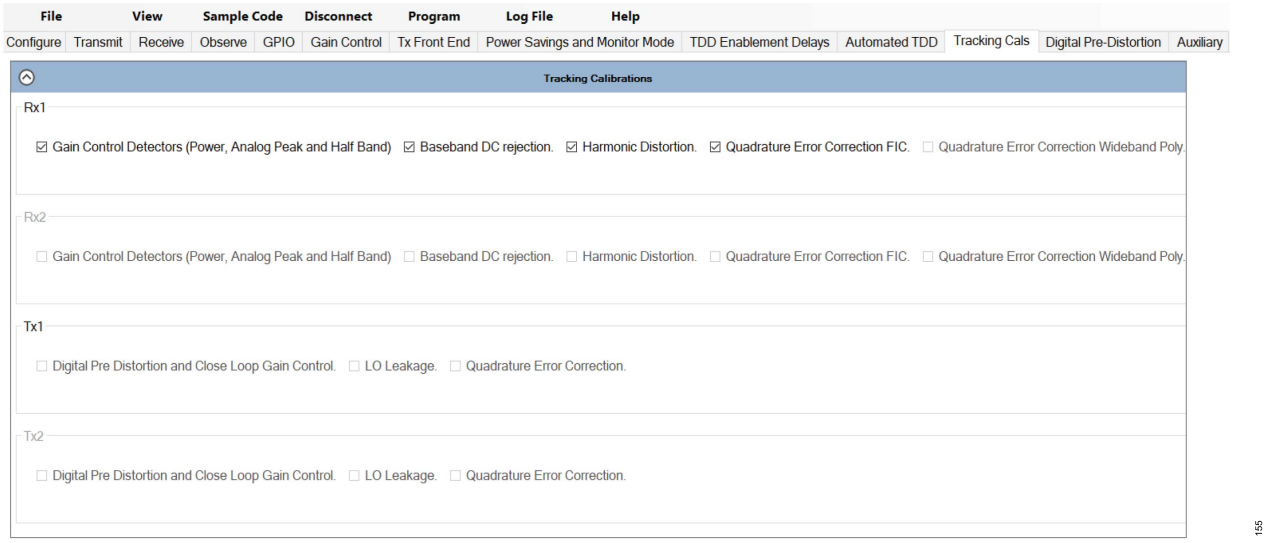


Figure 160. Tracking Tx/Rx Calibration Configuration Through TES

When an automated TDD configuration is used based on the TDD frame length, some of the tracking calibrations are automatically disabled in TES if the Rx/Tx frame length is less than 1 ms. If the receiver frame is less than 1 ms, RX_HD2, and RX_QEC_WBPOLY are disabled. If the transmitter frame length is less than 1 ms, TX_DPD_CLGC, TX_QEC, and TX_LO_LEAKAGE are disabled. This ensures the correct TDD operations with the user-provided TDD timing parameters. Therefore, it is recommended to follow the same requirement when configuring tracking calibrations in own software. To violate this rule for better possible performance, carefully evaluate the tracking configurations with a long TDD stability test.

Tracking Calibrations Scheduling

Tracking calibration scheduling refers to when the tracking calibrations will run based on the profile configurations. Tracking calibrations scheduling is handled differently in TDD and FDD profiles.

For FDD profiles, the tracking calibrations will run periodically while in the RF_ENABLED state. No user interaction is needed to perform the tracking calibrations.

For TDD profiles, including frequency hopping, the tracking calibrations will only run at the start of a frame. No user interaction is needed to perform the tracking calibrations but note that since tracking calibrations will only run at the start of a frame, tracking calibrations won't run periodically during a frame which might be significant for long frame times. The exception to this is the DPD/CLGC tracking calibrations as described in the DPD/CLGC Updates Scheduling section.

As described in the Transmitter/Receiver/Observation Receiver Signal Chain Calibrations section, the observational receiver (ORx) path is needed to perform the corresponding Tx tracking calibrations i.e. Tx1 uses ORx1 and Tx2 uses ORx2. User control of the ORx path will take priority over the tracking calibrations. This means that if the user is taking control of the ORx path, no Tx calibrations can run. In a case where it is needed to have user control of the ORx path as well as running Tx calibrations, a time-sharing scenario can be used.

For FDD profiles, the user can periodically take control of the ORx path (via raising the ORx enable pin) and when the user is not controlling the ORx path, the tracking calibrations will automatically resume. For TDD profiles, the user can take control of the ORx path for specific frames and frames where the user is not using the ORx path, tracking calibrations can run. For long frame times, it is possible to allow tracking calibrations at the start of a frame and then the user can take control of the ORx path for the remainder of the frame. In this case, tracking calibrations will need control of the ORx path for at least 2.2 ms at the start of the frame.

TRANSMITTER/RECEIVER/OBSERVATION RECEIVER SIGNAL CHAIN CALIBRATIONS

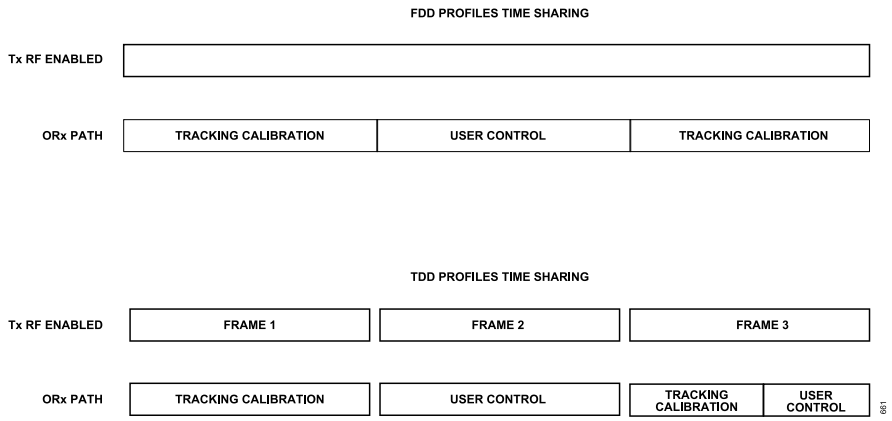


Figure 161. Tracking Calibration Time Sharing

RECEIVER GAIN CONTROL

The ADRV9001 receivers feature automatic and manual gain control modes for flexible gain control in a wide array of applications. It controls the gain at various stages of the receiver datapath to avoid overloading during the onset of a strong interferer. In addition, it ensures that the receiver digital output data is representative of the root mean square (RMS) power of the receiver input signal so that any internal front-end gain changes to avoid overloading are transparent to the baseband processor.

The two gain control modes in the ADRV9001 are called automatic gain control (AGC) and manual gain control (MGC). The AGC allows receivers to autonomously adjust the receiver gain depending on the variations of the input signal. It controls the gain of the device based on the information from signal detectors called peak detector and power detector. The receivers can also operate in the MGC mode, where changes in gain are initiated by the baseband processor through API commands or digital GPIO (DGPIO) pins. In the MGC mode, by enabling the signal detectors, the baseband processor can optionally use the information from the signal detectors through the DGPIO pins to properly control the gain.

The gain control is highly flexible and is configured differently in various scenarios. For example: for BTS receivers, the received signal is a multicarrier signal in most cases. Perform a gain change only under large overrange or underrange conditions, and gain changes should not occur very often for typical 3G/4G operations. Use peak detectors in such cases. Nevertheless, if an asynchronous blocker does appear, there is a “fast attack” mode, which reduces the gain at a fast rate. As another example, to support GSM blockers and radar pulses, which have fast rise and rapid fall times, there is a “fast attack and fast recovery” mode. This mode is capable of fast recovery in addition to the fast attack.

Section Topics

The following is the list of topics reviewed in detail.

- Receiver Datapath: This section outlines the gain control and signal observation elements of the receiver chain, followed by a description of the receiver gain table concept.
- Gain Control Modes: Advises on how to select between the AGC and MGC modes, followed by a detailed description of how to operate in each mode. The AGC mode section further discusses and compares peak and peak/power detect modes.
- Gain Control Detectors: Outlines the operation and configuration of various gain control detectors in the device.
- AGC Clock and Gain Block Timing: Describes the speed of the AGC clock and the various gain event and delay timers.
- Analog Gain Control API Programming: Outlines how to configure the analog gain control using API commands, explaining each parameter of the API structures. It also provides a summary of all supported API functions.
- Digital Gain Control and Interface Gain (Slicer): Outlines the various forms of digital gain control in the ADRV9001.
- Digital Gain Control and Interface Gain API Programming: Outlines how to configure the digital gain control and interface gain using API commands, explaining each parameter of the API structures. It also provides a summary of all supported API functions.
- Usage Recommendations: Provides a list of recommended gain control configurations to achieve optimal performance.
- TES Configuration and Debug Information : Advises on how to configure receiver gain control functionality through TES and perform simple debugging when encountering some gain control performance problems.

Important Terminology

The following is a summary of the important terms used in the following sections.

Manual Gain Control (MGC)

A use case when the user is in control of the currently applied gain settings in the receiver chain.

Automatic Gain Control (AGC)

The device’s own internal AGC, where the device is in control of the receiver gain settings. If the internal AGC is not used, an AGC is expected to run in the baseband processor. However, in this document, such a case is referred to as MGC because the gain of the receive path is under the user’s control.

Gain Attack

Indicates the reduction of the receiver gain due to an overloaded signal path.

Gain Recovery

Indicates the increase of the receiver gain due to a reduction in the power of the signal received.

Gain Compensation

RECEIVER GAIN CONTROL

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

Triggers the gain attack event. Some detectors can have multiple high thresholds.

Low Threshold

Triggers the gain recovery event. Some detectors can have multiple low thresholds.

Threshold Overload

When a threshold is exceeded in a signal detector, it is an overload.

Threshold Underload

When a threshold is not exceeded in a signal detector, it is an underload.

Overrange Condition

An overrange condition is 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 occurred within a defined period of time, or a power condition, where the measured power exceeds a high power threshold.

Underrange Condition

An underrange condition is 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 of time, or a power condition, where the measured power does not exceed a low power threshold.

RECEIVER DATAPATH

Figure 162 shows the simplified receiver datapath and gain control blocks. The receivers have front-end attenuators before the mixer stage, which are used to attenuate the signal in the analog domain to ensure the signal does not overload the receiver chain. Note: The ADRV9001 provides about 20dB gain so the front end gain attenuator further attenuates the signal from that level. There is also digital gain control capability in the digital domain.

The figure shows that the receiver chain has a number of observation elements that monitor the incoming signal. These are used in either the MGC or AGC mode. Firstly, there is an analog peak detector (APD) before the ADC. Being in the analog baseband, these peak detectors see the signals first, and also see the interfering signals that overload the ADC but are filtered as they progress through the digital chain. The second peak detector is called the HB peak detector because it monitors the data at the output of the half band (HB) filtering block in the receiver chain.

There is also a power measurement detection block at the same output of the HB filtering block, which takes the RMS power of the received signal over a configurable period of time.

Besides the front end gain control, this device also controls an external gain element through the analog GPIO (AGPIO) pins. In the digital domain, this device further controls the digital gain in both the wideband (WB) and narrowband (NB) modes. To avoid saturating the output signal due to the limitation of the bit-width of the data port, an optional interface gain (slicer) is applied at the end of the datapath by properly shifting the data. The interface gain is automatically controlled internally inside the device by using the information provided from the receiver signal strength indicator (RSSI) block or manually controlled through API commands.

Figure 162 shows that the gain control block has multiple inputs, which come from two peak detectors, and one power detector. By using the information from those detectors, the gain control block controls the gain of the signal chain using a predefined gain table. Note that the default gain table is loaded into the device during initialization. An API function **adi_adrv9001_Rx_GainTable_Write()** is called to load a custom gain table or reconfigure the gain table. Note that this operation is done before performing initial calibrations.

RECEIVER GAIN CONTROL

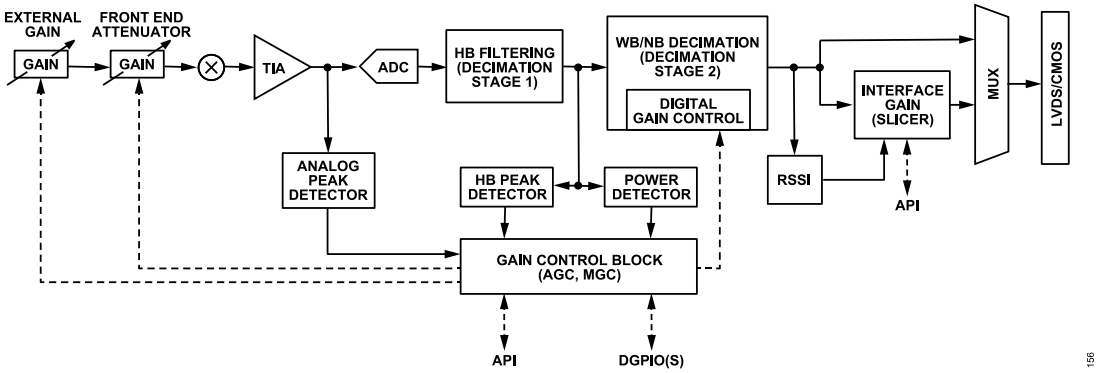


Figure 162. Rx Datapath and Gain Control Blocks

In this gain table, each row provides a unique combination of six fields, including front-end attenuator, TIA control, ADC control, external gain control, phase offset, and digital gain/attenuator. Among them, the TIA control that sets the TIA gain, ADC control that sets the ADC gain, and phase offset that compensates for the phase discontinuity during gain change are reserved for future use.

Based on the table row selected, either manually in the MGC mode or automatically in the AGC mode, the gain control block updates the variable gain elements depicted by the dash lines. In the MGC mode, the user controls the variable gain elements through the same gain table using the API commands and DGPIOs.

Table 71 shows the first three and the last three rows in a sample gain table.

Table 71. Sample Rows from the Default Rx Gain Table

Gain Table Index	Total Effective ADRV9001 Attenuation (dB)[1]	Front-End Attenuator Control Word [7:0]	TIA Control [2]	ADC Control [3]	External Gain Control [1:0]	Phase Offset	Digital Gain/Attenuator Control Word [10:0]
187	34	248	0	0	0	0	−2
188	33.5	247	0	0	0	0	−17
189	33	250	0	0	0	0	−4
…	…	…	…	…	…	…	…
253	1	28	0	0	0	0	−2
254	0.5	14	0	0	0	0	−1
255	0	0	0	0	0	0	0

[1] This column is not part of the Rx Gain Table and included only for illustrative purposes.

[2] TIA Control is a legacy component and won't be included in ADRV9001.

[3] ADC Control is a legacy component and won't be included in ADRV9001.

The gain table index is the reference for each unique combination of gain settings in the programmable gain table. The current possible range of the gain table is 187 to 255. The gain index region is user configurable. Call an API function **adi_adrv9001_Rx_MinMaxGainIndex_Set()** right after loading the gain table to load multiple gain table regions and switch between multiple gain table regions during runtime.

The front-end attenuator and digital gain/attenuator are the two fields used in the default gain table. The front-end attenuator is an 8-bit control word. The amount of attenuation applied depends on the value set in this column of the selected gain table index. The following equation provides an approximate relationship between the attenuation in dB and the front-end attenuation control word programmed in the gain table, N:

$$Attenuation\ \ (dB)\ \ =\ \ 20\log_{10}\left(\frac{256-N}{256}\right) \tag{6}$$

It is shown that index 255 denotes a 0 dB front-end attenuation and the step size between the adjacent gain index is approximately 0.5 dB. Note that when the index is below 195, the actual step size becomes less accurate.

The digital gain/attenuator column is used to apply the gain or attenuation digitally. The 11-bit signed word defines the digital gain applied, which equals the control word times 0.05 in dB. Table 71 shows that for gain index 253, the digital gain is calculated as -2 × 0.05 = -0.1 dB.

RECEIVER GAIN CONTROL

There are two types of receiver gain tables. One is for gain correction, in which the digital gain is for correcting the small step size inaccuracy in the front-end attenuator. The other one is for gain compensation, which compensates the entire front-end attenuation. The example Table 71 stands for a receiver gain correction table. The Receiver/Observation Receiver Signal Chain section mentions that the receiver can also be used as an observation channel (ORx).

The receive channel can also operate as an observation channel (ORx), as discussed in Receiver/Observation Receiver Signal Chain. In such a case, a different gain table is used. It provides 0 dB to 30 dB attenuation in 5 dB step size. The gain index is from 2 to 14. Gain index 14 provides 0 dB attenuation, then the next two indexes (both 13 and 12) provide 5 dB attenuation. The last two indexes (both 3 and 2) provide 30 dB attenuation. The digital gain specified in the gain table is only used for gain correction. In addition, only manual gain control is supported. Note that the ADRV9001 also uses the ORx gain table internally for initial and tracking calibrations (see Transmitter/Receiver/Observation Receiver Signal Chain Calibrations) so it should not be modified by user.

Gain Control with External Gain Control

The ADRV9001 can also control external gain such as the gain of an LNA. Configure the LNA gain by configuring the data structure **adi_adrv9001_RxLnaConfig_t** and then calling the API **adi_adrv9001_Rx_ExternalLna_Configure()**. Based on the user configuration, the ADRV9001 creates a new receiver gain table internally with extended gain indices to accommodate the additional LNA attenuation.

In the case of a single external gain control component, the minimum possible gain index becomes 137, which provides a maximum total of 59 dB attenuation, including the maximum LNA attenuation of 29 dB. For cases with more than one external gain control component, this limit is relaxed as described in the Multiple External Gain Control Components section. Note that with the extended gain table, the front-end attenuation included is only 30 dB to avoid gain step size inaccuracy from gain index 194 to 187.

As indicated in the default gain table, the external gain control uses a two bit control word through two AGPIO pins, which yields four different gain step sizes for each receive channel. The four step sizes are based on the attenuation relative to the max gain of LNA, and are defined as the following:

- Step 0 (control word 0) = 0 dB
- Step 1 (control word 1) = −N dB
- Step 2 (control word 2) = −N − M dB (optional)
- Step 3 (control word 3) = −N − M − L dB (optional)

Gain steps N, M, and L are multiple integers of 0.5 dB steps. In addition, N + M + L should not exceed 29 dB. Note that the N and M are optional. The gain table maintains a gain step of 0.5 dB between adjacent gain indices, and it assumes that the LNA step sizes are accurate.

Create the new receiver gain table first assuming the max LNA gain (0 dB) until the ADRV9001 front-end attenuator “runs out of” attenuation. Then new gain indices are produced by assuming the LNA gain of -N dB. To achieve the desired total attenuation by maintaining the 0.5 dB step size, recalculate the front end gain and set it properly in the new rows. Once the front-end attenuator “runs out of” attenuation again with LNA gain of -N dB, the new gain indices are further produced by assuming LNA gain of -N-M dB if LNA step 2 is configured. The same method applies to LNA step 3, if it is configured.

As an example, if the LNA is configured with step 1 of 10 dB attenuation only, the receiver gain table uses LNA with 0 dB attenuation for gain indices from 255 to 195 and sets the external gain control word to be 0. For gain entries below 195, it switches LNA 10 dB of attenuation as the gain index 195 has an attenuation of 30 dB to maintain 0.5 dB step size. The next gain index 194 represents 30.5 dB of total attenuation with a 10 dB external LNA. Therefore, the front-end attenuation is 30.5 - 10 = 20.5 dB, which reuses the setting associated with the index 255 – 20.5 × 2 = 214. So the entries for new gain index 194 are copied from index 214, plus the external control indicates that the LNA is enabled with 10 dB attenuation step by using control word 1. Each lower gain entry is simply copied from the next lower gain in table entries 213, 212, and until it reaches 195, which exhausts the maximum front-end attenuation.

Table 72 shows the generated new receiver gain table. It has new entries from 194 to 175 to accommodate the 10 dB LNA attenuation, and they are copied from 214 to 195 with external gain control set to be 1.

Table 72. New Rx Gain Table Created from the Default Rx Gain Correction Table

Gain Table Index	Total Effective ADRV9002 Attenuation (dB)[1]	Total Effective External Attenuation (dB)[1]	Front-End Attenuator Control Word [7:0]	TIA Control	ADC Control	External Gain Control [1:0]	Phase Offset	Digital Gain/Attenuator Control Word [10:0]
175	30	10	248 (copied from 195)	0	0	1	0	−2
…	…	…	…	…	…	…	…	…

RECEIVER GAIN CONTROL

Table 72. New Rx Gain Table Created from the Default Rx Gain Correction Table (Continued)

Gain Table Index	Total Effective ADRV9002 Attenuation (dB)[1]	Total Effective External Attenuation (dB)[1]	Front-End Attenuator Control Word [7:0]	TIA Control	ADC Control	External Gain Control [1:0]	Phase Offset	Digital Gain/Attenuator Control Word [10:0]
193	21	10	233 (copied from 213)	0	0	1	0	−20
194	20.5	10	232 (copied from 214)	0	0	1	0	−17
195	30	0	248	0	0	0	0	−2
196	29.5	0	247	0	0	0	0	−17
197	29	0	247	0	0	0	0	−7
…	…	…	…	…	…	…	…	…
253	1	0	28	0	0	0	0	−2
254	0.5	0	14	0	0	0	0	−1
255	0	0	0	0	0	0	0	0

[1] This column is not part of the Rx Gain Table and included only for illustrative purposes.

The above example uses a receiver gain correction table. A similar algorithm applies to generate the new receiver gain compensation table with only one difference: the digital gain also compensates the LNA attenuation in addition to the front end attenuation. Therefore, based on the external gain control word setting, the digital gain is further adjusted.

Note that the LNA must be powered down for initial calibrations. Once it is configured or bypassed during the radio on operation, it cannot be dynamically configured or bypassed.

It is recommended to use the external LNA gain control based on the default Rx gain table, as shown in the above example. However, the ADRV9001 allows to define the minimum gain index in the default Rx gain table (for example, set the minimum gain index to be 211 instead of 195) to accommodate the gain control requirements in the applications. The minimum gain index (minGainIndex) can be set in the LNA configuration structure **adi_adrv9001_RxLnaConfig_t**.

As mentioned, two AGPIO pins are used for each receiver to perform external gain control. Depending on the hardware register setting, the AGPIO pins for Receiver 1 and Receiver 2 are selected from AGPIO[3:0], AGPIO[7:4], and AGPIO[11:8]. Table 73 shows an example of Receiver 1 and Receiver 2 external gain element control when AGPIO[0:3] is selected (note that it is also possible to use AGPIO[1:0] for Receiver 2 and AGPIO[3:2] for Receiver 1). For more details, see the General-Purpose Input/Output (GPIO) and Interrupt Configuration section.

Table 73. An Example of Analog GPIOs for External Gain Element Control

Receiver	AGPIO Pins to Control External Gain Element
Rx1	AGPIO[1:0]
Rx2	AGPIO[3:2]

Enable these AGPIOs as outputs and set for external gain functionality. The programmed 2-bit value is directly related to the status of these AGPIO pins. For example, if the external gain word of the Receiver 1 gain table is programmed to three in the selected gain index, then the AGPIO[0] and AGPIO[1] are high if AGPIO[1:0] is used to control the external gain element, as the example shown in Figure 163.

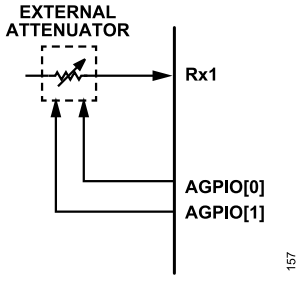


Figure 163. AGPIO Control of an External Gain Element to Rx1

See the Implement an External LNA in TES section for details on how to implement external gain control in the TES.

RECEIVER GAIN CONTROL

Measuring External Gain Control Path Delay

To accurately apply the attenuation for different components in the datapath, the user needs to measure the external path delay associated with the LNA and set the delay through the API command **adi_adrv9001_Rx_ExternalLna_DigitalGainDelay_Set()**.

- Properly load Rx gain table, set gain control and LNA configurations. Only LNA gain step 0 (0dB) and 1 (NdB) are needed for this measurement. Set the delay value as 0.
- Based on gain control and LNA configurations, the following info is known:
  - LNA gain step of N dB = LNAStepdB.
  - Min gain index allowed (when LNA gain is 0dB) = minGainIndex.
  - The adjacent gain index has a gain step size 0.5 dB.
- With above info, compute the following:
  - Gain index start and end that correspond to an N dB ADRV9001 RF attenuator only gain change is:
    - attenStartIndex = minGainIndex.
    - attenEndIndex = minGainIndex + 2 × LNAStepdB.
  - Gain index start and end that correspond to an N dB LNA only gain change is:
    - lnaStartIndex = minGainIndex − 2 × LNAStepdB.
    - lnaEndIndex = minGainIndex.
- Inject Rx input signal at desired level, through LNA into RX input port.
- Initialize ADRV9001 with all initial calibrations properly performed including Rx Gain Delay Initial Calibrations.
- Perform measurement 1—capture Rx data with attenuator only gain change.
- Perform measurement 2—capture Rx data with LNA only gain change.
- Analyze both captures to determine the timing when gain change starts (t1 for RF attenuator gain change and t2 for LNA gain change). The LNA delay is calculated as (*t2* − *t1*) × *ADRV9001 AGC clock*/4.

Multiple External Gain Control Components

As mentioned previously, the maximum gain of an external gain control component that the ADRV9001 can control is 29 dB due to the ADRV9001 having a maximum of 30 dB linear attenuation.

The ADRV9001 can also support up to 3 different external gain control components e.g. LNAs. In gain correction mode, each LNA can provide any gain up to 29 dB. This allows a maximum gain of −117 dB to be achieved using 3x 29 dB LNA's and the ADRV9001 i.e. N = 29 dB, M = 29 dB, L = 29 dB. The purpose of using multiple LNA's is to allow higher overall gain without compromising the ability to keep 0.5 dB step sizes.

In gain compensation mode, multiple LNAs are supported but the maximum external gain is limited to 29 dB total i.e. N + M + L <= 29 dB.

Customizing the Rx Gain Table

Recall the default Rx gain table Table 71. This default table has rows from 187 to 255 representing approximately 0.5 dB step sizes.

User may modify the default gain table. However, strict limitations apply to this modification:

- Gain step sizes should always be decreasing from 255.
- Each step size should be the same e.g. 255 = 0 dB gain, 254 = −2 dB gain, 253 = −4 dB gain.
- To do this, remove unwanted gain steps in the table but ensure to keep the index column the same.

Table 74 shows an example gain table where the step sizes were increased from 0.5 dB to 20 dB. To create this table, the following procedure was done:

- Row 255 remained the same at gain of 0 dB.
- Every 4th row from the default gain table was kept to create the equal 2 dB step sizes.
- The index column was maintained but now the table ends at row 238, due to needing less rows for 2 dB step sizes.

RECEIVER GAIN CONTROL

Table 74. Sample Rows from a Modified Rx Gain Table

Gain Table Index	Total Effective ADRV9002 Attenuation (dB)[1]	Front-End Attenuator Control Word [7:0]	TIA Control	ADC Control	External Gain Control [1:0]	Phase Offset	Digital Gain/Attenuator Control Word [10:0]
238	34	251	0	0	0	0	+13
239	32	250	0	0	0	0	+16
240	30	248	0	0	0	0	−2
…	…	…	…	…	…	…	…
253	4	94	0	0	0	0	−9
254	2	53	0	0	0	0	−4
255	0	0	0	0	0	0	0

[1] This column is not part of the Rx Gain Table and included only for illustrative purposes.

Notice that no new rows needed to be created. Rows which represented the more granular step changes were simply removed.

GAIN CONTROL MODES

Select the gain control mode through the API function **adi_adrv9001_Rx_GainControl_Mode_Set()** for a specified channel. For more details, refer to the API Doxygen document. **adi_adrv9001_RxGainControlMode_e** is an enum to select the gain mode. Table 75 shows the possible options.

Table 75. Definition of adi_adrv9001_RxGainControlMode_e

ENUM	Gain Mode
ADI_ADRV9001_RX_GAIN_CONTROL_MODE_SPI	Manual Gain Control SPI Mode
ADI_ADRV9001_RX_GAIN_CONTROL_MODE_PIN	Manual Gain Control PIN Mode
ADI_ADRV9001_RX_GAIN_CONTROL_MODE_AUTO	Automatic Gain Control Mode

The **adi_common_ChannelNumber_e** enum indicates the receiver channel to use

Table 76. Definition of adi_common_ChannelNumber_e

ENUM	Rx Channel
ADI_CHANNEL_1	Rx1
ADI_CHANNEL_2	Rx2

Automatic Gain Control (AGC)

In the AGC mode, a built-in state machine automatically controls the gain based on the user-defined configuration. The AGC is configured to 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 both the power detector and peak detectors are used jointly to make gain changes.

Peak Detect Mode

In this mode, only the peak detectors are used to inform the AGC to make gain changes. This section explains the basic premise of the operation, while the subsequent sections cover the more explicit details of configuring the peak detectors.

The APD and HB detector both have a high threshold and a low threshold, apdHighTresh, apdLowThresh, hbHighTresh, and hbUnderRange-HighThresh, respectively. These levels are user programmable, as well as the number of times to exceed a threshold to flag an overrange condition.

The high thresholds are used as limits on the incoming signal level and are principally 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 the signal level. When an underrange condition occurs, the AGC increases the gain (gain recovery). The AGC stable state (where it does not adjust gain) occurs when neither an underrange nor overrange condition occurs.

Each overrange/underrange condition has its own attack and recovery gain step, as shown in Table 77.

RECEIVER GAIN CONTROL

Table 77. Peak Detector Gain Steps

Overrange/Underrange	Gain Step
apdHighThresh overrange	Reduces gain by apdGainStepAttack
apdLowThresh underrange	Increases gain by apdGainStepRecovery
hbHighThresh overrange	Reduces gain by hbGainStepAttack
hbUnderRangeHighThresh underrange	Increases gain by hbGainStepHighRecovery

An overrange condition occurs when the high thresholds are exceeded a configurable number of times within a configurable period. An underrange condition occurs when the low thresholds are not exceeded a configurable number of times within the same configurable period. These counters make the AGC less sensitive to occasional peaks in the input signal, ensuring that a single peak exceeding a threshold does not necessarily cause the AGC to react. Table 78 outlines the counter parameters for the individual overload/underrange conditions.

Table 78. Peak Detector Counter Values

Overrange/Underrange	Counter
apdHighThresh overrange	apdUpperThreshPeakExceededCnt
apdLowThresh underrange	apdLowerThreshPeakExceededCnt
hbHighThresh overrange	hbUpperThreshPeakExceededCnt
hbUnderRangeHighThresh underrange	hbUnderRangeHighThreshExceededCnt

The AGC uses a gain update counter to time gain changes which happens when the counter expires. The counter value, and therefore the time spacing between possible gain changes, is user programmable through the gainUpdateCounter parameter. The user specifies the period in AGC clock cycles to make gain changes. Typically, this might be set to frame or sub-frame boundary periods.

Once the gain update counter expires, all the peak threshold counters are reset. The gain update period is, therefore, a decision period. The overload thresholds and counters are, therefore, set based on the number of overloads considered acceptable for the application within the gain update period.

Figure 164 shows an example of the AGC response to a signal versus the APD or HB peak detector threshold levels. APD and HB detector works in the same fashion in this mode. For ease of explanation, only APD is mentioned in the following discussions. The staircase line is representative of the peaks of the signal. Initially, the peaks of the signal are within the apdHighThresh and apdLowThresh. No gain changes are made. An interferer suddenly appears, whose peaks now exceed apdHighThresh. On the next expiry of the gain update counter (assuming a sufficient number of peaks occurred to exceed the counter), the AGC decrements the gain index (reduces the gain) by apdGainStepAttack. This is not sufficient to get the signal peaks within the threshold levels, and hence, the gain is decremented once more, with the peaks now between the two thresholds. The gain is stable in this current gain level until the interferer is removed and the peaks of the desired signal are now below the apdLowThresh, which results in an underrange condition. Hence, the AGC increases gain by the apdGainStepRecovery at the next expiry of the gain update counter, continuing to do so until the peaks of the signal are within the two thresholds once more.

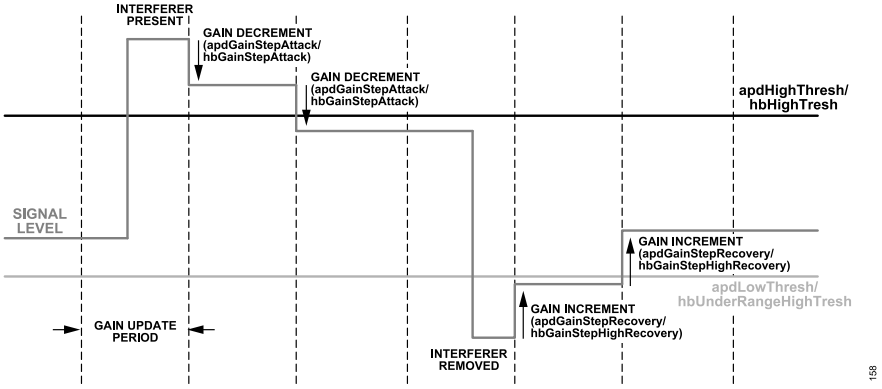


Figure 164. APD/HB Thresholds and Gain Changes Associated with Underrange and Overrange Conditions

It is possible to enable a fast attack mode, whereby the AGC is instructed to reduce gain immediately when an overrange condition occurs instead of waiting until the next expiry of the gain update counter using changeGainIfThreshHigh. This parameter has independent controls for the APD and HB detectors. Values from 0 to 3 are valid, as shown in Table 79.

RECEIVER GAIN CONTROL

Table 79. changeGainIfThreshHigh Settings

changeGainIfThreshHigh[1:0]	Gain Change Following APD Overrange	Gain Change Following HB Overrange
00	After expiry of gainUpdateCounter	After expiry of gainUpdateCounter
01	After expiry of gainUpdateCounter	Immediately
10	Immediately	After expiry of gainUpdateCounter
11	Immediately	Immediately

Figure 165 shows how AGC reacts when the changeGainIfThreshHigh is set for the APD or HB detector. In this case, when the interferer appears, the gain is updated as soon as the number of peaks exceeds the peak counter. It does not wait for the next expiry of the gain update counter. Hence, a number of gain changes are made in quick succession, providing a much faster attack than the default operation. The assumption here is that if the ADC is overloaded, then it is best to decrease the gain quickly rather than wait for a suitable moment in the received signal to change the gain. This mode is referred to as the "fast-attack" mode.

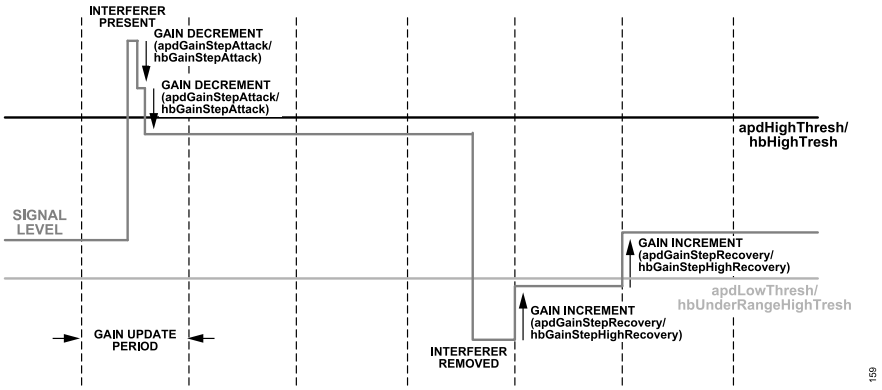


Figure 165. APD/HB Gain Changes with Fast Attack Enabled

Besides the "fast attack" mode, it is also possible to enable a "fast recovery" mode. This functionality is enabled with the enableFastRecovery-Loop parameter.

This "fast recovery" mode only works with the HB detector. Figure 166 shows this operation. In this mode, achieve the "fast recovery" using multiple low thresholds and step sizes as well as the update periods. When the signal level falls below hbUnderRangeLowThresh, the lowest threshold, the gain is incremented the most by hbGainStepLowRecovery following the expiry of a gain update period. Note that in the "fast recovery" mode, the agcUnderRangeLowInterval is used instead of the gain update counter to set the gain update period (this also applies to the APD). After sufficient gain increases to bring the signal level above hbUnderRangeLowThresh, the gain is incremented by hbGainStepMidRecovery after the expiry of agcUnderRangeMidInterval, which is a multiple of agcUnderRangeLowInterval. Finally, when the signal level is increased above hbUnderRangeMidThresh, the gain is incremented by hbGainStepHighRecovery following the expiry of agcUnderRangeHighInterval, which is a multiple of agcUnderRangeMidInterval.

The multiple thresholds and interval parameters allow for faster gain recovery. Typically, agcUnderRangeHighInterval is set to gain update counter, as shown in Figure 166. Therefore, when the signal level is below the middle and low thresholds, the recovery happens multiple times within a single gain update counter, making the recovery process much faster. Note that in the "fast recovery" mode, gain recovery might not always happen at the expiry of the gain update counter, which is different from the mode without "fast recovery". If the gain update counter is set to align with the frame or subframe boundary, it is possible that a fast recovery happens in the middle of a frame or subframe. Therefore, it is recommended not to use the "fast recovery" mode when there is a stringent requirement for keeping a constant gain for an entire frame or subframe.

RECEIVER GAIN CONTROL

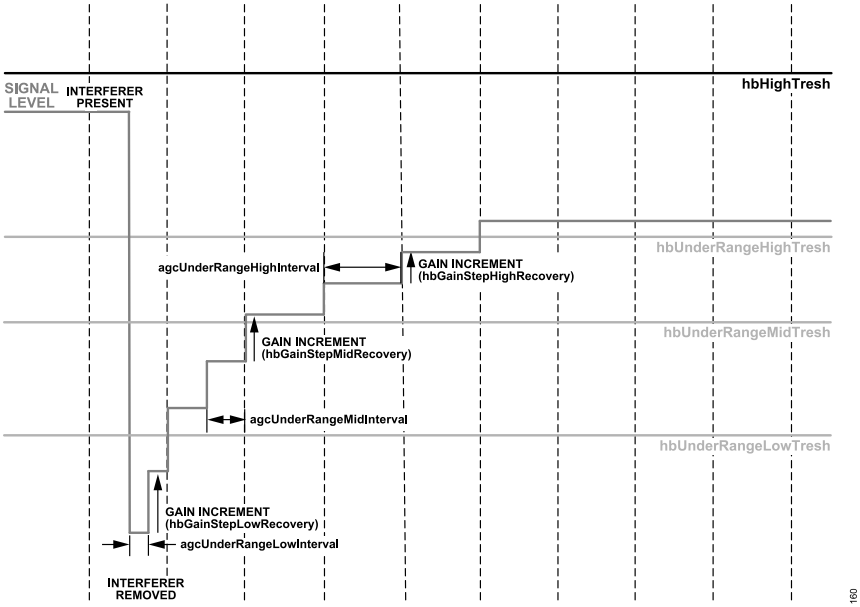


Figure 166. AGC Operation with HB Detector in Fast Recovery Mode

It is highly recommended that the apdHighThresh and hbHighThresh are set to an equivalent dBFS value. Likewise, it is highly recommended that the apdLowThresh and the hbUnderRangeHighThresh are set to equivalent values. 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 are exceeded. Table 80 shows the priorities between the detectors when multiple overranges occur.

Table 80. Priorities of Attack Gain Steps

apdHighThresh Overrange	hbHighThresh Overrange	Gain Change
No	No	No Gain Change
No	Yes	Gain Change by hbGainStepAttack Gain
Yes	No	Change by apdGainStepAttack
Yes	Yes	Gain Change by apdGainStepAttack

Note that in the case of apdHighThresh Overrange is Yes, but hbHighThresh Overrange is No, there are two possibilities: one is due to the out-of-band blocker, which is only observed by the APD, the other one is due to a low frequency signal near DC. In the second case, when the signal is near DC, due to the characteristics of the APD, it might generate positive overrange conditions. To overcome this issue, a mitigation mode is designed, which utilizes a secondary digital threshold to decide if the APD overrange is a false positive. When the mitigation mode is turned on, if the secondary threshold is exceeded, then it confirms a false positive due to a signal near DC, so the gain does not change. If the secondary threshold is not exceeded, then it confirms an out-of-band blocker, so the gain is decremented. In the current implementation, the mitigation mode is always turned on, and the user is not allowed to configure the secondary digital threshold.

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

1. hbUnderRangeLowThresh Underrange Condition
2. hbUnderRangeMidThresh Underrange Condition
3. hbUnderRangeHighThresh Underrange Condition
4. apdLowThresh 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; i.e., if hbUnderRangeLowThresh is reporting an underrange condition, then the AGC adjusts the gain by hbGainStepLowRecovery with two exceptions.

The apdLowThresh has priority in terms of preventing recovery. If apdLowThresh reports an overrange condition (sufficient signal peaks have exceeded its threshold in a gain update counter period), then no further recovery is allowed. apdLowThresh and hbUnderRangeHighThresh are

RECEIVER GAIN CONTROL

configured to be close to the same value of dBFS, but assuming some small difference between the thresholds, then as soon as apdLowThresh is exceeded, recovery no longer occurs. The reverse is not true. hbUnderRangeHighThresh does not prevent the gain recovery towards the apdLowThresh. Given the strong recommendation that the apdLowThresh and hbUnderRangeHighThresh be set equally, then a condition whereby apdLowThresh is at a lower dBFS level to hbUnderRangeLowThresh or hbUnderRangeMidThresh should not occur.

Another exception is if the recovery step size for a detector is set to zero. If so, the AGC makes the gain change of the highest priority detector with a non-zero recovery step. Figure 167 provides a flow diagram of the decisions of the AGC when recovering the gain in the peak detect mode.

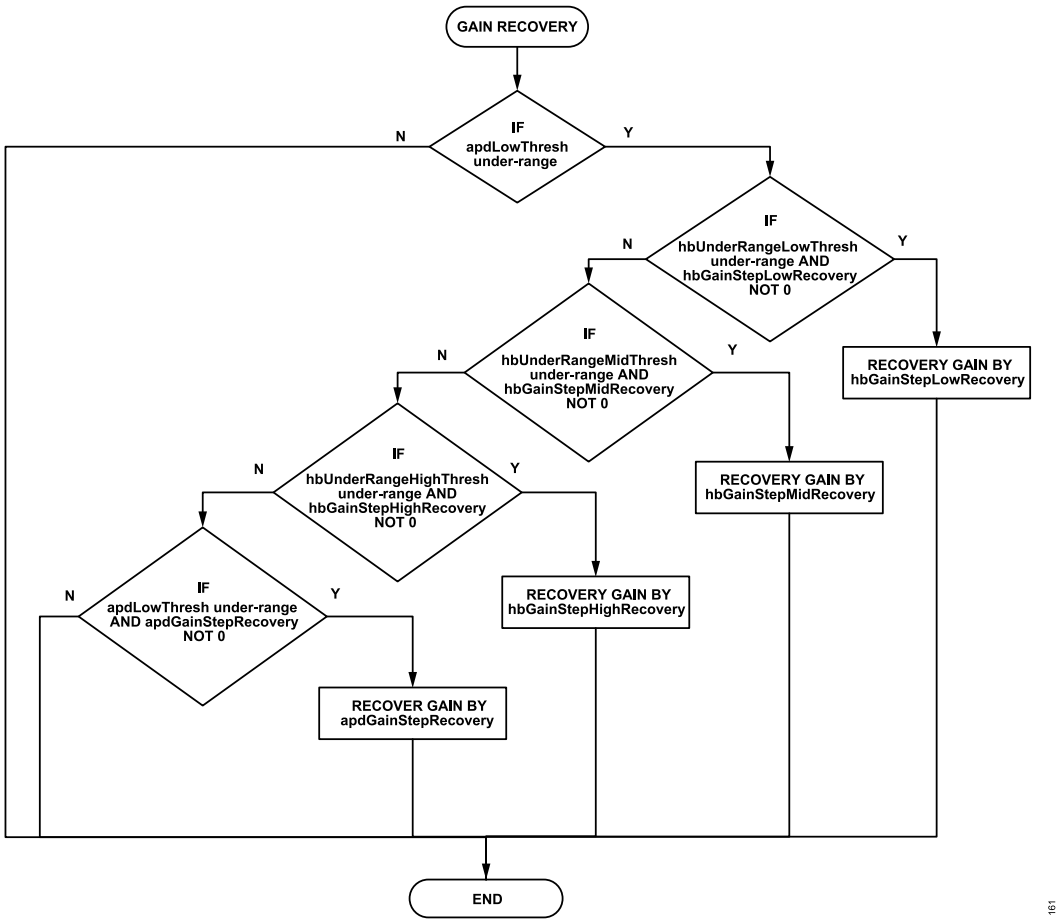


Figure 167. Flow Diagram for AGC Recovery in Peak Detect AGC Mode

Peak/Power Detect Mode

In this mode, the peak and power detect work jointly to control the gain of the receiver chain. In the event of an overrange condition, both the peak and the power detector can instantiate a gain decrement. In the event of an underrange, only the power detector can increment the gain. The power detector changes gain solely at the expiry of the gain update counter. As mentioned earlier, the peak detector is set in one of two modes (depending on the setting of gainChangeIfThreshHigh), whereby the AGC: 1) waits for the gain update counter to expire before initiating a gain change, or 2) immediately updates the gain as soon as the overrange condition occurs (see Figure 164 and Figure 165). Therefore, in the peak/power detect mode, if the gain attack is instantiated by peak detectors, it is possible to perform the fast attack.

The power detector provides the RMS power measurement of the receiver data at the output of the HB filtering block. In the power detect mode, the AGC compares the measured signal level to programmable thresholds, which provide a second order control loop, whereby the gain is changed by the larger amounts when the signal level is farther from the target level while making smaller gain changes when the signal is closer to the target level. This allows the gain to change faster when the level is farther from the targeted range.

RECEIVER GAIN CONTROL

Figure 168 shows the operation of the AGC when using the power detector. Considering the power detected in isolation from the peak detectors, the AGC does not modify the gain when the signal level is between overRangeLowPowerThresh and underRangeHighPowerThresh. This is the target range for the power measurement. The associated thresholds are also called inner thresholds.

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. underRangeLowPowerThresh and overRangeHighPowerThresh are also called outer thresholds.

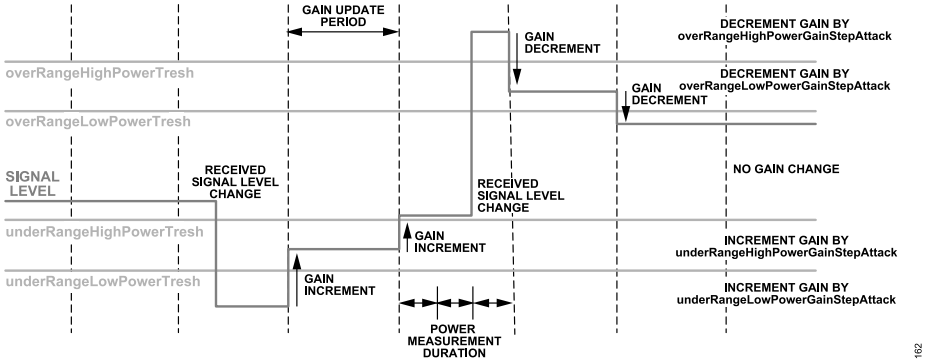


Figure 168. Power Detector Thresholds and Gain Changes for Underrange and Overrange Conditions

It is possible for the AGC to get contrasting requests from the power and peak detectors. An example is an interferer visible to the analog peak detector but significantly attenuated at the power detector. In this case, the APD is requesting a gain decrement, while the power detector is requesting a gain increment. The AGC has the following priority scheme in peak/power detect modes:

1. APD overrange
2. HB overrange
3. APD lower level peak exceeded
4. HB lower level peak exceeded
5. Power measurement

In this example, the gain is decremented because the APD overrange has a higher priority than the power measurement. However, the APD and HB lower level overload act differently in the peak detector and peak/power detect modes. In the peak detect mode, the lower level thresholds for these detectors are used to indicate an underrange condition, which causes the AGC to increase the gain. In the peak/power detect mode, these detectors are not used for gain recovery but used to control gain recovery by setting the API parameter lowThreshPreventGainInc. If this parameter sets and the signal leveling exceeds a lower level threshold, the AGC is prevented from increasing gain regardless of the power measurement.

When a signal has higher than the expected PAR, the power detector can indicate the gain increase while the peak detector's low threshold can still be exceeded. In such a case, the gain increase is prevented to avoid an overloading possibility. In addition, this can prevent an oscillation condition that can otherwise occur to an interferer visible to the APD but filtered before the power detects. In such a case, the peak detect can cause the AGC to decrease the gain. It does this until the interferer is no longer exceeding the defined threshold. At this point, the power detector can request an increase in the gain and does so until the detector’s low threshold is exceeded. This might cause an oscillation condition. By using these lower level thresholds of peak detection, the AGC is prevented from increasing the gain as the signal level approaches an overload condition, providing a stable gain level for the receiver chain under such a condition.

Comparing the Peak Detect and Peak/Power Detect Modes

Among the two detection modes, peak detector offers the quickest response time to overload signals by employing “fast attack” mode. It allows the AGC to respond within hundreds of nanoseconds in overload scenarios. In addition, the peak detector also provides a “fast recovery” option to increase the gain of the desired signal quickly when an interferer disappears. It can also avoid the possible gain index oscillation issue of peak/power detect when the signal has higher than expected peak-to-average ratio (PAR).

RECEIVER GAIN CONTROL

With power detection, the gain change can only happen at the expiration of the gain update counter, which is typically set in the order of hundreds of microseconds or milliseconds. However, power detectors are usually more stable and unlikely to cause frequent gain changes. In addition, it can provide tighter control of the signal level by using a set of inner and outer thresholds compared to the peak detector.

It is highly recommended to use peak detection, especially for fast gain control.

Manual Gain Control (MGC)

The gain control block applies the settings from the selected gain index in the gain table. In the MGC mode, the baseband processor is in control of selecting the gain index. There are two options: API command (**ADI_ADRV9001_RX_GAIN_CONTROL_MODE_SPI**) and pin control (**ADI_ADRV9001_RX_GAIN_CONTROL_MODE_PIN**). By default, if MGC is chosen, the part is configured for an API command.

In the API command mode, select a gain index in the gain table through the API function a**di_adrv9001_Rx_Gain_Set()**. The gain index selected for a channel is read back through the API function **adi_adrv9001_Rx_Gain_Get()**.

The pin control MGC mode is useful for the real time control of gain. The API command **adi_adrv9001_Rx_GainControl_PinMode_Configure()** is used to properly configure this mode. In this mode, out of 16 digital DGPIO pins, two pins per receiver are used, one increasing and the other decreasing the gain table index. Specify both the maximum and minimum gain index as well as the increment and decrement step size (in the range of 0 to 7 gain table indices). Apply a pulse to the relevant DGPIO pin to trigger an increment or decrement in gain, as shown in Figure 169. Hold this pulse high for at least two AGC clock cycles for a reliable detection of the rising edge to trigger the gain change. The following defines the configuration data structure for this mode.

```
typedef struct adi_adrv9001_RxGainControlPinCfg
{
uint8_t minGainIndex; //Minimum gain index. Must be >= gainTableMinGainIndex and < maxGainIndex
uint8_t maxGainIndex; //Maximum gain index. Must be > minGainIndex and <= gainTableMaxGainIndex
uint8_t incrementStepSize; //Number of indices to increase gain on rising edge on incrementPin (Range:
0 to 7)
uint8_t decrementStepSize; //Number of indices to decrease gain on rising edge on decrementPin (Range:
0 to 7)
adi_adrv9001_GpioPin_e incrementPin;// A rising edge on this pinincrements gain by incrementStepSize.
adi_adrv9001_GpioPin_e decrementPin; //A rising edge on this pin decrements gain by decrementStepSize.
} adi_adrv9001_RxGainControlPinCfg_t
```

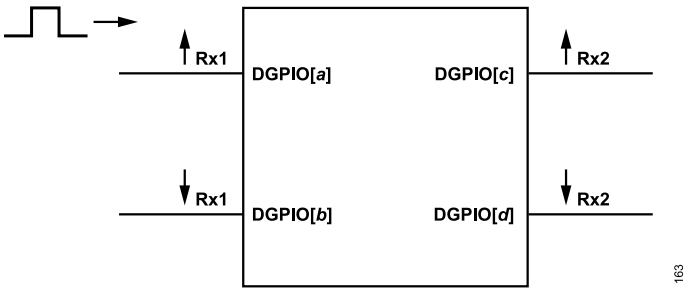


Figure 169. MGC Pin Mode: DGPIO (a to d) Represent Any of DGPIO[0:15]

In the MGC mode, to properly control the gain, optionally retrieve the status of peak detectors and the power detector in the device through a set of DGPIO pins (this can also be done in the AGC mode for observation). To make sure that the status information from the signal detectors is meaningful, it is important to first enable and configure the signal detectors properly, which is done through the API command. The feedback information is configured into two modes; the peak detect and peak/power detect modes. In the peak detect mode, the overrange and underrange conditions of both the APD and HB detectors are provided through the DGPIO pins. In the peak and power detect mode, the overrange and underrange conditions of the power detector and the overrange and underrange conditions of the APD are provided through the DGPIO pins.

Table 81 describes the feedback configuration and the bitfield definition and position. For example, when it is configured in peak mode, connect to a set of DGPIO pins to retrieve all the APD and HB detector status. Note that the DGPIO pins are selected from pin 0 to pin 15, and two

RECEIVER GAIN CONTROL

consecutive DGPIO pins should always be configured as a pair to retrieve two consecutive bitfields (Bit 0 and Bit 1 or Bit 2 and Bit 3 in both modes). The following enum type defines the DGPIO pin selection.

```
typedef enum adi_adrv9001_GpioPinCrumbSel
{
ADI__GPIO_PIN_CRUMB_UNASSIGNED,
ADI__GPIO_PIN_CRUMB_01_00,
ADI__GPIO_PIN_CRUMB_03_02,
ADI__GPIO_PIN_CRUMB_05_04,
ADI__GPIO_PIN_CRUMB_07_06,
ADI__GPIO_PIN_CRUMB_09_08,
ADI__GPIO_PIN_CRUMB_11_10,
ADI__GPIO_PIN_CRUMB_13_12,
ADI__GPIO_PIN_CRUMB_15_14,
} adi_adrv9001_GpioPinCrumbSel_e
```

In both the peak and peak/power modes, a pair of bits (Bit 0 and Bit 1 or Bit 2 and Bit 3) can choose any one pair of GPIO pins defined in "**adi_adrv9001_GpioPinCrumbSel_e**". If "**ADI_ADRV9001_GPIO_PIN_CRUMB_UNASSIGNED**" is selected, it means that no GPIO pins are assigned. So, the corresponding bitfields cannot be observed. The DGPIO pins are associated with either one of the receivers.

Table 81. DGPIO Configuration for Retrieving Signal Detector Information

Mode	Bit Field Definition	Feedback Mask Bit Position
Peak Mode	hb_low_threshold_counter_exceeded (low threshold (hbUnderRangeHighThresh) is exceeded counter times, no underload condition)	0
	apd_low_threshold_counter_exceeded (low threshold is exceeded counter times, no underload condition)	1
	hb_high_threshold_counter_exceeded (high threshold is exceeded counter times, overload condition)	2
	apd_high_threshold_counter_exceeded (high threshold is exceeded counter times, overload condition)	3
Peak and Power Mode	power_inner_low_threshold_exceeded (inner low threshold exceeded, no underload condition)	0
	power_inner_high_threshold_exceeded (inner high threshold exceeded, overload condition)	1
	apd_low_threshold_counter_exceeded (low threshold is exceeded counter times, no underload condition)	2
	apd_high_threshold_counter_exceeded (high threshold is exceeded counter times, overload condition)	3

As mentioned earlier, the status of signal detectors can also be retrieved in the AGC mode. Note that the signal detector status does not always correspond to the final AGC decision for gain increment or decrement, particularly due to the mitigation mode, in which the user can observe the APD high threshold exceeded, but no gain change happens as the secondary digital threshold helps to detect the false positive of the APD, which is caused by the near DC low frequency signal.

GAIN CONTROL DETECTORS

This section discusses three gain control detectors in more details.

Analog Peak Detector (APD)

The analog peak detector is located in the analog domain following the TIA filter and before the ADC input. It functions by comparing the signal level to programmable thresholds. When a threshold is exceeded a programmable number of times in a gain update period, the detector flags an overrange condition.

RECEIVER GAIN CONTROL

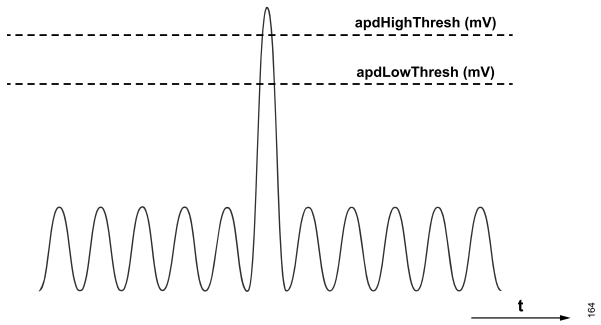


Figure 170. Analog Peak Detector Thresholds

There are two APD thresholds, as shown in Figure 170. These thresholds are contained in the agcPeak API structure, apdHighThresh, and apdLowThresh, respectively. The thresholds are typically considered relative to the full scale voltage of the ADC, which is 850 mV peak. The mV setting of the APD thresholds is determined using the following equations.

$$apdHighThresh\ (mV) = \frac{apdHighThresh\ \times 15\ (mV)}{0.91}$$

$$apdLowThresh\ (mV) = \frac{apdLowThresh\ \times 15\ (mV)}{0.91}$$

To determine the setting of the APD thresholds in terms of the closest possible setting in terms of dBFS of the ADC assuming apdHighdBFS and apdLowdBFS for apdHighThresh and apdLowThresh, respectively, the following equations can be used.

$$apdHighThresh = round\left(\frac{\left(10^{\left(\frac{apdHighdBFS}{20}\right)} \times 850\right) \times 0.91}{15}\right)$$

$$apdLowThresh = round\left(\frac{\left(10^{\left(\frac{apdLowdBFS}{20}\right)} \times 850\right) \times 0.91}{15}\right)$$

The APD threshold must be exceeded a programmable number of times within a gain update counter period before an overrange condition occurs. Both the upper and lower thresholds have a programmable counter in the API structure, as indicated in Table 82.

Table 82. APD Programmable Threshold Counters

Threshold	Counter
Upper Threshold (apdHighThresh)	apdUpperThreshPeakExceededCnt
Lower Threshold (apdLowThresh)	apdLowerThreshPeakExceededCnt

The Automatic Gain Control (AGC) section shows that the APD is used for both gain attack and gain recovery in the peak detect mode. In the peak/power detect mode, the APD can be used for gain attack and is used to prevent overloading during gain recovery. For more details, see the Automatic Gain Control (AGC) section.

In the AGC mode, the APD has programmable gain attack and gain recovery step sizes, as shown in Table 83.

Table 83. APD Attack and Recovery Step Sizes

Gain Change	Step Size
Gain Attack	apdGainStepAttack
Gain Recovery	apdGainStepRecovery

Step size refers to the number of indices in the gain table when the gain is changed. As mentioned earlier, the gain table is programmed with the largest gain in the Max Gain Index (typically index 255), with ever decreasing gain for decreasing gain index. Thus, if APD gain attack step size is programmed to six, then this means that the gain index is reduced by six when the apdHighThresh is exceeded more than apdUpperThreshPeakExceededCnt times. For example, if the gain index is 255 before this overrange condition, then the gain index reduces to 249. The amount of gain reduction this equates to is dependent on the gain table in use. The default table has 0.5 dB steps, which in this example equates to a 3 dB gain reduction upon an APD overrange condition.

The APD is held in reset for a configurable amount of time following a gain change to ensure the receiver path is settled at the new gain setting.

RECEIVER GAIN CONTROL

Half-Band Peak Detector

The HB peak detector is located in the digital domain at the output of the HB filtering block. It can, therefore, also be referred to as the decimated data overload detector because it works on decimated data. Like the APD detector, it functions by comparing the signal level to programmable thresholds. It monitors the signal level by observing individual samples (I2 + Q2 or peak I/peak Q) over a period of time and compares these samples to the threshold. If a sufficient number of samples exceed the threshold in the period of time, then the threshold is noted as exceeded by the detector. hbOverloadDurationCount controls the duration of the HB measurement, while hbOverloadThreshCount controls the number of samples that should exceed the threshold in that period.

Once the required number of samples exceeds the threshold for the duration required, the detector records that the threshold is exceeded. Like the APD detector, the HB detector requires a programmable number of times for the threshold to be exceeded in a gain update period before it flags an overrange condition.

Figure 171 shows the two-level approach, which is different from APD. It shows the gain update counter period, with the time broken into subsets of time based on the setting of hbOverloadDurationCount. Each period is considered separately, and hbOverloadThreshCount individual samples must exceed the threshold within hbOverloadDurationCount to declare an overload. These individual samples greater than the threshold are shown in gray. Two examples are shown; one where the number of samples exceeding the threshold is sufficient for the HB peak detector to declare an overload, and a second example where the number of samples exceeding the threshold is not sufficient to declare an overload. The number of overloads is counted, and if the number of overloads of the hbHighThresh exceeds the hbUpperThreshPeakExceededCount in a gain update counter period, an overrange condition is called. Likewise, if the number of overloads of the hbUnderRangeHighThresh does not exceed hbUnderRangeHighThreshExceededCount, an underrange condition is called. Note that if hbOverloadDurationCount is set to equal to the time duration of one sample and hbOverloadThreshCount is set to one, the HB two-level approach becomes similar to the APD algorithm.

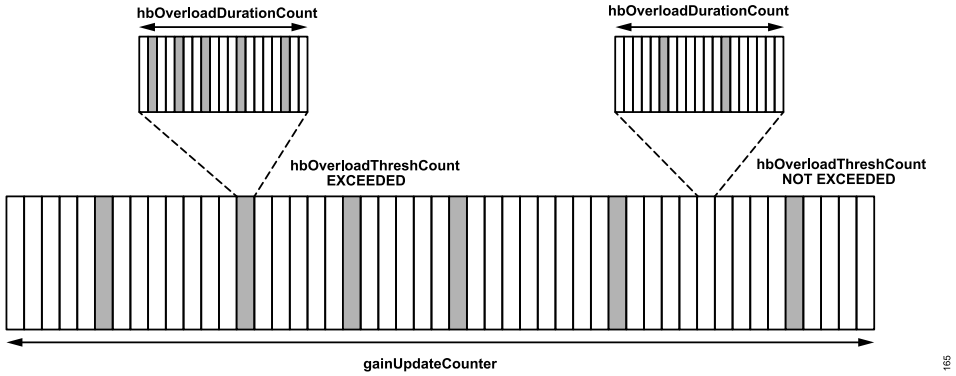


Figure 171. HB Detector, Two-Level Approach for an Overload Condition

The HB 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 84.

Table 84. HB Overload Thresholds

HB Threshold	Use
hbHighThresh	Gain attack in both peak and peak/power detect AGC modes.
hbUnderRangeHighThresh	Gain recovery in peak detect AGC mode. In the peak/power detect AGC mode, it prevents overloads during gain recovery.
hbUnderRangeMidThresh	Only when the fast recovery option of the peak detect AGC mode is being used.
hbUnderRangeLowThresh	Only when the fast recovery option of the peak detect AGC mode is being used.

For more details of how the AGC uses these thresholds, see the relevant sections in this document. The thresholds are related to an ADC dBFS value using the following equations.

$$hbHighThresh = 16384 \times 10^{\left(\frac{hbHighdBFS}{20}\right)}$$

$$hbUnderRangeHighThresh = 16384 \times 10^{\left(\frac{hbUnderRangeHighdBFS}{20}\right)} \quad (7)$$

RECEIVER GAIN CONTROL

Each threshold has an associated counter such that an overrange condition is not flagged until the threshold is exceeded this amount of times in a gain update period.

Table 85. Counters for HB Overrange and Underrange Conditions

HB Threshold	Counter
hbHighThresh	hbUpperThreshPeakExceededCount
hbUnderRangeHighThresh	hbUnderRangeThreshExceededCount
hbUnderRangeMidThresh	hbUnderRangeMidThreshExceededCount
hbUnderRangeLowThresh	hbUnderRangeLowThreshExceededCount

In the AGC mode, the HB peak detector has programmable gain attack and gain recovery step sizes.

HB Attack and Recovery Step Sizes

Gain Change	Step Size
Gain Attack	hbGainStepAttack
Gain Recovery (hbUnderRangeHighThresh)	hbGainStepHighRecovery
Gain Recovery (hbUnderRangeMidThresh)	hbGainStepMidRecovery
Gain Recovery (hbUnderRangeLowThresh)	hbGainStepLowRecovery

The HB peak detector is held in reset for a configurable amount of time following a gain change to ensure the receiver path is settled at the new gain setting.

Power Detector

The power measurement block measures the RMS power of the incoming signal at the output of the HB filtering block. The number of samples used in the power measurement calculation is configurable using the powerMeasurementDuration API parameter.

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

where:

receiver Sample Clocks is the number of clocks at the power measurement location.

It is important that this duration does 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 in a gain update period, the AGC uses the last fully completed power measurement, and any partial measurements are discarded.

Note that, currently, only inner thresholds of power detector are used. The inner thresholds defined in API are related to an ADC dBFS value using the following equations.

underRangeHighPowerThresh = -(underRangeHighPowerThresh _dBFS + 6)

overRangeLowPowerThresh = -(overRangeLowPowerThresh_dBFS + 6)

The power measurement block has a dynamic range of 60 dB. Signals lower than −60 dBFS cannot be measured accurately. The API function adi_adrv9001_Rx_DecimatedPower_Get() is used to read the the power measurement.

Detector Overload and QEC

If the device is operating in the AGC mode when an overload is detected, the Rx QEC calibration is frozen. The AGC can adjust the Rx Gain Index to bring the received signal to a level under the overload threshold so that the input signal no longer triggers the overload detector. Then, Rx QEC calibration resumes.

If the device is operating in the MGC mode when an overload is detected, the Rx QEC calibration is frozen as well. However, as the device is operating in the MGC mode, if the input signal remains at a high level above the overload threshold, then Rx QEC remains frozen and does not recover until the input signal is below the overload threshold.

RECEIVER GAIN CONTROL

AGC CLOCK AND GAIN BLOCK TIMING

The AGC clock drives the AGC state machine. In the ADRV9001 device, the default AGC clock (to support a set of standard sample rates) is at 184.32 MHz. When an arbitrary sample rate is adopted in the receiver, the AGC clock varies.

The AGC state machine contains three states: the gain update counter, followed by slow loop settling (SLS) delay, and 5 AGC clock cycles delay. The total time between the gain updates (gain update period) is a combination of slowLoopSettlingDelay and 5 AGC clock cycles. Note that the first slowLoopSettlingDelay in darker gray is a part of the gain update counter.

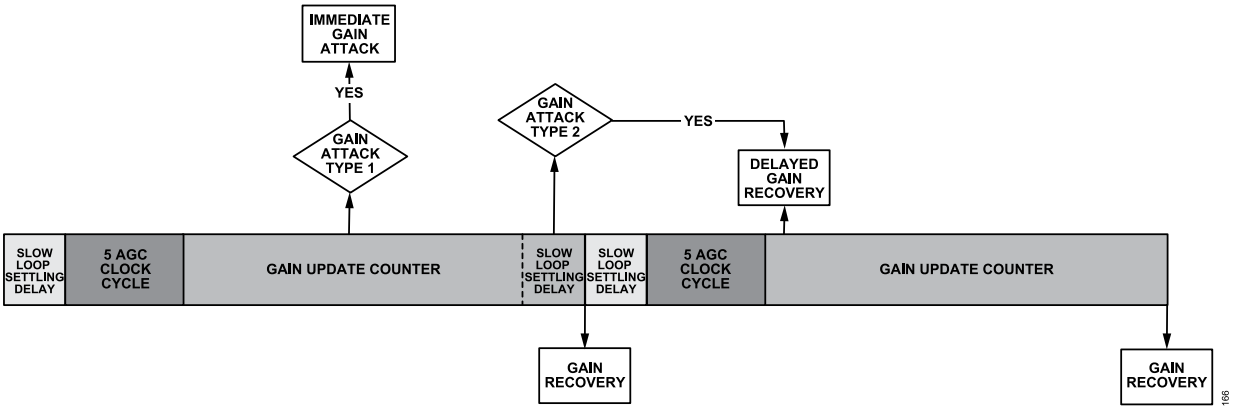


Figure 172. Delayed Gain Attack for Non-Delayed Gain Recovery

Figure 172 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:

- AGC gain attack within gain update counter, but more than an SLS delay before the gain update counter expiry: Because the SLS is typically several orders of magnitude smaller than the gain update counter, this is the most common gain decrement scenario. This type of AGC gain attack is called gain attack type 1, as shown in Figure 172.
- AGC gain attack within gain update counter, but within an SLS delay before the gain update counter expiry: This is a special case, which rarely occurs in applications. This type of AGC gain attack is called gain attack type 2, as shown in Figure 172.
- AGC gain recovery at the end of the gain update counter: Note that when the fast recovery is enabled, the gain update counter is substituted with the low underrange interval. The gain attack can occur within the gain update counter when the fast attack is enabled. A gain recovery event can only occur at the end of the gain update counter (or low underrange interval in the “fast recovery” mode), as previously discussed. This is mainly to align the gain recovery (for the desired signal) with the frame or subframe boundary. After a gain attack, a gain change counter with a value equal to the SLS delay is started. No further gain attacks are allowed while this counter is running. This allows to set the minimum time between gain changes.

However, the gain change counter also prevents the AGC from moving from the gain update counter state to the slow loop settling delay state as it must wait until the expiry of the SLS delay. Therefore, if a gain attack occurred very close to the end of the gain update counter state, the gain change counter delays the start of the SLS state and shifts the gain recovery event, as shown in Figure 173, whereas in Figure 172, the gain recovery event is always aligned with the end of the gain update counter period.

RECEIVER GAIN CONTROL

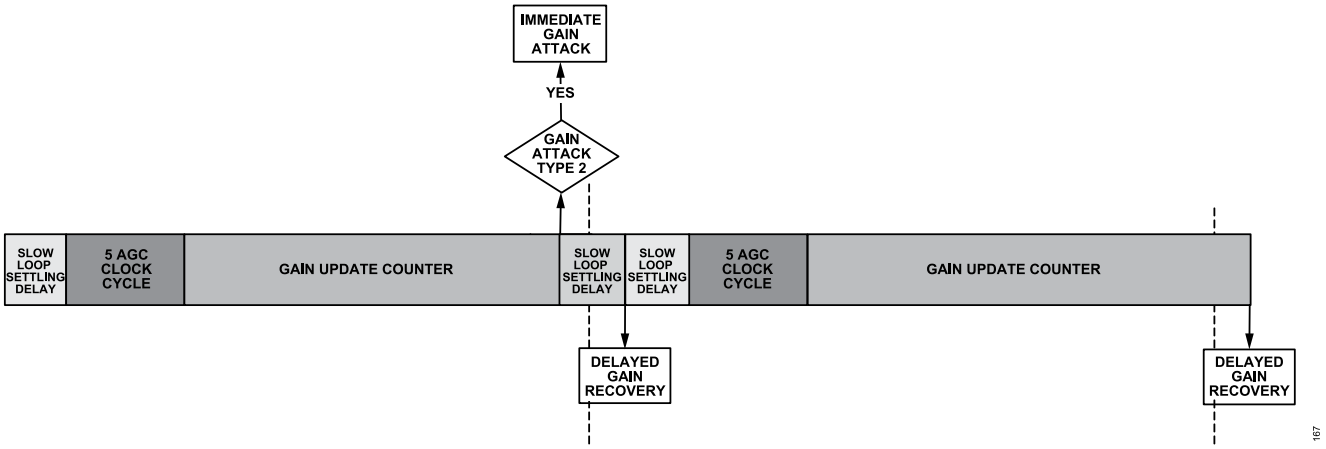


Figure 173. Immediate Gain Attack Causing Delayed Gain Recovery

To prevent this happening and to maintain a perfectly periodic gain recovery event, the gain attacks are prevented from happening towards the end of the gain update counter state, as shown in Figure 172. If a gain attack happens in this period, it is delayed until the start of the next gain update counter state. This can cause gain attacks to be held off for up to 2× SLS + 5 delay; therefore, it is recommended to keep the SLS delay as short as possible to minimize the gain attack delay. Note that it is possible to disable this blocking feature, thus allowing gain attacks to occur anywhere within the gain update counter state. However, the periodicity of the gain recovery event is no longer guaranteed as gain attacks towards the end of the gain update counter state causes the gain recovery event to be delayed, as shown in Figure 173.

At the expiry of the gain update counter (or low underrange interval in the “fast recovery” mode), all measurement blocks are reset, and any peak detector counts are reset back to zero. When the receiver is enabled, the counter begins. This might mean that its expiry is at an arbitrary phase to the slot boundaries of the signal. The expiry of the counter is aligned to the slot boundaries by setting the parameter enableSyncPulseForGainCounter. While this bit is set, the AGC monitors a DGPIO pin to find a synchronization pulse. This pulse causes the reset of the counter at this point; hence, if the user supplies at the DGPIO pulse time aligned to these slot boundaries, the expiry of the counter is aligned to the slot boundaries. Any of DGPIO pins 0 to 15 is used for this purpose. Note this feature is not supported currently.

For example, considering the 100 µs gain update period and a 184.32 MHz AGC clock, 18,432 AGC clocks exist in the gain update period.

Gain Update Period (*AGC Clocks*) = 184.32 MHz × 100 µs = 18,432

As noted, the full gain update period is the sum of the gainUpdateCounter, slowLoopSettlingDelay, and a number of AGC clock cycles. If the slowLoopSettlingDelay is set to 4, set the gain update counter to 18,423 from the following calculation.

Gain Update Period (*AGC Clocks*) = *gainUpdateCounter* + *slowLoopSettlingDelay* +5

Gain Update Period (*AGC Clocks*) = 18,423 + 4 + 5 = 18,432

When the receiver is enabled, attackDelay_us keeps the AGC inactive for a number of the AGC clock cycles. This means the user can specify one delay for AGC reaction when entering the receive mode and another after the gain change occurs (slowLoopSettlingDelay).

ANALOG GAIN CONTROL API PROGRAMMING

As mentioned earlier, the Rx gain control mode is configured as the MGC or AGC mode. In both the modes, the API function **adi_adrv9001_Rx_GainControl_Configure()** is used to configure the gain control blocks, such as the peak detectors and power detector for a specific channel. These detectors are used not only in the AGC mode but also in the MGC mode to feed important information. This API function also configures the DGPIO pins to retrieve the signal detectors information.

Note that although signal detector information is critical for the MGC mode, it can also be obtained in the AGC mode for observation and debugging.

The next section discusses the composition of the gain control configuration structure **adi_adrv9001_GainControlCfg_t** in detail. Once it is configured, the desired gain control mode can be enabled using **adi_adrv9001_Rx_GainControl_Mode_Set()** API.

RECEIVER GAIN CONTROL

If selecting the MGC mode, as discussed, manually control the gain through API commands or DGPIO pins. In the API command mode, select a gain index in the gain table through the API function **adi_ADRV9001_Rx_Gain_Set()**. The API function **adi_ADRV9001_Rx_Gain_Get()** can read back the gain index selected for a channel.

Figure 174 is a high level flowchart of Rx gain control programming. Note that the final step is to configure any GPIOs, as necessary, such as GPIO inputs, which directly control the gain index. Note that the API command adi_adrv9001_Rx_GainControl_Configure() includes the configuration of the DGPIO pins to retrieve signal detectors information. The operation of these is described earlier.

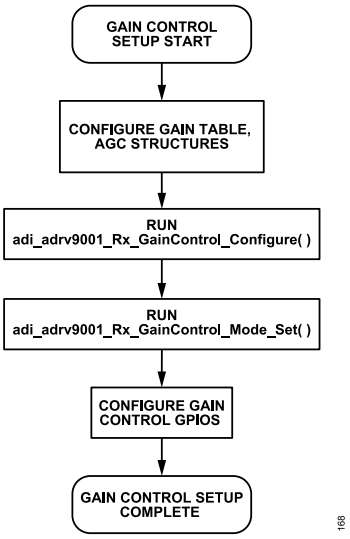


Figure 174. Gain Control Programming Flowchart

Gain Control Data Structures

Figure 175 shows a gain control programming flowchart with the member structure of adi_adrv9001_GainControlCfg_t and its substructures, adi_adrv9001_PeakDetector_t, adi_adrv9001_PowerDetector_t, and adi_adrv9001_ExtLna_t. Table 86 to Table 89 briefly explain each parameter. The previous relevant sections outline the wider context of these parameter settings.

RECEIVER GAIN CONTROL

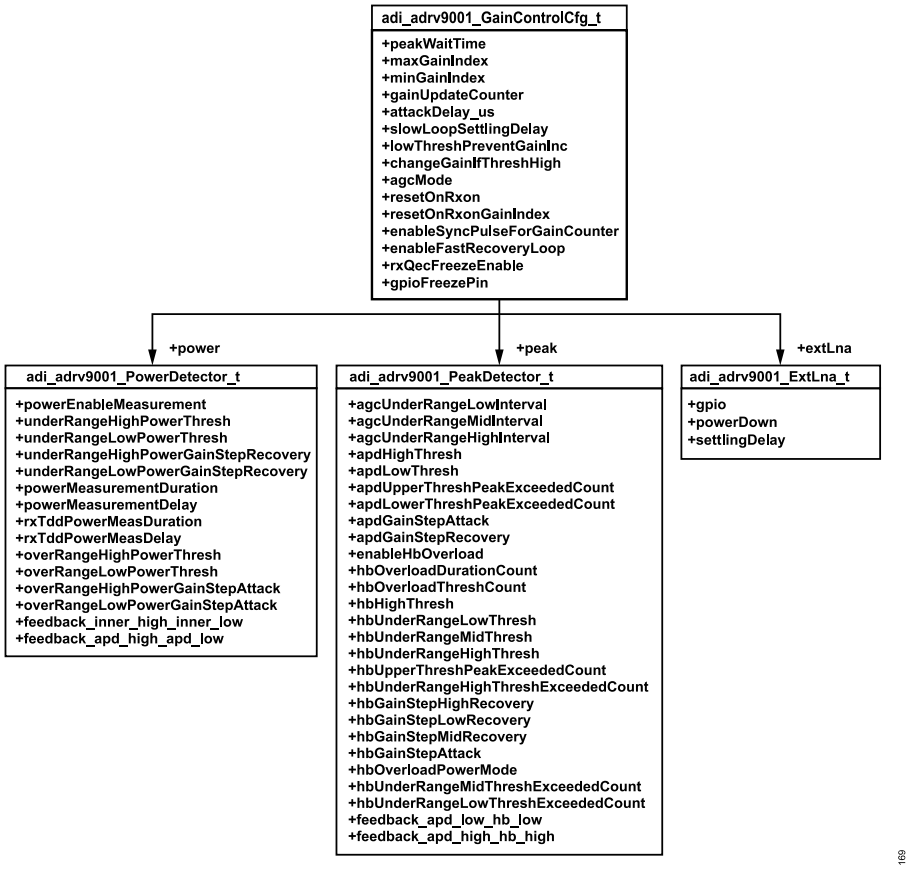


Figure 175. Member Listing of adi_adrv9001_GainControlCfg_t Data Structure

Table 86. adi_adrv9001_GainControlCfg_t Structure Definition

Parameter	Description	Min Value	Max Value	Default Value
peakWaitTime	Number of gain control clock cycles to wait before enabling peak detectors after a gain change.	0	31	4
maxGainIndex	Maximum gain index allowed. Must be greater than minGainIndex and be a valid gain index.	187	255	255
minGainIndex	Minimum gain index allowed. Must be less than maxGainIndex and be a valid gain index.	187	255	187
gainUpdateCounter	Used as a decision period, with the detectors reset on this period. Gain changes in the AGC mode can also be synchronized to this period (the expiry of this counter). The full period is a combination of the gainUpdateCounter and slowLoopSettlingDelay and a number of AGC cycles.	Depends on overload detector settings	4194303 AGC CLK cycles	11520
attackDelay_us	The duration the AGC is held in reset when the Rx path is enabled.	0	63	10
slowLoopSettlingDelay	Number of AGC clock cycles to wait after a gain change before the AGC changes gain again.	0	127	16
lowThreshPreventGainInc	Only relevant in peak and power detect AGC operation.1: If AGC is in peak and power detect mode, then gain increments requested by the power detector are prevented if there are sufficient peaks (APD/HB low threshold exceeded count) above the apdLowThresh or hbUnderRangeHighThresh.0: apdLowThresh and hbUnderRangeHighThresh do not care for gain recovery.	0	1	0
changeGainIfThreshHigh	Applicable in both the peak and peak and power detect modes.0: Gain changes wait for the expiry of the gain update counter if a high threshold	0	3	3

RECEIVER GAIN CONTROL

Table 86. adi_adrv9001_GainControlCfg_t Structure Definition (Continued)

Parameter	Description	Min Value	Max Value	Default Value
	count is exceeded on either the APD or HB detector.1: Gain changes occur immediately when initiated by HB. Gain changes initiated by the APD wait for the gain update to expire.2: Gain changes occur immediately when initiated by APD. Gain changes initiated by HB wait for the gain update to expire.3: Gain changes occur immediately when initiated by APD or HB detectors.			
agcMode	1: AGC in peak AGC mode, power-based gain changes are disabled.0: AGC in peak and power AGC mode where both peak and power detectors are used.	0	1	1
resetOnRxon	1: AGC state machine is reset when Rx is disabled. The AGC gain setting uses the "resetOnRxonGainIndex"after resuming the operation.0: AGC state machine maintains its state when Rx is disabled and the last AGC gain index is used after resuming the operation.	0	1	0
resetOnRxonGainIndex	The AGC index to start with when "resetOnRxon" is set as 1.	187	255	255
enableSyncPulseForGainCounter	1: Allows synchronization of AGC gain update counter to the time-slot boundary. GPIO setup required.0: AGC gain update counter free runs.	0	1	0
enableFastRecoveryLoop	1: Enables the fast recovery AGC functionality using the HB overload detector. Only applicable in peak detect mode. 0: AGC fast recovery is not enabled.	0	1	0
power	Structure containing all the power detector settings.	N/A	N/A	N/A
peak	Structure containing all the peak detector settings.	N/A	N/A	N/A
extLna	Structure containing all external LNA settings.	N/A	N/A	N/A
rxQECFreezeEnable	RXQEC freeze enable/disable	0	1	1
gpioFreezePin	GPIO pin to activate to freeze AGC	0 (not assigned)	16	0 (not assigned)

Table 87. adi_adrv9001_PowerDetector_t Structure Definition

Parameter	Description	Min Value	Max Value	Default Value
powerEnableMeasurement	1: Power measurement block enabled.0: Power measurement block disabled.	0	1	1
underRangeHighPowerThresh	Threshold (negative sign assumed) that defines the lower boundary on the stable region of the power detect gain control mode.	0	127	10
underRangeLowPowerThresh	Offset (negative sign assumed) from underRangeHighPowerThresh that defines the outer boundary of the power based AGC convergence. Typically, recovery is set to larger steps than when the power measurement is less than this threshold.	0	15	4
underRangeHighPowerGainStep Recovery	The number of indicesthe gain index pointer is increased (gain increase) in the event of the power measurement being less than underRangeHighPower-Thresh but greater than underRangeLowPowerThresh.	0	31	2
underRangeLowPowerGainStepRecovery	The number of indices the gain index pointer is increased (gain increase) in the event of the power measurement being less than underRangeLowPowerThresh.	0	31	4
powerMeasurementDuration	Number of IQ samples on which to perform the power measurement. The number of samples corresponding to the 4-bit word is 8 × 2^(pmdMeasDuration[3:0]). This value must be less than AGC gain update counter.	0	31	10
powerMeasurementDelay	Measurement delay to detect power for specific slice of gain update counter	0	255 AGC clock cycles	2
rxTddPowerMeasDuration	Following an Rx Enable, the power measurement block is requested to perform a power measurement for a specific period of a frame. This is	0	65535 AGC clock cycles	0

RECEIVER GAIN CONTROL

Table 87. adi_adrv9001_PowerDetector_t Structure Definition (Continued)

Parameter	Description	Min Value	Max Value	Default Value
	applicable in TDD modes. This parameter sets the duration of this power measurement. A value of 0 causes the power measurement to run until the next gain update counter expiry.			
rxTddPowerMeasDelay	Following an Rx Enable, the power measurement block is requested to perform a power measurement for a specific period of a frame. This is applicable in TDD modes. This parameter sets the delay between the Rx Enable and the power measurement starting on Rx.	0	65535 AGC clock cycles	0
overRangeHighPowerThresh	Offset (positive sign assumed) from threshold overRangeLowPowerThresh that defines the outer boundary on the stable region of the power detect gain control mode. Typically, attack is set to larger steps than when the power measurement is greater than this threshold.	0	15	0
overRangeLowPowerThresh	Threshold (negative sign assumed) that defines the upper boundary on the stable region of the power detect gain control mode.	0	127	7
overRangeHighPowerGainStepAttack	The number of indices the gain index pointer is decreased (gain reduction) in the event of the power measurement being greater than overRangeHighPowerThresh.	0	31	4
overRangeLowPowerGainStepAttack	The number of indices that the gain index pointer should be decreased (gain decrease) in the event of the power measurement being less than OverRangeHighPowerThresh but greater than OverRangeLowPowerThresh.	0	31	4
feedback_inner_high_inner_low	A pair of DGPIO pins to retrieve power detector inner low threshold exceeded status and inner high threshold exceeded status.	0 (not assigned)	9 (select DGPIO pins 14 and 15)	0 (not assigned)
feedback_apd_high_apd_low	A pair of DGPIO pins to retrieve the APD detector low threshold counter exceeded status and APD high threshold counter exceeded status.	0 (not assigned)	9 (select DGPIO pins 14 and 15)	0 (not assigned)

Table 88. adi_adrv9001_PeakDetector_t Structure Definition

Parameter	Description	Min Value	Max Value	Default Value
agcUnderRangeLowInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hbUnderRangeLowThresh. Only applicable when the fast recovery option is enabled in the peak detect AGC mode.	Depends on HB detector settings	65535	50
agcUnderRangeMidInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hbUnder-RangeMidThresh. Calculated as (underRangeMidInterval + 1) ×underRangeLowInterval. Only applicable when the fast recovery option is enabled in the peak detect AGC mode.	0	63	2
agcUnderRangeHighInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hbUnderRangeHighThresh. Calculated as (underRangeHighInterval + 1) *underRangeMidInterval. Only applicable when the fast recovery option is enabled in the peak detect AGC mode.	0	63	4
apdHighThresh	This sets the upper threshold of the analog peak detector. When the input signal exceeds this threshold a programmable number of times (set by its corresponding overload counter) within a gain update period, the overload detector flags. In AGC modes, the gain is reduced when this overload occurs.	apdLowThresh	63	21
apdLowThresh	This sets the lower threshold of the analog peak detector. When the input signal exceeds this threshold a programmable number of times (set by its corresponding overload counter) within a gain update period, the overload detector flags. In the peak AGC mode,the gain is increased when this overload is	0	apdHighThresh	12

RECEIVER GAIN CONTROL

Table 88. adi_adrv9001_PeakDetector_t Structure Definition (Continued)

Parameter	Description	Min Value	Max Value	Default Value
	not occurring. In the power AGC mode, this threshold prevents further gain increases if the lowThreshPreventGainInc bit is set.			
apdUpperThreshPeakExceededCount	Sets number of peaks to detect above apdHighThresh to cause an APD high overrange event. In AGC modes, this results in a gain decrement set by apdGainStepAttack.	0	255	6
apdLowerThreshPeakExceededCount	Sets number of peaks to detect above apdLowThresh to cause an APD low overload event. In the peak detect AGC mode, if an APD low overload event is not occurring, then this results in a gain increment set by apdGainStepRecovery.	0	255	3
apdGainStepAttack	The number of indices the gain index pointer is decreased in the event of an APD high overrange in AGC modes. The step size in dB depends on the gain step resolution of the gain table (default 0.5 dB per index step).	0	31	2
apdGainStepRecovery	The number of indices the gain index pointer is increased in the event of an APD underrange event occurring in the peak detect AGC mode. The step size in dB depends on the gain step resolution of the gain table (default 0.5 dB per index step).	0	31	0
enableHbOverload	1: HB overload detector enabled 0: HB overload detector disabled	0	1	1
hbOverloadDurationCount	The number of clock cycles (at the HB output rate) within which hbOverloadThreshCount must be exceeded for an overload to occur. An HB overload flag is only raised when the number of these overloads exceeds hbUpperThreshPeakExceededCnt or hbLowerThreshPeakExceededCnt within a gain update period.	0	7	1
hbOverloadThreshCount	Sets the number of individual samples exceeding hbHighThresh or hbLowThresh necessary within hbOverloadDurationCount for an overload to occur. The HB overload flag is only raised when the number of these overloads exceeds hbUpperThreshPeakExceededCnt or hbLowerThreshPeakExceededCnt within a gain update period.	1	15	1
hbHighThresh	This sets the upper threshold of the HB detector.	0	16383	13044
hbUnderRangeLowThresh	This sets the lower threshold of the HB underrange threshold detectors. Used only when the fast recovery option of the peak detect AGC mode is being used.	0	16383	5826
hbUnderRangeMidThresh	This sets the middle threshold of the HB underrange threshold detectors. Used only when the fast recovery option of the peak detect AGC mode is being used.	0	16383	8230
hbUnderRangeHighThresh	Peak detect mode: Threshold used for gain recovery. Peak detect with fast recovery mode: This sets the highest threshold of the HB underrange threshold detectors. Power detect mode: Threshold used to prevent further gain increases if lowThreshPreventGainInc is set.	0	16383	7335
hbUpperThreshPeakExceededCount	Sets number of individual overloads above hbHighThresh (number of times hbOverloadThreshCount is exceeded inhbOverloadDuractionCount) to cause an HB High overrange event. In AGC modes, this results in a gain decrement set by hbGainStepAttack.	0	255	6
hbUnderRangeHighThreshExceededCount	Sets number of individual overloads above hbUnderRangeHighThresh (number of times hbOverloadThreshCount is exceeded in hbOverloadDurationCount) to cause an HB underrange high threshold overload event. In the peak detect AGC mode, not having sufficient peaks to cause the overload is flagged	0	255	3

RECEIVER GAIN CONTROL

Table 88. adi_adrv9001_PeakDetector_t Structure Definition (Continued)

Parameter	Description	Min Value	Max Value	Default Value
	as an underrange event and the gain is recovered by hbGainStepHighRecovery.			
hbGainStepHighRecovery	The number of indices the gain index pointer is increased in the event of an HB underrange high threshold underrange event.	0	31	2
hbGainStepLowRecovery	Only applicable in the fast recovery mode of peak detect AGC. This sets the number of indices the gain index pointer is increased in the event of an HB underrange low threshold underrange Event.	0	31	6
hbGainStepMidRecovery	Only applicable in the fast recovery mode of peak detect AGC. This sets the number of indices the gain index pointer is increased in the event of an HB underrange mid threshold underrange Event.	0	31	4
hbGainStepAttack	The number of indices the gain index pointer is decreased in the event of an HB high threshold overrange event in AGC modes. The step size in dB depends on the gain step resolution of the gain table (default 0.5dB per index step).	0	31	2
hbOverloadPowerMode	Sets the measurement mode of the HB detector. HB uses I^2 + Q^2 when set to 1.Otherwise the HB uses max(I, Q) per sample.	0	1	0
hbUnderRangeMidThreshExceededCount	Only applicable in the fast recovery mode of peak detect AGC. Sets number of individual overloads above hbUnderRangeMidThresh (number of times hbOverloadThreshCount is exceeded in hbOverloadDurationCount) to cause an HB underrange mid threshold overload event. In the peak detect AGC mode, not having sufficient peaks to cause the overload is flagged as an underrange event and the gain is recovered by hbGainStepMidRecovery.	0	255	3
hbUnderRangeLowThreshExceededCount	Only applicable in the fast recovery mode of peak detect AGC. Sets number of individual overloads above hbUnderRangeLow-Thresh (number of times hbOverloadThreshCount is exceeded in hbOverloadDurationCount) to cause an HB underrange low threshold overload event. In the peak detect AGC mode, not having sufficient peaks to cause the overload is flagged as an underrange event and the gain is recovered by hbGainSte-pLowRecovery.	0	255	3
feedback_apd_low_hb_low	A pair of DGPIO pins to retrieve the HB low threshold counter exceeded status and APD low threshold counter exceeded status.	0 (not as-signed)	9 (select DGPIO pins 14 and 15)	0 (not assigned)
feedback_apd_high_hb_high	A pair of DGPIO pins to retrieve the HB high threshold counter exceeded status and APD high threshold counter exceeded status.	0 (not assigned)	9 (select DGPIO pins 14 and 15)	0 (not assigned)

Table 89. adi_adrv9001_ExtLna_t Structure Definition

Parameter	Description	Min Value	Max Value	Default Value
settlingDelay	External LNA settling delay	0	255	0

There is a set of receiver gain control APIs to interact with the ADRV9001 device. The previous sections mention some of them. Table 90 summarizes the list of API functions with a brief description for each one. Refer to the Doxygen document for more up-to-date information and detailed descriptions.

RECEIVER GAIN CONTROL

Table 90. A List of Rx Gain Control APIs

Rx Gain API Function Name	Description
adi_adrv9001_Rx_GainControl_Mode_Set	Configures the Rx gain control mode for a specific channel.
adi_adrv9001_Rx_GainControl_Mode_Get	Retrieves the currently configured Rx gain control mode.
adi_adrv9001_Rx_Gain_Get	Reads the Rx gain index for the requested Rx channel.
adi_adrv9001_Rx_Gain_Set	Sets the current AGC gain index for the requested Rx channel.
adi_adrv9001_Rx_GainTable_Write	Programs the gain table settings for Rx channels.
adi_adrv9001_Rx_GainTable_Read	Reads the gain table entries for requested Rx channels.
adi_adrv9001_Rx_DecimatedPower_Get	Gets the decimated power for the specified channel.
adi_adrv9001_Rx_GainControl_Configure	Sets up the device Rx Gain Control for a specified channel
adi_adrv9001_Rx_GainControl_Inspect	Inspects the device Rx gain control for a specified channel.
adi_adrv9001_Rx_GainControl_MinMaxGainIndex_Set	Sets the min/max gain indexes for gain control operation for the specified channel.
adi_adrv9001_Rx_GainControl_MinMaxGainIndex_Get	Gets the min/max gain indexes for gain control for the specified channel.
adi_adrv9001_Rx_GainControl_Reset	Resets all state machines within the gain control block.
adi_adrv9001_Rx_GainControl_PinMode_Configure	Configures gain control for the MGC PIN mode.
adi_adrv9001_Rx_GainControl_PinMode_Inspect	Inspects gain control configurations for the MGC PIN mode.

DIGITAL GAIN CONTROL AND INTERFACE GAIN (SLICER)

The digital gain control has two major purposes: gain correction to correct the small step size inaccuracy in the analog front-end attenuation, and gain compensation to compensate for the entire analog front-end attenuation. In the gain compensation mode, for example, if 5 dB analog attenuation is applied at the front end of the device, then 5 dB of digital gain is applied. This ensures the digital data is representative of the RMS power of the signal at the receiver input port (plus the nominal receiver analog gain) so that any internal front-end attenuation changes in the device to prevent ADC overloading are transparent to the baseband processor. In this way, the device's AGC is used to react quickly to incoming blockers without the baseband processor needing to track the current gain index for the level of the received signal at the input to the device for signal strength measurements.

The receiver gain table controls the digital gain block, as mentioned earlier. Note that different digital gain is applied when configured in the gain correction or gain compensation mode. The receiver gain table has a unique front-end attenuator setting with a corresponding amount of digital gain, programmed at each index of the table, as shown in Table 71.

For the gain compensation mode, it is used in either the AGC or MGC mode. The digital gain compensates for both the internal analog attenuator and an external gain component (such as a DSA or LNA). After the digital gain compensation, the signal power should only depend on the input signal power.

Around the end of the receiver datapath, the receiver interface gain is further applied using a "Slicer" block for two major purposes: to avoid digital saturation due to the bit-width limitation of the data port in the gain compensation mode, and to ensure the overall SNR is limited only by analog noise and is unaffected by quantization noise. When the gain compensation mode is used, any analog attenuation is compensated by a corresponding digital gain, such that the sum of the analog and digital gain is always equal to the nominal receiver analog gain of 20 dB. At the ADC input, the full scale input signal is approximately 8.6 dBm. This value translates to 0 dBFS in the digital datapath for either the I or Q channel. As an example, assuming a 5 dBm signal is applied at the receiver input port, at the receiver output, the signal power is 5 + 20 = 25 dBm or 25 − 8.6 = 16.4 dBFS. This causes the clipping at the 16-bit output signal. Therefore, the interface gain (less than 0 in this case) is applied to attenuate the signal to avoid clipping. On the other hand, for a very low signal level, at the receiver input, within the RF bandwidth of interest, it must be assured that the analog noise dominates the quantization noise. In the receiver datapath, there are different modes of receiver data interface , which are classified into two major categories, one using a final 15-bit or 16-bit quantizer to round the 22-bit receiver data to 15-bit or 16-bit, and the other passing the 22-bit receiver data entirely without using any quantizer. In the first mode, the quantizer becomes the dominant noise source as a result of the final interface quantization. This quantization noise, as a result of the final 15-bit or 16-bit quantizer, is spread over a bandwidth equivalent to its output sampling frequency. For NB applications where the output sampling frequency is low, the total quantization noise per Hz is larger than the analog noise per Hz. By applying interface gain (greater than 0 in this case), prior to the final quantizer, the signal level and analog noise level are both increased. Therefore, the analog noise dominates over the quantization noise so that SNR is dominated by analog front-end noise in the RF bandwidth of interest. For WB applications, as the sampling frequency is higher, the total quantization noise becomes much smaller. In such a case, the analog noise is way above the quantization noise. Therefore, interface gain is not required. In the second mode, it passes the full 22-bit I/Q data from the receiver data path to the interface without rounding. Therefore, it lowers the quantization noise without the need for additional interface gain. It uses 32 bit I data and 32 bit Q data on the interface for CMOS 1-lane (64-bit) and LVDS 2-lane (32-bit I data and 32-bit Q data). Besides including the 22-bit I/Q data, the 32-bit data also includes

RECEIVER GAIN CONTROL

some extra fields, which are 1 bit of slicer gain or AGC gain change indicator and one of the following: zeros, interface gain, or AGC gain index. For more details, see the Data Interface section.

Figure 176 is a block diagram of the digital gain control portion of the receiver chain, showing the locations of the various blocks in the simplified datapath.

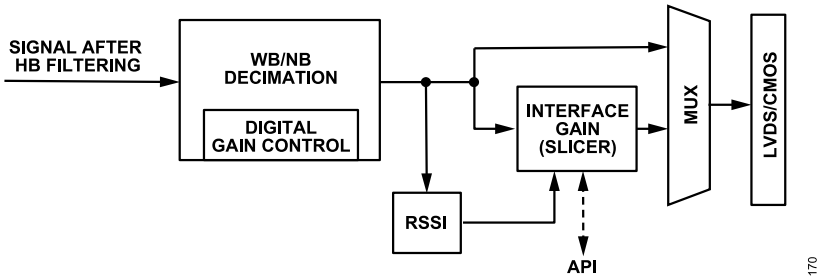


Figure 176. Gain Control and Slicer Section of the Receiver Datapath

Figure 176 shows that digital gain control is performed in the WB/NB decimation block. In NB and WB applications, the digital gain control is actually performed at different stages of the receiver data chain to achieve optimal performance, which is simplified in Figure 176. The slicer must depend on the desired signal power alone and must be done only when all the interfering signals are filtered out, for example, close to the end of the datapath. The slicer operation can either be controlled automatically by the device internally or externally through API commands. When controlled internally, the RSSI block is used to determine the amount of interface gain. Note that there is also a fast attack algorithm for automatic interface gain control algorithm. This is mainly to solve the signal saturation problem at the beginning when the Rx output signal level is increased significantly in NB applications (such as in a TDD operation). As previously discussed, when the Rx output signal is very low, a high interface gain is applied and when the Rx output signal level becomes very high, the interface gain must lower quickly to avoid saturation. However, it takes some time to react to the signal level change in a normal operation mode. Therefore, the fast attack algorithm is designed to solve this issue. Provide the maximum signal level target and the signal peak to average power ratio (PAPR) to help the automatic interface gain control algorithm to produce the proper interface gain values.

The following sections describe four different digital gain control modes in the device.

Mode 1: No Digital Gain Compensation with Internal Interface Gain Control

In this mode, the digital gain block is used for gain correction. It applies a small amount 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 steps in dB across its entire range, the digital gain block is used to even out the steps for consistency (the default table uses the digital gain block to provide consistent 0.5 dB steps).

With internal control, the device automatically applies the interface gain determined by RSSI, which measures the input signal power right before the slicer. Note that in the gain correction mode, interface gain less than 0 is not needed as the receiver output level should not exceed 0 dBFS through either AGC or MGC. When in NB applications, the interface gain range can be from 0 dB to 18 dB in 6 dB step size (0, 6, 12, and 18) to improve the sensitivity when a quantizer is used. In WB applications, as discussed earlier, the sensitivity is already satisfied by the high sampling rate. So, the interface gain is always 0.

After applying the interface gain, the signal is provided to the data port. The baseband processor can retrieve the interface gain through API commands 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 included as a part of the digital gain compensation).

Mode 2: No Digital Gain Compensation with External Interface Gain Control

This mode is similar to mode 1, except the interface gain is controlled manually. Similarly, when in NB applications, the interface gain range is selected from 0 dB to 18 dB in 6 dB step size when a quantizer is used, while in WB applications, the interface gain is fixed at 0 dB.

Mode 3: Digital Gain Compensation with Internal Interface Gain Control

The gain compensation is used in this mode, and the interface gain is determined internally. The device is loaded with the gain tables that apply compensation for the analog front-end attenuation. Thus, as the analog front-end attenuation is increased, equal amount of digital gain is

RECEIVER GAIN CONTROL

applied. The interface gain is determined by RSSI. If the power level is too high, the slicer shifts the signal properly before sending it to the data port to avoid saturation.

Let us look at a slicer example that considers three different input signal power levels. The power level 1 fits a data length of 16 bit-width. Power level 2 is 0 dB to 6 dB higher than power level 1, which increases the bit-width by 1. Power level 3 is 6 dB to 12 dB higher than power level 1, which further increases bit-width by 1. Figure 177 outlines this effect, with gray boxes indicating the valid (used) bits in each case.

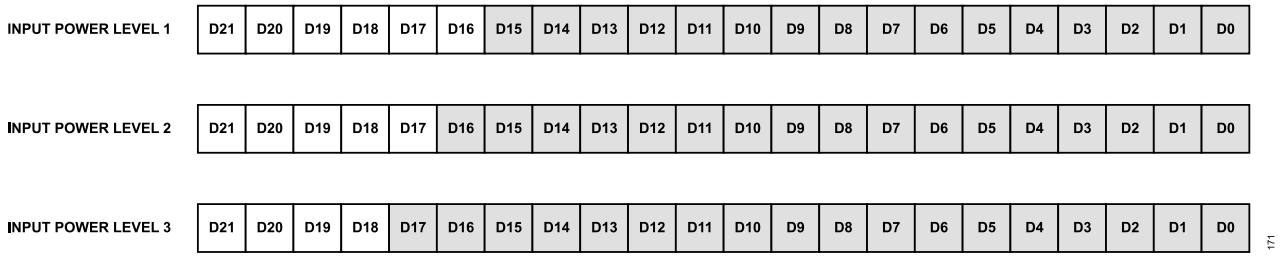


Figure 177. Bit Width of Input Signal with Increasing Power Levels

The slicer is used to attenuate the data such that it can fit into the resolution of the data port. As the output is a shifted version of the input, the slicer can only handle gains in ±6 dB steps.

Figure 178 explains the slicer operation. For power level 1, the slicer shift value is calculated as 0, so the 16-bit output data is taken from D15 – D0. As the power level increases, the bit-width of the signal increases. For power level 2, now the bit-width is 17. The slicer shift value becomes 1, so the 16-bit output data is taken from D16 – D1. This is equivalent to applying 6 dB of attenuation by a slicer, which ensures that the bit-width of the signal is 16 once more; i.e., the 16 MSBs are selected (sliced) with the LSB dropped. When the power level further increases, as power level 2, the signal bit-width becomes 18-bit. The slicer shift value becomes 2, so the 16-bit output data is taken from D17 – D2, which is equivalent to applying 12 dB of attenuation by slicer, or slice the 16 MSBs dropping the 2 LSBs.

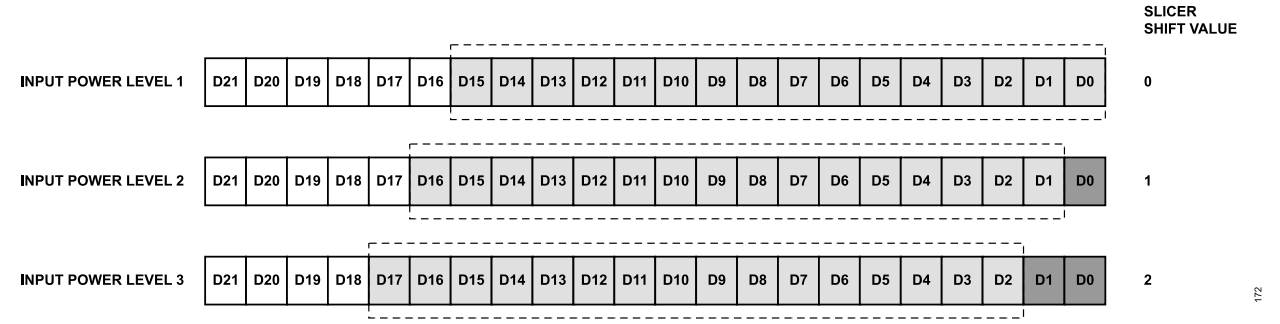


Figure 178. Slicer Bit Selection with Different Input Power Levels

The slicer algorithm assumes a max PAR of 15 dB, and it adjusts the interface gain such that the measured signal power +15 dB is less than a predefined threshold such as 0 dBFS. For NB applications, the interface gain is from −36 dB to +18 dB, and for WB applications, the interface gain is from −36 dB to 0 dB in 6 dB step size.

Similarly, the baseband processor retrieves the interface gain through API commands 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 included as a part of the digital gain compensation).

Mode 4: Digital Gain Compensation with External Interface Gain Control

This mode is similar to mode 3, except the interface gain is controlled by selecting a proper value. The baseband processor measures the input signal power or uses the power measurement done by RSSI in the device to determine the interface gain. Then, through API commands, the slicer operates in the same way as mentioned in mode 3. For NB applications, the interface gain is from −36 dB to +18 dB, and for WB applications, the interface gain is from −36 dB to 0 dB in 6 dB step size. Use this mode especially when the baseband processor input signal clipping is observed.

RECEIVER GAIN CONTROL

DIGITAL GAIN CONTROL AND INTERFACE GAIN API PROGRAMMING

The API function **adi_adrv9001_Rx_InterfaceGain_Configure()** configures the interface gain. The configuration structure **adrv9001_RxInterfaceGainCtrl_t** is defined as the following:

```
typedef struct adi_adrv9001_RxInterfaceGainCtrl
{
adi_adrv9001_RxInterfaceGainUpdateTiming_e updateInstance; /* Time at which receiver interface gain
control must be updated. 0: To be updated at start of next frame 1: To be updated immediately */
adi_adrv9001_RxInterfaceGainCtrlMode_e controlMode; /* 0: Uses internal receiver interface gain value
1: Uses external Rx interface gain value. Gain value must be provided in this case. */
adi_adrv9001_RxGainTableType_e gainTableType; /* 0: Gain Correction table 1: Gain Compensation table
adi_adrv9001_RxInterfaceGain_e gain; /* a value between 0 and 9 (0x0 = 18 dB, 0x1 = 12 dB, 0x2 = 6dB,
0x3 = 0 dB, 0x4 = -6 dB, 0x5 = -12 dB, 0x6 = -18 dB, 0x7 = -24 dB, 0x8 = -30dB, 0x9 = -36dB). */
} adrv9001_RxInterfaceGainCtrl_t
```

It is clear from the above structure that there are two interface gain control modes: internal (automatic) and external control. In addition, there are two options to apply the interface gain. The first option is to apply it at the start of the next frame and the second option is to apply it immediately. The interface gain is selected from −36 dB to +18 dB in 6 dB step size, a total of 10 options, which is defined by "adi_adrv9001_RxInterfaceGain_e" as the following:

```
typedef enum adi_adrv9001_RxInterfaceGain
{
ADI__RX_INTERFACE_GAIN_18_DB = 0, /* 18 dB */
ADI__RX_INTERFACE_GAIN_12_DB, /* 12 dB */
ADI__RX_INTERFACE_GAIN_6_DB, /* 6 dB */
ADI__RX_INTERFACE_GAIN_0_DB, /* 0 dB */
ADI__RX_INTERFACE_GAIN_NEGATIVE_6_DB, /* -6 dB */
ADI__RX_INTERFACE_GAIN_NEGATIVE_12_DB, /* -12 dB */
ADI__RX_INTERFACE_GAIN_NEGATIVE_18_DB, /* -18 dB */
ADI__RX_INTERFACE_GAIN_NEGATIVE_24_DB, /* -24 dB */
ADI__RX_INTERFACE_GAIN_NEGATIVE_30_DB, /* -30 dB */
ADI__RX_INTERFACE_GAIN_NEGATIVE_36_DB, /* -36 dB */
} adi_adrv9001_RxInterfaceGain_e
```

Note: As discussed before, depending on the gain table type and the bandwidth of the application, the interface gain can be limited to a subset of the 10 options. Call this API in the "calibrated" state.

In the external mode, to change the interface gain on-the-fly while the channels are operational, use the API function **adi_adrv9001_Rx_InterfaceGain_Set()**. Select the gain from one of the 10 options.

In the internal (automatic) mode, the ADRV9001 allows to notify an interface gain it wants to seed using the API **adi_adrv9001_Rx_InterfaceGain_SeedGain_Set()**. Then, by using the rising edge of an assigned GPIO pin, the interface gain is applied to the Rx output signal. After that, the automatic interface gain algorithm continues. Inspect the seeded interface gain using the API **adi_adrv9001_Rx_InterfaceGain_SeedGain_Get()**. Also use the falling edge of the GPIO pin to retrieve the latest interface gain at the end of the frame.

The following table summarizes the list of API functions currently available for digital gain control and interface gain. For more up-to-date information and detailed descriptions, refer to the API Doxygen document.

Table 91. A List of Rx Interface Gain Control APIs

Rx Gain API Function Name	Description
adi_adrv9001_Rx_Rssi_Read	Reads back the RSS status for the given channel.
adi_adrv9001_Rx_InterfaceGain_Configure	Sets the Rx interface gain control configuration parameters for the given Rx channel.
adi_adrv9001_Rx_InterfaceGain_Set	Sets the Rx interface gain for the given Rx channel.
adi_adrv9001_Rx_InterfaceGain_Inspect	Gets the Rx interface gain control configuration parameters for the given Rx channel.
adi_adrv9001_Rx_InterfaceGain_Get	Gets the Rx interface gain for the given Rx channel.

RECEIVER GAIN CONTROL

Table 91. A List of Rx Interface Gain Control APIs (Continued)

Rx Gain API Function Name	Description
adi_adrv9001_Rx_InterfaceGain_SeedGain_Set	Sets the seed for the Rx interface gain. seedGain is applied by the rising edge on associated GPIO.
adi_adrv9001_Rx_InterfaceGain_SeedGain_Get	Gets the seed for the Rx interface gain. seedGain is applied by the rising edge on associated GPIO.
adi_adrv9001_Rx_InterfaceGain_EndOfFrameGain_Get	Gets the EndOfFrameGain for the Rx interface gain. EndOfFrameGain is updated by the falling edge on associated GPIO.

USAGE RECOMMENDATIONS

This section summarizes a list of recommendations to achieve optimal gain control performance. It is recommended to start with AGC and default configurations.

- When changing the default configurations, it is better to change the parameters one by one.
- The high thresholds are used as limits on the incoming signal level and should be set based on the maximum input of the ADC. Set the high thresholds at least 3 dB to 6 dB lower than the full input scale of the ADC.
- Set the apdHighThresh and hbHighThresh to an equivalent dBFS value. Likewise, set the apdLowThresh and the hbUnderRangeHighThresh to equivalent values.
- Set the apdUpperThreshPeakExceededCount and hbUpperThreshPeakExceededCount properly to achieve the desired level of AGC sensitivity to input signal peaks.
- Align the gain change period typically to the frame or subframe boundary periods.
- Peak detect achieves a faster response time compared to peak/power detect. For applications requiring “fast attack” and “fast recovery”, peak detect provides better performance.
- Do not use the “fast recovery” mode to increase gain only at the frame or subframe boundary.
- For applications requiring “fast attack” and “fast recovery”, set SLS delay as short as possible to minimize the delay while allowing sufficient time to set the gain changes.

TES CONFIGURATION AND DEBUG INFORMATION

Interact with the receiver gain control functionality through API commands, as discussed in previous sections, or through the ADRV9001 transceiver evaluation software (TES). While using the TES, first configure the receiver gain control and signal detector operation modes, as shown in Figure 179.

The configuration page is under the **GPIO** tab in TES. The gain control mode provides three options: **Automatic**, **Manual Pin,** and **Manual SPI**, which correspond to the AGC, MGC with pin control, and MGC with SPI control. By default, it is set to the **Manual SPI**. When the **Manual Pin** is selected, further select the GPIO pins for the gain increment and decrement. After selecting the gain control mode, further configure the **Detection Mode**, which has two options: **Peak Only** and **Peak and Power**. By default, it is set to the **Peak Only** mode. The **Detection Mode** indicates which AGC mode is configured when the **Gain Control Mode** is selected as **Automatic**. When the **Gain Control Mode** is selected as **Manual Pin** or **Manual SPI**, it further enables the ADRV9001 internal signal detectors in either the **Peak Only** or **Peak and Power** mode. By configuring the GPIO pins, the user can retrieve the signal detector status, which can be used to control the receiver gain in the **Manual** mode. Note that the signal detector status can also be retrieved in the AGC mode for observation.

RECEIVER GAIN CONTROL

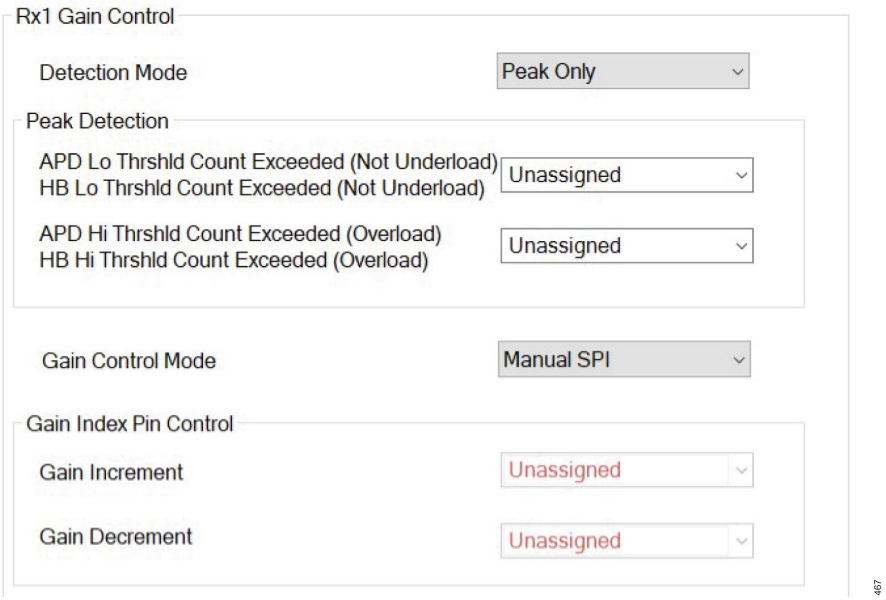


Figure 179. TES Configuration for Rx Gain Control Mode and Signal Detector Operation Mode

After configuring the **Gain Control Mode** and **Detection Mode**, further configure the interface gain, signal detection parameters, and manual control parameters under the **Gain Control** tab in the TES, as shown in Figure 180 , Figure 181, and Figure 182.

The interface gain control parameters shown in Figure 180 are previously discussed. Choose the **Manual** or **Automatic** receiver interface gain control mode and the associated parameters. When the **Manual** mode is selected, further enter the desired interface gain value. Note that the interface gain range relates to the selection of the gain table. First select the receiver gain table type. In both modes, configure when the gain update should take effect. Also configure the RSSI related parameters and choose if the fast attack mode should be enabled. In the **Automatic** mode, further configure the **Max Signal Level Target**, **Peak-to-Average Ratio of Applied Signal** and **Seed/Save Gain Value** by specifying a GPIO pin.

For signal detection parameters shown in Figure 181, configure the signal detection parameters such as **Peak Overload Threshold**, **Peak Underload Threshold** (they apply to both APD and HB detectors), **Overload Gain Step**, **Underload Gain Step**, **Gain Refresh Period**, **Max Gain Index**, and **Min Gain Index** when the **Peak Only** mode is selected. Note that some selections are grayed out as they are not applicable in the selected modes. For example, **Power Overload Threshold** and **Power Underload Threshold** are only applicable for the **Peak and Power** mode. The TES provides sanity checks for the entered parameters so when the value is out of range, the user input is not allowed.

There are several additional configurations:

1. Freeze the signal detection using a specified GPIO pin. If this pin is high during the **Automatic** mode, it freezes signal detections. So, Rx gain is not changed regardless of the input Rx signal level.
2. Enable the "lowThreshPreventGainInc" functionality, as discussed previously.
3. Freeze the Rx QEC algorithms when overload happens.
4. Enable the HB detector (if not selected, it is disabled on both the detection modes).
5. Reset the Rx gain index when Rx is enabled.

For the manual control parameters shown in Figure 182, if MGC SPI is configured, further configure other parameters such as the **Manual Gain Index**. With the **MGC Pin** mode, configure the **Increment Step Size** and **Decrement Step Size**.

RECEIVER GAIN CONTROL

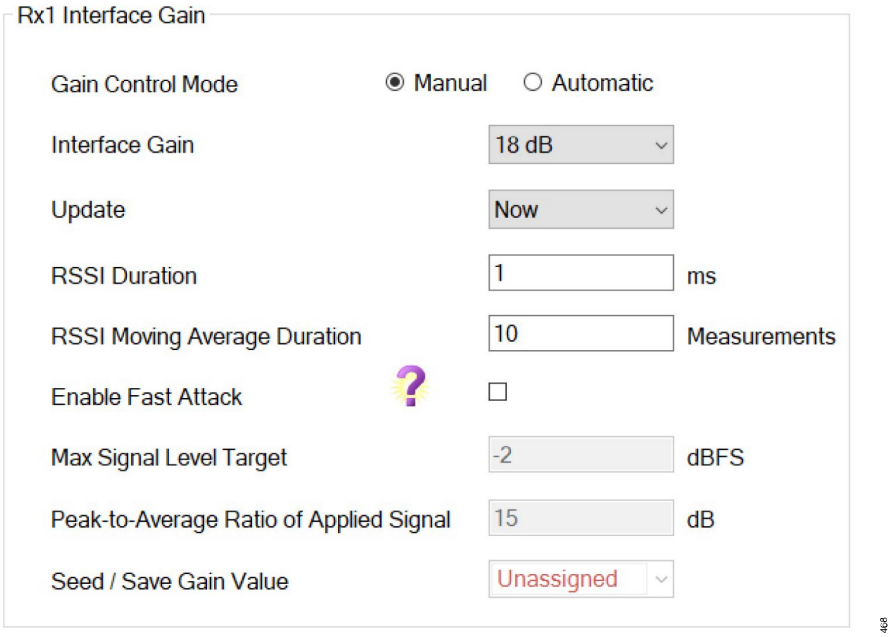


Figure 180. TES Configuration for Rx Interface Gain

RECEIVER GAIN CONTROL

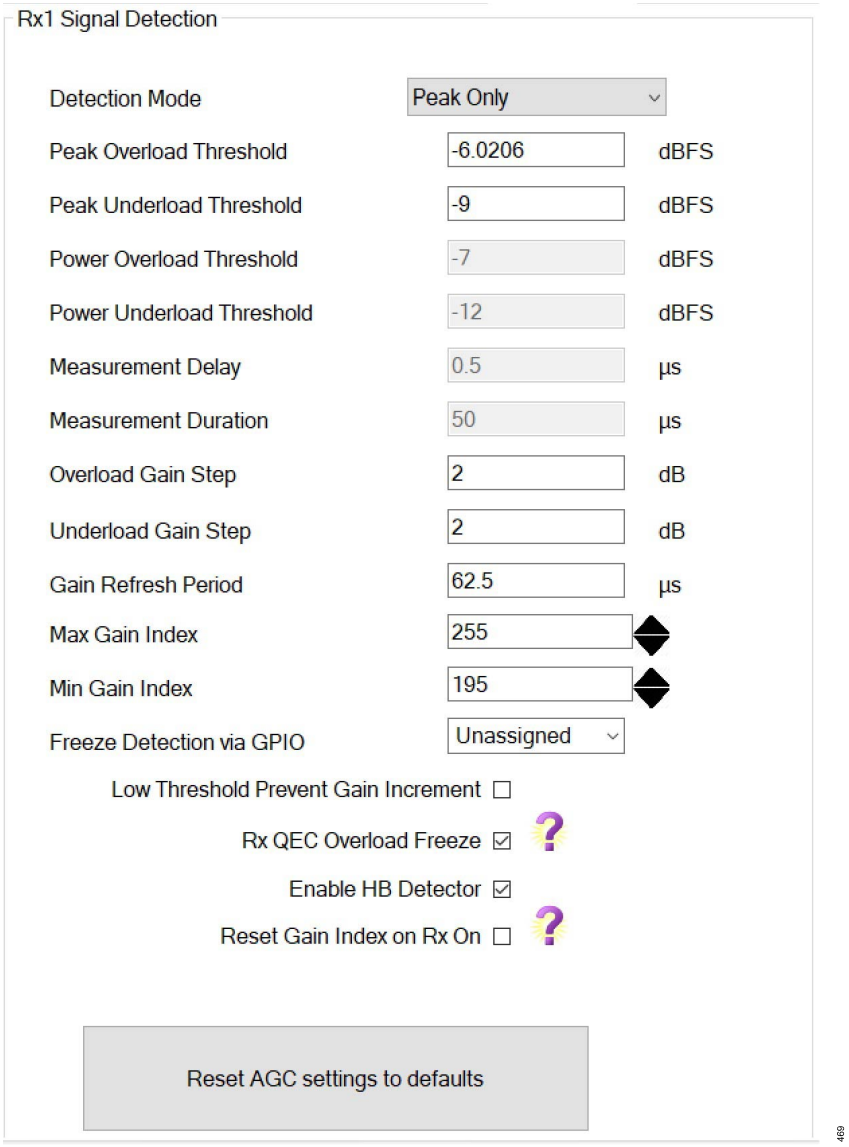


Figure 181. TES Configuration for Signal Detection Parameters

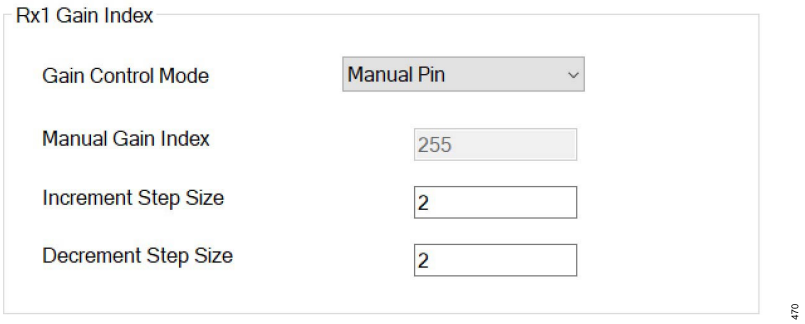


Figure 182. TES Configuration for Manual Control Mode Parameters

After finishing all the configurations, start the receive operations and observe the receiver gain changes. It is recommended to start from the default settings and change the parameters one by one as needed. "**Reset AGC settings to defaults**" resets all the parameters to their default values.

RECEIVER GAIN CONTROL

When the receiver gain control is not working as expected, perform the following simple self-debugging.

- Check if the gain control mode is set as AGC or MGC.
- Check if the MAX and MIN index are set properly. When set improperly, the gain control capability can be significantly impacted.
- If detectors are not working, check if the gain step is set to 0, which disables gain attack or gain recovery.
- When signal saturation is observed, adjust the slicer/interface gain.

RECEIVER DEMODULATOR

RECEIVER NARROWBAND DEMODULATOR SUBSYSTEM

The ADRV9001 receiver narrowband demodulator subsystem, denoted by rxnbdem, is the digital baseband back-end partition of the ADRV9001 receiver channel. Note that the narrowband is commonly used in wireless communication systems. If the channel spacing (also known as channel bandwidth) is 1 MHz or less, it is known as a "narrowband system," otherwise it is called a "wideband system." Figure 183 shows the rxnbdem subsystem, incorporating signal buffering, carrier frequency offset correction, programmable channel filtering, frequency discrimination, narrowband programmable pulse shaping, and resampling function. The input of rxnbdem, driven by the receiver decimation filters, is the zero intermediate frequency (ZIF) digital baseband IQ signal. There is programmability to bypass each block in the rxnbdem subsystem. The output of rxnbdem is directly sent to the receiver SSI. Depending on the programmed functionality, the output can be frequency deviation (I)-only or IQ.

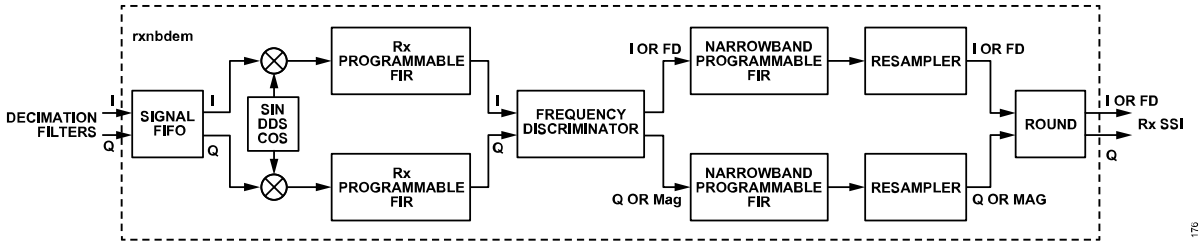


Figure 183. Functional Diagram of rxnbdem

Signal FIFO

The Signal FIFO buffers the input IQ data stream, and it is applicable only in the CMOS SSI operation mode. The Signal FIFO depth is 2048. As a result, it can store more than 85 ms at the sampling rate of 24 kHz.

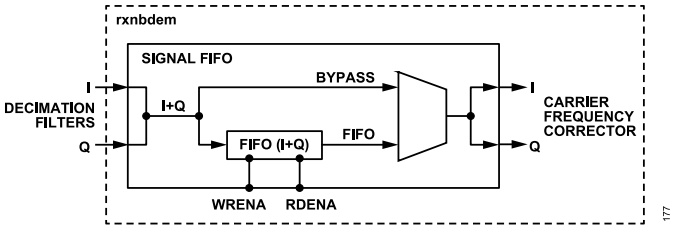


Figure 184. Functional Diagram of Signal FIFO

Disable or enable the Signal FIFO based on requirements. As the Signal FIFO is disabled, this block is bypassed and cannot be written or read. As the Signal FIFO is enabled, the writing and reading control of the Signal FIFO can be manipulated separately. The FIFO reading clock is configurable and can be 1×, 2×, 4×, or 8× of the FIFO writing clock. Only 1× and 2× are supported for wideband modes, and the reading clock rate cannot be above 61.44 MHz.

In the Signal FIFO, as shown in Figure 184, there is an output mux, which has two inputs: “Bypass” is driven by the input IQ stream to the FIFO and “FIFO,” is driven by the output IQ stream from the FIFO. The mux inputs can be switched on demand to drive the following modules in rxnbdem.

The following example explains how to use the Signal FIFO.

Keep the mux at "Bypass" during the signal capturing phase before the wireless data link is established. The "FIFO" writing control is enabled and reading control is disabled. Through the receiver SSI port, the baseband integrated circuit (BBIC) can keep on detecting the received signal. Meanwhile, the Signal FIFO keeps on buffering the IQ stream. The FIFO writing overflow may or may not happen. If this happens, the FIFO always stores the latest 2048 IQ data.

As the BBIC detects the desired receiver frame from the input data stream and estimates the right starting point of the wanted receiver frame, and further the synchronous parameters, the BBIC can switch the mux from “Bypass” to “FIFO,” then enable the reading control of the FIFO, to process the stored data stream and the following data stream seamlessly.

RECEIVER DEMODULATOR

Carrier Frequency Corrector (CFC)

The carrier frequency corrector (CFC) in rxnbdem removes the carrier frequency offset. This module can be bypassed.

In a communication system, the desired signal is transmitted by the transmitter at RF over the air. As the clock reference at the transmitter and receiver are independent, this can result in the RF carrier frequency offset between the transmitter and the receiver. This frequency difference is called the carrier frequency offset (CFO). The CFC in rxnbdem enables the BBIC to remove the CFO before the channel selection filtering (receiver PFIR) at the receiver side. The BBIC must estimate and input the correction value applied to the CFC. The correction value change can occur immediately or relative to the receiver frame boundary.

The CFC is implemented as a digital down converter (DDC), which consists of an NCO and a complex multiplier in the datapath. As the correction value, ensure the NCO frequency word is in the range of minute (±20 k, 20% of sample rate).

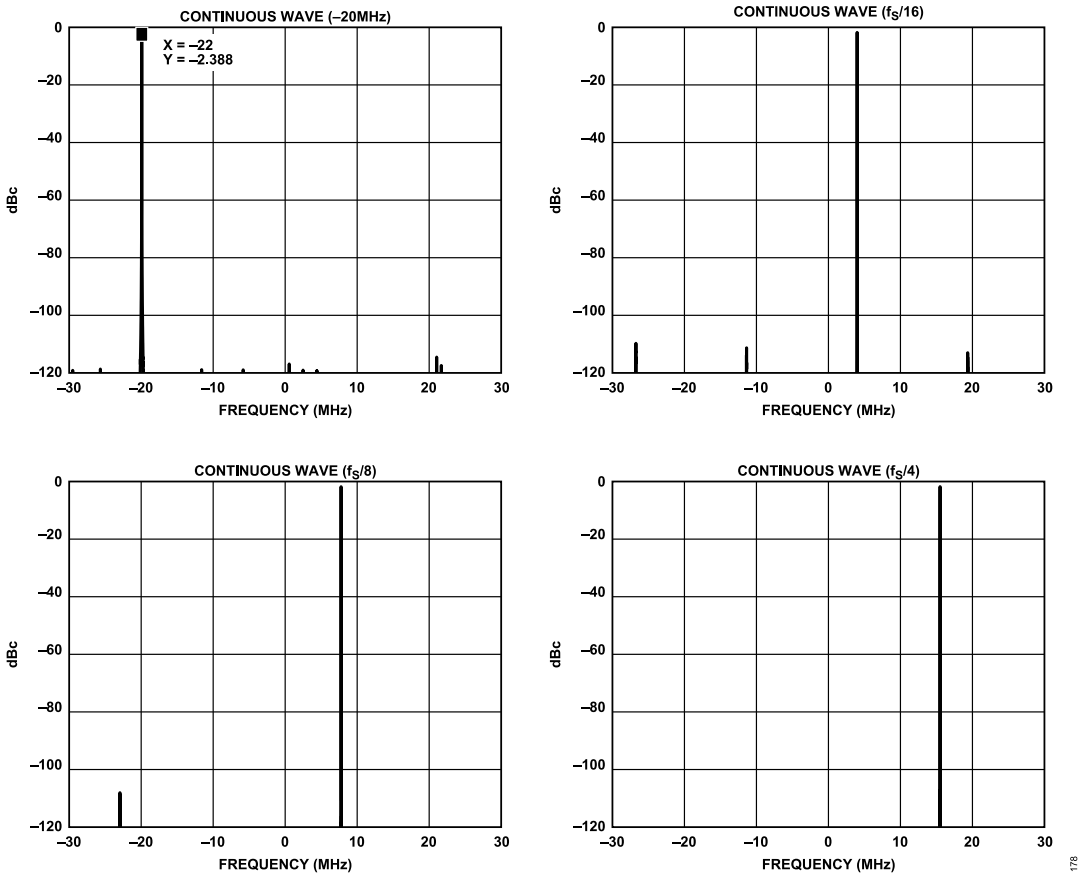


Figure 185. Output Spectrum of the CFC/NCO as f_S = 61.44 MHz

Figure 185 shows the spectrum of the desired tone and generated NCO spurs levels relative to the desired tone for the CFC NCO at 61.44 MHz sampling frequency. The outlined plots show a typical case {-20 MHz Continuous Wave (CW)} and some worst cases. Figure 185 shows that the NCO output spurs are −100 dBc below the desired tone across the range of [$-f_S/2$, $f_S/2$]:

- −20 MHz CW
- fs/16 CW
- fs/8 CW
- fs/4 CW

Note**:** These four tones' frequencies are not within the defined NCO range, but are just for the NCO spurs level demonstration.

Receiver Programmable FIR Filter

The receiver programmable FIR filter in rxnbdem is multifunctional and customizable. However, this module can be bypassed.

RECEIVER DEMODULATOR

The receiver programmable FIR supports up to 128 taps. Each tap is 24 bits in width with the signed bit included. Four sets of customized FIR profiles can be stored at the initialization phase. One of the four stored FIR profiles can be switched to load on the fly under the control of the BBIC.

The receiver programmable FIR can be loaded with a customized low-pass filter profile to stop the adjacent channel interference, which helps to achieve better channel selectivity. For example, as shown in Figure 186, before the CFO is corrected, the BBIC can program a loose filter profile onto the receiver programmable FIR to perform common filtering. However, after the CFO is corrected through the carrier frequency corrector block, a tight filter profile can be loaded to perform the deep channel selection filtering. The BBIC can initiate the change in the filter profile on demand in the RF_Enabled state.

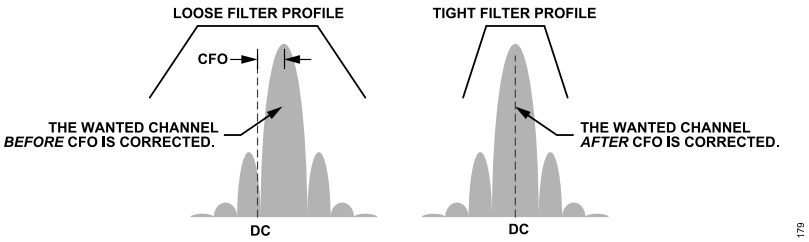


Figure 186. Loose Filter Profile vs. Tight Filter Profile

Frequency Discriminator

The frequency discriminator in rxnbdem translates the IQ signal into the frequency deviation (FD) signal, performing the frequency demodulation in the digital domain. However, this module can be bypassed.

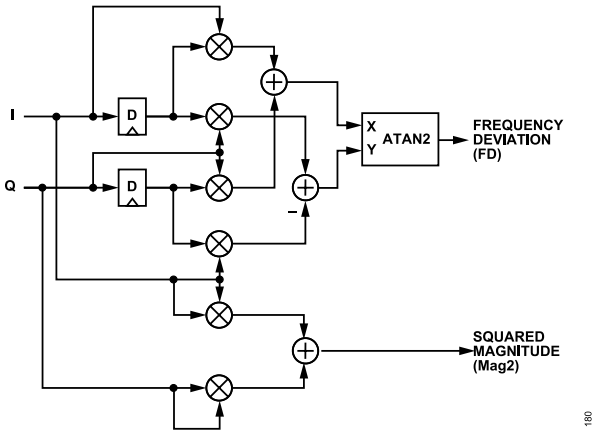


Figure 187. Functional Diagram of Frequency Discriminator

Figure 187 shows that the frequency discriminator outputs the transient frequency deviation (FD) and the transient squared magnitude (Mag2) sample by sample. The output FD and the output Mag2 are defined as the following:

$$FD(n) = \frac{1}{\pi}a\tan 2[I(n-D)Q(n) - I(n)Q(n-D), I(n)I(n-D) + Q(n)Q(n-D)] \quad (8)$$

$$Mag2(n) = I(n)^2 + Q(n)^2 \quad (9)$$

where:

$atan2$ is same as the function in Octave.

D is the programmable delay.

Typically, *D* is chosen as '1', which means 1 sampling clock delay.

Assuming the input IQ signal is the complex single tone, given by:

RECEIVER DEMODULATOR

$$A \times e^{\frac{j2\pi f_t n}{f_S}}, \quad n = 0, 1, 2, \ldots \tag{10}$$

where:

A is the signal magnitude.

f_t is the tone frequency.

f_S is the sampling frequency.

The output of the frequency discriminator (FD) is $D \times 2f_t/f_S$ while the output Mag2 is A^2.

Narrowband Programmable FIR

The narrowband programmable FIR in rxnbdem performs the pulse shaping filtering or low-pass filtering at the output of the frequency discriminator. This module can be bypassed.

The narrowband programmable FIR supports up to 128 taps. Each tap is 24 bits in width, including the sign-bit, and can load only one set of customized FIR profiles.

Resampler

Resampler in rxnbdem adjusts the sampling phase of the IQ or FD signal. This module can be bypassed.

The Figure 188 shows the functional diagram of the resampler. The resampler resamples the incoming received signal by reconstructing intermediate samples between every two input samples according to the resample phase parameter, $\mu, \text{where} |\mu| \leq 0.5$.

In the frequency domain, the ideal digital resampler has the transfer function, as shown in the following equation:

$$H_{ideal}\left(j\omega, \quad \mu\right) = \begin{cases} T_s \times e^{j\omega\mu T_S}, & |\omega/2\pi| \leq B_x \\ do \quad not \quad care, & |\omega/2\pi| > B_x \end{cases}$$

where:

B_x is the maximum bandwidth of the input signal.

Filtered by the ideal resampling filter, the input signal, $x(kT_S)$, is converted to the output signal $y(kT_S) = x((k + \mu) \times T_S)$

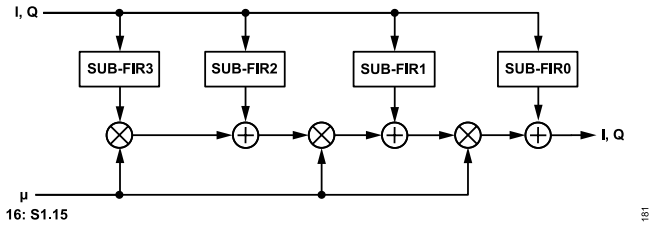


Figure 188. Functional Diagram of the Resampler

Scanning the sampling phase μ from 0 to 0.5, the maximum magnitude difference of the frequency transfer function between the Resampler and the ideal digital resampler is collected and plotted at the upside of the Figure 189. The maximum phase error is collected and plotted at the downside of the Figure 189.

The resampler can be used in both the wideband and narrowband modes.

RECEIVER DEMODULATOR

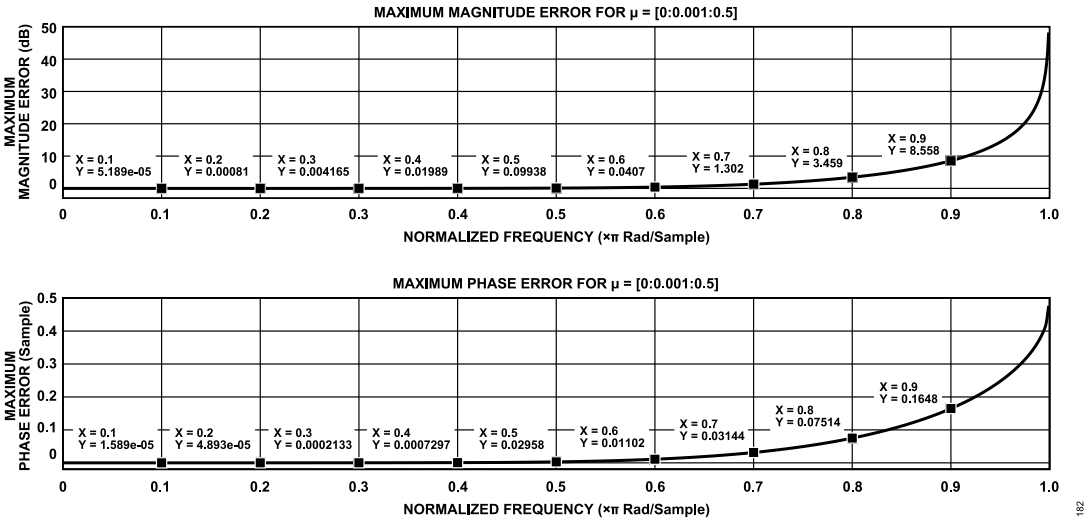


Figure 189. Maximum Magnitude/Phase Error of the Resampler

Note: The current ADRV9001 software release does not support the resampler configuration.

Round Module

The round module in rxnbdem maps the ADRV9001 internal data path bit-width to the receiver SSI output. This module can be bypassed if the IQ-22bit mode is chosen.

The round module can output 16-bit or 15-bit I/Q data to the receiver SSI output. If required, the round module can output 16-bit I data to the receiver SSI output with the 16-bit output Q data set as '0'.

NORMAL IQ OUTPUT MODE

The normal IQ output mode is applicable for both wideband and narrowband as the frequency discriminator is bypassed. Except for the rounding, all other modules can be bypassed. Figure 190 shows that the ADRV9001 receiver narrowband demodulator can be the common output stage of the receiver channel.

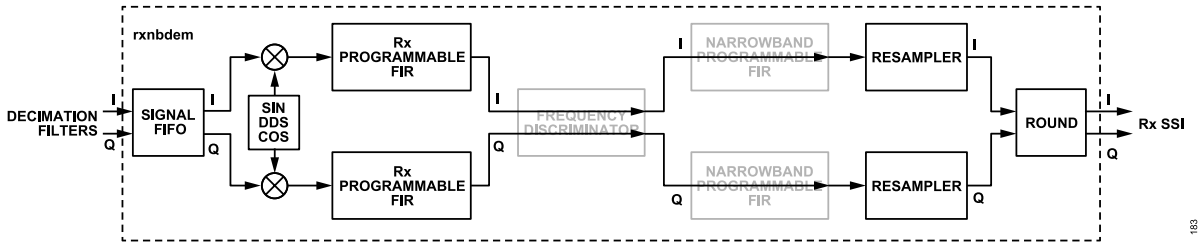


Figure 190. Rxnbdem in Normal IQ Output Mode

In the normal IQ output mode, the rxnbdem can provide the signal processing resources as the following:

- A spur-free carrier frequency offset correction
- A profile-switchable channel filter
- A precise IQ resampler (not supported by software yet)

The BBIC can manually enable and control all the above resources by API (the future software release provides support).

FREQUENCY DEVIATION OUTPUT MODE

The frequency deviation output mode is only applicable for the narrowband modes. Except for the round module and frequency discriminator, all other modules can be bypassed. See Figure 191. The ADRV9001 receiver narrowband demodulator contains a frequency discriminator

RECEIVER DEMODULATOR

hardware block. Cooperating with other hardware blocks, such as the CFC/DDC, programmable FIR filters, and so on, the ADRV9001 receiver narrowband demodulator can perform frequency-shift key (FSK) and FM demodulation under the control of the BBIC.

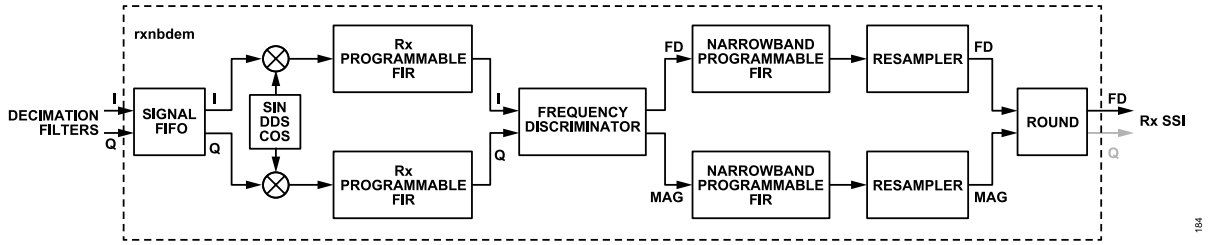


Figure 191. Rxnbdem in Frequency Deviation Output Mode

The FSK/FM demodulation can cover these standards:

- Analog FM with 12.5 kHz channel bandwidth
- Analog FM with 20 kHz channel bandwidth
- Analog FM with 25 kHz channel bandwidth
- P25 phase I with 12.5 kHz channel bandwidth
- DMR with 12.5 kHz channel bandwidth

For each standard, Table 92 lists the usage suggestion of each module in the frequency deviation output mode.

Table 92. Suggestion of rxnbdem Usages in the Frequency Deviation Output Mode

	FM, 12.5 kHz	FM, 20 kHz	FM, 25 kHz	P25 I, 12.5 kHz	DMR, 12.5 kHz
Input Sampling Clock	24 kHz or 48 kHz		48 kHz	24 kHz or 48 kHz	
Signal FIFO	Optional				
CFC/DDC	Enable for manual control				
Rx Programmable FIR	Enabled for channel filtering with 2 sets of FIR profiles				
Freq. Discriminator	Enabled with D = 1				
NB Programmable FIR	Low-pass filtering for smoothing the output FD signal		Pulse shaping		
Resampler	Disable			Enable for manual control	

In summary, in the frequency deviation output mode, the rxnbdem can provide more important resources such as:

- A 2k-depth FIFO
- A spur-free carrier frequency offset correction
- A profile-switchable channel filter
- An accurate digital frequency discriminator
- A low-pass filter or a pulse shaping filter for the frequency deviation
- A precise frequency deviation resampler

The BBIC can manually enable and control all the above resources.

APPLICATION PROGRAMMING INTERFACE (API) PROGRAMMING

The configuration for blocks in the rxnbdem subsystem is profile related so far. All relative blocks are enabled or bypassed in the profile by selecting the Rx “IQ” or “frequency deviation” mode.

Carrier Frequency Corrector API Programming

The API function **adi_adrv9001_Rx_FrequencyCorrection_Set()** sets the CFC frequency word. Ensure the frequency correction word is in the range of minute (±20 k, 20% of sample rate). The correct frequency operation can take effect immediately or at the start of the next available frame by setting the parameter “immediate” to “true” or “false.”

RECEIVER DEMODULATOR

Receiver Programmable FIR (PFIR) Filter API Programming

Profile predefined receiver PFIR coefficients or customized receiver PFIR coefficients are automatically loaded during chip initialization. Therefore, there is no need for a baseband processor to call any PFIR coefficients loading API function.

The configuration structure adi_adrv9001_PfirWbNbBuffer_t is defined as the following for the PFIR filter coefficients.

```
typedef struct adi_adrv9001_PfirWbNbBuffer
{
uint8_t numCoeff; /* number of coefficients */
adi_adrv9001_PfirSymmetric_e symmetricSel; /* symmetric selection */
adi_adrv9001_PfirNumTaps_e tapsSel; /* taps selection */
adi_adrv9001_PfirGain_e gainSel; /* gain selection */
int32_t coefficients[ADI_ADRV9001_WB_NB_PFIR_COEFS_MAX_SIZE]; /* coefficients */
} adi_adrv9001_PfirWbNbBuffer_t;
```

The baseband processor can prepare new FPIR coefficients in one or more **adi_adrv9001_PfirWbNbBuffer_t** instances and call the API function **adi_adrv9001_arm_NextPfir_Set()** in the “PRIMED” or “RF_ENABLED” state to load each required instance into the ADRV9001 hardware. Multiple PFIRs using the same coefficients can be loaded in a single call. However, note that the old coefficients remain in effect until **adi_adrv9001_arm_Profile_Switch()** is called.

The API function **adi_adrv9001_arm_NextRxChannelFilter_Set()** calls **adi_adrv9001_arm_NextPfir_Set()** once or twice as needed to update the channel filter coefficients for Rx1, Rx2, or both. Either PFIR pointer can be NULL to prevent modifying the corresponding PFIR, but it is an error if both PFIR pointers are NULL.

The ADRV9001 performs the PFIR coefficients switch for all channels that have new coefficients prepared and waiting when the API command **adi_adrv9001_arm_Profile_Switch()** is called. If the ADRV9001 is in a PRIMED state, the new coefficients take effect on the subsequent transition to RF_ENABLED. If it is in RF_ENABLED, they take effect immediately.

The following code is an example python code for the receiver PFIR coefficients switch.

```
pfir_dmr_12p5k_coeff = [1,4,10,14,10,-8,-36,-56,-43,18,110,176,140,-36,-296,\
-477,-385,65,717,1164,945,-116,-1630,-2655,-2161,224,3612,5917,4823,-660,-8930,\
-15835,-16492,-8267,6681,21178,26054,15428,-8503,-34452,-46572,-33192,4645,50326,\
77802,65235,9575,-66663,-122526,-118715,-41686,81623,189288,211654,109809,-93600,\
-309489,-411032,-287232,101328,691337,1327974,1817574,2001178,1817574,1327974,\
691337,101328,-287232,-411032,-309489,-93600,109809,211654,189288,81623,-41686,\
-118715,-122526,-66663,9575,65235,77802,50326,4645,-33192,-46572,-34452,-8503,\
15428,26054,21178,6681,-8267,-16492,-15835,-8930,-660,4823,5917,3612,224,-2161,\
-2655,-1630,-116,945,1164,717,65,-385,-477,-296,-36,140,176,110,18,-43,-56,-36,\
-8,10,14,10,4,1,0]
pfir_dmr_12p5k = adi_adrv9001_PfirWbNbBuffer_t()
pfir_dmr_12p5k.numCoeff = 128
pfir_dmr_12p5k.symmetricSel = adi_adrv9001_PfirSymmetric_e.ADI_ADRV9001_PFIR_COEF_NON_SYMMETRIC
pfir_dmr_12p5k.tapsSel = adi_adrv9001_PfirNumTaps_e.ADI_ADRV9001_PFIR_128_TAPS  # PFIR_128_TAPS
pfir_dmr_12p5k.gainSel = adi_adrv9001_PfirGain_e.ADI_ADRV9001_PFIR_GAIN_ZERO_DB  # PFIR_GAIN_0DB
for i in range(pfir_dmr_12p5k.numCoeff):pfir_dmr_12p5k.coefficients[i] = pfir_dmr_12p5k_coeff[i]

Adrv9001.arm.NextPfir_Set(1, pfir_fm_12p5k) # put in the right filter object
Adrv9001.arm.Profile_Switch()
```

Narrowband PFIR API Programming

Like receiver PFIR, a profile predefined set of narrowband PFIR coefficients (customized narrowband PFIR coefficients are not supported currently) are automatically loaded during chip initialization. Therefore, there is no need for a baseband processor to call any PFIR coefficients loading API function.

POWER SAVING AND MONITOR MODE

The ADRV9001 is a high-performance integrated transceiver with low power considerations. To accommodate different user cases, the ADRV9001 provides flexibility for users can flexibly trade-off between power consumption and performance with some of the following static configuration options:

- Clock phase-locked loop (PLL) option of high performance and low power
- Clock PLL power option of high, medium, and low
- ADC option of high performance or low power
- ADC clock rate option of high, medium, and low
- RF PLL local oscillator generator (LOGEN) optimization option of best phase noise and best power consumption
- RF PLL power option of high, medium, and low
- ARM clock rate option divided by 1, 2, and 4

Choose and configure these static options in the chip initialization stage. These are not allowed to change dynamically, except the ADC option, high-performance ADC, and low-power ADC can be dynamically switched after the chip is initialized. Refer to the related sections in this user guide for the above-mentioned options.

For TDD applications, the ADRV9001 defines different power-saving modes to meet the power-saving requirement in various user cases. Some standards like DMR require the radio to enter periodical sleep and carrier detection cycles to save power (monitor mode) when it is not in use. The ADRV9001 has dedicated hardware to meet this monitor mode requirement, and the ADRV9001 software has additional static and dynamic power saving schemes to extend the power saving feature to a broader market beyond DMR.

The ADRV9001 defines five extra power-down modes that provide low to high power saving but short to long recovery time, and the details are introduced in the following section.

There are three power-saving schemes for different power-saving applications.

- Temporarily powering up/down the unused Tx/Rx channel in the calibrated state.
- Dynamic interframe power saving runs automatically during all regular TDD transmitter/receiver operations. Configure DGPIO pins to support additional power savings. Set all configurations by API through fast messages on the fly. The power saving software smartly handles the powering up/down hardware components based on the PLL mapping and selected power saving mode. There are two power-saving choices in interframe operations:
  - Channel power saving. Powers down a channel (both transmitter and receiver) based on power-down mode 0 to 2.
  - System power saving. Powers down the whole chip by power-down mode 3 to 5
- Monitor Mode. Allows the baseband processor to move into a sleep state after it configures and moves the ADRV9001 into the monitor mode. The ADRV9001 software controls the dedicated hardware and timers for periodical sleeping and detecting. Only power-down modes 3 to 5 are allowed in the monitor mode.

Choose proper power-down modes and power-saving schemes according to their application scenarios. The following sections explain the detailed power-saving schemes.

POWER-DOWN MODES

Power-down modes dynamically power down and power up different levels of the ADRV9001 components. Table 93 describes five extra power-down modes from low power saving but short recover time to high power saving but long recover time. Each higher power-down mode powers down additional components than the lower mode. Power-down mode 3 is the exception.

Table 93. Power-Down Modes and Related Power-Down Components

		Power-Down Modes	0 (Default)	1	2	3	4	5
Components	TX	Analog and Digital datapath	x	x	x	x	x	x
		TX Internal PLLs		x	x	x	x	x
		TX LDOs			x		x	x
	RX	Analog and Digital datapath	x	x	x	x	x	x
		RX Internal PLLs		x	x	x	x	x
		RX LDOs			x		x	x
	System	CLK PLL				x	x	x
		Converter and CLK PLL LDOs					x	x

POWER SAVING AND MONITOR MODE

Table 93. Power-Down Modes and Related Power-Down Components (Continued)

	Power-Down Modes	0 (Default)	1	2	3	4	5
	ARM (+ memories)						x
Approximate Power-Up Time (µs) [1, 2]	DEV_CLK = 30Mhz	4.5	350	500	250	650	3200
	DEV_CLK = 50Mhz		180	380			
	DEV_CLK = 100Mhz		170	370			
Approximate Power-Up Time (µs) [3, 2]	DEV_CLK = 30Mhz		100	300			
	DEV_CLK = 50Mhz		60	260			
	DEV_CLK = 100Mhz		40	240			

[1] RF PLL is in normal calibration mode, power up time varies with DEV_CLK frequency.

[2] At 184.32MHz processor clock

[3] RF PLL is in fast calibration mode, power up time varies with DEV_CLK frequency.

- Power-down mode 0 is the default power saving if power-saving API is not called to set other values. In this mode, the TX/RX enable pin can automatically trigger powering up/down the TX/RX analog and digital datapath components such as mixer, A/D, filters, and so on. The transition time is very short (around 4.5 µs).
- Power-down mode 1 powers down internal RF PLLs in addition to mode 0 power down. After powering up, PLL requires recalibration. So, it takes more time to power up.
- Power-down mode 2 powers down some LDOs related to channels and RF PLLs in addition to mode 1 power down.
- Power-down mode 3 powers down the TX/RX channels and PLLs (clock PLL, RF PLLs) only. No LDOs are powered down.
- Power-down mode 4 powers down clock PLL and system LDOs related to TX/RX channels in addition to mode 3 power down.
- Power-down mode 5 powers down almost the whole ADRV9001 chip, including ARM and memory, except for some wake-up circuits.

POWER-DOWN/UP CHANNEL IN CALIBRATED STATE

Dynamically power down/up individual channels (TX1/TX2/RX1/RX2) in the calibrated State if these channels are statically enabled in the device profile. Call **adi_adrv9001_Radio_Channel_PowerDown()** to power down the specified channel, and it powers down the channel-related LDOs and PLL for the channel in addition to the datapath power down. **adi_adrv9001_Radio_Channel_PowerUp()** powers up the specified channel. The two APIs can only be called in the calibrated state.

Figure 192 shows a DMR radio switch from transmitter-only frames into transmitter/receiver alternate frames. The ADRV9001 is initialized with the transmitter, and the receiver is enabled at the beginning of the transmitter-only frames. The baseband processor can bring the receiver channel into the calibrated state and power it down. Then, before the transition of the TX/RX alternate frames, the baseband processor can power up the receiver and move it into a primed state. The power saving of the transmitter channel in the gray area is addressed by power-down modes discussed in the following sections.

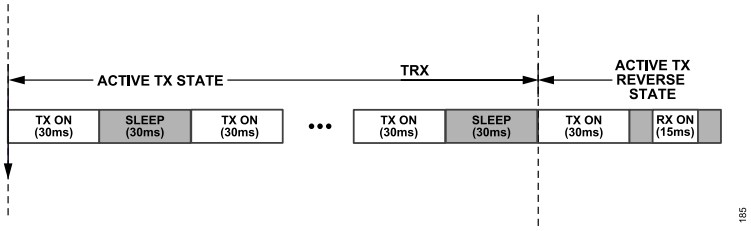


Figure 192. DMR Typical State Transition

Another use case example is if four channels ((TX1/TX2/RX1/RX2) are enabled in the profile, the unused channels can be temporarily powered down by moving them to the calibrated state.

DYNAMIC INTERFRAME POWER SAVING

Dynamic interframe power saving runs automatically during all regular TDD transmitter/receiver operations. A higher level power-down mode can be configured to get more power saving if the application has a longer transmitter/receiver transition time. DGPIO pins can be configured to support additional power savings.

POWER SAVING AND MONITOR MODE

There are two power-saving choices that can be applied for various TDD interframe scenarios: channel power saving (CPS) and system power saving (SPS). Configure either or both options according to the system specifications.

Channel Power Saving (CPS)

Channel power saving saves power-on channel granularity for dynamic TDD interframe operations. Two kinds of power-saving events are triggered by either transmitter/receiver enable pins or DGPIO pins, respectively. The configuration selects power-saving modes for both kinds of events. Only power-down modes 0 to 2 can be configured for CPS.

TX_ENABLE/RX_ENABLE Pin Triggers Power Saving

The power saving triggered by the TX_ENABLE/RX_ENABLE pin powers up/down based on the TX_ENABLE/RX_ENABLE rising or falling edges. The TX_ENABLE/RX_ENABLE rising edge powers up the components based on the power-down mode, and the falling edge powers down them.

Figure 193 shows the TX/RX Enable pin powers up/down channels. If the transmitter/receiver enable pin power-down mode is set to mode 1, the TX1/RX1 enable falling edge powers down the TX1/RX1 PLL and TX1/RX1 datapath, and the rising edge powers them up. The higher the power-down mode, the longer the recovery time. Ensure the system has enough transition time between the power-down and power-up of the same component if a high power-down mode is set. For example, if TX1 and RX1 use the same internal PLL and there is very short transition time between the transmitter enable falling edge, and receiver enable rising edge, ensure mode 1 and 2 are not selected because the same PLL and LDOs are always used.

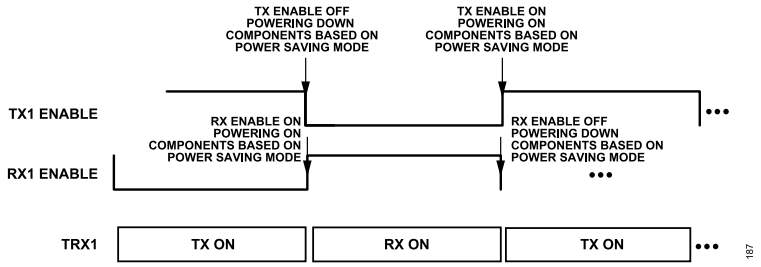


Figure 193. TX/RX Enable Pin Triggers Power Saving

DGPIO Triggers Power Saving

DGPIO pin-triggered channel power saving provides additional power saving than the TX/RX enable pin when it is enabled. Therefore, if enabled, ensure the power-down mode triggered by DGPIO is larger than the TX/RX enable pin-triggered power-down mode. Both the transmitter and receiver channels can be powered down at the DGPIO rising edge and powered up at the DGPIO falling edge. This is because only one DGPIO is assigned for the transmitter and receiver channels. The DGPIO can only be allowed to pull up when both the TX enable and RX enable are low.

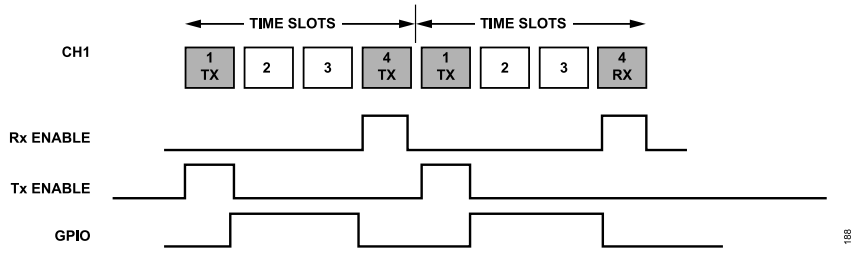


Figure 194. TX/RX Pin Triggers Power Saving and DGPIO Triggers Power-Down Saving

Figure 194 shows an example when both the TX/RX enable and DGPIO pin trigger power saving is enabled. The grey time slots are the ones when the transmitter/receiver must be active. If the transmitter and receiver transition time is not long enough to allow power-down mode 1 or 2, select the TX/RX enable pin power-down mode 0. However, DGPIO power saving can be engaged during time slots 2 and 3 by selecting power-down mode 2 to power down both the transmitter/receiver LDOs and PLLs in slot 2 and slot 3 areas in which the transmitter and receiver are inactive.

POWER SAVING AND MONITOR MODE

The API command **adi_adrv9001_powerSavingAndMonitorMode_ChannelPowerSaving_Configure()** configures the channel power-saving modes for a specified channel. Call it in the calibrated, primed, or RF-enabled state. The new setting cannot take effect immediately after mailbox acknowledgment but at the start of the power-down pin edge (enable falling edge and DGPIO rising edge). Therefore, the baseband processor should allow enough time to send this command and receive or acknowledge it before the next power-down event.

The following data structure defines the channel power-saving trigger modes.

```
typedef struct adi_adrv9001_PowerSavingAndMonitorMode_ChannelPowerSavingCfg
{
adi_adrv9001_PowerSavingAndMonitorMode_ChannelPowerDownMode_e channelDisabledPowerDownMode;
adi_adrv9001_PowerSavingAndMonitorMode_ChannelPowerDownMode_e gpioPinPowerDownMode;
} adi_adrv9001_PowerSavingAndMonitorMode_ChannelPowerSavingCfg_t;
```

The enumerator **adi_adrv9001_PowerSavingAndMonitorMode_ChannelPowerDownMode** defines three power-down modes described in the following data structure.

```
typedef enum adi_adrv9001_PowerSavingAndMonitorMode_ChannelPowerDownMode
{
ADI_ADRV9001_POWERSAVINGANDMONITORMODE_CHANNEL_MODE_DISABLED = 0, /*!< Default radio operation, no ex▶
tra power down */
ADI_ADRV9001_POWERSAVINGANDMONITORMODE_CHANNEL_MODE_RFPLL = 1, /*!< RF PLL power down */
ADI_ADRV9001_POWERSAVINGANDMONITORMODE_CHANNEL_MODE_LDO = 2, /*!< Channel LDO power down */
} adi_adrv9001_PowerSavingAndMonitorMode_ChannelPowerDownMode_e;
```

adi_adrv9001_powerSavingAndMonitorMode_ChannelPowerSaving_Inspect() inspects the channel power-saving settings for the specified channel.

System Power Saving (SPS)

System power saving achieves more power saving but with a longer transition time. The SPS mode uses an additional DGPIO pin to trigger the whole ADRV9001 chip into sleep in the power-saving mode 3 to 5.

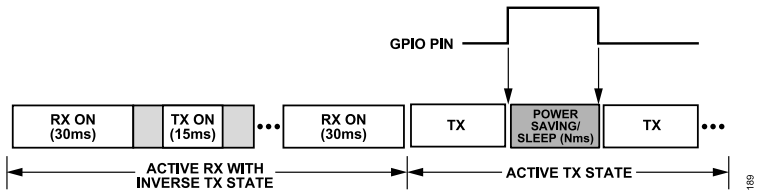


Figure 195. Combined CPS and SPS for Power Saving

Figure 195 shows an example how CPS and SPS combine for the best power savings. Select the channel power saving to the TX/RX enable pin power-down mode 2, so that the TX/RX enable falling edge powers down channel LDOs and TX/RX PLL, and the rising edge can power them up. After switching to the transmitter-only state, although the receiver channel can be powered down by the command **adi_adrv9001_Radio_Channel_PowerDown()** in the calibrated state, the dark gray area only has transmitter LDOs, and PLL powered down by the transmitter enable falling edge. Another option is to power down more by using SPS if the dark gray area is very long. Set the power-down mode 3 to 5 to power down most of the ADRV9001 components to save power and wake them up by the DGPIO falling edge early enough before the transmitter enable rising edge.

Similar to the DGPIO usage in the CPS mode, the DPGIO in the SPS mode can only be pulled up when both the Tx Enable and Rx enable are low. SPS can only be enabled when both Rx and Tx for a channel are in the PRIMED state.

adi_adrv9001_powerSavingAndMonitorMode_SystemPowerSavingMode_Set() sets the SPS modes. The enumerator **adi_adrv9001_PowerSavingAndMonitorMode_SystemPowerDownMode_e** defines three power-down modes described in Table 93.

```
typedef enum adi_adrv9001_PowerSavingAndMonitorMode_SystemPowerDownMode
{
ADI_ADRV9001_POWERSAVINGANDMONITORMODE_SYSTEM_MODE_CLKPLL = 3, /*!< CLK PLL power down */
```

POWER SAVING AND MONITOR MODE

```
ADI_ADRV9001_POWERSAVINGANDMONITORMODE_SYSTEM_MODE_LDO = 4, /*!< LDO power down */
ADI_ADRV9001_POWERSAVINGANDMONITORMODE_SYSTEM_MODE_ARM = 5 /*!< ARM power down */
} adi_adrv9001_PowerSavingAndMonitorMode_SystemPowerDownMode_e;
```

MONITOR MODE

Monitor mode is an enhanced sleep mode which includes automatic wake-up on signal detection. Monitor mode allows for autonomous detection of carrier signals in power saving mode. This enables the ADRV9001 and the host digital baseband processor (DBB) to stay in sleep mode during the idling (not active) state.

During the monitor mode operation, the ADRV9001 goes through cycles of sleeping and detection. In the sleep mode, most of the ADRV9001 is powered down as described in the Power Saving and Monitor Mode section. In the detection mode, a portion of the Rx signal chain and embedded ARM processor are woken to perform the detection. Since the carrier detection is performed autonomously by the ADRV9001, no input is needed from the DBB so it can stay in full sleep mode until a carrier signal is detected. Monitor mode allows for greater system power savings by enabling both the ADRV9001 and the DBB to stay in sleep mode for the duration of the idling state.

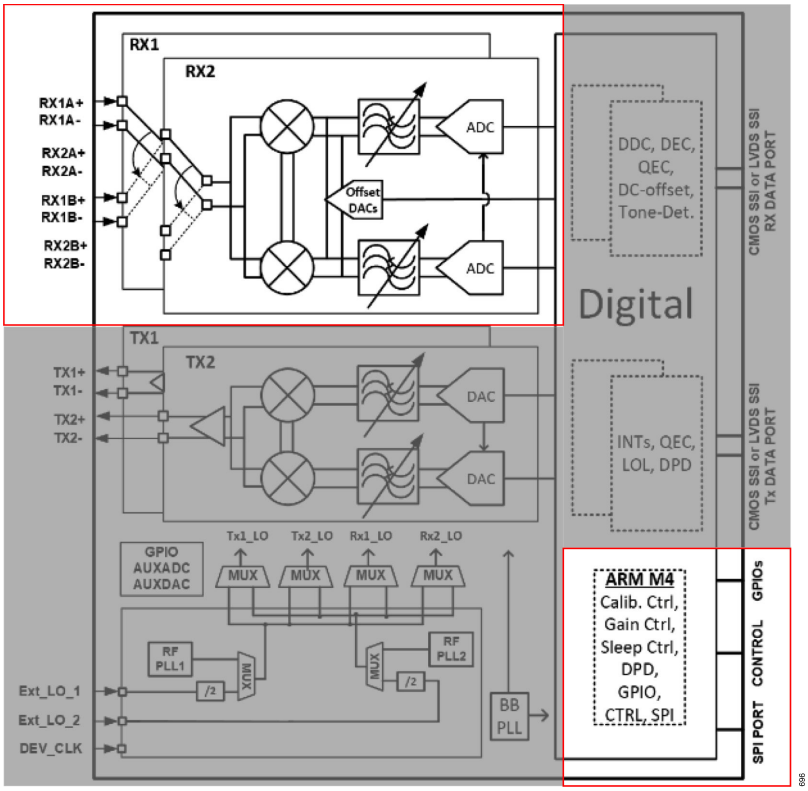


Figure 196. Monitor Mode Block Diagram

Figure 196 shows a block diagram of the waking/sleeping sections of the ADRV9001 during monitor mode operation. In normal operation, ADRV9001 and the DBB are fully powered up. During the idle time, ADRV9001 periodically sleeps and detects. The DBB can sleep for the full idle time. In the sleep state, ADRV9001 is in full power down mode. In the detecting state, only a portion of the Rx signal chain and the embedded ARM processor are woken up (highlighted).

Figure 197shows a typical monitor mode operation, in which the baseband processor fully controls the operation before it enables the ADRV9001 into the monitor mode. First, the baseband processor sets the monitor mode configuration through an API command. Then, the baseband processor asserts the monitor enable pin (specified by a DGPIO) to move the ADRV9001 into the monitor mode, and the baseband processor itself can go into a sleep state until it is woken up by the ADRV9001 or other system interrupt. During the monitor mode, the ADRV9001 fully controls itself to perform the sleep-detection cycling. The sleep/detection cycle of the ADRV9001 is continuous unless the configured detection method detects a valid signal, or it is terminated by a baseband processor pulling down the monitor enable pin. The initial sleep and detection durations are user configurable, and users can decide on detection first or sleep first when the ADRV9001 is transitioned

POWER SAVING AND MONITOR MODE

into the monitor mode. The ADRV9001 triggers the wake-up pin to wake up a baseband processor once the carrier is detected in any detection cycle.

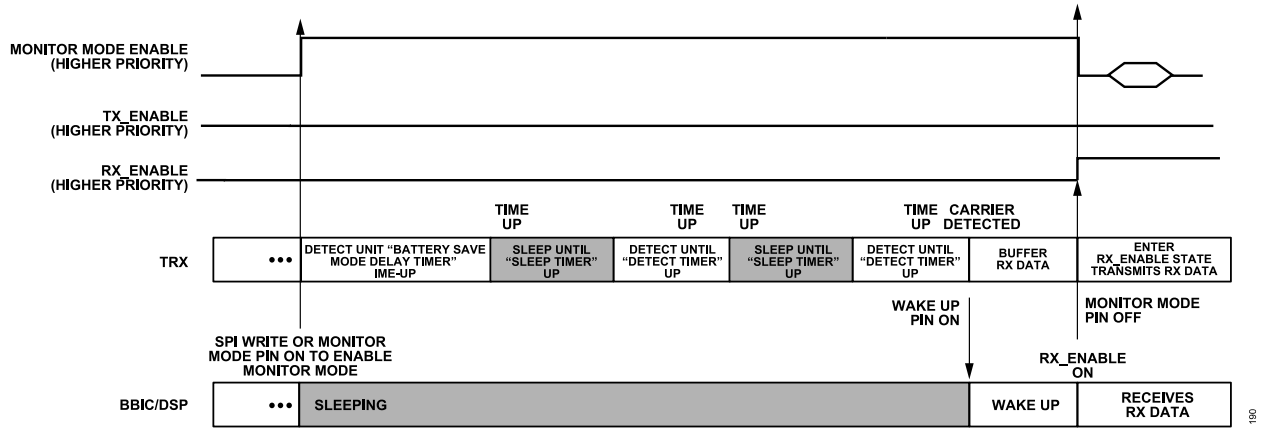


Figure 197. Monitor Mode: Baseband Processor in Sleep and ADRV9001 in Alternate Sleep and Carrier Detection

The monitor mode uses the same power-down modes as System Power Saving (SPS) and they can use the same DGPIO as the power saving trigger interface. Use either the SPS or monitor mode. These two modes can also be dynamically switched for different time slots. Send an SPS or monitor mode API command to switch between modes. The ADRV9001 can buffer the latest incoming data in the monitor mode "detecting " cycle. Then, once a valid incoming signal is detected, and the ADRV9001 wakes up the baseband processor, it can send out the buffered receiver data to the baseband processor. This procedure ensures the baseband processor does not miss the valid incoming signal when it is in the sleep state.

Detection Modes

Monitor mode supports five detection modes for use in different radio standards. The detection modes define under what condition the system detects a valid signal and triggers the wake-up pin. Some modes are standard dependent.

There are three methods of detection of a valid signal: RSSI, DMR and FFT detection. RSSI detection can be used in conjunction with DMR detection and FFT detection to make dual-detection modes. All five detection modes are described in the following sections.

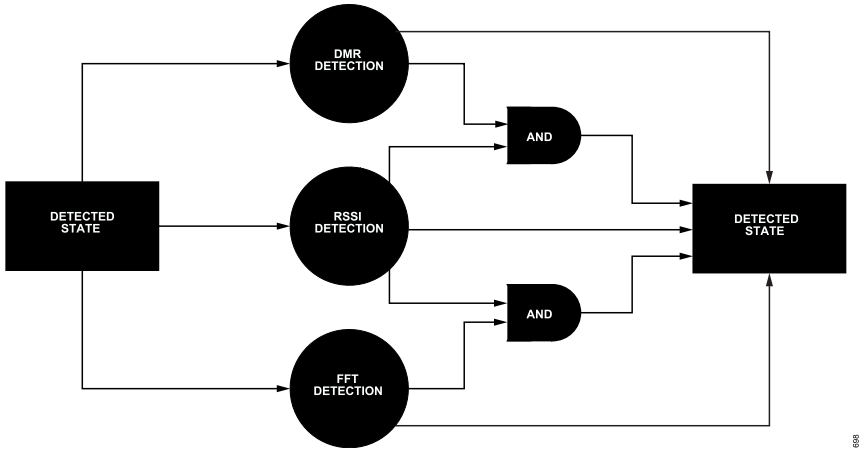


Figure 198. Monitor Mode Detection Modes

RSSI Mode

The "RSSI" detection mode is radio standard independent. The RSSI detection mode settings are independent of the standard receive mode RSSI settings outlined in *RSSI*, however both settings are similar to each other. The number of measurements to average, the measurement duration and the measurement start period are configurable. The detection threshold is also configurable (between −98 dBFS to 0 dBFS) which defines the level for when the signal RSSI power contains a valid signal.

POWER SAVING AND MONITOR MODE

A “DETECTED” state is asserted once the input signal level is beyond the threshold in the “DETECTING” state, and then the following “Wake Up” procedure is started. For example, if the threshold is set to −50 dBFS, when a signal strength higher than this threshold is detected then it is moved to the “DETECTED” state.

Note: this mode is currently only available in DMR profile.

SYNC Mode

The “SYNC” detection monitor mode is intended only for the DMR standard. Configure the receiver datapath in the “frequency deviation” mode to enable the related functions of the Receiver Demodulator. In the NB FSK block, there is a correlator that can compare received code to some expected vectors. The correlator detects the SYNC code from the incoming data stream and up to 14 different SYNC code detection can be supported simultaneously. Similarly, once a SYNC code is detected, the ADRV9001 starts the wake-up procedure.

In this mode, it is required that the Rx interface gain is set to 0 dB as the interface gain is applied prior to the ‘NB FSK’ block which expects 0dB of interface gain.

A path delay calibration is required to run in this mode. This calibration measures the delay between the buffer and the SYNC detection block. This path delay is needed to synchronize the buffer read function. The calibration result can also be saved and stored and uploaded later to avoid needing to run the calibration on each boot.

FFT Mode

The “FFT” detection mode is only available for the standards that use the FM/FSK modulation scheme. Configure the receiver datapath in the “frequency deviation” mode to enable the related functions of the Receiver Demodulator. This mode uses the frequency discriminator hardware block (described in the Frequency Deviation Output Mode section) to demodulate FM/FSK signals. The data is passed to the ARM MCU which performs a real-time 64 bin FFT of the signal and subsequently uses an FM/FSK detection algorithm to check for a valid FM/FSK signal. Once a valid FM/FSK signal is detected, the ADRV9001 starts the wake-up procedure.

Note: this mode is currently not available. The maximum channel bandwidth of this mode is limited by the frequency discriminator which is currently 25 kHz.

RSSI SYNC Mode

RSSI SYNC mode combines the functionality of RSSI and SYNC modes to achieve a more specific method of detection using a dual detection process. Simultaneously, the signal RSSI is compared to a configurable threshold level similar to the RSSI detection mode and analyzed for inclusions of DMR Sync codes similar to Sync Detection Mode. If the RSSI detection and SYNC detection both detect a valid signal the ADRV9001 starts the wake-up procedure.

The settings for RSSI SYNC mode are similar to RSSI Mode and SYNC Mode respectively and the same limitations apply. Use this mode to avoid false positive detection of SYNC codes by checking that the signal level is valid also.

Note: this mode is currently not available.

RSSI FFT Mode

RSSI FFT mode combines the functionality of RSSI and FFT modes to achieve a more specific method of detection using a dual detection process. Simultaneously, the signal RSSI is compared to a configurable threshold level and analyzed for a valid FM/FSK signal. If the RSSI detection and FFT detection both detect a valid signal the ADRV9001 starts the wake-up procedure.

The settings for RSSI FFT mode are similar to RSSI Mode and FFT Mode respectively and the same limitations apply. Use this mode to avoid false positive detection of FM/FSK signal by checking that the signal level is valid also.

Note: this mode is currently not available.

Monitor Mode Advanced Features

Fast Buffer Read

A fast read of the buffer can be enabled when valid data has been detected. The incoming Rx data is continuously buffered and when the monitor mode enters the ‘detected’ state, the buffered data can be read out at 2, 4, or 8x the sample rate until the buffer is cleared. This mode

enables all the buffer data to be retrieved by the user, so no information is lost. To use this mode, it is required to use a higher clock rate as described in the Receive CSSI with Two, Four, and Eight Times Data Clock Rates section. When performing the fast buffer read operation, the IQ data is constantly read out onto the SSI bus with no idle time until the buffer is emptied.

As described in the Signal FIFO section, the buffer depth is 2048.

Note: this mode is currently only available in DMR profile and is limited to a 4x sample rate buffer read.

Enable External PLL

If using an external PLL, this needs to be enabled when the monitor mode enters the 'detecting' state. The ADRV9001 uses its SPI Main function to send a wakeup command to an external PLL. To use this function, set the SPI Main trigger source to 'Monitor' and upload the required SPI data needed to enable the external PLL. Whenever the monitor mode enters the 'detecting' state, the SPI data will be transmitted by the SPI main.

This feature allows further control of external components in monitor mode without the use of the BBIC. The BBIC is not needed in this process so can stay in power saving mode.

An example of how to use this feature is given in the Monitor Mode in TES GUI section.

Note: this feature is currently not enabled.

Reference Timer

In normal operation, the BBIC might uses a reference timer to synchronize timing between the base station and the mobile radio. When the BBIC goes into sleep mode, this reference timer could be lost. The ADRV9002 provides the ability to save this timer during the monitor mode operation and allow the BBIC to retrieve the reference timer when it wakes up. Before the monitor mode starts, the BBIC programs the ADRV9002 to mimic the reference timer. The reference timer then stays active during the full monitor mode cycle. On wake of the BBIC, the reference timer can be retrieved to synchronize its timing again.

There are two ways to set and retrieve the reference timer:

- SPI mode: the reference timer value is written to the ADRV9001 via an API. Note that API writes are non-deterministic which may introduce inconsistency.
- Pin mode: the reference timer is pulsed on a DGPIO which is read by the ADRV9001 to generate the reference timer internally. To retrieve the reference timer, the ADRV9001 pulses the timer onto a DGPIO for the BBIC to read.

Note: this feature is currently not available.

Monitor Mode with Frequency Hopping

Monitor mode can operate with frequency hopping. The difference between fixed frequency and frequency hopping in this mode is the requirement to retune to different frequencies during the monitor mode operation. Frequency hopping can operate as normal in either PLL Mux mode or PLL Retune mode. Monitor mode can continue to cycle between DETECTING and SLEEP states. When a valid signal is detected, the wake-up procedure is started.

Since the ARM processor is needed to operate frequency hopping, the hop signals must align with the detecting cycle of monitor mode. Alternatively, use power down modes 1 to 4 to keep ARM alive for the full monitoring cycle.

As the HOP signal needs to be externally generated by the BBIC, the BBIC will need to wake at least partially to send this signal and any other necessary procedures to operate frequency hopping.

POWER SAVING AND MONITOR MODE

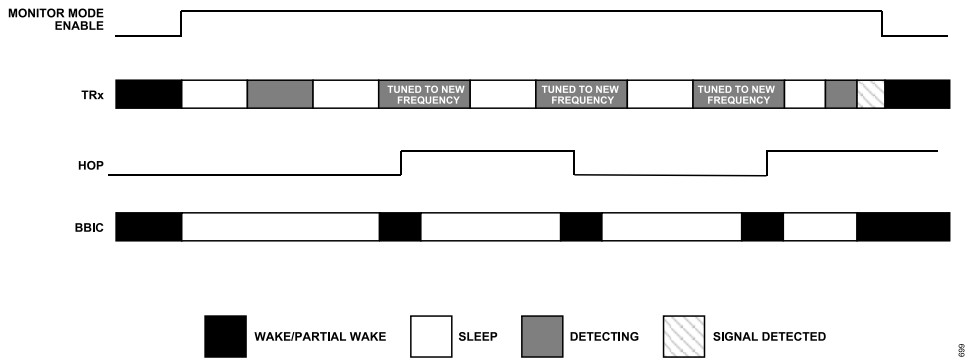


Figure 199. Monitor Mode with Frequency Hopping Timing Diagram

In Figure 199, the monitor mode with frequency hopping diagram is given. The monitor mode pin is enabled to start the process and is only disabled when a signal is detected. When the monitor mode is started, the ADRV9001 starts the sleep/detect cycle. BBIC can stay asleep during this process but must wake partially to send the HOP signal. The HOP signal is aligned with the DETECTING state and when the hop signal is received, the ADRV9001 is tuned to the required frequency based on the hop table. When a signal is detected, both the ADRV9001 and BBIC are woken to continue in normal operation.

Note: this feature is currently not available.

Monitor Mode in TES GUI

There are two monitor mode sections in the TES GUI.

The first section is under Configure -> Advanced features.

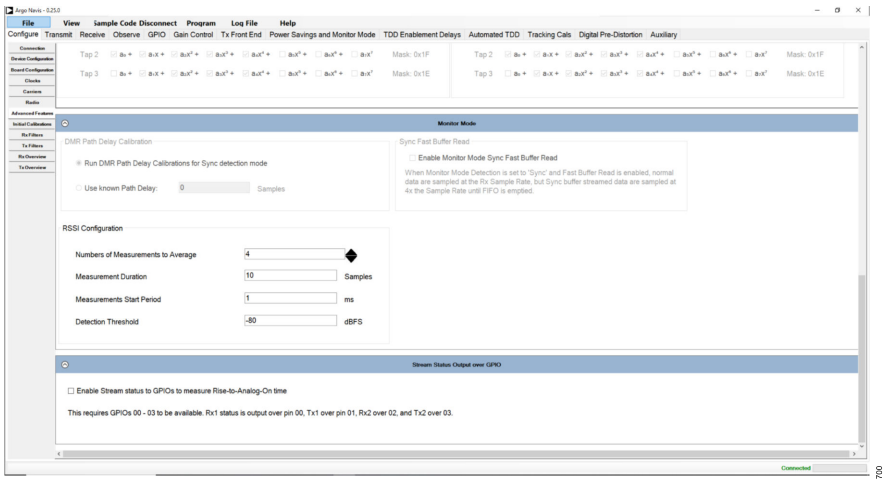


Figure 200. Monitor Mode TES GUI (Configure)

In this section, the following settings are available:

- DMR path delay calibration (see **adi_adrv9001_powerSavingAndMonitorMode... _MonitorMode_RxDmrPd_Run()**)
- Sync Fast Buffer Read (see the Fast Buffer Read section)
- RSSI Configuration (see the RSSI Mode section)

The second section is under 'Power Savings and Monitor Mode'.

POWER SAVING AND MONITOR MODE

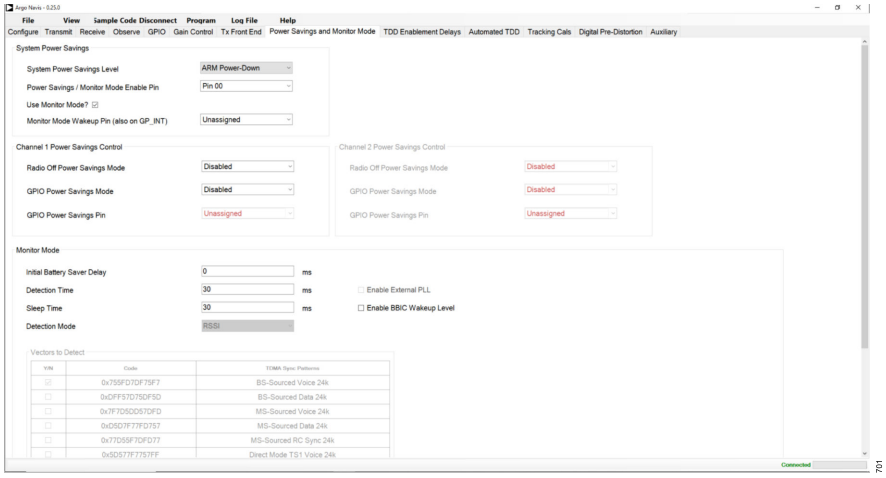


Figure 201. Monitor Mode TES GUI (Power Savings and Monitor Mode)

In this section, the following settings are available:

- Initial battery saver delay—the first detection time
- Detection time—all subsequent detection times
- Sleep time—all sleep times
- Detection mode
- Enable External PLL
- Enable BBIC Wakeup level
- Vectors to detect - for SYNC detection

Configure TES to your desired profile. As described in the Detection Modes section, some modes are radio standard dependent. Note: currently monitor mode is only enabled in DMR mode.

To enable Monitor Mode, ensure that the System Power Savings is not disabled and assign a pin to the Monitor Mode Enable Pin.

RSSI Detection Mode

To use RSSI detection mode, follow these steps:

1. Set the desired monitor mode RSSI settings.

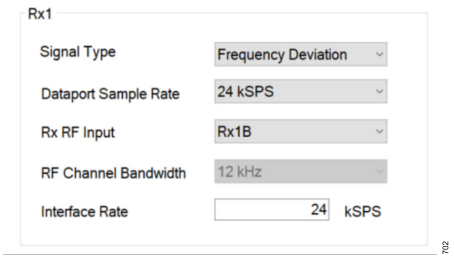


Figure 202. Sync Detect Settings 1

2. In the radio tab, set the Rx ADC to ‘High Performance’ (Note: this is a software limitation and could be expanded to the ‘Low Power’ option. However, LP ADC has reduced performance which might impact detection capabilities.).

POWER SAVING AND MONITOR MODE

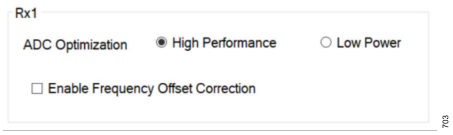


Figure 203. Sync Detect Settings 2

3. In the Receive tab, set the interface gain to 0 dB.

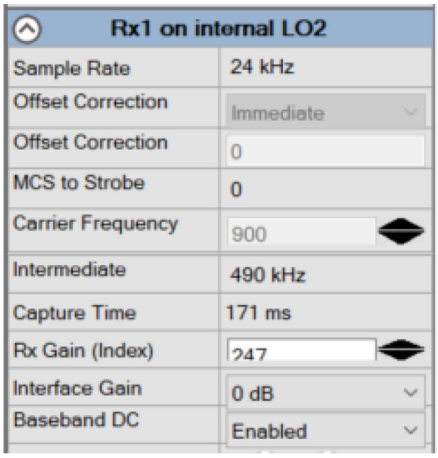


Figure 204. Sync Detect Settings 3

4. Either run the SYNC path delay or give a known value.

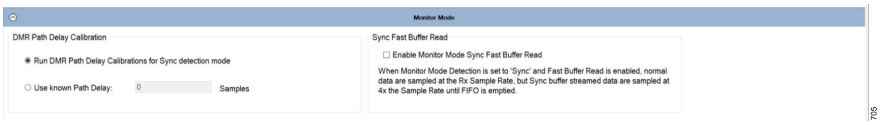


Figure 205. Sync Detect Settings 4

5. Now, select 'Sync' detection mode and enable the desired vectors.

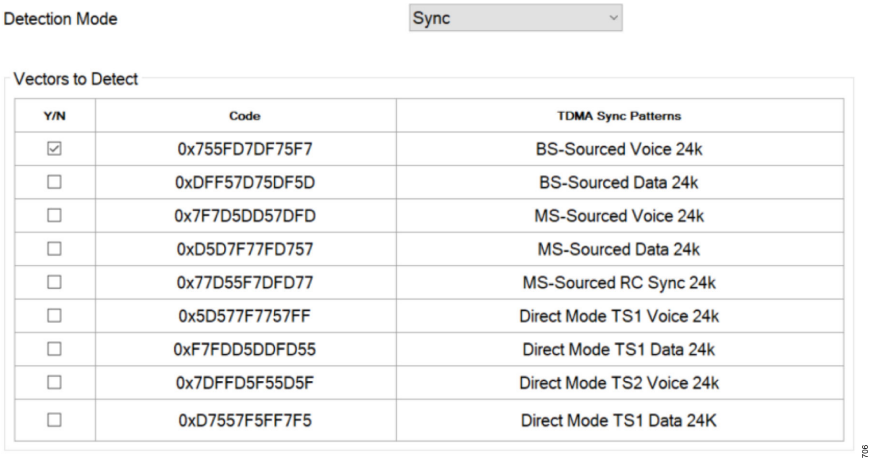


Figure 206. Sync Detect Settings 5

Enable External PLL with SPI Main

To enable an external PLL, follow these steps:

1. Use SPI main to send a wakeup command when the monitor mode enters the 'detecting' state.
2. Program the device, enable SPI Main from the 'Auxiliary' tab and set the trigger source to 'Monitor'.

POWER SAVING AND MONITOR MODE

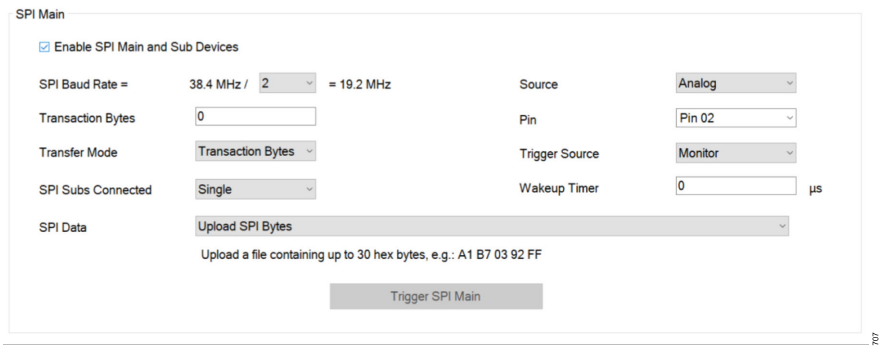


Figure 207. Enable External PLL with SPI Main Settings 1

3. Upload the SPI data needed to wake the external PLL.
4. Select the 'Enable External PLL' option.

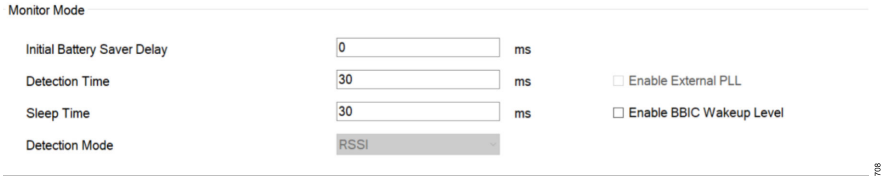


Figure 208. Enable External PLL with SPI Main Settings 2

DIGITAL PREDISTORTION (DPD)

BACKGROUND

One main criteria of a power amplifier (PA) operation is its ability to maintain linearity, i.e., the gain is constant regardless of the input amplitude. However, in practice, a PA can only maintain linearity up to a specific input level, beyond which the gain starts to lower, and the PA enters into a nonlinear or compression region, as shown in Figure 209. Most low-power linear amplifiers operate in the linear region, as shown in the "LINEAR REGION" circle. The PA that operates mostly in the linear region has lower efficiency. PA efficiency is defined as the ratio of the output RF power to the DC supply power. Therefore, it is desirable to operate the PA at high efficiency to save DC power and reduce heat dissipation.

To achieve higher PA efficiency, the highest input signal peak is usually set at around 1dB (P1dB) compression region, as shown in the "1dB COMPRESSION REGION" circle in Figure 209. However, compression of the peak signals produces harmonics and hence intermodulations. Some intermodulations fall back right into or next to the carrier spectrum, not only distorting the transmit signal but also widening the spectrum of the transmit signal, so called spectral regrowth. If left untreated, the error vector magnitude (EVM) performance of the transmit signal can degrade, and the spectral regrowth can interfere with adjacent channels, resulting in worse than required adjacent channel power ratio (ACPR) performance. Digital predistortion (DPD) mitigates this problem.

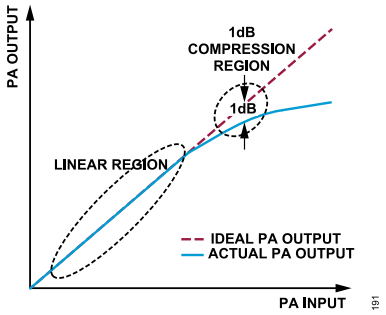


Figure 209. Ideal Power Amplifier Output vs. Actual Power Amplifier Output

ADRV9001 DPD FUNCTION

The ADRV9001 device provides a fully integrated DPD function that supports both narrowband and wideband applications. It is a hardware/software combined solution that linearizes the PA by predistorting the digital transmit signal with the inverse of the PA's nonlinear characteristics. After amplifying by the PA, the predistortion compensates for the PA's nonlinearity. So, the amplified RF transmit signal becomes linear. Therefore, the integrated DPD solution allows the PA to operate at very high efficiency while achieving a satisfactory EVM and adjacent channel power ratio (ACPR) performance.

Figure 210 depicts a high-level block diagram of the DPD algorithm and shows that before the PA, a "Predistortor" block is added in the transmit datapath, which distorts the transmit signal d(t) with the inverse of the PA's nonlinear characteristics, as shown by the first input/output figure curve. Spectral regrowth is introduced after the predistortion. However, after the "predistorted" transmit signal x(t) is amplified by the PA, the PA nonlinear characteristics, as shown by the second input/output figure curve, cancel out the predistortion. Therefore, the PA y(t) output becomes linear, as shown by the third input/output figure curve. In addition, the spectral regrowth after predistortion is also corrected. The "DPD Coefficients Computation" block is used to compute the predistortion parameters using the "Predistortor" output signal x(t) as well as the PA output signal y(t) through a feedback path. It models the behavior of the PA in the reverse direction, i.e., from output to input. Therefore, it characterizes the inverse of the PA nonlinearity and then feeds the parameters to the "Predistortor."

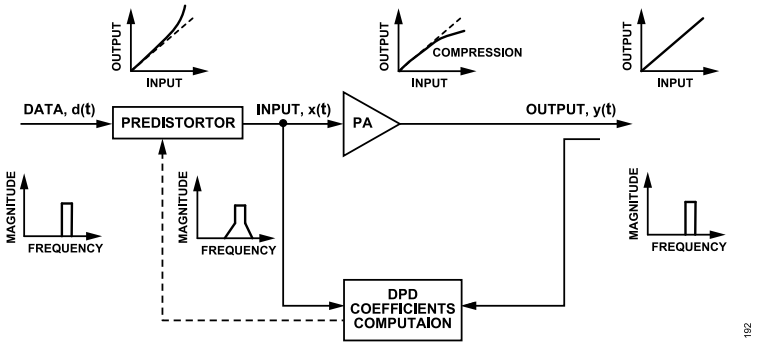


Figure 210. High Level Block Diagram of DPD Algorithm

DIGITAL PREDISTORTION (DPD)

In the ADRV9001 device, DPD is considered one of the transmitter tracking calibrations. It is a real-time signal processing with iterative updates to account for hardware variations such as temperature and power level changes. Similar to other transmitter tracking calibrations, it requires a loopback path from the transmitter to the observation receiver (ORx) to perform the calibration. In this case, an external loopback path (ELB) type 2 is required (see the Receiver/Observation Receiver Signal Chain section for more details about the loopback paths), in which the transmitter output signal after PA is looped back to the ORx, as shown in Figure 211. Ensure to establish this path before enabling the integrated DPD. In FDD applications, where only one receiver is used or in the TDD applications during transmit time slots, unused receiver path can be used for DPD calibration and other transmitter tracking calibrations. See the ADRV9001 Example Use Cases section for more details.

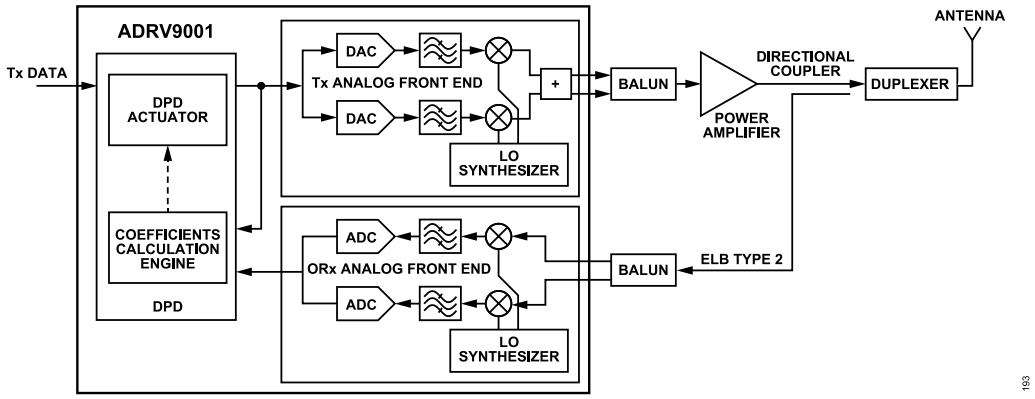


Figure 211. High Level Block Diagram of ADRV9001 DPD Implementation

Similar to what is shown in Figure 211, DPD includes two major components; a “DPD Actuator” and a “Coefficients Calculation Engine.” The “Coefficients Calculation Engine” computes the DPD coefficients periodically and then updates the “DPD Actuator” for real-time predistortion of the transmit signal. The predistortion coefficients are associated with polynomial terms defined by the PA model. To meet the real-time processing requirement, polynomial terms associated with a common time-delay input data are precomputed and stored into look-up tables (LUT) in the “DPD Actuator.” In the device, without frequency hopping, two LUTs are used for all waveforms. One is active to perform predistortion, while the other is updated in the background to track the changes and replace the current LUT when ready, resulting in complete transmit operation. The “DPD Actuator” also includes the functionality to calculate the amplitude of the input signal, which is used to search the LUT. The outputs of the LUT are then multiplied with different time-delayed input data according to the configured DPD model and combined to form the final predistorted transmit data.

ADRV9001 DPD SUPPORTED WAVEFORMS

The integrated DPD supports narrowband waveforms such as TETRA. Note: some narrowband standard waveforms, such as direct modulation types with constant envelope, do not require DPD. See Table 94 for TETRA's different modes of operations. The integrated DPD supports all TETRA 1 and 2 modes.

Table 94. Supported Narrowband Standards and Associated Operational Parameters

Standard	Bandwidth (kHz)	Modulation	Number of Carriers	PAR (dB) Before CFR
TETRA1	25	DQPSK	1	3.1
TETRA2	25	4 QAM	8	9.6
TETRA2	25	16 QAM	8	9.5
TETRA2	25	64 QAM	8	10.3
TETRA2	50	4 QAM	16	10.2
TETRA2	50	16 QAM	16	10.3
TETRA2	50	64 QAM	16	10
TETRA2	100	64 QAM	32	11.2
TETRA2	150	64 QAM	48	10.8

Besides that, the integrated DPD supports some wideband long-term evolution (LTE) and LTE-like waveforms with their associated operational parameters. Other wideband waveforms can be supported if the PA behavior fits the designed hardwired amplifier model, as well as the sampling rates and transceiver bandwidth.

DIGITAL PREDISTORTION (DPD)

Table 95 shows the supported wideband LTE standards and their associated operation parameters.

Table 95. Supported Wideband Standards and Associated Operational Parameters

LTE Bandwidth (MHz)	Number of Carriers	PAR (dB) Before CFR
1.4	Multicarrier	~11
3	Multicarrier	~11
5	Multicarrier	~11
10	Multicarrier	~11
15	Multicarrier	~11
20	Multicarrier	~11

Note: The ADRV9001 DPD can also be enabled for the LTE 40MHz profile with a slightly degraded performance, due to the limited DPD observation bandwidth.

Table 94 and Table 95 show that the multicarrier TETRA2 has a PAR between 9.6 dB to 11.2 dB, and the multicarrier LTE signal typically has a PAR of about 11dB. Therefore, to achieve higher PA efficiency and DPD algorithm stability, a waveform with a large PAR is expected to perform crest factor reduction (CFR) in baseband processors before DPD operation. Ensure that CFR is applied to a multicarrier signal before transmission as it keeps the average power higher while maintaining the same peak back off in the digital transmit data. Also, CFR suppresses large peaks to below a preset threshold, thereby eliminating occasional large peaks that can make DPD unstable if the peak is beyond the compression threshold limit of the power amplifier. For example, an LTE signal has a PAR of about 11 dB and the PAR can be reduced by CFR to about 7 dB by trading off the EVM. Note that the the baseband processor is responsible to perform CFR with an appropriate trade-off between the PAR reduction and EVM degradation before sending the transmit data to the ADRV9001. Note: The additional EVM degradation caused by the integrated DPD is negligible compared to the degradation caused by the CFR.

DPD WITH FREQUENCY HOPPING (FH)

The ADRV9001 also supports DPD operation during frequency hopping (FH). In this case, there is an option to identify multiple frequency regions, and for any LO frequency in one region, the same DPD solution is applied. The ADRV9001 allows a maximum of eight frequency regions. Define each region by specifying the lower bound and upper bound LO frequencies ([lower bound, upper bound)) for up to seven regions. The rest LO frequencies are in the eighth region. Refer to the Frequency Hopping chapter for more details.

ADRV9001 DPD PERFORMANCE

The integrated DPD algorithm has been tested using both metal–oxide–semiconductor (MOS) type and gallium nitride (GaN) type of PA. For example, Figure 212 and Figure 213 show the AM-AM and AM-PM performance of the raw transmit signal input vs. the MOS type of PA output before and after DPD without FH. The test waveform is TETRA1. Figure 213 clearly shows that the integrated digital predistortion successfully corrects the PA's nonlinearity.

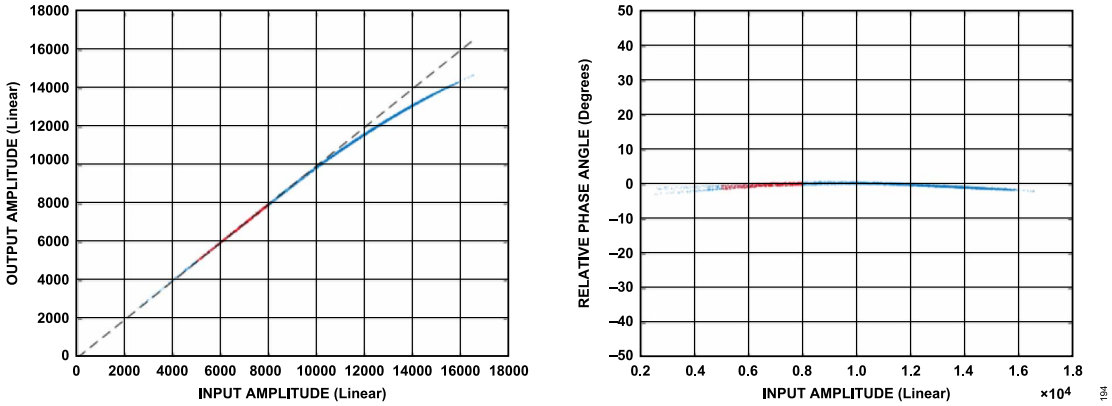


Figure 212. Raw Transmit Signal Input vs. Nonlinearized Power Amplifier Output

DIGITAL PREDISTORTION (DPD)

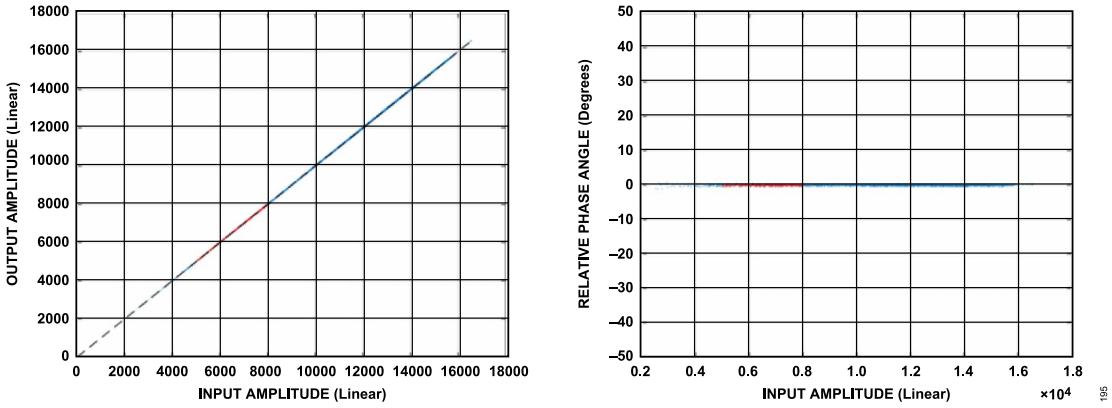


Figure 213. Raw Transmit Signal Input vs. Linearized Power Amplifier Output

Figure 214 shows an example of the ACPR performance before and after DPD. The blue curve represents the ACPR performance before DPD, which shows spectral regrowth. The black curve represents the ACPR performance after DPD. The ACPR performance significantly improves.

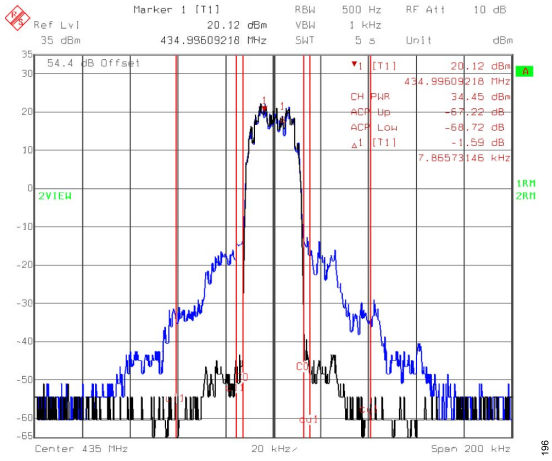


Figure 214. ACPR Performance Before and After DPD

Satisfactory DPD performance also depends on the successful completion of some operations inside the device, such as the ADC and DAC calibrations. Most of these operations are guaranteed internally in the device, but the following operations are user configurable. Therefore, perform all these optional operations is important to achieve the optimal DPD performance.

- Time alignment between transmit x(t) and loopback y(t) for data capture
- Transmit closed loop gain control (CLGC) tracking calibration
- Transmit local oscillator leakage suppression (LOL) calibration
- Transmit and receive quadrature error correction (QEC) calibration

CLOSED LOOP GAIN CONTROL (CLGC)

The ADRV9001 provides CLGC as one of the transmit tracking calibration algorithms. The sole purpose of CLGC is to maintain a fixed gain between the PA output and transmit digital input amplitude. The primary source of gain variation comes from the changes in the gain of the PA, which can vary due to transistor temperature change in response to power output in the short term, RF carrier frequency change within the bandwidth of the amplifier, and slow hardware degradation in the long term. Therefore, it is important to maintain a fixed gain in the transmit path to maintain a precise calibrated transmit power at the antenna port.

CLGC helps to achieve a fixed gain target by adjusting the transmit attenuation automatically by monitoring the gain variation of the PA. It can work with or without the DPD algorithm. When DPD is not activated, CLGC is an optional feature. Ensure the CLGC is always activated, when DPD is activated. Note: The ADRV9001 provides the option to enable DPD without CLGC, but it is not recommended. DPD compensates for the instantaneous compression of the peak signal by expanding the input signal so that the peak signals are linearized. When linearization is

achieved, the gain in the compression region is adjusted (increased) to match the gain of the lower linear region so that the overall gain is independent of the input or output average power level. This has the advantage of having CLGC focusing on compensating only the PA gain variation mainly due to temperature change. Furthermore, by setting a proper gain target, CLGC can also help to monitor and limit the transmit power level to keep the amplifier output power from rising beyond the linearization capable power limit of the PA. Therefore, it is crucial always to enable CLGC while DPD is active.

As the first step of CLGC, set up a target transmit gain. The target transmit gain can be measured through the ADRV9001 using the "CLGC Loop Open" method. The detailed steps of measuring the target gain are discussed later. After the measurement, the ADRV9001 provides an unfiltered and filtered transmit gain value. Based on these, further adjust the value and set a proper gain target, then close the loop and start the ADRV9001 CLGC algorithm to continuously track the gain variation based on the determined gain target.

When DPD is active, the PA gain is defined as the gain in the linear region of the AM-AM curve, as shown in Figure 212. To estimate this reference gain, data samples are usually selected in the upper linear region and below the compression region, as indicated in red in Figure 212. Note: This same gain plus relative phase is used by the DPD to scale the y(t) loopback data to match the predistorted transmit x(t) data. The PA gain derived from data between these bounds represents the actual gain in the linear region of the amplifier while excluding the distorted gain at the upper end due to compression. However, after DPD has converged, the gain in the compression region increases to match the gain in the lower region. When DPD is off, it must not correct the compression at the top region. Hence, it is necessary to include all samples for integration to estimate the total power, including the compressed region, to define the transmit gain. Define the region to calculate the gain through API configurations.

Similar to the DPD algorithm, the CLGC algorithm requires the time alignment between the transmit x(t) and loopback y(t) for data capture. Measure the delay and provide it to the ADRV9001, which is essential for wideband profiles. When the DPD is enabled, the same delay measurement serves both the DPD and CLGC algorithm.

Note: The ADRV9001 CLGC algorithm has a limit to track gain variations not exceeding ±3 dB (this must accommodate most types of PA), and for each CLGC iteration, the maximum gain adjustment is limited to ±0.5 dB to prevent the DPD algorithm becoming unstable.

DPD/CLGC CONFIGURATION

To use the integrated DPD/CLGC properly and ensure optimal performance, properly configure the DPD/CLGC parameters. Configure through the ADRV9001 TES or SDK. The configuration consists of two sets of DPD/CLGC parameters:"preinitial calibration" parameters, as they should be configured before performing initial calibration when the device is at the "STANDBY" state, and"post initial calibration" parameters, as they must be configured after performing initial calibration when the device is at the "CALIBRATED" state. The following subsections explain these DPD/CLGC parameters in detail.

DPD/CLGC Preinitial Calibration Parameters Configuration

To properly set the preinitial calibration parameters of the DPD/CLGC, there should be a general understanding of the DPD model used in the device. The following equations describe the DPD model.

$$x(n) = \sum_{t=0}^{T-1} \psi_t\left(|d(n-l_t|\right) d\left(n-k_t\right)$$

$$\psi_t\left(|d(n-l_t|\right) = \sum_{i=0}^{7} b_{t,l_t,i} a_{t,l_t,i} |d(n-l_t)|^i$$

where:

T is the total number of taps in the DPD model.

$\psi_t(|d(n-l_t)|)$ is the function implemented by the LUT for tap"t".

l_t and k_t are part of the hardware model representing the amplitude and data delay, respectively.

Optionally include/exclude each individual power term in $\psi_t(|d(n-l_t)|)$ by controlling the corresponding $b_{t,l_t,i}$, setting it to either 0 for excluding or 1 for including to better model the PA.

$a_{t,l_t,i}$ are coefficients estimated by the coefficients calculation engine and used to generate the LUTs by the DPD actuator. For $b_{t,l_t,i}$ and $a_{t,l_t,i}$, the subscript t represents the index for the tap, l_t represents the amplitude delay, and i represents the order of the power term. The ADRV9001 only supports 0 to 7th order power term in the function $\psi_t(|d(n-l_t)|)$.

DIGITAL PREDISTORTION (DPD)

Configure this set of DPD/CLGC parameters before initial calibration. It is defined by the following API data structure.

```
typedef struct adi_adrv9001_DpdInitCfg
{
     bool enable;
     adi_adrv9001_DpdAmplifier_e amplifierType;
     adi_adrv9001_DpdLutSize_e lutSize;
     adi_adrv9001_DpdModel_e model;
     bool changeModelTapOrders;
     uint32_t modelOrdersForEachTap[4];
     uint8_t preLutScale;
     uint8_t clgcEnable;
} adi_adrv9001_DpdInitCfg_t;
```

Table 96 briefly summarizes all the DPD/CLGC preinitial calibration parameters described in the above data structure.

Table 96. DPD Preinitial Calibration Parameters

Parameter	Type	Description	Default	Note
enable	bool	Sets "TRUE" to place the "DPD Actuator" in the datapath on the specified channel to prepare for DPD operation.	FALSE	Setting "TRUE" starts to apply the predistortion with the current DPD coefficients. But DPD update starts when the corresponding tracking calibration bit is set.
amplifierType	enum	Selects the type of amplifier. ADI_ADRV9001_DPD_AMPLIFIER_NONE = 0 ADI_ADRV9001_DPD_AMPLIFIER_DEFAULT = 1 ADI_ADRV9001_DPD_AMPLIFIER_GAN = 2	1	"1" is the only allowed PA type currently for both the MOS and GaN type of PA.
lutSize	enum	Determines the LUT size ADI_ADRV9001_DPD_COMPANDER_SIZE_256 = 0 ADI_ADRV9001_DPD_LUT_SIZE_512 = 1	1	Only 2 LUT sizes are supported currently.
model	enum	Selects the DPD model. ADI_ADRV9001_DPD_MODEL_0 = 0 ADI_ADRV9001_DPD_MODEL_1 = 1 ADI_ADRV9001_DPD_MODEL_3 = 3 ADI_ADRV9001_DPD_MODEL_4 = 4 Model 4 is the ADRV9001 model.	4	"4" is the only allowed DPD model currently. Always choose "4."
changeModelTapOrders	bool	Sets "TRUE" to use the model tap orders defined by "modelOrdersForEachTap". Sets "FALSE" to ignore "modelOrdersForEachTap" and use the default order.	FALSE	The default model tap order for the DPD model 4 is: [0] = 0x001F, [1] = 0x007F, [2] = 0x001F, [3] = 0x001E
modelOrdersForEachTap	array	A bit map for each of the taps in a model to indicate the power terms included in the model. Tap 0 and 2 must have the same order.	[0] = 0x001F, [1] = 0x007F, [2] = 0x001F, [3] = 0x001E	The default bit map represents the default tap order for model 4.
preLutScale	uint8_t (U2.2)	Describes the prescaling factor for the LUT.	2.0	Min = 1, Max = 3.75
clgcEnable	bool	Enables CLGC functionality.	False	Setting "TRUE" does not start the CLGC operation. CLGC starts when the corresponding tracking calibration bit is set.

The following sections describe each parameter.

DIGITAL PREDISTORTION (DPD)

enable, clgcEnable

The “enable” parameter places the “DPD Actuator” on the datapath of the specified channel to prepare for DPD operation. The “clgcEnable” parameter enables the CLGC functionality. Do this through TES under the “Advanced Features” tab, as shown in Figure 215. Note: Enable CLGC When DPD is enabled.

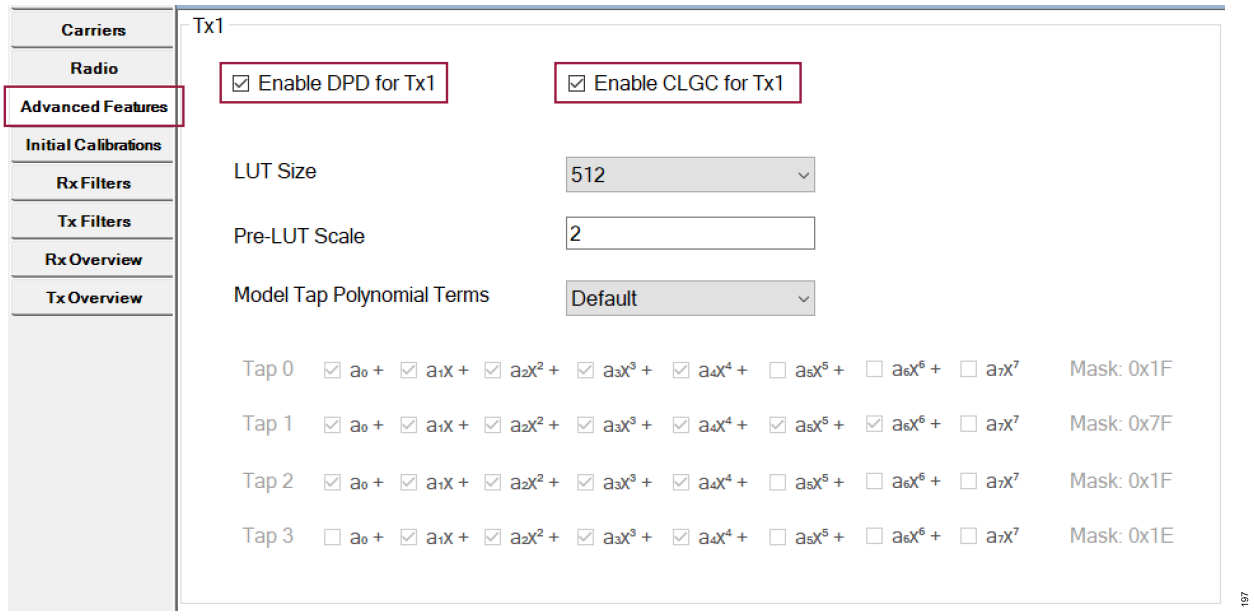


Figure 215. TES Configuration for Placing the DPD Actuator and Enable CLGC Functionality in the Datapath

Note: Setting it to “TRUE” is necessary but not sufficient to start the DPD/CLGC operation. The DPD/CLGC starts when the corresponding tracking calibration bit is also set, as shown in Figure 216.

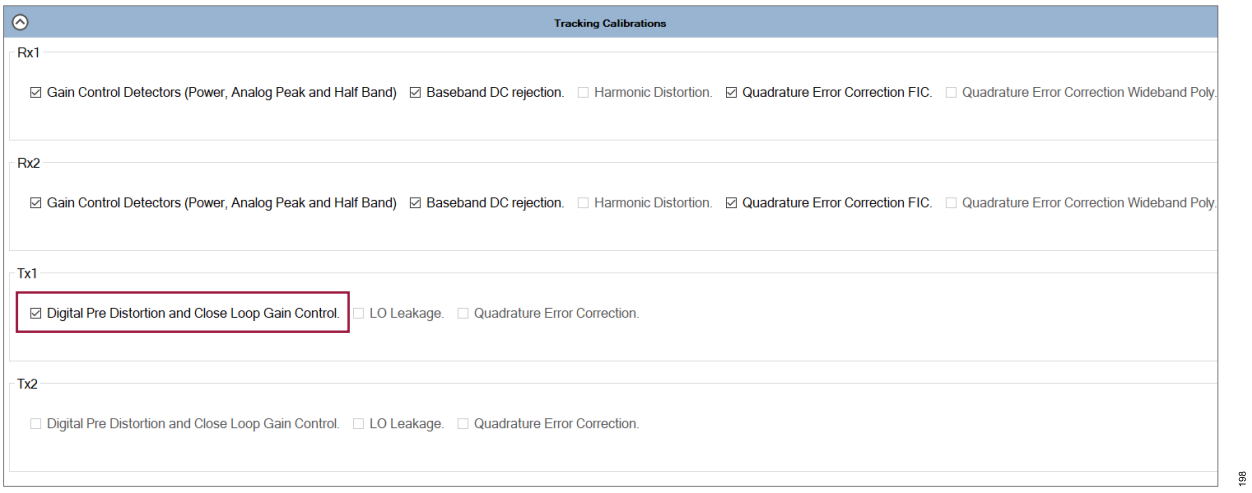


Figure 216. TES Configuration for Enabling DPD/CLGC Tracking Calibration

For the DPD/CLGC to work, the profile must indicate an external loopback connection and an external PA for this channel. Do this by setting “Board Configurations” in the TES properly, as shown in Figure 217.

DIGITAL PREDISTORTION (DPD)

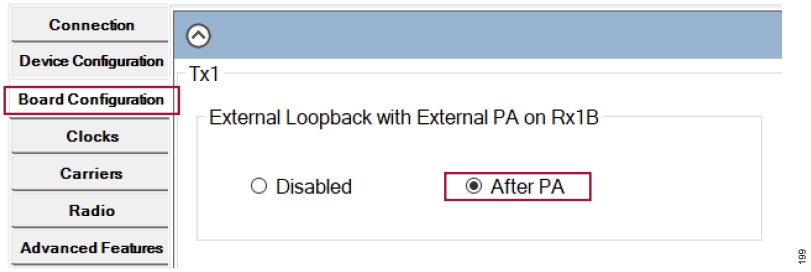


Figure 217. TES Configuration for External Loopback with External Power Amplifier

amplifierType

Currently, set the PA type to ADI_ADRV9001_DPD_AMPLIFIER_DEFAULT if DPD is enabled. The default PA type refers to both the MOS and GaN types of PA.

lutSize

Currently, the supported LUT sizes are 256 and 512. The size determines the number of entries in the DPD LUT. A more significant number of entries provides better LUT granularity.

model

Currently, set the model to ADI_ADRV9001_DPD_MODEL_4 only. There are other models for backward compatibility with other transceivers. Do not use them at this time. Model 4 consists of four taps (T=4), which can be described by the following equation:

$$x(n) = \sum_{t=0}^{3} \psi_t(\lfloor d(n-l_t) \rfloor) d(n-k_t) \quad (11)$$

Table 97 describes the delays for each tap.

Table 97. Delays of DPD Model 4

Tap	Delay of Data (k)	Delay of Magnitude (l)
0	0	0
1	1	1
2	2	2
3	1	2

Based on this, rewrite the equation as:

$$x(n) = \psi_0(\lfloor d(n) \rfloor) d(n) + \psi_1(\lfloor d(n-1) \rfloor) d(n-1) + \psi_2(\lfloor d(n-2) \rfloor) d(n-2) + \psi_3(\lfloor d(n-2) \rfloor) d(n-1) \quad (12)$$

where:

$\psi_0(|d(n)|)$, $\psi_1(|d(n\text{-}1)|)$, $\psi_2(|d(n\text{-}2)|)$ and $\psi_3(|d(n\text{-}2)|)$ represent four taps, generated by the LUT.

Figure 218 shows the DPD model 4 tap configuration used to generate the final predistorted data x(t).

DIGITAL PREDISTORTION (DPD)

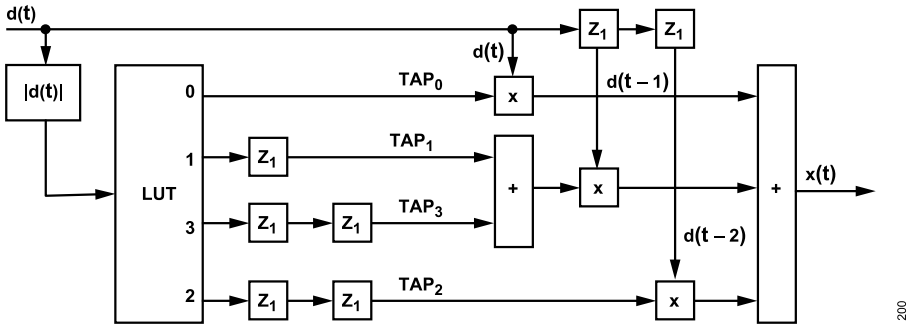


Figure 218. ADRV9001 DPD Model 4 LUT Configuration

Figure 218 shows that d(t) is the raw complex transmit signal before predistortion. Its amplitude is the basis used by the DPD actuator to predistort the d(t) through its LUT. The LUT consists of four taps calculated with precomputed DPD coefficients α, as shown here:

$TAP_0 = a_{0,0,0} + a_{0,0,1}|d(t)| + a_{0,0,2}|d(t)|^2 + a_{0,0,3}|d(t)|^3 + a_{0,0,4}|d(t)|^4$

$TAP_1 = a_{1,1,0} + a_{1,1,1}|d(t-1)| + a_{1,1,2}|d(t-1)|^2 + a_{1,1,3}|d(t-1)|^3 + a_{1,1,4}|d(t-1)|^4 + a_{1,1,5}|d(t-1)|^5 + a_{1,1,6}|d(t-1)|^6$

$TAP_2 = a_{2,2,0} + a_{2,2,1}|d(t-2)| + a_{2,2,2}|d(t-2)|^2 + a_{2,2,3}|d(t-2)|^3 + a_{2,2,4}|d(t-2)|^4$

$TAP_3 = a_{3,2,1}|d(t-2)| + a_{3,2,2}|d(t-2)|^2 + a_{3,2,3}|d(t-2)|^3 + a_{3,2,4}|d(t-2)|^4$

Note: The TAPx equations represent the default power term setting for each tap in the model 4, from which, derive $b_{t,l_t,i}$ as the following depending if a power term is included or excluded:

Tap 0: $b_{0,0,0} = 1$, $b_{0,0,1} = 1$, $b_{0,0,2} = 1$, $b_{0,0,3} = 1$, $b_{0,0,4} = 1$, $b_{0,0,5} = 0$, $b_{0,0,6} = 0$, $b_{0,0,7} = 0$

Tap 1: $b_{1,1,0} = 1$, $b_{1,1,1} = 1$, $b_{1,1,2} = 1$, $b_{1,1,3} = 1$, $b_{1,1,4} = 1$, $b_{1,1,5} = 1$, $b_{1,1,6} = 1$, $b_{1,1,7} = 0$

Tap 2: $b_{2,2,0} = 1$, $b_{2,2,1} = 1$, $b_{2,2,2} = 1$, $b_{2,2,3} = 1$, $b_{2,2,4} = 1$, $b_{2,2,5} = 0$, $b_{2,2,6} = 0$, $b_{2,2,7} = 0$

Tap 3: $b_{3,2,0} = 0$, $b_{3,2,1} = 1$, $b_{3,2,2} = 1$, $b_{3,2,3} = 1$, $b_{3,2,4} = 1$, $b_{3,2,5} = 0$, $b_{3,2,6} = 0$, $b_{3,2,7} = 0$

If using an array B for 4 taps, and for each tap using a byte to represent the above setting (the least significant bit represents the 0 power term), it is clear that the default setting is equivalent to B[0] = 0x1F, B[1] = 0x7F, B[2] = 0x1F and B[3] = 0x1E.

Combine the connections from the four outputs to produce the final output, x(t), as the following:

$x(t) = TAP_0[|d(t)|] \times d(t) + \{TAP_1[|d(t-1)|] + TAP_3[|d(t-2)|]\} \times d(t-1) + TAP_2[|d(t-2)|] \times d(t-2)$

changeModelTapOrders

This flag selects the default model tap orders or chooses customized model tap orders. If this flag is “TRUE,” use the next field in the data structure, “modelOrdersForEachTap, to set the model tap orders for the specified channel. If it is “FALSE”, then “modelOrdersForEachTap” is ignored and it uses the default tap orders as discussed (B[0] = 0x1F, B[1] = 0x7F, B[2] = 0x1F and B[3] = 0x1E).

modelOrdersForEachTap

This is an array of bitmaps $b_{t,l_t,i}(i = 0 \quad to \quad 7)$ for each tap t (t=0 to 3), formulated in the same way as discussed for the default setting. It provides option to customize the order so that a power term can be included or excluded in the polynomial to better model the PA . Table 98 shows recommendations to set this field. Try these suggestions and find the best model through tests. The DPD Tuning and Testing section discusses the method to select the best model tap orders as a part of DPD tuning recommendations.

Table 98. Suggested Model Orders for Narrowband Waveforms

Taps	Model Orders for Each Tap
Tap 1	B[1] = 0x1F, 0x3F, 0x7F, 0xFF
Tap 0 and Tap 2	B[0] = B[2] = 0x03, 0x07, 0x0F, 0x1F, (Tap 0 and Tap 2 should be the same.)
Tap 3	B[3] = 0x0, 0x02, 0x06, 0x0E, 0x1E, 0x3E

DIGITAL PREDISTORTION (DPD)

Configure the changeModelTapOrders and modelOrdersForEachTap through TES, as shown in Figure 219 and Figure 220. Figure 219 shows the default model tap configuration and Figure 220 shows a customized model tap configuration, which is equivalent to B[0] = 0x07, B[1] = 0x7F, B[2] = 0x07 and B[3] = 0x06. Note: Tap 0 and 2 are recommended to be the same. For simplicity, the GUI uses X to represent $|d(n-l_t)|$).

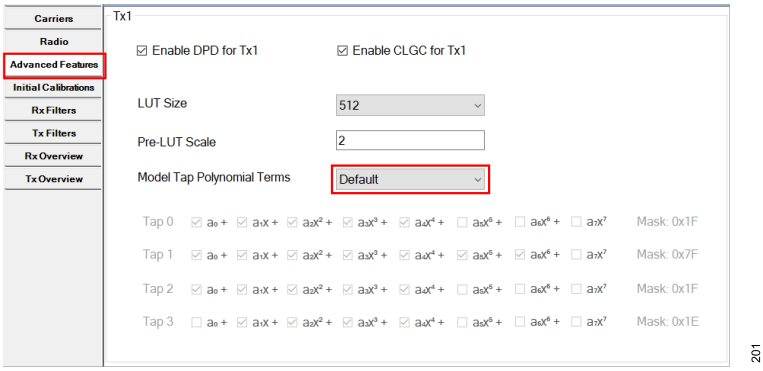


Figure 219. Configuring Default Model Tap Order Through TES

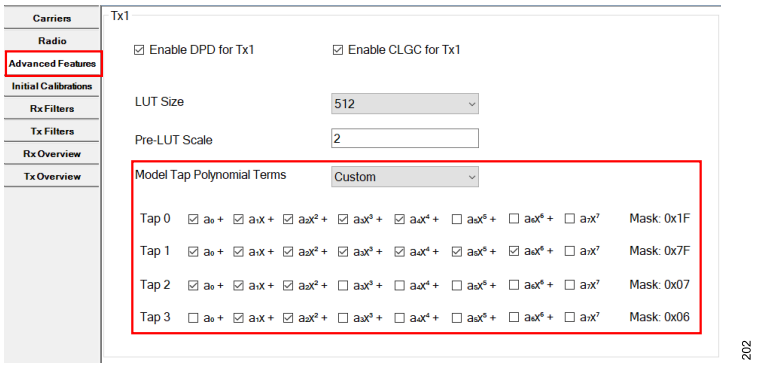


Figure 220. Configuring Customized Model Tap Order Through TES

preLutScale

This value, given as a fixed point U2.2 number, sets the scaling factor before searching the LUT. Set the scaling factor as 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, and 3.75. Scale the input signal magnitude to cover close to the full range of the LUT for better DPD performance. If the signal input to the compander is too small, then use only part of the LUT. When the input signal is small, try different scaling factors to increase the signal level, which might improve the DPD performance. The scaling factor can be determined according to the dBFS of the input data peak. As an example, if the signal peak power is less than −4 dBFS, apply the scaling factor 3.5. Figure 219 and Figure 220 shows the default value of “pre-LUT Scale” as 2, which can be further changed.

DPD/CLGC Post Initial Calibration Parameters Configuration

Configure the second set of DPD/CLGC parameters after initial calibration. It is defined by the following data structure:

```
typedef struct adi_adrv9001_DpdCfg
{
uint32_t numberOfSamples;
uint32_t additionalPowerScale;
uint32_t rxTxNormalizationLowerThreshold;
uint32_t rxTxNormalizationUpperThreshold;
uint32_t detectionPowerThreshold;
uint32_t detectionPeakThreshold;
uint16_t countsLessThanPowerThreshold;
uint16_t countsGreaterThanPeakThreshold;
```

DIGITAL PREDISTORTION (DPD)

```
bool immediateLutSwitching;
bool useSpecialFrame;
bool resetLuts;
uint32_t timeFilterCoefficient;
uint32_t dpdSamplingRate_Hz;
uint8_t clgcLoopOpen;
int32_t clgcGainTarget_HundredthdB;
uint32_t clgcFilterAlpha;
int32_t clgcLastGain_HundredthdB;
int32_t clgcFilteredGain_HundredthdB;
uint32_t captureDelay_us;
} adi_adrv9001_DpdCfg_t
```

Table 99 briefly summarizes all the DPD/CLGC post initial calibration parameters described in the data structure.

Table 99. DPD/CLGC Post Initial Calibration Parameters

Parameter	Type	Description	Min	Max	Default	Note
numberOfSamples	uint32_t	Specifies the number of samples to use for each iteration of DPD computation.	1024	4096	4096	The maximum value is preferred.
additionalPowerScale	uint32_t	Provides an estimate of the standard deviation of the modem input data magnitude to scale the data for internal DPD computation.	0	2^{32-1}	4	
rxTxNormalizationLowerThreshold	uint32_t (U2.30)	Signal power for the lower threshold for the normalization of the magnitude and phase of the RX and TX data.	0	1.0	0.0031622776602 (−25 dBFS)	
rxTxNormalizationUpperThreshold	uint32_t (U2.30)	Signal power for the upper threshold for the normalization of the magnitude and phase of the RX and TX data.	0	1.0	0.031622776602 (-15 dBFS)	
detectionPowerThreshold	uint32_t (U1.31)	Power threshold used for invalid capture detection.				
detectionPeakThreshold	uint32_t (U1.31)	Peak threshold used for invalid capture detection.				
countsLessThanPowerThreshold		If the number of samples below the detectionPowerThreshold exceeds this number, the capture is discarded.				To disable the detection, set it to 4096
countsGreaterThanPeakThreshold		If the number of samples above the detectionPeakThreshold is less than this number, the capture is discarded.				To disable the detection, set it to 0
immediateLutSwitching	bool	Determines whether the LUT switches immediately or at the end of Tx data frame.			TRUE	
useSpecialFrame	bool	DPD only runs on a user-indicated special frame.			FALSE	Currently not supported.
resetLuts	bool	Reset LUTs so that no predistortion is applied.			FALSE	Reset LUTs at the start of DPD operation.
timeFilterCoefficient	uint32_t	Coefficient of a time filter to remove spectral spikes from LUT switching.	0	1.0	0	It helps if there are spectral spikes from LUT switching, but can hurt convergence.

DIGITAL PREDISTORTION (DPD)

Table 99. DPD/CLGC Post Initial Calibration Parameters (Continued)

Parameter	Type	Description	Min	Max	Default	Note
						A coefficient of zero means no filtering.
dpdSamplingRate_Hz	uint32_t	Sampling rate in Hz for the DPD actuator and capture.				Read only. No effect on DPD configuration.
clgcLoopOpen	uint8_t	Open or close the gain loop.			0	If true, the loop is open and the TX attenuators are not updated. Used to measure a target gain.
clgcGainTarget_HundredthdB	int32_t	Set as the gain target.				
clgcFilterAlpha	uint32_t	Gain filter coefficient.	0	1	0.75	When it is 0, the filter is disabled.
clgcLastGain_HundredthdB	int32_t	Unfiltered gain measured when loop is open.				Only valid for user to retrieve.
clgcFilteredGain_HundredthdB	int32_t	Filtered gain measured when loop is open.				Only valid for user to retrieve.
captureDelay_us	uint32_t	Amount of time that DPD capture is delayed (beyond normal) relative to the start of the frame.	0		0	This parameter applies to both DPD and CLGC.

The following sections describe each parameter.

numberOfSamples

It specifies the number of samples used per DPD/CLGC data capture, with a limit of 4096. The DPD/CLGC performance can be improved if more samples are used. Currently, the linearization channel (LCH) is not supported for TETRA 1. So, set the numberOfSamples to 4096. When LCH is supported, change the numberOfSamples to a different value.

For LTE, set the number of samples to 4096 without frequency hopping.

additionalPowerScale

This parameter scales the higher power terms when calculating the auto-correlation matrix using transmit data d(n). It keeps the nominal magnitude of each of the power terms about the same to avoid ill condition of the correlation matrix. The scaling factor α can be defined as $\alpha \approx 2 \times std(d(n))$, where “std” stands for standard deviation. Measure α and then pass the information through this parameter.

rxTxNormalizationLowerThreshold/rxTxNormalizationUpperThreshold

These are required parameters to normalize the magnitude and phase of the receive and transmit data. These thresholds are used to pick a linear region to normalize the data. The chosen region must be below the compression point but above the noise. In the AM-AM and AM-PM plot shown in Figure 212, a possible choice of the linear region is highlighted in red. In general, set the rxTxNormalizationUpperThreshold to 0.5 of the peak signal amplitude, and set the rxTxNormalizationLowerThreshold to 0.3 of the peak signal amplitude. Once set, fix the threshold values and these should not vary from capture to capture. Therefore, by knowing the peak transmit signal in dBFS, set the rxTxNormalizationUpperThreshold to “peak Tx dBFS – 6 dB”, and set the rxTxNormalizationLowerThreshold to “peak Tx dBFS – 10.5 dB”. The peak transmit signal is usually set at P1 dB by adjusting the transmitter attenuation setting.

Enter these thresholds in dBFS through TES. If using API, use the linear numbers, which can be calculated as $10^{(threshold\_dBFS/10)}$.

detectionPowerThreshold

It detects an invalid data capture for the DPD/CLGC operation if a specified number of samples (countsLessThanPowerThreshold) are below the defined power threshold.

DIGITAL PREDISTORTION (DPD)

detectionPeakThreshold

It detects a valid data capture for the DPD/CLGC operation if a specified number of samples (countsGreaterThanPeakThreshold) is greater than the defined peak threshold. The DPD/CLGC operation needs a good set of large signal samples to properly model the PA compression behavior.

countsLessThanPowerThreshold

It defines the number of samples below the detectionPowerThreshold to determine an invalid capture. To disable it, set it to be the DPD/CLGC maximum number of capture samples of 4096.

countsGreaterThanPeakThreshold

It defines the number of samples greater than the detectionPeakThreshold to determine a valid capture. To disable it, set it to 0.

immediateLutSwitching

When a new DPD solution is formed, load the new solution into a spare LUT, which can then be switched with the active LUT. There are two options for LUT switching. When immediateLutSwitching is set to “TRUE”, the new LUT immediately, when ready, swaps out the active LUT. When immediateLutSwitching is set to “FALSE”, after the updated LUT is available, LUT swapping is triggered after the next data frame is completed. Note: This only applies to TDD operations. FDD systems should always use immediate LUT switching.

useSpecialFrame

To achieve optimal performance, DPD must capture the peaks of the signal. Some standards allow the transmission of special data used for DPD estimation. In other cases, the baseband processor can know in advance that a frame contains good data for DPD estimation. In these cases, the user can control which frames are used for DPD. The flag indicates the control of the frames that the DPD can capture. This is currently not supported. So, it should be disabled.

resetLuts

To start the DPD operation from a known state, reset the LUTs. By setting resetLuts to 1, it sets most polynomial terms to 0 to remove the predistortion at the beginning of the DPD operation.

timeFilterCoefficient

This parameter defines the coefficient of a single-pole filter, which mitigates the spectral spikes caused by LUT switching, especially in the FDD and narrowband use cases. Avoid spectral spikes in TDD by not performing an immediate LUT switch. The range of “timeFilterCoefficient” is between 0 and 1. If setting it to 0, it is equivalent to disabling this functionality. Experiment with this parameter and pick the optimal value to reduce the spikes. Note that enabling this feature might hurt the convergence time. The bigger this parameter is, the slower the convergence might be. Therefore, consider this method as an available tool when all other methods are exhausted and employ it with caution.

dpdSamplingRate_Hz

This parameter shows the sampling rate in Hz for the DPD actuator and capture. It is read only and does not have any effect on DPD configurations.

clgcLoopOpen

This parameter opens or closes the gain control loop. When it is set to 1, the transmit attenuators are not updated. This measures the gain to help determine the target gain. Once the target gain is set, set it to 0 to close the loop and start the CLGC operation.

clgcGainTarget_HundredthdB

Configure this parameter to notify the ADRV9001 about the gain target for CLGC with an accuracy of a hundredth of a dB.

DIGITAL PREDISTORTION (DPD)

clgcFilterAlpha

This parameter stands for the coefficient of a single-pole filter to smooth the gain measurement. The minimum value is 0, which is equivalent to disable this filter by using the instantaneous gain measurement result. The maximum value is 1, and the default value is 0.75. Set the bigger value to achieve smoother gain measurement results.

clgcLastGain_HundredthdB

This parameter represents the unfiltered gain measurement results when the loop is open. Use API to retrieve this value, which is in an accuracy of a hundredth of a dB. Note: This parameter is only valid for retrieving.

clgcFilteredGain_HundredthdB

This parameter represents the filtered gain measurement results when the loop opens (based on the user-configured filter coefficient). Use API to retrieve this value, which is in an accuracy of a hundredth of a dB. Note: This parameter is only valid for retrieving.

captureDelay_us

This parameter delays the DPD capture from the beginning of the transmitter frame by an amount of time (in µs) specified. By default, it is set to 0. However, It can help avoid using transmitter data at the beginning of a transmitter frame, such as preamble or control signal, which has a characteristic not suitable for DPD operation. Note: This parameter applies to both DPD and CLGC.

Configure the DPD/CLGC post initial calibration parameters through TES, as shown in Figure 221. Note: All data capture related configurations such as “Number of Samples,” “Capture Delay (us),” “Rx/Tx Normalization”, and “Activation” are also used by the CLGC algorithm.

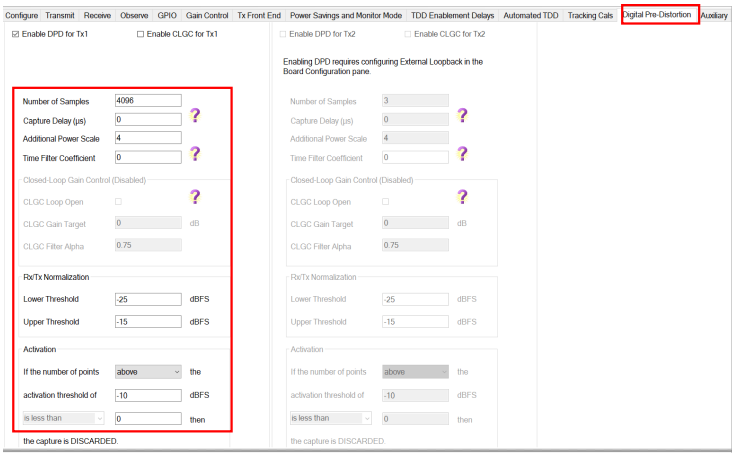


Figure 221. Configuring DPD/CLGC Post Initial Calibration Parameters Through TES

DPD/CLGC Updates Scheduling

As described in the Tracking Calibrations Scheduling section, tracking calibrations will run periodically in FDD profiles and at the start of a frame in TDD profiles. DPD will be scheduled to run approximately every 200 ms for the first four iterations and every 2 sec thereafter.

Optionally, user can allow DPD/CLGC tracking calibrations to run periodically during a TDD frame. This can be used in long frame times where DPD/CLGC may need to update during a frame time. In TES, this can be selected via the 'Repeated Estimation TDD' in a frequency hopping profile.

DIGITAL PREDISTORTION (DPD)

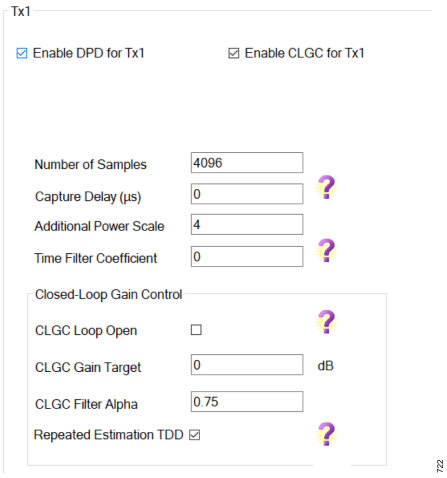


Figure 222. Periodic Updates of DPD/CLGC Enablement in TES

BOARD CONFIGURATION

Besides configuring the DPD/CLGC preinitial and post initial calibration parameters, configure two other parameters related to the board configuration: externalLoopbackPeakPower and externalLoopbackPathDelay. Provide these to the ADRV9001 before performing initial calibrations.

externalLoopbackPeakPower

It indicates the peak power of the ORx input signal loop backed from the transmitter output. For the DPD/CLGC to achieve optimal performance, set the ideal externalLoopbackPeakPower at about -18 dBm with a tolerance of ±5 dBm. Adjust the peak power using an external step attenuator after the PA.

externalLoopbackPathDelay

DPD/CLGC requires the alignment of the transmit signal capture x(t) with the external loopback capture y(t). The externalLoopbackPathDelay parameter compensates for additional delays on the external loopback path from the ADRV9001 transmit output to the ORx input. Measure this delay and provide it to the ADRV9001 before initial calibration. Then use the measured delay to compensate for the delay between x(t) and y(t). This parameter is critical, especially for wideband applications, due to the high sample rate. In narrowband applications, it is less critical. So, set it to 0 unless there is a larger delay in the external loopback path.

Set these two configurations through TES, as shown in Figure 223.

DIGITAL PREDISTORTION (DPD)

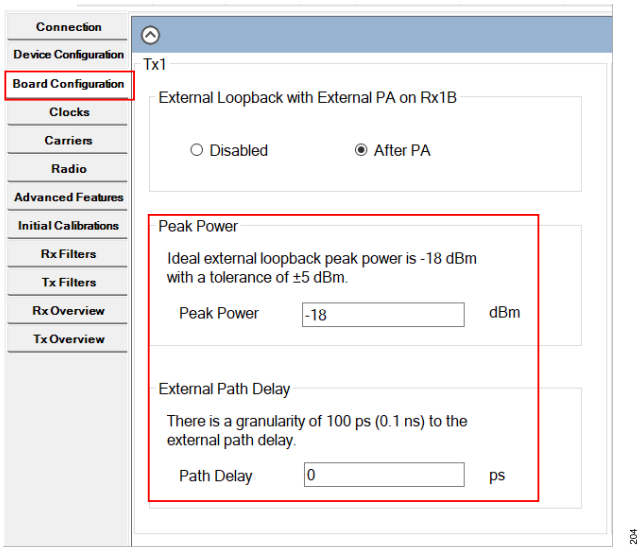


Figure 223. Configuring Board Configuration Related Parameters Through TES

SAVE AND LOAD DPD COEFFICIENTS FROM LAST TRANSMISSION

The ADRV9001 DPD also allows to save and load DPD coefficients from the last transmission. Therefore, the DPD can either start from scratch (unity coefficients) or a set of known coefficients. This is a very useful option to reach convergence quickly under a similar transmit operation condition. Use this feature as shown here in the TES.

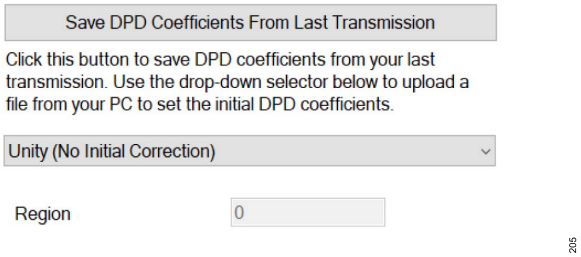


Figure 224. Save and Load DPD Coefficients from Last Transmission Through TES

DEFINE THE FREQUENCY REGION WHEN PERFORMING DPD WITH FH

Perform the DPD operation during FH. Define the seven frequency regions as shown here in the TES.

DIGITAL PREDISTORTION (DPD)

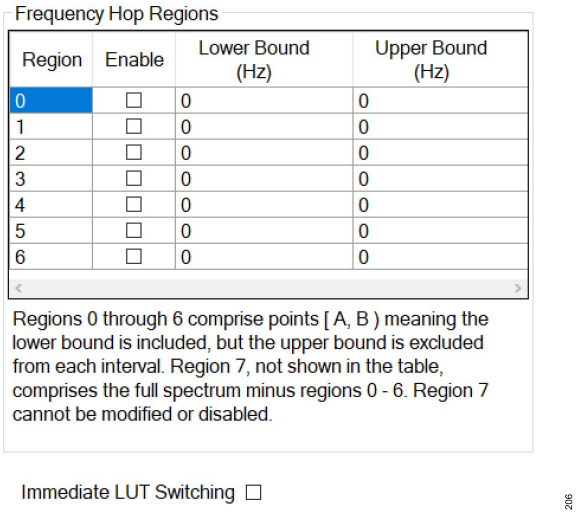


Figure 225. Define Frequency Hop Regions Through TES

Note: In this case, disable immediate LUT switching as the LUT switch only happens at the beginning of each hopping frame with the correct LUT for that LO frequency. In addition, the length of each hopping frame should be sufficient to capture a specified number of samples at the DPD sampling rate plus the additional time it takes for the system to set up the DPD tracking calibration. If this condition is not satisfied, DPD cannot be performed successfully with FH. For example, if the number of samples is set as 4096, for LTE20, the DPD sampling rate is at 184.32 MSPS, and the hopping frame must be longer than 4096/184.32 = 22.22 µs.

DPD/CLGC API PROGRAMMING

A set of API commands set and inspect the DPD/CLGC parameters summarized in Table 100. Set the board configuration parameters through the ADRV9001 initialization structure. Refer to the Doxygen document for more details.

Table 100. DPD APIs

DPD Rx Function Name	Description
adi_adrv9001_dpd_Initial_Configure	Configures the preinitial calibration DPD parameters. Called by adi_adrv9001_Utilities_InitRadio_Load() as part of device initialization.
adi_adrv9001_dpd_Initial_Inspect	Inspects the preinitial calibration DPD parameters.
adi_adrv9001_dpd_Configure	Configures the post-initial calibration DPD parameters.
adi_adrv9001_dpd_Inspect	Inspects the post-initial calibration DPD parameters.
adi_adrv9001_dpd_coefficients_Set	Sets DPD coefficients to be used at the next start of DPD.
adi_adrv9001_dpd_coefficients_Get	Gets DPD coefficients for the last solution.
adi_adrv9001_dpd_CaptureData_Read	Reads DPD captured data.
di_adrv9001_dpd_fh_regions_Configure	Configures DPD FH frequency regions.
adi_adrv9001_dpd_fh_regions_Inspect	Inspects DPD FH frequency regions.

DPD TUNING AND TESTING

Figure 226 describes an example setup to test the integrated DPD with the ADRV9001 evaluation board in narrowband applications (in narrowband applications such as TETRA, PA input should be connected to the TX1 output, and PA output should be connected to RX1B). Figure 226 shows that an LPF is required at the Tx1 output port to filter out the transmitter harmonics before feeding the signal to the PA driver (If the PA driver has an internal LPF, then the external LPF is not needed). As the device uses a square-wave mixer, it produces strong odd-order harmonics. Without filtering those harmonics, the DPD performance can be impacted. The step attenuators external to the ADRV9001 evaluation board are optional. Note: It is important to set up the external loopback path before operating the integrated DPD. To achieve optimal DPD performance for TETRA waveforms, it is recommended to use an external LO source for the transmitter due to possible better phase noise performance, while the receiver LO remains internal because the RF receive signal is downconverted to an intermediate frequency (IF) instead of directly to baseband. For wideband applications, the setup is similar, but both transmitter and receiver LOs can be set to be internal because a wideband signal is less sensitive to phase noise. Set up a spectrum analyzer to observe the adjacent channel power ratio (ACPR) performance during a DPD operation.

DIGITAL PREDISTORTION (DPD)

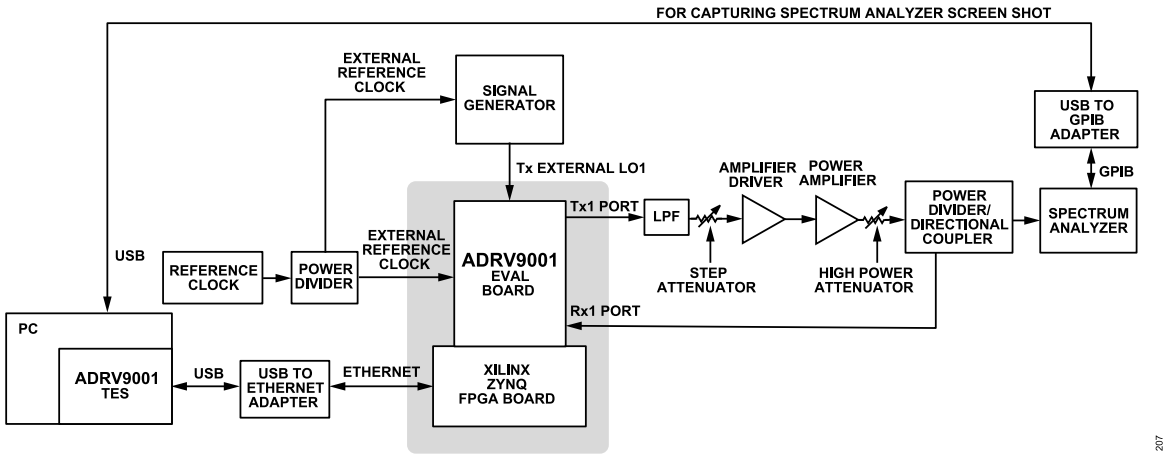


Figure 226. An Example Setup for Testing the Integrated DPD in Narrowband Applications

Once the setup is ready, further configure the TES and available external components properly, which includes the following major steps:

- Select the desired profile.
- Perform board configuration to indicate that external loopback path with external PA is available.
- Enter the peak power of the loopback signal (ideally, adjust it to -18 dBm ± 5 dB. Do this by tuning the external step attenuator).
- Measure the external loopback delay and provide it through TES. Do this through API commands, which are discussed at the end of this section.
- Configure other initialization parameters such as RF frequency, LO source, and so on, as desired. Also, enable DPD for the transmitter and configure the model tap polynomial terms. It is recommended to start with the default model tap. The next section discusses the method to tune the model tap order.
- Turn on the DPD tracking calibration and all other available tracking calibrations, and start with the default DPD post calibration parameter settings provided in the TES.
- After programming, load, and play the provided sample transmit input file.
- Properly tune the transmitter attenuation and/or the step attenuator to make sure that the ACPR performance at the device transmitter output is satisfactory before passing to the PA. In addition, make sure that the transmit peak signal is around the P1 dB compression region for optimal DPD performance.

Compare the ACPR performance through a spectrum analyzer with and without using the integrated DPD. Significant ACPR performance improvement with the integrated DPD should be observed even with internal LO sources. For TETRA waveforms, the ACPR after the second iteration of DPD is between −70 dB and −60 dB at an amplifier compression of P1 dB. For LTE waveforms, the ACPR after the second iteration of DPD is between −55 dB and −50 dB at an amplifier compression of P1 dB.

Tuning the Model Tap Order

DPD can be considered an adaptive filter modeled according to the behavior of the PA. The ADRV9001 default model (model 4) consists of four taps. Each tap consists of a series of polynomial terms to fit the nonlinear behavior due to compression at higher output power. The order of polynomial terms is determined by intermodulation falling closer to the carrier spectrum. In DPD, the orders of intermodulations to consider are usually the third, fifth, and seventh orders, with decreasing magnitude, respectively. An nth-order intermodulation expands the signal bandwidth 'n' times. By inspecting the bandwidth expansion factor on a spectrum analyzer, estimate how many orders of intermodulations to include in the polynomial terms, in order to suppress the spectral regrowth down to the required ACPR. It is important not to include higher order power terms than needed, which might cause the DPD to be unstable.

DPD model 4 consists of four taps, as shown in the example tap arrangement diagram in Figure 227. The four taps can be classified into three categories:

The main tap: The main tap of the DPD adaptive filter suppresses most of the spectral regrowth due to intermodulation. Hence, it has the greatest number of polynomial terms. It is labeled TAP_1.

DIGITAL PREDISTORTION (DPD)

The side taps: There are two side taps on each side of the main taps. They are memory terms that compensate for frequency-dependent distortion in the frequency domain and time misalignment between the transmit and receive captured data. The side taps have the same number of polynomial terms, and each side tap has about half of the number of polynomial terms of the main tap. They are labeled TAP_0 and TAP_2.

The cross-term tap: The cross-term further suppresses the residual spectral regrowth left over by the other three taps. The number of polynomial terms is usually equal to or less than that of each side tap. It is labeled TAP_3.

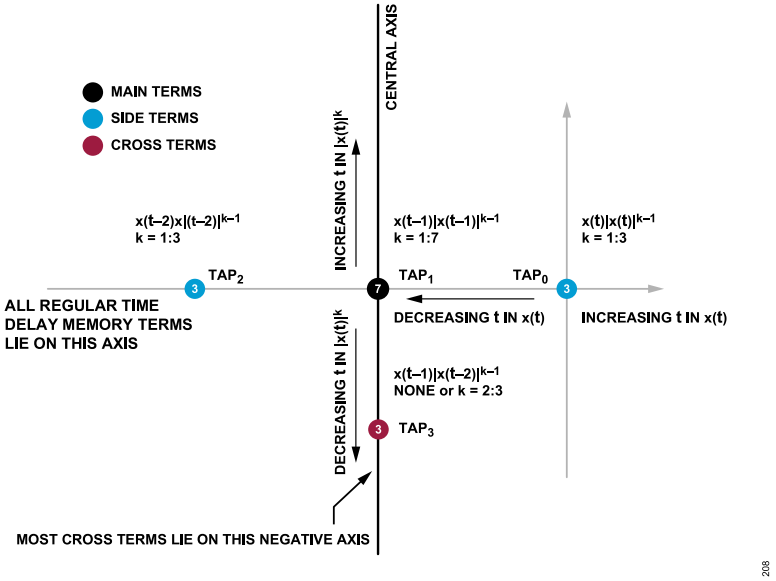


Figure 227. Example Polynomial Constellation Configuration for Model 4

Note that in Figure 227, k represents the order. To handle the seventh harmonics, the main tap must include the power terms up to k = 7.

To find the best model tap order for a specific PA design model, use the following recommended procedures:

1. Set the amplifier output power to have a compression ratio of 1 dB or slightly less, as shown in Figure 209, i.e., the maximum peak of the output signal is 1 dB below ideal linearity.
2. Determine the initial highest polynomial order of the main tap, TAP1, by measuring the spectral regrowth bandwidth to carrier bandwidth ratio. Include all lower-order polynomial terms. For example, if the bandwidth ratio is 5, set the initial highest polynomial to k = 5, i.e., $x(t-1)|x(t-1)|^4$
3. Set the other taps to zeros, I.e., turn off the other taps.
4. Use only TAP1 to execute DPD with order 5, then 6, and 7 (i.e., up to two orders above the initial estimate). Measure the ACPR for each case. Select the one that yields a better ACPR. If the difference is small, select a lower-order one, say 5.
5. While keeping the main tap determined above, set the side taps, TAP0 and TAP2, to about half the order of the main tap. In the above example, the main tap has an order of 5. Select the initial side tap order as 2. Execute DPD with the main tap of order 5 and the side tap order of 2, then 3, and 4. Select the side tap order that yields the lowest ACPR, say 3.
6. While keeping the main tap and side tap orders determined above, set the initial cross-term tap order, TAP 3, to be 2. Execute DPD with the cross-term tap order of 2 and 3. Select none, 2, and 3, which yields the best ACPR. If the difference is small, select the lower order one, including “none” taps. Some power amplifiers do not need a cross-term.
7. Iterate the above procedure as necessary with different combinations until confident with the selections. Keep all tap order selections to be minimal, that satisfies the ACPR requirement with a 5 dB margin, which helps to keep DPD more stable. For example, if the ACPR requirement is −60 dB, set the ACPR target as −65 dB.

Measuring the External Path Delay

Call the following API commands to measure and check the external path delay:

DIGITAL PREDISTORTION (DPD)

1: **adi_adrv9001_cals_ExternalPathDelay_Calibrate()**. Call this API when the channel state is CALIBRATED. It internally calls the following functions (no need to call these two lower level APIs).

- ExternalPathDelay_Run(), which runs external path delay calibrations.
- ExternalMinusInternalPathDelay_Measure(), which measures and gets the result of the difference in the path delays between the ILB and ELB and calculates the delay.

2: **adi_adrv9001_cals_ExternalPathDelay_Set()**. This API sets the external path delay value measured by "**adi_adrv9001_cals_External-PathDelay_Calibrate()**." Call this API when the channel state is in STANDBY and CALIBRATED only.

3: **adi_adrv9001_cals_ExternalPathDelay_Get()**. It gets the current external path delay value. Use this API to check the delay.

DPD Monitoring

Use DPD monitoring to check the running operation and status of the DPD process. Use the **adi_adrv9001_dpd_channel_Status_Get()** API to retrieve the DPD status for Channel 1 or Channel 2.

The following status and power measurements can be retrieved via the API:

- Number of iterations—the count of times the DPD process has started a capture
- Number of Successful iterations—the count of times the DPD process has completed and processed a capture successfully
- Tx Peak/RMS Power—the peak and average output power
- Rx Peak/RMS Power—the peak and average ORx input power

An example status from the TES GUI is shown in Figure 228.

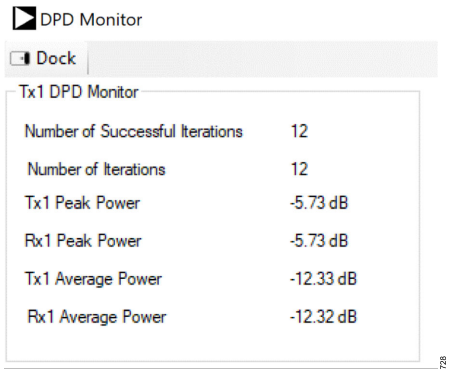


Figure 228. Example DPD Monitor Status

MEASURING THE CLGC TARGET GAIN

The user is responsible to set the gain target for the choice of PA. Determine the gain target in the lab using a PA with a typical gain at room temperature. The following guidelines provide the optimal operation of the ADRV9001. Fix the optimal gain target and adjustable gains in transmit and external loopback receive during operation, except for the transmit attenuation adjustment by the CLGC.

1. Set the transmit frequency to the middle of the band of interest.
2. Adjust the peak digital transmit signal amplitude to be between −6 dBFS at the DAC.
   - Margin 3 dB
   - Maximum DPD expansion 3 dB
3. Adjust the transmit attenuator so the PA input ACPR (before power amplifier) is about −70 dBc.
4. Connect the amplifier drivers and PA to the RF output.
5. Increase the transmit attenuation so that the peak compression is about 1 dB.
6. If a peak compression of 1 dB cannot be reached, increase the gain of a PA.
7. Adjust the gain of the external loopback path to make the feedback signal peak amplitude to around -18 dBm with a tolerance of ±5 dB.
8. Do not change these settings during normal operation. Otherwise, a new gain target has to be determined.

DIGITAL PREDISTORTION (DPD)

9. Use the TES to measure the transmit gain to help determine the gain target. To do this, enable the “CLGC Loop Open” option under the “Digital Pre-Distortion” tab in TES, as shown in Figure 229. Play a Tx modulated input signal and observe the “CLGC last gain” and “CLGC filtered gain” at the “Transmit” tab. Note: Apply a single pole filter to smooth the gain measurement, and set the filter coefficient (0<α<1), as shown in Figure 229. The measurement is smoother when a bigger 'α' is used. By default, it is set as 0.75.
10. By determining the final gain target based on the “CLGC last gain” and “CLGC filtered gain” provided by the ADRV9001, configure “CLGC Gain Target”, as shown in the Figure 229. After that, disable the “CLGC Loop Open” to close the loop to allow CLGC to achieve the gain target. Note: If the transmit attenuation is changed by anything else rather than the CLGC, factor the change in the gain target. To achieve the most accurate gain control, compensate the gain variation of the external components and ORx data path components by adjusting the gain target.

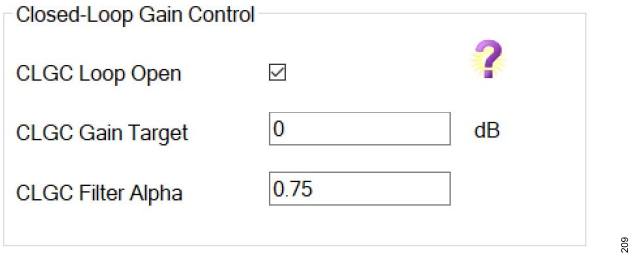


Figure 229. CLGC Configuration Parameters

DYNAMIC PROFILE SWITCHING (DPS)

OVERVIEW

Dynamic Profile Switching (DPS) is a feature supported by the ADRV9001 to switch among several predefined profiles with different signal bandwidths and sampling rates on the fly. With a switching time of about 50 µs, DPS enables a very fast change to a different profile without the need to reinitialize the chip. However, the sampling rates of different profiles must satisfy a relationship of multiple integers of two among each other. All profiles must be wideband profiles with a signal bandwidth of no less than 1MHz. For example, the set of LTE profiles including the sampling rate of 61.44 MSPS, 30.72 MSPS, 15.36 MSPS, 7.68 MSPS, 3.84 MSPS, and 1.92 MSPS satisfy the requirements. Therefore, DPS could be performed on these profiles. Besides the standard LTE profiles, the ADRV9001 also supports DPS on profiles with arbitrary sample rates. The maximum number of dynamic profiles is limited to six.

The ADRV9001 supports DPS for both TDD and FDD operations. When performing switching, the BBIC should first deactivate all channels. In the current release, the new profile is applied on all the configured transmitter and receiver channels simultaneously so that DPS cannot operate on channels individually. Figure 230 depicts a high-level diagram showing the DPS operation in a TDD system and Figure 231 depicts a high-level diagram showing the DPS operation in an FDD system, respectively.

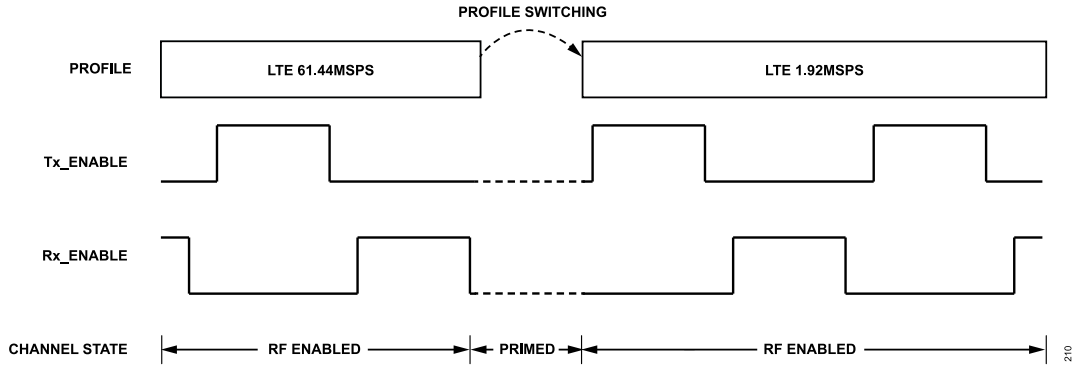


Figure 230. DPS Operation in TDD System

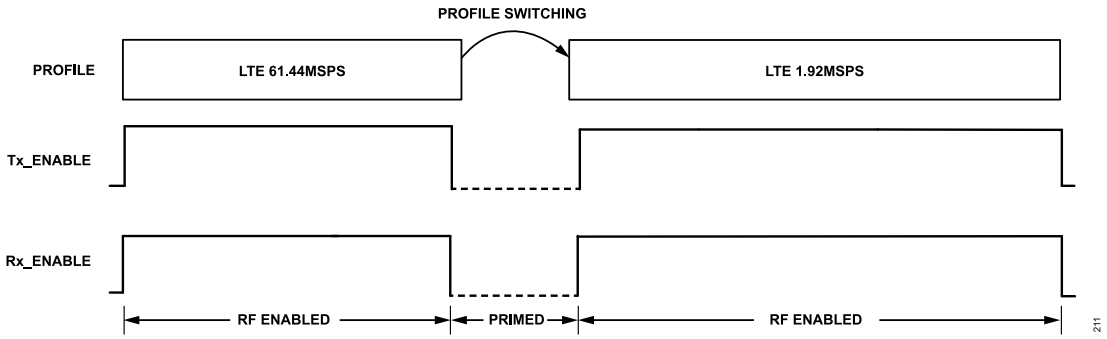


Figure 231. DPS Operation in FDD System

INITIAL CALIBRATION WITH DPS

From the ADRV9001 point of view, a dynamic profile change is considered a change in signal bandwidth and sampling rate relative to the main profile through changing decimation/interpolation settings in the datapath and receiver PFIR coefficients. As an example, for the set of LTE profiles , LTE 61.44 MSPS with the highest sampling rate is considered the main profile. During the initialization, configure more than one profile to enable DPS. Each profile has a profile index associated with it. Index 0 denotes the profile with the lowest bandwidth and sampling rate, and Index 5 denotes the profile with the highest bandwidth and sampling rate. During initialization, the user could enable receiver PFIR for each profile using either the default PFIR coefficients or providing a set of custom PFIR coefficients.

The ADRV9001 is first calibrated with the main profile as in the regular operation mode without DPS. Then, it is further calibrated for all the dynamic profiles using the API function **adi_adrv9001_cals_Dynamic_profiles_calibrate()**. When calibration is complete, the main profile is set as the initial profile to operate. Note: The SSI rate is configured based on the main profile, and it does not change during the entire profile switching operation.

Figure 232 describes the initial calibration procedure when DPS is enabled.

DYNAMIC PROFILE SWITCHING (DPS)

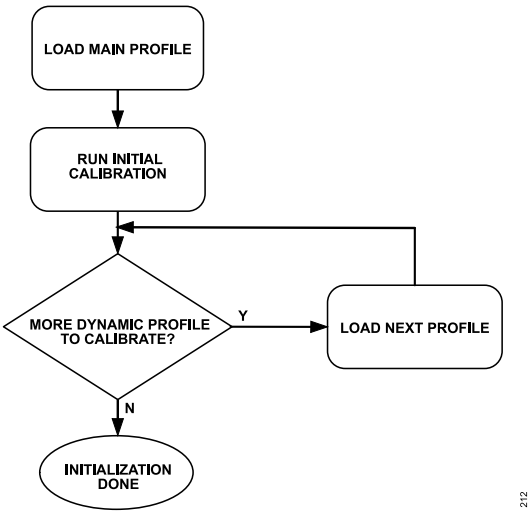


Figure 232. Initial Calibration with DPS

PERFORMING DPS ON THE FLY

After initialization, the ADRV9001 operates on the main profile with the fixed SSI rate, which does not change during the entire profile switching operation. To prepare for operating with the next profile, the BBIC should properly configure the ratio between the interface rate and new sampling rate before requesting profile switching. It must also notify the ADRV9001 about the next profile by calling the API command **adi_adrv9001_arm_NextDynamicProfile_Set()**. Furthermore, as an option, the BBIC can also set the receiver and transmitter PFIR coefficients associated with the next profile on the fly by calling the API command **adi_adrv9001_arm_NextPfir_Set()**. Note: These two APIs can be called in different channel states, including “Standby”, “Calibrated”, “Primed”, and “RF_enabled”. When the BBIC is ready to perform profile switching, it should first move all transmitter and receiver channels from the “RF_enabled” state to the “Primed” state and call the API command **adi_adrv9001_arm_Profile_Switch ()** to request the ADRV9001 to switch to the new profile. On receiving this command from the BBIC, the ADRV9001 starts to perform switching by applying the new profile and PFIR coefficients the BBIC set earlier, and it does not respond to any signals on the Tx_enable and Rx_enable pins. The ADRV9001 takes about 50 µs to complete the switch. After that, the BBIC can move the channels from the “Primed” state back to the “RF_enabled” state, and continue the transmit and receive operations with the new profile.

Figure 233 shows the procedure to perform DPS, and the communication between the BBIC and ADRV9001.

DYNAMIC PROFILE SWITCHING (DPS)

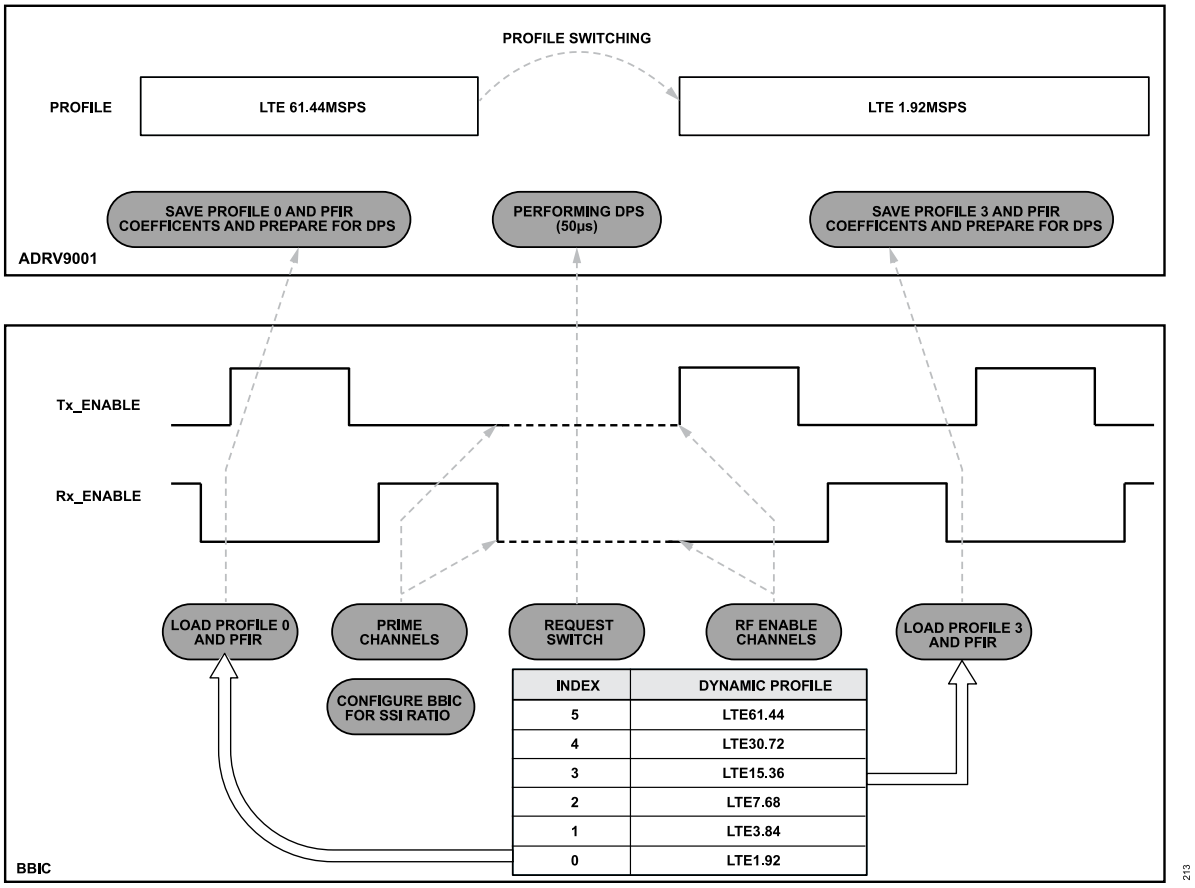


Figure 233. BBIC and ADRV9001 Interaction to Perform DPS

DPS API PROGRAMMING

Table 101 summarizes the set of ADRV9001 API commands for DPS. Refer to the API Doxygen document for more details.

Table 101. DPS APIs

DPD Rx Function Name	Description
adi_adrv9001_cals_Dynamic_profiles_calibrate	Runs the initial calibrations for dynamic profiles.
adi_adrv9001_arm_NextDynamicProfile_Set	Sends the next dynamic profile to the ADRV9001 and waits for it to process when profile switching is performed.
adi_adrv9001_arm_NextPfir_Set	Sends a bank of PFIR coefficients to the ADRV9001 and waits for it to process when profile switching is performed.
adi_adrv9001_arm_Profile_Switch	Requests the ADRV9001 to perform dynamic profile switching.

SUMMARY OF DPS LIMITATIONS

DPS allows to switch between different profiles very fast on the fly. However, to operate it properly, it is important to understand the limitations. The following list provides a summary.

- Arbitrary sampling rate is supported for DPS. But the highest sampling rate in a set of dynamic profiles should be no greater than 61.44MSPS and all profiles should be WB profiles with the bandwidth no less than 1MHz.
- The maximum number of profiles to configure for DPS is six.
- DPS operates simultaneously on all configured channels and cannot operate on any channels individually.
- All profiles support only the LVDS interface, and switching is not allowed between LVDS and CMOS.
- Switching among different TDD channel modes, different FDD channel modes, or TDD and FDD channel modes is not permitted with a profile change.
- The LO frequency of any channel cannot be modified with a profile change.
- The BBPLL frequency cannot be modified with a profile change.

DYNAMIC PROFILE SWITCHING (DPS)

- Transmitter/receiver TIA bandwidth and DAC/ADC sample rate cannot change with a profile switch.
- The physical user interface (for example, CMOS vs. LVDS, or interleaved data vs. non-interleaved data) cannot be modified with a profile change.
- SSI rate is the highest rate of all dynamic profiles.
- ADRV9001 operates in the frequency hopping or DPS mode, never both.

DPS OPERATIONS IN TES

The TES provides an interface to experiment with DPS. The user can only choose six profiles in the TES. So, the user must configure a profile with the sampling rate no less than 32 MSPS under the “Device Configuration” page in either the TDD or FDD mode. To enable DPS, under the “Rx Filters” tab, choose the “Number of Dynamic Profiles” as 6, and then define the receiver PFIR coefficients for all six enabled dynamic profiles. There is an option to disable PFIR, using the default PFIR or by uploading a set of custom PFIR coefficients, as shown in Figure 234.

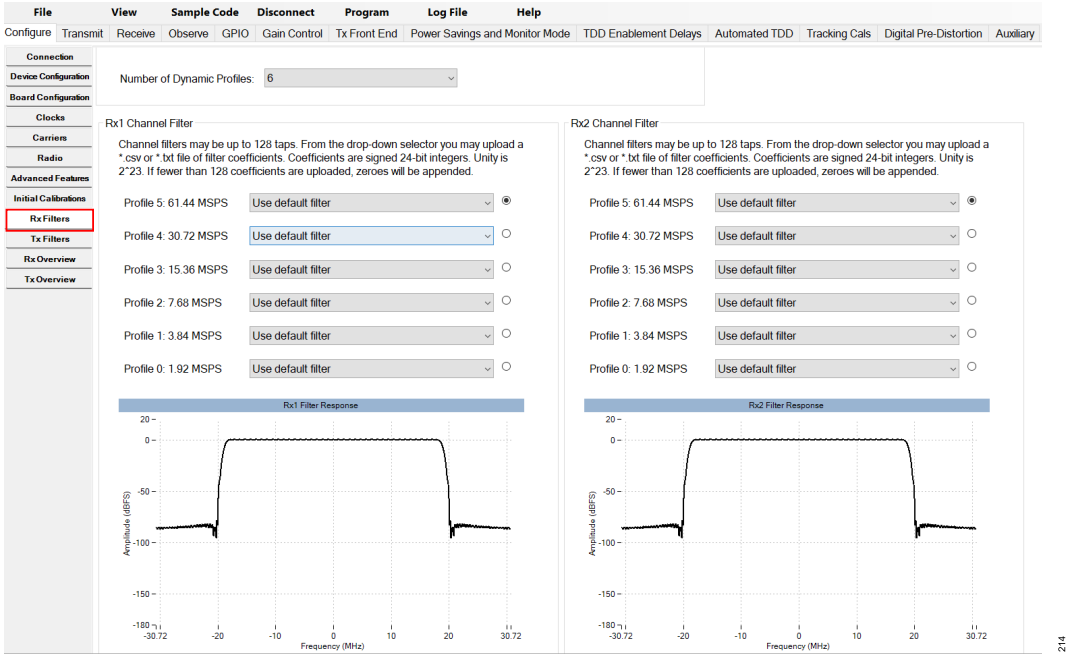


Figure 234. DPS Initialization in TES

The TES displays the frequency response of the PFIR for a certain dynamic profile upon selecting this profile.

After programming, the main profile is used as the initial profile to operate. To perform DPS, as shown in Figure 235, first move all channels (all transmitter and receiver channels) to the “Primed” state, then pick a new profile to request a profile switch from the “Sample Rate” dropdown under either “Transmit” or “Receive” tab. After that, play transmitter and/or receiver to observe the profile change and verify all channels work as expected.

DYNAMIC PROFILE SWITCHING (DPS)

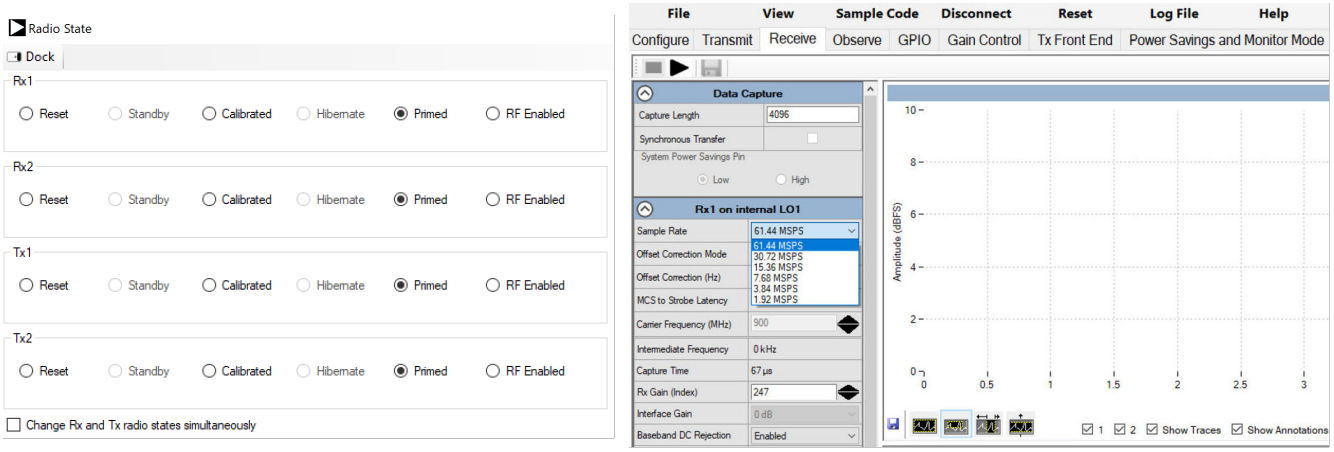


Figure 235. Performing DPS in TES

POWER AMPLIFIER RAMP CONTROL

For a normal TDD burst, the instantaneous transmitter power levels are constrained to the mask defined by the corresponding standards. The mask assures that the near-far situation results in co-channel and adjacent channel interference on the alternate or non-transmission slot. The mask also assures that the power level is adequate for acceptable bit error rate (BER) performance.

The RF power amplifier is an active device closest to the antenna. There should be precise control for the power amplifier output ramping up and down to fully comply with the transmitter mask requirements of the TDD standards. Ramp up/down the baseband IQ signals to control the power amplifier output level, or control the power amplifier power supply and bias voltage directly, to ramp up/down the power amplifier output level.

The direct power amplifier ramp control needs an independent microcontroller (MCU) through the monitoring channels to sense the power amplifier status and further generate the control targets based on the given function routine. Therefore, the ADRV9001 has highly integrated hardware and software functions to achieve the direct power amplifier ramp control function.

- The integrated AuxDACs, AuxADCs, and GPIOs in ADRV9001 provide sufficient monitoring and controlling channels.
- A fully digital hardware control loop in the ADRV9001 offers high controlling precision to overcome all sorts of non-ideal facts in the loop.
- The integrated ARM core in the ADRV9001 has the intelligence to protect and control the power amplifier. This intelligence includes auto calibration, always-on-time monitoring independent of SPI communication speed, customized self-decision response, and autonomous recovering.

The ADRV9001 supports open-loop and closed-loop power amplifier ramp controls. Apply either of them based on the system requirements.

POWER AMPLIFIER OPEN-LOOP RAMP CONTROL

Tune the power amplifier bias voltage (Vgs) to control the power amplifier output level. The Vgs range varies with different power amplifier types. An external amplifying and buffering circuit cooperating with the ADRV9001-integrated AuxDACs is necessary because the ADRV9001 AuxDAC 0.05 V ~ 1.75 V output voltage may not be enough for the power amplifier bias voltage control, Figure 236 shows an example of the power amplifier open-loop ramp control circuits, where the operational amplifier (OPA) can be AD8542 or ADA4692. This example uses two AuxDAC channels: one to generate the ramp waveform and another optional one to generate the offset voltage to compensate for the power amplifier characteristic difference.

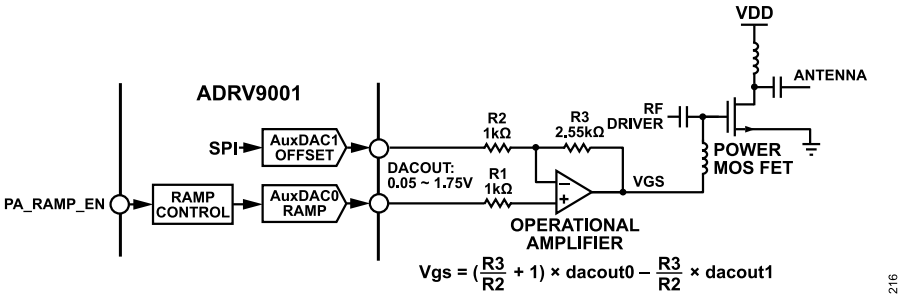


Figure 236. Power Amplifier Open Loop Ramp Control with ADRV9001 AuxDAC Output Amplifying and Buffering Circuits

Each transmitter channel of the ADRV9001 has an independent power amplifier ramp control logic, and an independent look-up table (LUT) storing the DAC codes for the ramp up/down waveform. The maximum depth of the LUT is 256. Flexibly configure the ramp-up and ramp-down waveforms with this 256 size LUT. Also set the full 256 size LUT for ramp-up and down simultaneously if the transmitter mask has symmetric up/down behaviors.

Power amplifier ramp enable can be triggered by SPI, TX_ENABLE, or a DGPIO. The TX_ENABLE, or DGPIO mode is recommended for accurate TDD operation timing. Figure 237 shows the power amplifier open-loop ramp control and timing. Power amplifier ramp up and down is triggered at the rising edge and falling edge of the power amplifier ramp EN (TX_Enable or DGPIO), respectively, with an optional start delay. The stored DAC codes in LUT is sent to AuxDAC accordingly. Calculate the ramp up/down duration by the ramp clock period times the relative LUT length.

POWER AMPLIFIER RAMP CONTROL

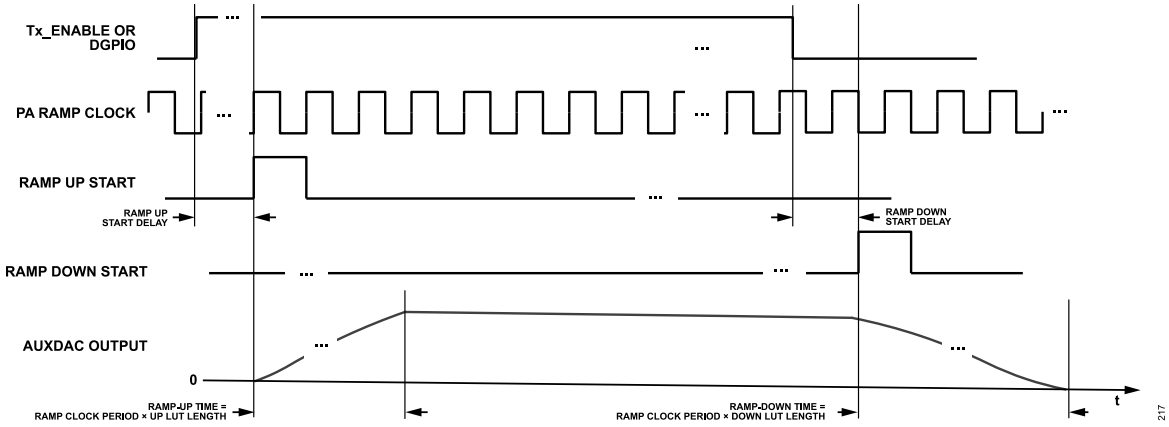


Figure 237. ADRV9001 Power Amplifier Open Loop Ramp Control and Timing

Use the API **adi_adrv9001_Tx_PaRamp_Configure()** to configure the necessary parameters for the power amplifier open-loop ramp control.

POWER AMPLIFIER CLOSE LOOP RAMP CONTROL

The power amplifier close loop ramp control has a current sensor loop through AuxADC, dedicated digital feedback control logic, and the power amplifier bias control through AuxDAC. Figure 238 shows an example of the circuits with the ADRV9001 power amplifier close-loop ramp control.

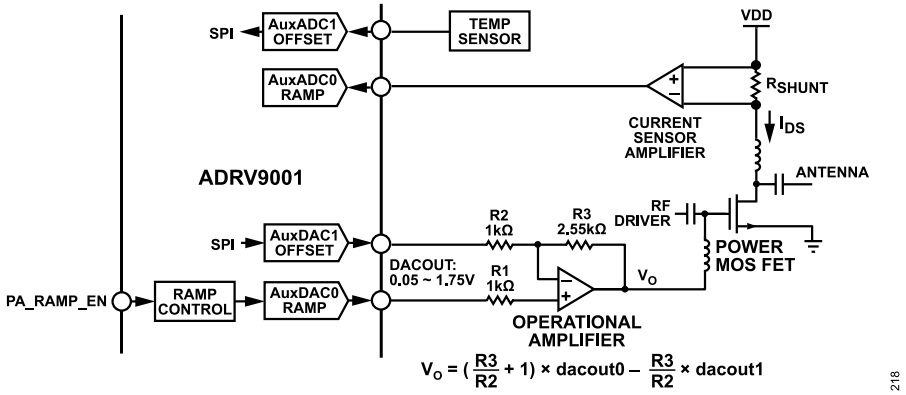


Figure 238. Power Amplifier Close Loop Ramp Control with ADRV9001 AuxDAC/AuxADC Amplifying and Buffering Circuits

In the open-loop control mode, the look-up table (LUT) stores the AuxDAC words, but in the close-loop control mode, the LUT stores the expectation of the AuxADC inputs vector. The manual control mode of the power amplifier close-loop ramp control can be applied in the power amplifier ramp factory calibration, and it helps to build the LUT vectors.

More details of power amplifier close loop ramp control are available when the software is supported.

GENERAL-PURPOSE INPUT/OUTPUT (GPIO) AND INTERRUPT CONFIGURATION

The ADRV9001 has a number of software configurable GPIO pins. Using API functions, users can configure the GPIO pins to operate with a variety of control or monitoring functions. The ADRV9001 has two types of GPIO:

- 16 digital GPIO pins (referenced to VDIGIO_1P8 supply, designated DGPIO_0 through DGPIO_15)
- 12 analog GPIO pins (referenced to VAGPIO_1P8 supply, designated AGPIO_0 through AGPIO_11)

The digital and analog GPIO pins can be used as real-time status signals that provide device status information from the ADRV9001 to the baseband processor when the GPIO pins are configured as outputs with respect to the ADRV9001. When set as inputs, the GPIO pins can be used as real-time control signals that can alter the device's state. The API functions related to the GPIO configuration allow the configuration of pins as inputs or outputs and the assigning of functionality to specific pins.

Figure 239 shows the different functionalities that can be enabled in the device and then controlled using the digital and analog GPIO pins. Not all functionalities can be enabled at the same time.

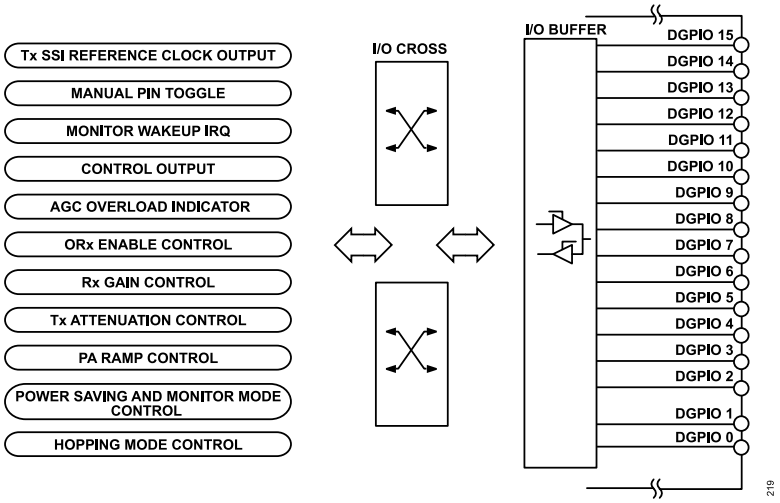


Figure 239. Digital GPIO Features Overview

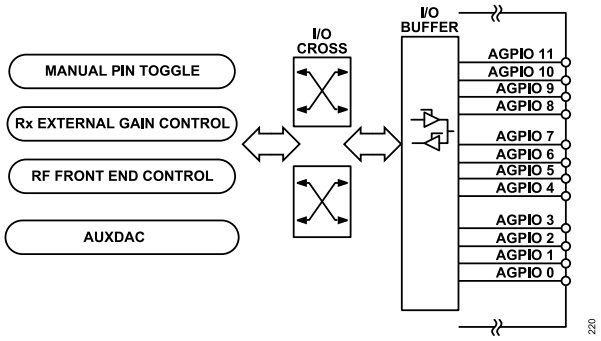


Figure 240. Analog GPIO Features Overview

In configuring the GPIO, two major factors to consider are: the GPIO output enable control and GPIO source control.

The output enable determines the direction of the pin. If a pin is set as output, then the GPIO I/O buffer is configured as an output.

The GPIO source control determines the functionality of the pin. The digital GPIO source control is assigned in groups of two. This means that DGPIO_0 to DGPIO_1 share a single source control, DGPIO_2 to DGPIO_3 share a single source control, and so on. The analog GPIO source control is assigned in groups of four, which means that AGPIO_0 to AGPIO_3 share a single source control, AGPIO_4 to AGPIO_7 share a single source control, and so on.

Use the API commands **adi_adrv9001_gpio_ControlInit_Configure()** and **adi_adrv9001_gpio_Configure()** to configure the digital or analog GPIO work modes. Some of the GPIO work modes are configured with the specified functions, for example, the pin-based transmitter attenuation, etc. The following subsections explain the operation details of the digital and analog GPIO.

GENERAL-PURPOSE INPUT/OUTPUT (GPIO) AND INTERRUPT CONFIGURATION

GPIO OPERATION

Set each digital GPIO pin to either the input or output mode. The input mode allows the baseband processor to drive pins on the ADRV9001 to execute specific tasks. The output mode allows the ADRV9001 to output various control or status signals to the baseband processor.

Note: There can be conflicts regarding GPIO usage when using combinations of certain features. Ensure that multiple functions are not assigned to the same GPIO pins.

Digital GPIO Input Features

Table 102 provides a list of digital GPIO input features that interact with datapath control elements on the ADRV9001. For these features, the APIs automatically set the I/O direction of the GPIO pins assigned for the feature.

Table 102. Summary of Input Digital GPIO Features

Feature	Description	GPIO Pins Available for Feature
ORx Enable	Configures specific digital GPIO pins to enable/disable the Rx observation channel.	DGPIO_0 through DGPIO_11: ORx enable control pin select.
Pin-Based Tx Attenuation Increment and Decrement	Configures specific digital GPIO pins to increase or decrease attenuation on any Tx channel after a rising edge on the assigned pin.	DGPIO_0 through DGPIO_15: Tx attenuation increment pin select. DGPIO_0 through DGPIO_15: Tx attenuation decrement pin select.
Pin-Based Rx Gain Index Increment and Decrement	Configures specific digital GPIO pins to increase or decrease gain index on any Rx or ORx channel after a rising edge on the assigned pin.	DGPIO_0 through DGPIO_15: Rx/ORx gain index increment pin select. DGPIO_0 through DGPIO_15: Rx/ORx gain index decrement pin select.
Tx Power Amplifier Ramp Control	Configures specific digital GPIO pins to ramp up the power amplifier controlling on any Tx channel after a rising edge on the assigned pin and ramp down the power amplifier controlling on any Tx channel after a falling edge on the assigned pin	DGPIO_0 through DGPIO_15: Power amplifier ramp controlling pin select.
Power Saving and Monitor Mode Control	Configures specific digital GPIO pins to enable or disable channel power saving, or system power saving, or monitor mode.	DGPIO_0 through DGPIO_11: Mon_enable pin select.
Hopping Mode Control	Configures specific digital GPIO pins to control the hopping mode, including the hopping enable, update gain value, frequency index, and so on.	DGPIO_0 through DGPIO_11: Hopping event pin select. DGPIO_0 through DGPIO_15: Hopping gain value pin select. DGPIO_0 through DGPIO_15: Hopping frequency index pin select.

Observation Receiver (ORx) Enable Control

The ADRV9001 Receiver can be reused as an observation channel through either port A or B. Assign a DGPIO pin as an ORx enable signal once the ORx channel is configured. The BBIC can toggle the DGPIO in the transmitter RF-enabled state to enable/disable the ORx channel. Note that ORx enable timing should only be the subset of the transmitter enable timing.

The enum **ADI_ADRV9001_GPIO_SIGNAL_ORX_ENABLE_1/ ADI_ADRV9001_GPIO_SIGNAL_ORX_ENABLE_2** is defined for DPGIO as the ORx enable signal. Set the DPGIO assignment by the API **adi_adrv9001_gpio_Configure()**.

Pin-Based Transmitter Attenuation Control

The Transmitter Signal Chain section provides a complete description of the transmitter attenuation control.

Pin-based Transmitter attenuation control provides an interface for attenuation adjustments with precise timing control. The pin-based control offers lower latency than SPI-based attenuation change operations. In pin-based attenuation control, certain digital GPIO pins are assigned the “increment attenuation” or “decrement attenuation” functionality. By applying a high pulse on the assigned GPIO pin, the attenuation for a specific channel is either increased or decreased, depending on the assigned functionality. The pulse-width requirement is at least two system clock cycles (184.32 MHz in standard profiles) in the logic-high state. Assign the increment and decrement functionality to any digital GPIO from DGPIO_0 to DGPIO_15. Pin-based transmitter attenuation control allows multiple increments or decrements of transmitter attenuation.

GENERAL-PURPOSE INPUT/OUTPUT (GPIO) AND INTERRUPT CONFIGURATION

Set the transmitter attenuation control mode to “MODE_PIN” by the API function **adi_adrv9001_Tx_AttenuationMode_Set()**, and select the appropriate GPIOs for each channel by the API function **adi_adrv9001_Tx_Attenuation_PinControl_Configure()**. The baseband processor can send the pulses to the ADRV9001 through the specific digital GPIO pins to increase or decrease the transmitter attenuation.

Pin-Based Receiver Gain Control

The Receiver Gain Control section provides a complete description of the receiver gain control.

Pin-based receiver gain control is relevant for applications that require manual gain control (MGC) and precise timing for gain change events. The pin-based control offers lower latency than SPI-based gain change operations. In pin-based gain control, certain digital GPIO pins are assigned the “increment gain index” or “decrement gain index” functionality for a particular receiver channel. By applying a high pulse on the assigned GPIO pin, the gain index for a specific channel is either increased or decreased, depending on the assigned functionality. The pulse-width requirement is at least two system clock cycles (184.32 MHz in standard profiles) in the logic high state. Assign the increment and decrement functionality to any digital GPIO from DGPIO_0 to DGPIO_15.

Note that for a programmed gain table that operates in a subset of the full gain table range (i.e., using index 195 to 255), once the gain Index reaches the min/max gain index, the subsequent pin-based receiver gain control rising edge does not change the gain index.

Set the receiver gain control mode to “MODE_PIN” by the API **adi_adrv9001_Rx_GainControl_Mode_Set()**. Configure the appropriate digital GPIO pins for the gain increase and decrease control and other control parameters by the API **adi_adrv9001_Receiver_GainControl_Pin-Mode_Configure()**. Then, the baseband processor can send the pulses to the ADRV9001 through the specific digital GPIO pins to increase or decrease the receiver gain index.

Power Amplifier Ramp Control

When the power amplifier ramp control function is used in the ADRV9001, an optional digital GPIO pin can be assigned as the “power amplifier ramp control enable” functionality driven by the baseband processor. The rising edge of power amplifier ramp control enable with programmable delay acts as the ramp up trigger signal, and the falling edge of power amplifier ramp enable with optional programmable delay as the ramp down trigger signal.

The DPGIO assignment for the power amplifier ramp control can be set by the API **adi_adrv9001_Tx_PaRamp_Configure()**.

Power Saving and Monitor Mode Control

The DPGIO can be used as the channel power saving, system power saving, and monitor mode enable signal. See the Power Saving and Monitor Mode section for the details.

Enum **ADI_ADRV9001_GPIO_SIGNAL_MON_ENABLE_SPS** and **ADI_ADRV9001_GPIO_SIGNAL_POWER_SAVING_CHANNEL1/ ADI_ADRV9001_GPIO_SIGNAL_POWER_SAVING_CHANNEL2** are for the DPGIO as system power saving/monitor mode and channel power saving enable, respectively.

When the CPS and/or SPS/monitor mode is enabled, the BBIC can call the API **adi_adrv9001_gpio_Configure()** to set the power saving and monitor mode control to enable signals on the DGPIOs.

Hopping Mode Control

Assign a DPGIO as a frequency hopping control signal. Also use the DGPIOs to choose the hop table index, receiver gain table index, transmitter attenuation index, etc. See the Frequency Hopping section for more details.

Set the hopping mode control GPIO functions through the API **adi_adrv9001_fh_Configure()**.

Digital GPIO Output Features

Table 103 shows the available digital GPIO output features. The relative API automatically sets the GPIO I/O directions.

Table 103. Summary of Digital GPIO Output Features

Feature	Description	GPIO Pins Available for Feature
Control Out Mux	Allows a choice of Main/RX/TX control signals to output from the ADRV9001 to monitor the status of the device.	DGPIO_0 through DGPIO_11

GENERAL-PURPOSE INPUT/OUTPUT (GPIO) AND INTERRUPT CONFIGURATION

Table 103. Summary of Digital GPIO Output Features (Continued)

Feature	Description	GPIO Pins Available for Feature
Manual Pin Toggle	Manually control the GPIO output level. API functions set the output pin levels and read the input pin levels.	DGPIO_0 through DGPIO_11
Monitor WakeUp Baseband Processor/DSP	Interrupt signal to wake the baseband processor/DSP when it is in the sleep state.	DGPIO_0 through DGPIO_11
Rx AGC Overload Indicator	Allows output of the AGC overload signals.	DGPIO_0 through DGPIO_11
TX DCLK OUT	Allows output of the SSI reference clock for the baseband processor to generate the TX SSI clock, data, and strobe to the ADRV9001.	DGPIO_12 through DGPIO_13 TX Channel 1 SSI reference clock out pin select DGPIO_14 through DGPIO_15 TX Channel 2 SSI reference clock out pin select

Control Out Mux

Control out mux (monitor out) allows status signals within the ADRV9001 to be output to digital GPIOs, such as the AGC mode of the gain change flag, gain index can be mapped to the DGPIO for BBIC observation by the API **adi_adrv9001_Rx_GainIndex_Gpio_Configure()**.

Map the ADRV9001 internal stream status to the DGPIO for the accurate transmitter/receiver enable control effective timing measurement. Call **adi_adrv9001_Stream_Gpio_Debug_Set()** to enable this feature. Configure DGPIO0~3 to represent the Tx/Rx enable control effective timing. See Pin Control Mode Timing Measurement for more details.

Manual Pin Toggle

The manual pin toggle feature controls the logic level of individual digital GPIO pins. **adi_adrv9001_gpio_ManualOutput_Configure()** configures the relative GPIO to the manual control mode. The **adi_adrv9001_gpio_OutputPinLevel_Set()** command sets the output levels of the GPIO pins. **adi_adrv9001_gpio_OutputPinLevel_Get()** command reads the output levels of the GPIO pins.

Additionally, the **adi_adrv9001_gpio_InputPinLevel_Get()** command reads the input GPIO level if the relative GPIO is configured as the input by the **adi_adrv9001_gpio_ManualInput_Configure()** command.

Receiver AGC Overload Indicator

Retrieve the status of peak and power detectors in the receiver channel to the baseband processor through a set of DGPIO pins. One DGPIO configuration uses the peak detect mode, which has the overrange and underrange conditions of both the analog peak detector (APD) and half band (HB) detectors. The other DGPIO configuration uses the peak/power detect mode, which has the overrange and underrange conditions of the APD and power detector.

The data structure of **adi_adrv9001_GainControlCfg_t**, and its substructures, **adi_adrv9001_PeakDetector_t** and **adi_adrv9001_PowerDetector_t** initialize the necessary gain control parameters as well as the digital GPIO pins assignment for the overload indicator. The API command **adi_adrv9001_Rx_GainControl_Configure()** sets the parameters. (See the Receiver Gain Control section for more details.)

Monitor Wake-Up Baseband Processor/DSP

Assign certain digital GPIO pin as “wake-up baseband processor/DSP” to output the interrupt signal to wake the baseband processor/DSP when the ADRV9001 works in the monitor mode and specific detection conditions are met.

Use the enum “ADI_ADRV9001_GPIO_SIGNAL_MON_BBIC_WAKEUP” as the DPGIO for the monitor wake-up interrupt signal. Call the API **adi_adrv9001_gpio_Configure()** to enable this function.

TX DCLK OUT

This mode configures the DGPIO pins to a pair of differential or a single-ended reference clock for the baseband processor if the TX SSI and RX SSI run at the different lane rate. Use this reference clock to generate the TX LSSI clock, data, and strobe when the RX SSI and TX SSI run at the different clock rate. Assign DGPIO_12 and DGPIO_13 to the TX1_DCLK_OUT± functionality when it is in the LVDS mode, or use either of DGPIO_12 or DGPIO_13 as the Tx1 DCLK out if it is in the CMOS mode. Similarly, assign DGPIO_14 and DGPIO_15 to the

GENERAL-PURPOSE INPUT/OUTPUT (GPIO) AND INTERRUPT CONFIGURATION

TX2_DCLK_OUT± functionality when it is in the LVDS mode, or use either DGPIO_14 or DGPIO_15 as the Tx1 DCLK out if it is in the CMOS mode.

Note: When the Tx DCLK OUT function is disabled, reuse the corresponding DGPIOs (DGPIO12/13 or DGPIO 14/15) only as input functions.

ANALOG GPIO OPERATION

The analog GPIO pins serve as the control pins for the external control elements, such as a digital step attenuator (DSA), low-noise amplifier (LNA), external local oscillator (LO)/voltage-controlled oscillator (VCO) components, T/R switch of the TDD system, and so on. Alternatively, analog GPIO pins provide the auxiliary DAC output.

Table 104 provides a high-level overview of the analog GPIO features.

Table 104. Summary of Analog GPIO Features

Feature	Description	GPIO Pins Available for Feature
RX Gain Table External Control Word	The RX gain table can include a column for 2-bit control of an external gain element (LNA). Each Rx channel has 2 Analog GPIO pins associated with it.	Any analog GPIO, but Rx1/Rx2 external gain word must be in one AGPIO nibble.
RF Front-End Control	Allows AGPIO timing to be associated with Tx/Rx_Enable to control the RF front end.	AGPIO_0 for Tx1 AGPIO_1 for Rx1 AGPIO_8 for Tx2 AGPIO_9 for Rx2
Manual Pin Toggle	Manually control the GPIO output level. API functions set output pin levels and read the input pin levels.	Any analog GPIO
Auxiliary DAC Output	Allows the auxiliary DAC output on analog GPIO pins.	AGPIO_0: AuxDAC0 output pin select AGPIO_1: AuxDAC1 output pin select AGPIO_2: AuxDAC2 output pin select AGPIO_3: AuxDAC3 output pin select

Receiver Gain Table External Control Word

The Receiver Gain Control section provides a complete description of the receiver gain table external control.

The ADRV9001 AGPIO output can control the external LNA gain. Each channel has two AGPIO control signals and achieves control of up to four external LNA gain steps. **adi_adrv9001_Rx_ExternalLna_Configure()** enables and configures the external LNA gain control.

The AGPIOs for channel 1 and channel 2 must be in one analog GPIO (AGPIO) nibble, which means the four AGPIOs for external gain control must be AGPIO[3:0] or AGPIO[7:4] or AGPIO[11:8]. For example, AGPIO_7/AGPIO_6 is for Rx1 external gain control, and AGPIO_5/AGPIO_4 for Rx2.

RF Front-End Control

To save the baseband processor control pins, the ADRV9001 provides the function to output the control signals through analog GPIO pins to power up/down the external RF front-end components (i.e., LNA, transmitter gain blocks, Ext PLL) or switch the T/R switch of a TDD system. For example, use a TX_ON, RX_ON output signal through the analog GPIOs and associated with the ADRV9001 transmitter/receiver enable timing and state to enable/disable the power amplifier and LNA, respectively, or do the antenna switch.

To get the best timing control performance, there are dedicated AGPIOs for transmitter/receiver front-end control. AGPIO_0, AGPIO_1, AGPIO_8, AGPIO_9 are associated with Tx1_Enable, Rx1_Enable, Tx2_Enable, Rx2_Enable, respectively.

The AGPIO for external RF front-end control is initialized in **adi_adrv9001_gpio_ControlInit_Configure()**, and the relative AGPIOs are configured by the API **adi_adrv9001_gpio_Configure()**.

Note: Once the AGPIO external RF front-end control is enabled, the receiver gain table external control can only use AGPIO[7:4].

Manual Pin Toggle

Like the manual pin toggle for digital GPIOs, this feature controls the logic level of individual analog GPIO pins. The **adi_adrv9001_gpio_ManualAnalogOutput_Configure()** and **adi_adrv9001_gpio_ManualAnalogInput_Configure()** manually configure the analog GPIO output and input, respectively. **adi_adrv9001_gpio_OutputPinLevel_Set()** and **adi_adrv9001_gpio_InputPinLevel_Get()** set and read the relative

analog GPIO level, respectively. Use the manual AGPIO level toggle to control the external RF components, like to power up/down the PA, and so on.

Auxiliary DAC Output

The auxiliary DAC can supply bias voltages, analog control voltages, or other system functionality. See the Auxiliary Converters and Temperature Sensor section for the details. Analog GPIO 0 through 3 provide the alternative function for the Aux DAC 0 through 3 output, respectively.

INTERRUPT

The ADRV9001 features the general-purpose interrupt output pin (GP_INT). The GP_INT pin can alert the baseband processor that an important event or error regarding the device operation occurred. These events include PLL locking, stream processor errors, ARM exceptions, and so on.

Table 105 describes the interrupt sources and their bit positions. An Interrupt source can be masked so it is not transmitted to the BBIC on the GP_INT pin or in status registers. An interrupt is masked when the corresponding mask bit is set to '1'. The GP_INT pin represents a logical OR of the enabled GP_INT mask sources. It is not necessary to enable all the interrupt sources.

Table 105. GP_INT Bitmask Description

Bit Position	Description	Component
0	ARM Error	ARM
1	Force Set an Interrupt	ARM
2	ARM System Error	ARM
3	ARM Calibration Error	ARM
4	Monitor Interrupt	ARM
5	Tx1 Power Amplifier Protection Error	Transmitter
6	Tx2 Power Amplifier Protection Error	Transmitter
7	Low-Power Clock PLL Lock Indicator	Lower Power Clock PLL
8	RF PLL 1 Lock Indicator	RF PLL1
9	RF PLL 2 Lock Indicator	RF PLL2
10	Aux PLL Lock Indicator	Aux PLL
11	Clock PLL Lock Indicator	Clock PLL
12	Main Clock 1105 MCS	Clock Distribution
13	Main Clock 1105 Second MCS	Clock Distribution
14	RX1 LSSI MCS	RX SSI
15	RX2 LSSI MCS	RX SSI
16	Main Stream Processor Error	Stream Processor
17	Stream Processor 0 Error	Stream Processor
18	Stream Processor 1 Error	Stream Processor
19	Stream Processor 2 Error	Stream Processor
20	Stream Processor 3 Error	Stream Processor
21	Not Used	
22	Not Used	
23	Not Used	
24	Not Used	

There is full control (through public API functions) to set/get the mask, sticky mask, and status registers (although the status register is read-only). Hence, tailor solutions (recovery actions) to handle the different events/interrupts.

Use **adi_adrv9001_gpio_GpIntMask_Set()** to mask the corresponding interrupt events after device initialization. When a rising edge is detected on the GP_INT pin, the baseband processor should call the API command **adi_adrv9001_gpio_GpIntStatus_Get()** to find out which interrupt sources trigger the interrupt signal.

AUXILIARY CONVERTERS AND TEMPERATURE SENSOR

The ADRV9001 device features auxiliary data converters, including four 12-bit auxiliary digital-to-analog converters (AuxDAC) and four 10-bit auxiliary analog-to-digital converters (AUXADC). There is an integrated diode-based temperature sensor to read back the approximate die temperature of the device.

These features are included to simplify control tasks and reduce pin count requirements on the baseband processor by offloading these tasks to the ADRV9001. An example usage of the auxiliary converters includes static voltage measurements performed by the AuxADC and flexible voltage control performed by the AuxDAC. This section outlines the operation of these features along with the API commands for configuration and control.

AUXILIARY DIGITAL-TO-ANALOG CONVERTER (AUXDAC)

The ADRV9001 has four independent, 12-bit AuxDACs. The auxiliary DACs have an output voltage of approximately 0.05 V to VAGPIO_1P8 – 0.05 V. The AuxDACs use the enumeration adi_adrv9001_AuxDac_e when referenced in the API. Table 106 shows the pins used for the AuxDAC features.

Table 106. AuxDAC Pin Mapping and adi_adrv9001_AuxDac_e Enum Description

Aux DAC Number	Pin Name	Pin Number	Enum Name
AUXDAC[0]	AGPIO_0	F12	ADI_ADRV9001_AUXDAC0
AUXDAC[1]	AGPIO_1	F10	ADI_ADRV9001_AUXDAC1
AUXDAC[2]	AGPIO_2	F3	ADI_ADRV9001_AUXDAC2
AUXDAC[3]	AGPIO_3	F5	ADI_ADRV9001 _AUXDAC3

The capacitive-load of the AuxDAC pins should not exceed more than 100 pF. Otherwise, there can be stability issues.

The AuxDAC uses the AGPIO pins on the device. There can be conflicts between the AGPIO and AuxDAC functionality. In case of these conflicts, the AuxDAC takes precedence over all other AGPIO functionality when the AuxDAC is enabled for a specific pin. When the AuxDAC is disabled, the configured AGPIO functionality is applied. Enable the AuxDAC one pin at a time for flexibility between the AuxDAC and AGPIO functionality.

The AuxDAC is typically used in applications requiring analog control signals. The data interface used to set the output level of the AuxDAC is SPI (API) or internal LUT (power amplifier RAMP function enabled) based. There is no CMOS/LVDS data interface to provide input data to the AuxDAC.

The (ideal) output voltage expressed on the AuxDAC is based on the following equation (in volts):

$$V_{AUXDAC} = 0.9 + \frac{AuxDacValue - 2048}{4096} \times 1.7 \quad (13)$$

where:

AuxDacValue is the 12-bit digital code applied to the AuxDAC.

The AuxDAC is not a precision converter. It is best used in feedback systems. Figure 241 shows the DAC code and corresponding DAC output voltage characterization plot. The AuxDAC output represents good linear behavior when the DAC code is within the range Code_min_linear and Code_max_linear. Table 107 shows the detailed test data summary, which is performed with the following conditions:

- Sample set: 20 AUX DACs
  - 5 parts
  - 4 AUX DACs per part
- Nominal process corner
- Nominal supplies
- Measured at ambient temperatures of 25°C, 85°C, and -40°C.

AUXILIARY CONVERTERS AND TEMPERATURE SENSOR

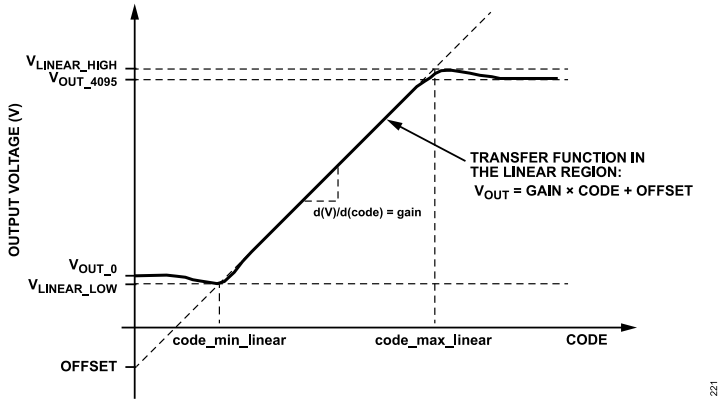


Figure 241. AuxDAC DAC Code vs. Output Voltage Characterization

Table 107. AuxDAC DAC Code and Output Voltage Characterization Data Summary

Conditions/ Metric	Vout_0 worst (mV)	Vout_4095 worst (V)	Mean Gain without DAC0 (mV/code)	Mean Gain with DAC0 (mV/code)	Gain Error with DAC0 (+/- %)	Mean Gain DAC0 (mV/Code)	Mean Offset (mV)	Offset Error (+/- mV)	Vlinear_low worst (mV)	code_min_linear worst	Vlinear_high worst (mV)	code_max_linear worst
25 °C	16.385	1.753	0.471	0.467	--	--	-38.283	--	15.662	262	1.753	3955
85 °C	18.767	1.742	0.459	0.456	--	--	-29.742	--	18.281	224	1.742	4007
-40 °C	13.799	1.760	0.487	0.482	--	--	-65.765	--	12.984	332	1.759	3874
m	14.649	1.754	0.472	0.469	--	0.457	-45.497	--	14.266	220	1.753	3990
s	2.387	0.007	0.012	0.013	--	0.009	80.674	--	2.456	50	0.007	51
Overall Performance Number	**18.767**	**1.742**		**0.469**	**8.429**		**-45.497**	**242.023**	18.281	**332**	1.742	**3874**

AuxDAC API Programming

A set of API commands set and inspect the AuxDAC, as shown in Table 108.

Table 108. AuxDAC API List

AuxDAC Function Name	Description
adi_adrv9001_AuxDac_Configure	Sets the configuration for AuxDACs. Enables/disables the selected AuxDAC.
adi_adrv9001_AuxDac_Inspect	Gets the configuration of the selected AuxDAC.
adi_adrv9001_AuxDac_Code_Set	Sets 12 bit DAC code of the selected AuxDAC.
adi_adrv9001_AuxDac_Code_Get	Reads the DAC word of the selected AuxDAC.

AUXILIARY ANALOG-TO-DIGITAL CONVERTER (AUXADC)

The ADRV9001 has four dedicated AuxADCs: AUXADC_0, AUXADC_1, AUXADC_2, and AUXADC_3. The AuxADC is a 10-bit output delta-sigma converter useful for measuring DC and near-DC signals (<8 kHz). The input voltage range of the AuxADC is 150 mV to 800 mV. API commands perform the readback of the AuxADC input voltage.

The AuxADCs use the enumeration adi_adrv9001_AuxAdc_e when referenced in the API. Table 109 shows the pins used for the AuxADC features.

Table 109. AuxADC Pin Mapping and adi_adrv9001_AuxAdc_e Enum Description

Aux ADC Number	Pin Name	Pin Number	Enum Name
AUXADC[0]	AUXADC_0	H11	ADI_ADRV9001_AUXADC0
AUXADC[1]	AUXADC _1	B8	ADI_ADRV9001_AUXADC1
AUXADC[2]	AUXADC _2	B7	ADI_ADRV9001_AUXADC2
AUXADC[3]	AUXADC _3	H4	ADI_ADRV9001_AUXADC3

AUXILIARY CONVERTERS AND TEMPERATURE SENSOR

The AuxADC clock rate is set to 30.72 MHz (or close when the ADRV9001 ARM system clock is changed) to get the best ADC performance. No on-chip calibrations are executed for the AuxADC. The ADC accuracy is limited to the accuracy of the supply reference. A simplified procedure is performed to measure and account for the AuxADC gain and offset errors. These AuxADC gain and offset errors are used to compensate for the AuxADCs measure results.

AuxADC API Programming

Table 110 shows the AuxADC relative API commands. Find more details in the API Doxygen document.

Table 110. AuxADC API List

AuxADC Function Name	Description
adi_adrv9001_AuxAdc_Configure	Sets to enable/disable the selected AuxADC.
adi_adrv9001_AuxAdc_Inspect	Gets the configuration of the selected AuxADC.
adi_adrv9001_AuxAdc_Voltage_Get	Gets the ADC code of selected AuxADC and converts to mV.

TEMPERATURE SENSOR

The ADRV9001 device features one instance of an on-chip temperature sensor that measures the temperature on the die. The temperature sensor uses an ADC similar to the AuxADC. However, it is a separate instantiation and has no connections to a device pin. The temperature sensor functions during the normal operation mode as well as the deep-sleep or monitor mode of the device.

The ADRV9001 internal microprocessor performs the initialization of the temperature sensor hardware without any user interaction. An API function "**adi_adrv9001_Temperature_Get()**" retrieves the current temperature measurement result, which is in celsius.

As the temperature sensor clock rate can depend on the user-configured profile, the measurement time depends on the user profile and other factors such as the number of average ADC performed. It usually takes several milliseconds to complete one measurement.

Note: The obtained temperature measurement represents an estimation, with limited accuracy.

RF PORT INTERFACE INFORMATION

The RF ports of the ADRV9001, consisting of the transmit (TX1±, TX2±) and receive ports (RX1A±, RX1B±, RX2A±, and RX2B±), support an operational frequency range from 30 MHz to 6 GHz. This wide frequency range fulfills the requirements of many application spaces. However, for optimized performance within a narrowband with minimal amplitude roll-off and optimized linearity and noise performance, the RF ports are impedance-matched.

This section provides some examples of impedance matching networks to select frequency bands. Locating a balun/transformer to cover the entire frequency range of the ADRV9001 with minimal phase and amplitude imbalance is a challenging task given the limited selection of commercially available baluns. So, the example RF matching networks are chosen based on the available baluns/transformers and RF trace implementations on the evaluation PCB.

The matching networks are divided into two categories, wideband and narrowband. The wideband matching networks cover a range of almost 3 GHz and possess more amplitude roll-off compared to the narrowband match. This roll-off is often dominated by the characteristic performance of the balun/transformer. For more optimized performance within a narrowband, use the frequency-specific narrowband matches.

TRANSMIT PORTS: TX1± AND TX2±

The ADRV9001 uses a direct conversion transmitter architecture consisting of two identical and independently controlled transmitter channels. The differential output impedance of transmitter outputs is matched to 50 Ω, as shown in Figure 242. Additionally, bias the transmitter outputs to a low noise 1.8 V supply.

RECEIVE PORTS: RX1A±, RX1B±, RX2A±, AND RX2B±

The ADRV9001 has two RF inputs for each receiver to accommodate different matching for each RF band of interest. The mixer architecture is very linear and inherently wideband, which facilitates wideband impedance matching. The differential input impedance of the receiver inputs is 100Ω, as shown in Figure 243 and Figure 244.

When selecting a balun/transformer for the receive paths, use a 2:1 impedance transformation to accommodate the 50 Ω single-ended impedance to 100 Ω differential impedance, as required by the ADRV9001 receiver inputs.

The receiver input pins are self-biased internally to 650mV and, therefore, require AC-coupling/DC-blocking capacitors at their inputs.

EXTERNAL LOCAL OSCILLATOR PORTS: LO1± AND LO2±

Apply two external LO inputs (LO1 and LO2) to the ADRV9001, and use each external LO signal for any of the two receivers or two transmitters instead of an internally generated LO signal. The AC-coupling interface is needed for both the positive and negative sides of the external LO input pins, which are internally biased. Similar to the receiver RF interface, a balun with 2:1 impedance transformation is necessary to accommodate the 50 Ω single-ended impedance to 100 Ω differential impedance, as required by the ADRV9001 external LO inputs.

DEVICE CLOCK PORT: DEV_CLK1±

There are two low-frequency (below 100 MHz) clock interface modes and an LVDS-type clock interface mode that can support the clock signal running as fast as 1 GHz. For the high-frequency clock interface, off-chip 100 Ω resistive termination is required, along with AC-coupling caps. The subsequent section Connection for External Device Clock (DEV_CLK_IN) provides more information.

RF RECEIVER/TRANSMITTER PORTS IMPEDANCE DATA

This section provides the port impedance data for all the transmitters and receivers in the ADRV9001 integrated transceiver. Note the following:

- The ADRV9001 ball pads are the reference plane for this data.
- 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 the real and imaginary components by two.

Contact Analog Devices Applications Engineering for the impedance data in the Touchstone format.

RF PORT INTERFACE INFORMATION

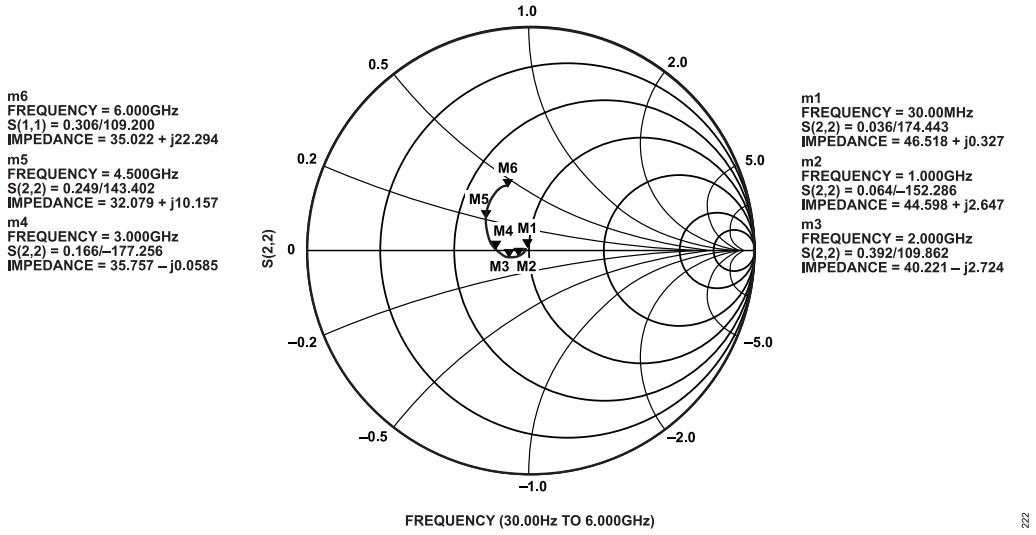


Figure 242. ADRV9001 Transmitter Port Series Equivalent Differential Impedance

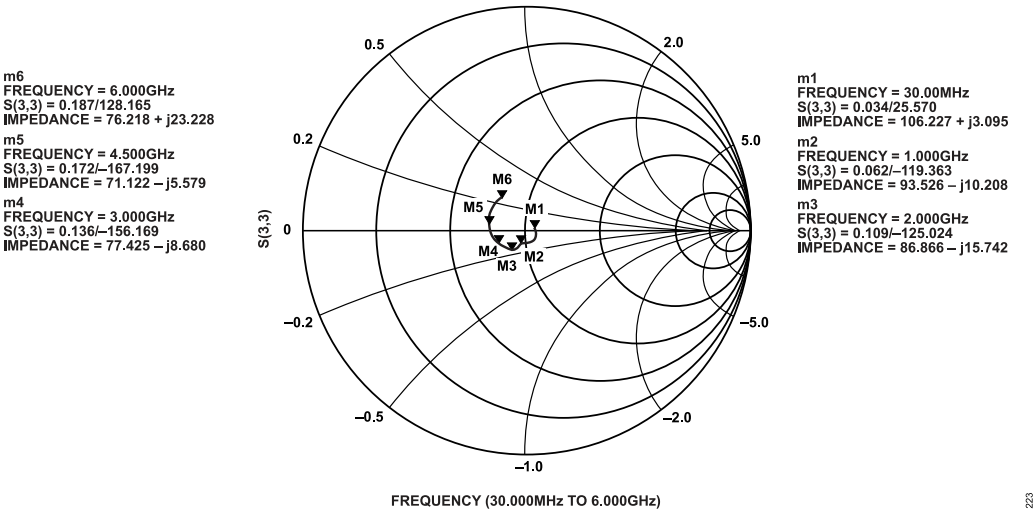


Figure 243. ADRV9001 Receiver A Port Series Equivalent Differential Impedance

RF PORT INTERFACE INFORMATION

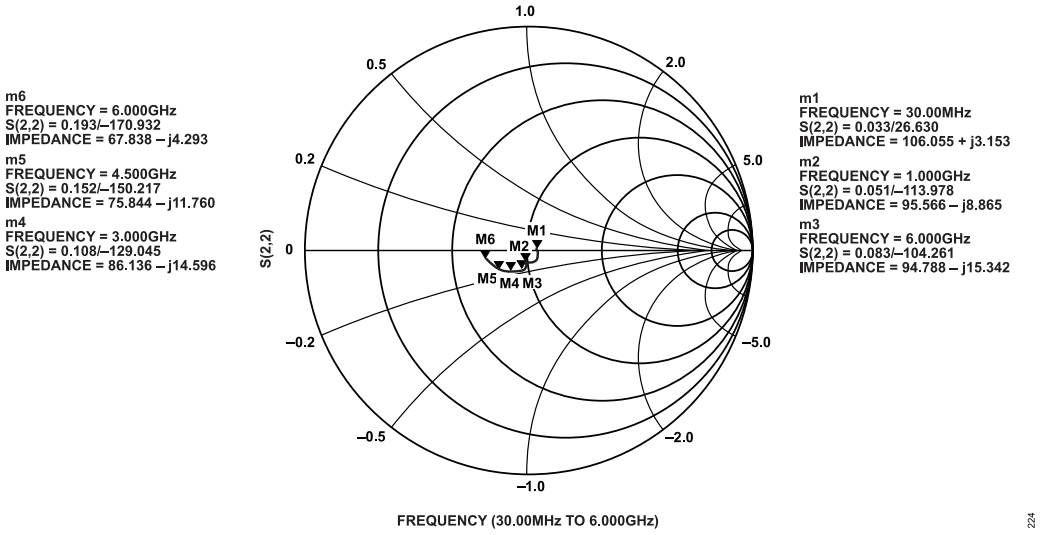


Figure 244. ADRV9001 Receiver B Port Series Equivalent Differential Impedance

Advanced Design System (ADS) Setup Using DAC and SEDZ File

Analog Devices supplies the port impedance as an *.s1p series equivalent differential Z (impedance) file. This format allows the simple interface to the advanced design system (ADS) using the data access component (DAC). Term1 is the single-ended input or output and Term2 represents the differential input or output RF port on the ADRV9001. See Figure 245. The Pi on the single-ended side and the 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 the appropriate component population.

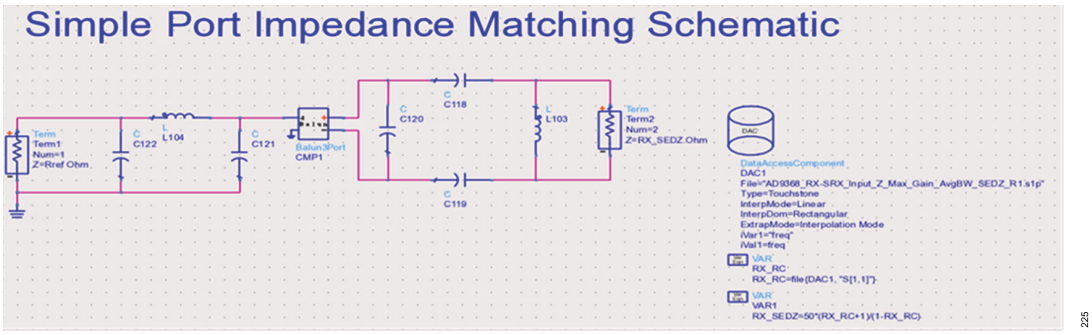


Figure 245. Simulation Setup in ADS with SEDZ S1P Files and DAC Component

The 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 result is the RX_SEDZ variable.
3. The RF port calculated complex impedance (RX_SEDZ) defines the Term2 impedance.

    a. Term2 is used in the differential mode and Term1 is used in the single-ended mode.

Setting up the simulation this way allows to measure the S11, S22, and S21 of the 3-port system without complex math operations within the display page.

Note: For the highest accuracy, use the EM modeling result of the PCB artwork and S-parameters of the laminate (also supplied by Analog Devices), matching components and balun in the simulations.

Insert the first differential shunt reactive component, such as L103 in Figure 245, to tune out the parallel parasitic reactance of the input impedance of the device. If the AC-coupling cap is necessary, use C118 and C119.

RF PORT INTERFACE INFORMATION

For a wideband match application, because of well-controlled input/output impedance characteristics of the ADRV9001 for the entire range of its RX/TX operational frequency band, implement the minimal matching network to control undesirable impedance deviation typically associated with the high side of the frequency range a balun operates. Additionally, select a balun with the same differential side termination impedance as the impedance of the receiver inputs and transmitter outputs, accomplish a broadband match by avoiding unnecessary impedance transformation network, which is inherently band-limiting.

Also consider adding additional differential series capacitive components on the balanced side of the balun to facilitate AC-coupling and Pi match on both sides of the balun, as shown in Figure 246.

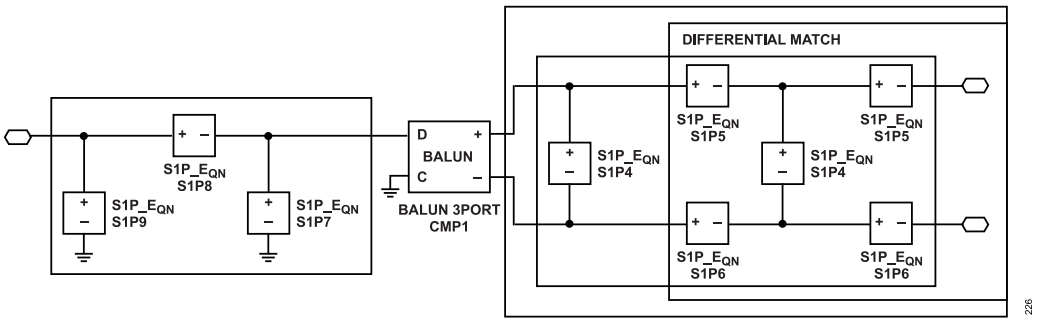


Figure 246. RF Matching Network with Additional Series AC-Coupling Capacitors

For a narrowband impedance match application to filter out signals outside of the frequency band of interest, use the Pi match technique for the desired bandwidth of impedance match with a selected balun's terminal impedance. Consider the Pi match as two L match networks back-to-back and allow independent control of Q and impedance ratio obtainable from a matching network. A narrowband matching network tuned for frequency bands of the receiver and transmitter further improves the out-of-band rejection of a transceiver for frequency duplexed systems.

GENERAL RECEIVER PORT INTERFACE

The ADRV9001 has two independent receive input channels (Rx1 and Rx2). Both receiver channels can support up to 40 MHz bandwidth and use a differential signaling interface. Apply the differential input signals to an integrated mixer. The mixer input pins are internally biased to 0.65 V and can be AC- coupled depending on the common-mode voltage level of the external circuit.

The important considerations for the receiver RF port interface are as follows:

1. Device to be interfaced: filter, balun, T/R switch, external LNA, and so on. Does this device represent a short-to-ground at DC?
2. Rx1 and Rx2 maximum safe input power is 18 dBm (peak).
3. Rx1 and Rx2 optimum DC bias voltage is 0.65 V bias to ground.
4. Board Design: reference planes, transmission lines, impedance matching, and so on. Figure 247 shows possible differential receiver port interface circuits. The options in Figure 247 and Figure 248 are valid for all receiver inputs operating in differential mode, though only the Rx1 signal names are indicated. Impedance matching can be necessary to obtain data sheet performance levels.

Receiver Port A/B Switching

The ADRV9001 supports a wide range of RF frequencies from 30 MHz to 6 GHz. However, typical RF baluns do not support all frequencies but only a smaller range. There is a special feature to support switching the receiver A and B ports, which allows to use both of them as receiver channels, and they can be switched at run time depending on the carrier frequency. Port A and B can be connected to different RF baluns and RF matching networks to support different RF frequencies. The frequencies are configurable.

RF PORT INTERFACE INFORMATION

Differential Receiver Input Interface Circuits

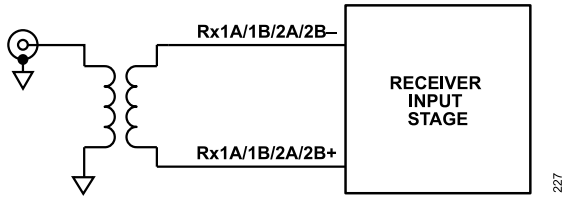


Figure 247. Differential Receiver Interface Using a Transformer

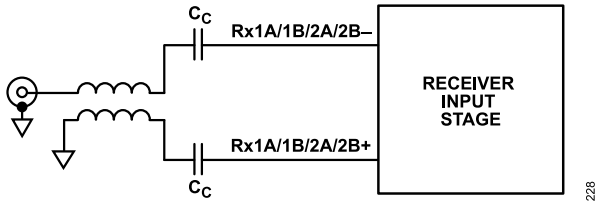


Figure 248. Differential Receiver Interface Using a Transmission Line Balun

Given the wide RF bandwidth applications, SMD balun devices function well, offering acceptable differential balance and insertion loss in a relatively small (0603, 0805) package.

Example of RX1 A Port Frequency Match Simulation

Advanced Design System (ADS) Setup Using DAC and SEDZ File shows that using the S parameter files for the laminate, matching components and balun are essential to get the most accurate match. This example uses the receiver port file and laminate file provided by Analog Devices in the design files package. It uses a more detailed ADS setup than mentioned previously. Figure 249 shows an overview of the ADS setup.

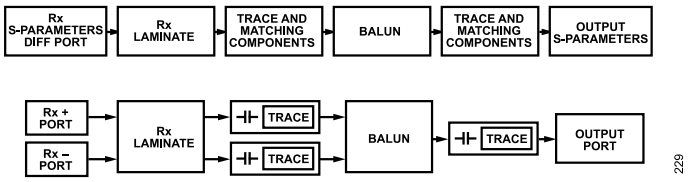


Figure 249. ADS Example Overview Block Diagram

Manipulate the S1P files for the receiver port to be used in the simulation. The file is generated from a differential port and packaged as a single port equivalent file. An ADS DAC is used to import the file into ADS and the equations in Figure 250.

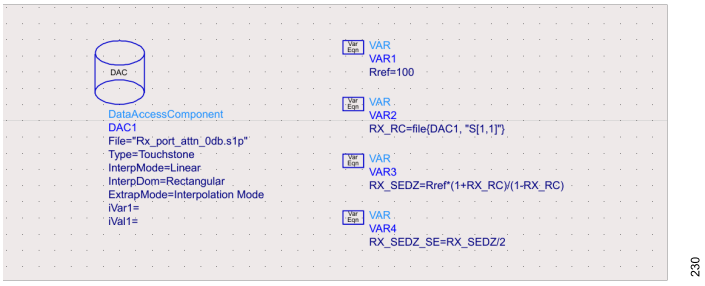


Figure 250. ADS Example Calculations

This allows to use the RX_SEDZ_SE as a single-ended port term. Figure 251 shows the setup with the laminate S4P file attached.

RF PORT INTERFACE INFORMATION

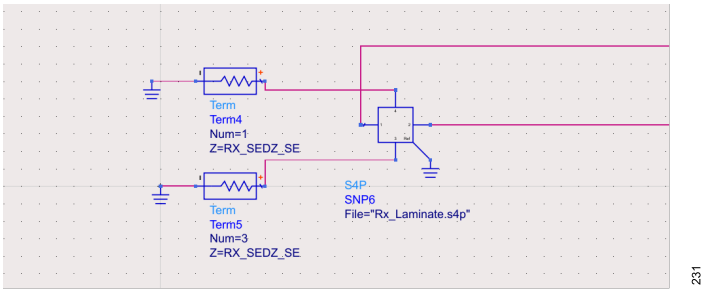


Figure 251. Differential Receiver Port and Laminate

Add traces and components to emulate the matching network being implemented with various ADS trace blocks, like MLINE, or with a simulated S parameter block from the layout Gerber files. A simulation of the board files is more accurate to include different trace lengths or curves.

Calculating the reflection results is the final step. Given the setup being used with three individual terms, two on the differential Rx port, use the following equations to get the single-ended equivalent responses. Note: In this case, the receiver port terms are numbered 1 and 3, and the single-ended output term is number 3.

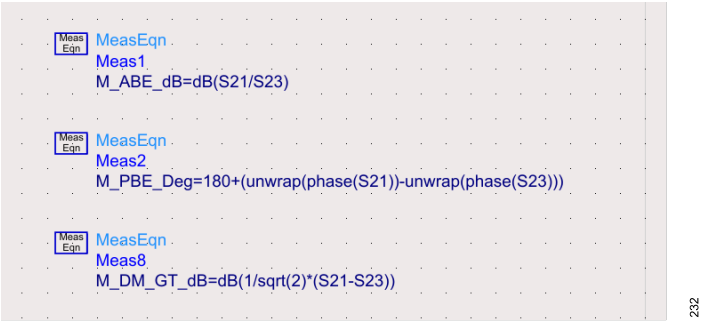


Figure 252. Reflection Result Calculations

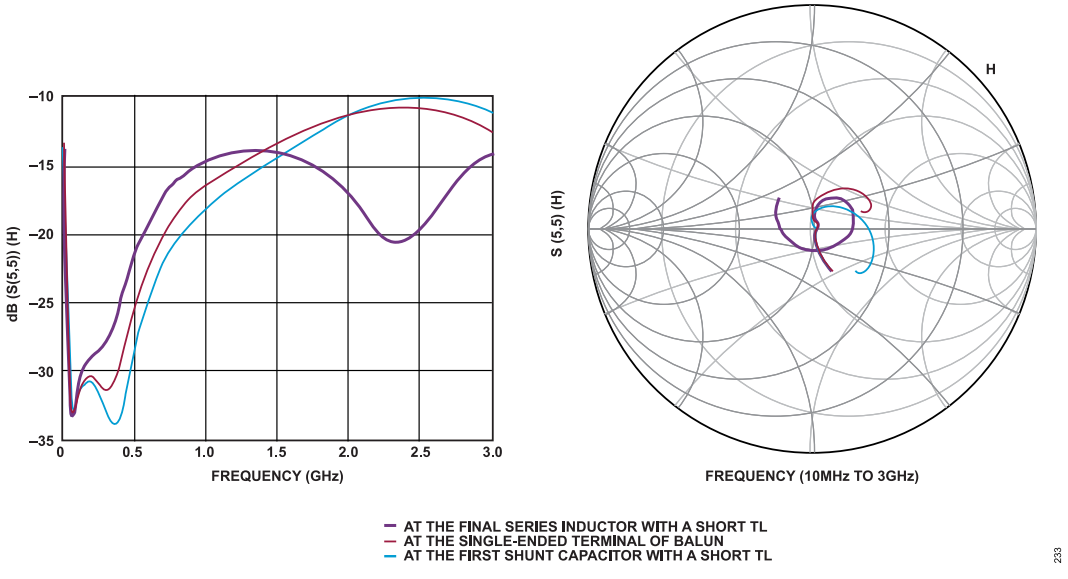


Figure 253. ADS Simulation Results of Return Loss Curve

S parameters shown in Figure 253 are an example, and the actual results depend heavily on the response of the chosen balun. Add shunt and series matching components with short TLs to represent possible PCB traces associated with these matching components on the single side of the balun.

RF PORT INTERFACE INFORMATION

For a more detailed breakdown of this example, refer the ADI Engineer Zone forum tutorials page.

GENERAL TRANSMITTER BIAS AND PORT INTERFACE

This section considers the DC biasing of the ADRV9001 transmitter outputs and how to interface with each transmitter port. The ADRV9001 transmitters operate over a range of frequencies. Each differential output side draws approximately 100mA of DC bias current at full output power. The transmitter 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 ensures 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 254. The R_{DCR} resistors indicate the parasitic elements. As the impedance of the parasitics increases, the voltage drop (ΔV) across the parasitic element increases, causing the transmitter RF performance (i.e., PO 1 dB PO MAX, and so forth) to degrade. Select the choke inductance (LC) high enough relative to the load impedance such that it does not degrade the output power.

Figure 255 shows the recommended DC bias network. This network has fewer parasitic and total components.

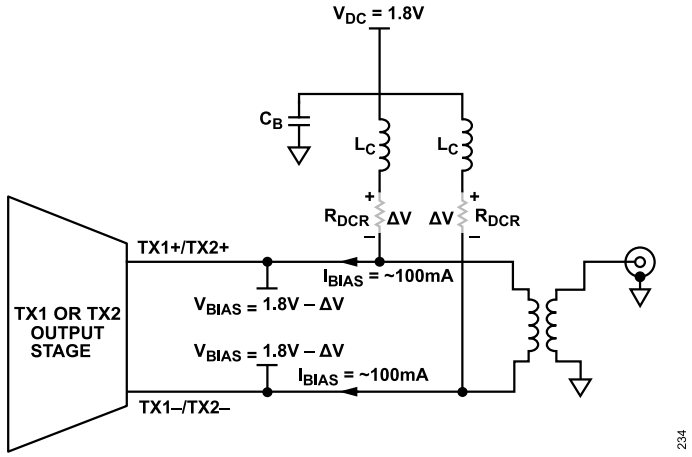


Figure 254. ADRV9001 RF DC Bias Configurations Depicting Parasitic Losses Due to Wire Wound Chokes

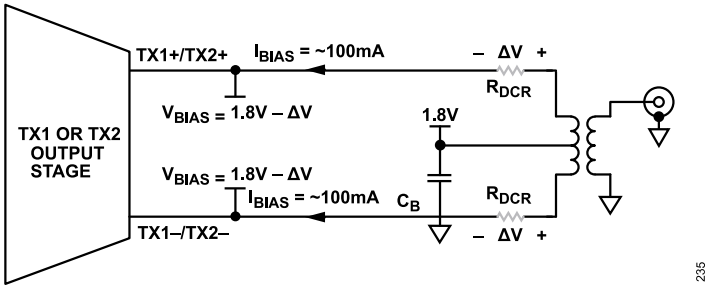


Figure 255. ADRV9001 RF DC Bias Configurations Depicting Parasitic Losses Due to Center Tapped Transformers

Figure 256 , Figure 257, Figure 258, and Figure 259 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 ADRV9001. 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.

Figure 256 shows the recommended RF transmitter interface featuring a center-tapped balun. This configuration offers the lowest component count of the available options.

The transmitter port interface schemes are briefly described 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 (CC) are added in the creation of a transmission line balun.

RF PORT INTERFACE INFORMATION

- 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.

Transmitter Interface Configurations

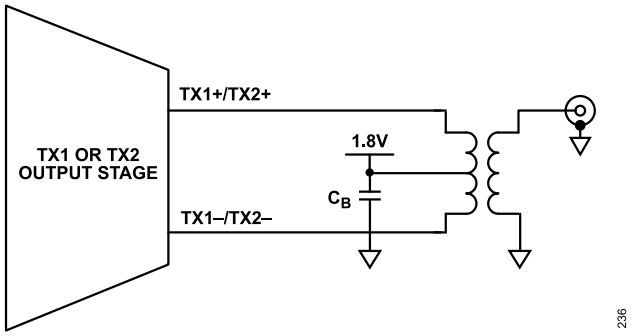


Figure 256. ADRV9001 RF Transmitter Interface Configuration A

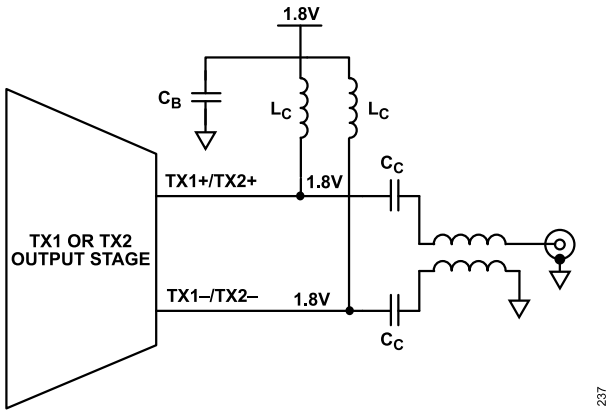


Figure 257. ADRV9001 RF Transmitter Interface Configuration B

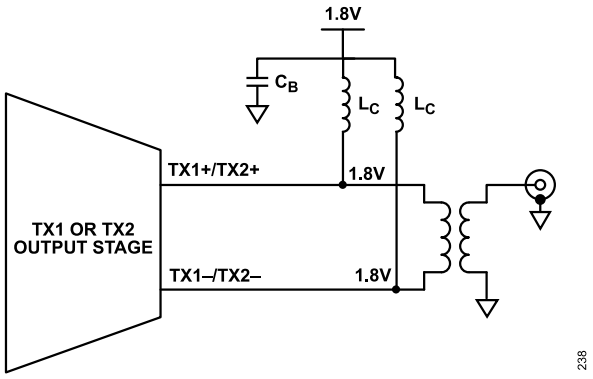


Figure 258. ADRV9001 RF Transmitter Interface Configuration C

RF PORT INTERFACE INFORMATION

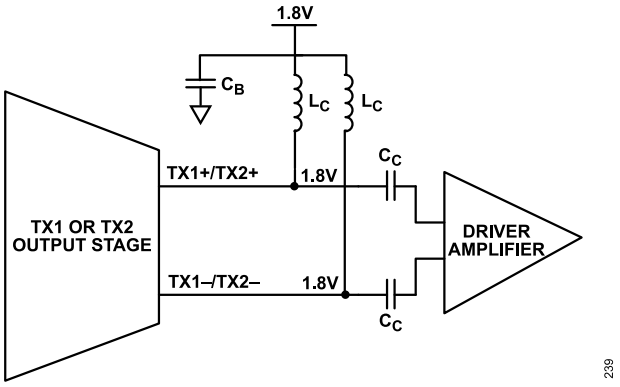


Figure 259. ADRV9001 RF Transmitter Interface Configuration D

Careful planning is required to select a transmitter balun with a set of external DC bias chokes. It is necessary to find the optimum compromise between the physical choke 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 around 50 mV.

Table 111. Sample Wire-Wound DC Bias Choke Resistance vs. Size vs. Inductance

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

When selecting a DC bias choke inductor, the shunting impedance of the choke inductor must be high for the transmitter frequency band to minimize its loading to outputs. Therefore, the self resonant frequency of the selected choke inductor must be higher than the intended transmitter frequency.

Additionally, the ADRV9001 provides a built-in Transmitter power ramp-up pattern generator to bring transmit power level in a predetermined way to protect internal devices from sudden voltage spikes, which can happen due to the in-rush current passing through an external DC bias choke inductor. Also tie the supply side of choke inductors to a capacitor with its self-resonant frequency higher than the transmitter frequency. When both the transmitter channels are active, tie each transmitter output to its own supply plane through a bias chock inductor or ferrite bead to reduce coupling between the two transmitters through the same supply feed line.

IMPEDANCE MATCHING NETWORK EXAMPLES

Impedance matching networks are required to achieve the performance levels noted on the data sheet. This section provides example topologies and components used on the evaluation board. The impedance matching networks provided in this section have not been evaluated in terms of mean time to failure (MTTF) in high volume production. Contact the component vendors for long-term reliability concerns. Contact the balun vendors to determine appropriate conditions for DC biasing.

The schematics in Figure 260 and Figure 261 show two or three circuit elements in parallel marked do not include(DNI). This is done on the evaluation board schematic to accommodate different component configurations for different frequency ranges. Only one set of standard microcircuit drawing (SMD) component pads is placed on the board to provide a physical location that can be used for the selected parallel circuit element. For example, R216, L216, and C216 components only have one set of SMD pads for one SMD component. The schematic shows that in a generic port impedance matching network, the series elements can be either a resistor, inductor or a capacitor, whereas the shunt elements can be either an inductor or a capacitor. Only one component of each parallel combination is placed in a practical application. Note: In some matching circuits, some shunt elements may not be required. All components for a given physical location remain DNI in those particular applications.

RF PORT INTERFACE INFORMATION

RECEIVER RF PORT IMPEDANCE MATCHING NETWORK

RX1A± and RX2A± Impedance Matching Network

The ADRV9001 evaluation board uses both the top and bottom layers of the PCB evaluation platform to accommodate two balun footprints. The 0805 footprint accommodates the high-frequency narrowband baluns, while the backside accommodates the larger DB1627 case-style transformer.

The PCB traces of the evaluation board are included in the simulation when designing the impedance match. Table 112 provides impedance matching networks specific to the ADRV9001 evaluation board. The component values apply to RX1A± and RX2A±.

RX1B± and RX2B± Impedance Matching Network

Both the RXA and RXB paths share the same input S-parameters. However, given the ball locations of the RXB paths are in an inner row and column, and the layout of both paths are slightly different, the RXB path has its own distinct impedance matching network different from the RXA path. On the RXB path, the low-frequency DB1627 case style transformer is located on the top side of the evaluation platform, and the high-frequency 0805 footprint transformer is located at the bottom of the board. This configuration is the opposite of the RXA path. This is done to minimize the coupling of the receive paths.

The PCB traces of the evaluation board are included in the simulation when designing the impedance match. Table 113 provides impedance matching networks specific to the ADRV9001 evaluation board. The component values apply to RX1B± and RX2B±.

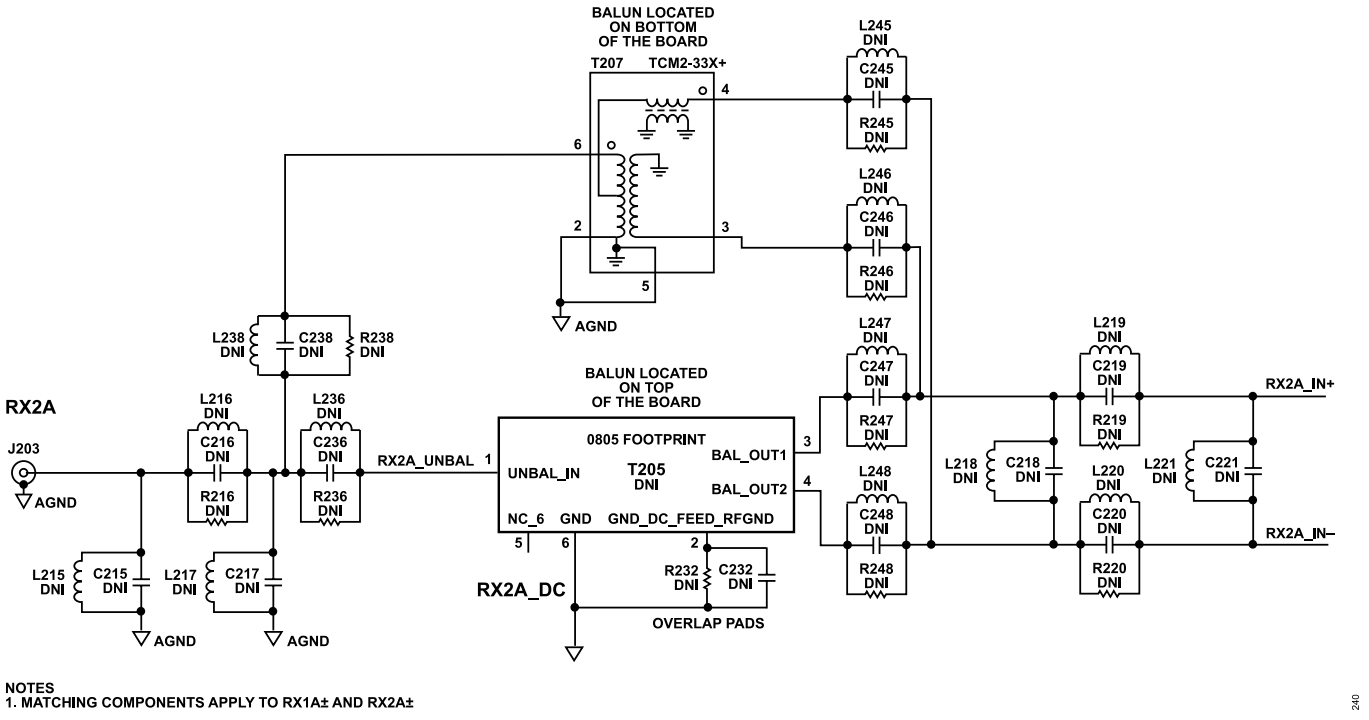


Figure 260. RX1A and RX2A Impedance Matching Network

Table 112. RX1A± and RX2A± Impedance Matching Network

Frequency	Balun	L/C/R 215	L/C/R 216	L/C 217	L/C/R 238	L/C/R 236	L/C/R 245 L/C/R 246	L/C/R 247 L/C/R 248	L/C 218	L/C/R 219 L/C/R 220	L/C 221	C232 R232
30MHz to 3GHz	MiniCircuits TCM2-33WX+	L215: DNI C215: DNI	L216: 1.2nH AVX Ind (L0201 Series)	L217: DNI C217: DNI	L238: DNI C238: DNI R238: 0Ω	L236: DNI C236: DNI R236: DNI	L245/246: DNI C245/246: DNI R245/246: 0Ω	L247/248: DNI C247/248: DNI R247/248: DNI	L218: DNI C218: DNI	L219/220: DNI C219/220: 470 pF Murata	L221: DNI C221: DNI	C232:DNI R232:DNI

RF PORT INTERFACE INFORMATION

Table 112. RX1A± and RX2A± Impedance Matching Network (Continued)

Frequency	Balun	L/C/R 215	L/C/R 216	L/C 217	L/C/R 238	L/C/R 236	L/C/R 245 L/C/R 246	L/C/R 247 L/C/R 248	L/C 218	L/C/R 219 L/C/R 220	L/C 221	C232 R232
			C216: DNI R216: DNI							(GRM03 Series) R219/220: DNI		
3GHz to 6GHz	Johanson 4400	L215: DNI C215: DNI	L216: 0.82nH AVX Ind (L0201 series) C216: DNI R216: DNI	L217: DNI C217: DNI	L238: DNI C238: DNI R238: DNI	L236: DNI C236: DNI R236: 0Ω	L245/246: DNI C245/246: DNI R245/246: DNI	L247/248: 1.0nH AVX Ind (L0201 series) C247/248: DNI R247/248: DNI	L218: DNI C218: 0.1pF Murata (GRM03 series)	L219/220: DNI C219/220: DNI R219/220: 0Ω	L221:DNI C221: DNI	C232: 1.5pF Murata (GJM03 Series) R232:DNI
30MHz to 1GHz	MiniCircuits TCM2-33WX+	L215: DNI C215: DNI	L216: 3.3nH AVX Ind (L0201 series) C216: DNI R216: DNI	L217: DNI C217: 1.5pF Murata (GJM03 Series)	L238: DNI C238: DNI R238: 0Ω	L236: DNI C236: DNI R236: DNI	L245/246: DNI C245/246: DNI R245/246: 0Ω	L247/248: DNI C247/248: DNI R247/248: DNI	L218: DNI C218: DNI	L219/220: DNI C219/220: 470pF Murata (GRM03 Series) R219/220: DNI	L221: DNI C221: DNI	C232:DNI R232:DNI
625MHz to 2.8GHz	Johanson 1720BL15B0100	L215: DNI C215: DNI	L216: 0.82nH AVX Ind (L0201 series) C216: DNI R216: DNI	L217: DNI C217: DNI	L238: DNI C238: DNI R238: DNI	L236: DNI C236: DNI R236: 0Ω	L245/246: DNI C245/246: DNI R245/246: DNI	L247/248: DNI C247/248: 10pF Murata (GRM03 Series) R247/248: DNI	L218: DNI C218: DNI	L219/220: DNI C219/220: DNI R219/220: 0Ω	L221: DNI C221: DNI	C232:DNI R232:DNI
2.8GHz to 5GHz	Anaren BD3150L50100AAF	L215: DNI C215: 0.3pF Murata (GJM03 Series)	L216: 1.2nH AVX Ind (L0201 series) C216: DNI R216: DNI	L217: DNI C217: 0.3pF Murata (GJM03 Series)	L238: DNI C238: DNI R238: DNI	L236: DNI C236: DNI R236: 0Ω	L245/246: DNI C245/246: DNI R245/246: DNI	L247/248: DNI C247/248: 10pF Murata (GJM03 Series) R247/248: DNI	L218: DNI C218: DNI	L219/220: 1.2nH AVX Ind (L0201 series) C219/220: DNI R219/220: DNI	L221: DNI C221: 0.1pF Murata (GRM03 Series)	C232:DNI R232:DNI
4.5GHz to 6GHz	Johanson 5400BL15B100	L215: DNI C215: 0.2pF Murata (GJM03 Series)	L216: 1.0nH AVX Ind (L0201 series) C216: DNI R216: DNI	L217: DNI C217: 0.3pF Murata (GJM03 Series)	L238: DNI C238: DNI R238: DNI	L236: DNI C236: DNI R236: 0Ω	L245/246: DNI C245/246: DNI R245/246: DNI	L247/248: DNI C247/248: 10pF Murata (GJM03 Series) R247/248: DNI	L218: DNI C218: DNI	L219/220: 1.2nH AVX Ind (L0201 series) C219/220: DNI R219/220: DNI	L221: DNI C221: DNI	C232:DNI R232:DNI

RF PORT INTERFACE INFORMATION

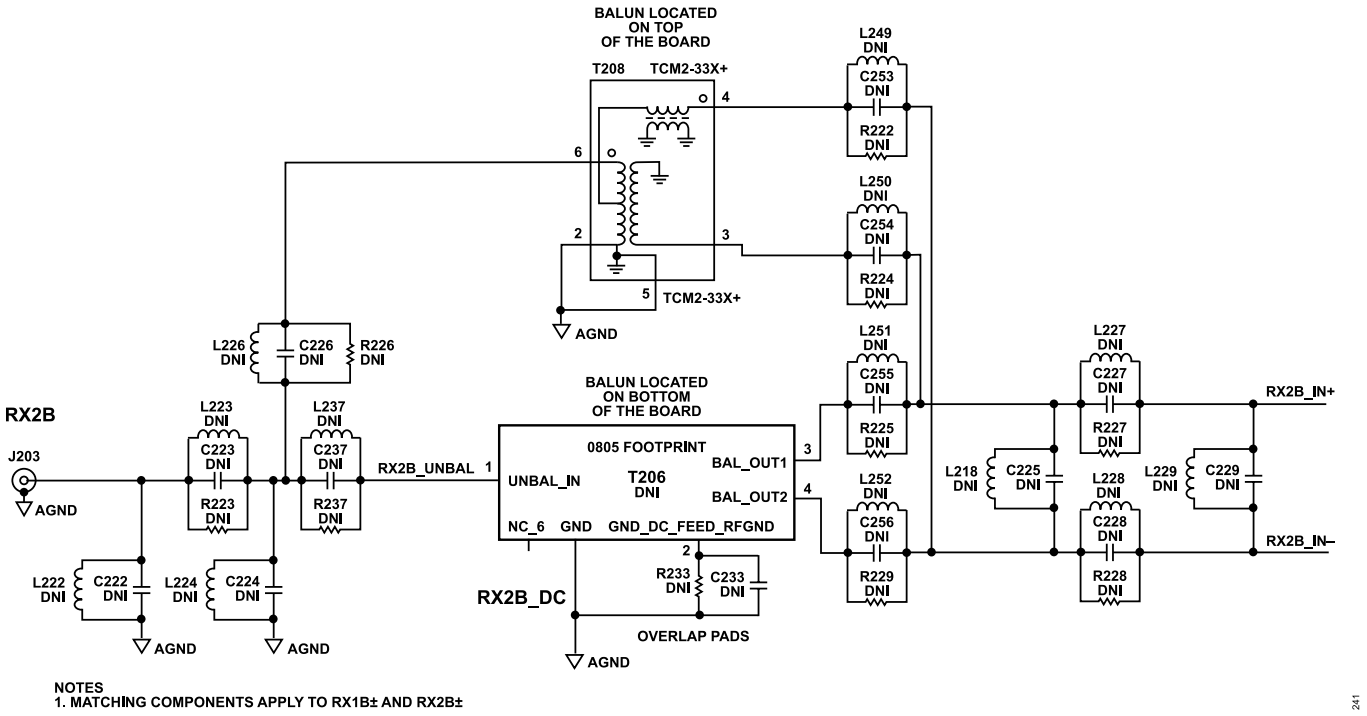


Figure 261. RX1B and RX2B Impedance Matching Networks

Table 113. RX1B± and RX2B± Impedance Matching Network

Frequency	Balun	L/C 222	L/C/R 223	L/C 224	L/C/R 226	L/C/R 237	L/C/R 249 L/C/R 250	L/C/R 251 L/C/R 252	L/C 225	L/C/R 227 L/C/R 228	L/C 229	C233 R233
30MHz – 3GHz	MiniCircuits TCM2-33WX+	L222: DNI C222: 0.5pF Murata (GJM03 Series)	L223: 2.2nH AVX Ind (L0201 series) C223: DNI R223: DNI	L224: DNI C224: DNI	L226: DNI C226: DNI R226: 0Ω	L237: DNI C237: DNI R237: DNI	L249/250: DNI C253/254: DNI R222/224: 0Ω	L251/252: DNI C251/252: DNI R225/229: DNI	L225: DNI C225: DNI	L227/228: DNI C227/228: 470pF Murata (GRM03 Series) R227/228: DNI	L229: DNI C229: DNI	C233:DNI R233:DNI
3GHz – 6GHz	Johanson 4400	L222: DNI C222: DNI Note: C257: 0.2pF Murata (GJM03 Series)	L223: 1.2nH AVX Ind (L0201 series) C223: DNI R223: DNI	L224: DNI C224: DNI	L226: DNI C226: DNI R226: DNI	L237: DNI C237: DNI R237: 0Ω	L249/250: DNI C249/250: DNI R222/224: DNI	L251/252: 0.33nH AVX Ind (L0201 series) C251/252: DNI R251/252: DNI	L225: CNI C225: 0.1pF Murata (GRM03 Series)	L227/228: 1.0nH AVX Ind (L0201 series) C227/228: DNI R227/228: DNI	L229: DNI C229: 0.1pF Murata (GRM03 Series)	C233: 1.4pF Murata (GJM03 Series) R233:DNI
30MHz – 1GHz	MiniCircuits TCM2-33WX+	L222: DNI C222: DNI	L223: DNI C223: DNI R223: 0Ω	L224: DNI C224: DNI	L226: DNI C226: DNI R226: 0Ω	L237: DNI C237: DNI R237: DNI	L249/250: DNI C249/250: DNI R222/224: 0Ω	L251/252: DNI C251/252: DNI R225/229: DNI	L225: DNI C225: DNI	L227/228: DNI C227/228: 470pF Murata (GRM03 Series)	L229: DNI C229: DNI	C233:DNI R233:DNI

RF PORT INTERFACE INFORMATION

Table 113. RX1B± and RX2B± Impedance Matching Network (Continued)

Frequency	Balun	L/C 222	L/C/R 223	L/C 224	L/C/R 226	L/C/R 237	L/C/R 249 L/C/R 250	L/C/R 251 L/C/R 252	L/C 225	L/C/R 227 L/C/R 228	L/C 229	C233 R233
										R227/228: DNI		
625MHz – 2.8GHz	Johanson 1720BL15B0100	L222: DNI C222: 0.7pF Murata (GJM03 Series)	L223: 2.7nH AVX Ind (L0201 series) C223: DNI R223: DNI	L224: DNI C224: 0.4pF Murata (GJM03 Series)	L226: DNI C226: DNI R226: DNI	L237: DNI C237: DNI R237: 0Ω	L249/250: DNI C249/250: DNI R222/224: DNI	L251/252: DNI C251/252: DNI R225/229: 0Ω	L225: DNI C225: DNI	L227/228: DNI C227/228: 470pF Murata (GRM03 Series) R227/228: DNI	L229: DNI C229: DNI	C233:DNI R233:DNI
2.8GHz – 5GHz	Anaren BD3150L50100AAF	L222: DNI C222: 0.5pF Murata (GJM03 Series)	L223: 1.5nH AVX Ind (L0201 series) C223: DNI R223: DNI	L224: DNI C224: 0.5pF Murata (GJM03 Series)	L229: DNI C229: DNI R229: DNI	L237: DNI C237: DNI R237: 0Ω	L249/250: DNI C251/252: DNI C249/250: DNI R222/224: DNI	L251/252: DNI C251/252: 100pF Murata (GRM03 Series) R251/252: DNI	L225: DNI C225: DNI	L227/228: 0.82nH AVX Ind (L0201 series) C227/228: DNI R227/228: DNI	L229: DNI C229: DNI	C233:DNI R233:DNI
4.5GHz– 6GHz	Johanson 5400BL15B100	L222: DNI C222: 0.2pF Murata (GJM03 Series	L223: 1.2nH AVX Ind (L0201 series) C223: DNI R223: DNI	L224: DNI C224: 0.2pF Murata (GJM03 Series	L226: DNI C226: DNI Series R226: DNI	L237: DNI C237: DNI R237: 0Ω	L249/250: DNI C249/250: DNI R222/224: DNI	L251/252: DNI C251/252: 30pF Murata (GRM03 Series) R251/252: DNI	L225: DNI C225: DNI	L227/228: 0.68nH AVX Ind (L0201 series) C227/228: DNI R227/228: DNI	L229: DNI C229: DNI	C233:DNI R233:DNI

RECEIVER RF PORT IMPEDANCE MATCH MEASUREMENT DATA

Receiver RF Port Impedance Match Measurement Data for 30 MHz to 3 GHz Band Match

Return loss is measured on RX1(2) A and RX1(2) B RF ports of evaluation boards and plotted as shown in Figure 262, Figure 263, and Figure 264. Blue and pink curves represent four different return loss measurements and the black dotted line represents simulated return loss curve in Figure 262 and Figure 263. Simulated insertion loss curve including balun loss is plotted in Figure 264.

RF PORT INTERFACE INFORMATION

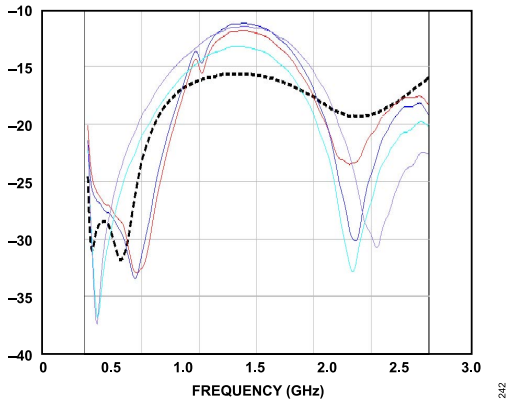


Figure 262. Return Loss of RX1(2) A Port

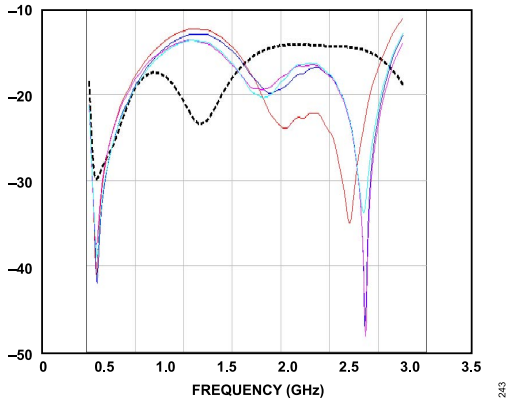


Figure 263. Return Loss of RX1(2) B Port

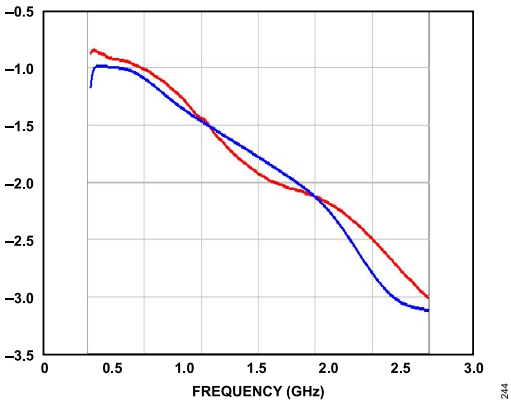


Figure 264. Insertion Loss – Simulated RX1(2) A Port – Red Curve RX1(2) B Port – Blue Curve

TRANSMITTER RF PORT IMPEDANCE MATCHING NETWORK

TX1± and TX2± Impedance Matching Network

For the transmitter path, the ADRV9001 evaluation board uses both the top and bottom layers of the PCB evaluation platform to accommodate two balun footprints. The 0805 footprint accommodates the high- frequency narrowband baluns while the backside accommodates the larger AT224-1A case style transformer.

RF PORT INTERFACE INFORMATION

The ADRV9001 evaluation board provides two options in providing the DC common-mode bias for the transmitter outputs. For transformers that provide a DC feed pin, use this to bias the transmitter output. For transformers that do not provide a DC feed pin, bias the transmitter outputs to 1.8 V through pull-up inductors. Choose only one bias option, and disable the unused path.

The PCB traces of the evaluation board are included in the simulation when designing the impedance match. Figure 265 and Table 114 provide impedance matching networks specific to the ADRV9001 evaluation board. The component values apply to TX1± and TX2±. Place the C335 close to the DC feed pin of balun T302 as its purpose is to eliminate transmitter spectrum spurs and dampen the transients. Tie the ground terminal of the C335 to a ground plane and orient the cap in the same direction as the ground plane surrounding the transmitter input trace so that the return current forms as small a loop as possible with the ground plane.

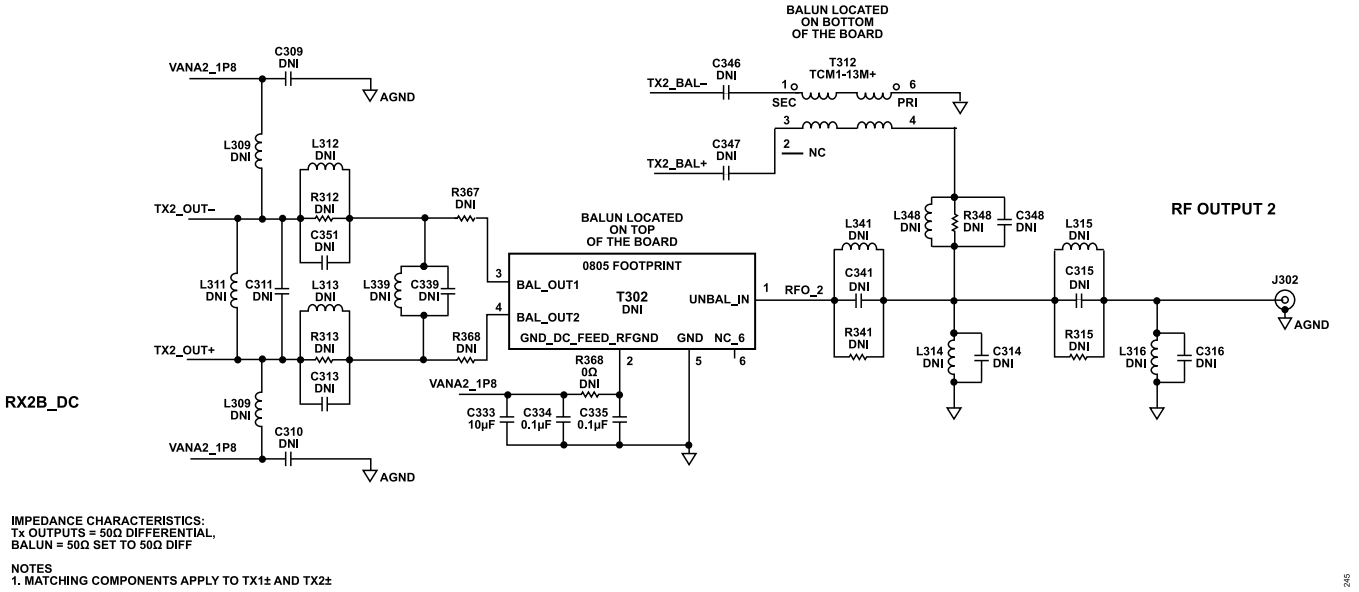


Figure 265. TX1 and TX2 Impedance Matching Network

Table 114. TX1± and TX2± Impedance Matching Network

Frequency	Balun	L/C 311	L309 L310	C309 C310	L/C/R 312 L/C/R 313	L339 C339	C346/347 R367/368	R361	C333 C334 C335	L/C/R 341	L/C/R 348	L/C 314	L/C/R 315	L/C 316
30MHz – 3GHz	MiniCircuits TC-1-13M+	L311: DNI C311: 0.3pF (Murata GJM03)	L309/310: 220nH (CoilCraft LQW18AN)	C309/ C310: 10nF (Murata GRM03)	L312/313: 1.2nH AVX Ind L0201 Series C312/313: DNI R312/313: DNI	L339: DNI C339: 0.8pF (Murata GJM03)	C346/347: 330pF (Murata GRM03) R367, R368: DNI	R361: DNI	C333: DNI C334: DNI C335: DNI	L341: DNI C341: DNI R341: DNI	L348: 2.2nH AVX Ind (L0201 Series C348: DNI R348: DNI	L314: DNI C314: 1pF (Murata GJM03)	L315: 2.2nH AVX Ind (L0201 Series) C315: DNI R315: DNI	L316: DNI C316: DNI
3GHz – 6GHz	Johanson 4400	L311: DNI C311: 0.2pF (Murata GJM03)	L309/310: DNI	C309/ C310: DNI	L312/313: 0.68nH AVX Ind (L0201 Series) C312/313: DNI	L339: DNI C339: 0.3pF (Murata GJM03)	C346/347: DNI R367/368: 0Ω	R361: 27nH Murata Ind (LQW18)	C333: 10µF C334: 0.1µF C335: 2.7pF	L341: 0.33nH AVX Ind (L0201 Series)	L348: DNI C348: DNI R348: DNI	L314: DNI C314: 0.3pF (Murata GJM03)	L315: 0.82nH AVX Ind (L0201 Series) C315: DNI	L316: DNI C316: DNI

RF PORT INTERFACE INFORMATION

Table 114. TX1± and TX2± Impedance Matching Network (Continued)

Frequency	Balun	L/C 311	L309 L310	C309 C310	L/C/R 312 L/C/R 313	L339 C339	C346/347 R367/368	R361	C333 C334 C335	L/C/R 341	L/C/R 348	L/C 314	L/C/R 315	L/C 316
					R312/313: DNI				(Murata GJM03)	C341: DNI R341: DNI			R315: DNI	
30MHz – 1GHz	MiniCircuits TC-1-13M+	L311: DNI C311: 1.4pF (Murata GJM03)	L309/310: 220nH (CoilCraft LQW18AN)	C309/ C310: 10nF (Murata GRM03)	L312/313: DNI C312/313: DNI R312/313: 0Ω	L339: DNI C339: DNI	C346/347: 330pF (Murata GRM03) R367/368: DNI	R361: DNI	C333: DNI C334: DNI C335: DNI	L341: DNI C341: DNI R341: DNI	L348: DNI C348: R348: 0Ω	L314: DNI C314: 1pF (Murata GJM03)	L315: 3.9nH (CoilCraft 0201 DS C315: DNI R315: DNI	L316: DNI C316: DNI
625MHz – 2.8GHz	Johanson 1720BL15 B0050	L311: DNI C311: DNI	L309/310: DNI	C309/310: DNI	L312/313: DNI C312/313: DNI R312/313: 0Ω	L339: DNI C339: 0.2pF (Murata GJM03)	C346/347: DNI R367/368: 0Ω	R361: 120nH Murata Ind LQW15 ANR12J00	C333: 10µF C334: 0.1µF C335: 47pF (Murata GRM03)	L341: DNI C341: DNI R341: 0Ω	L348: DNI C348: DNI R348: DNI	L314: DNI C314: DNI	L315: 0.33nH AVX Ind (L0201 Series) C315: DNI R315: DNI	L316: DNI C316: 0.1pF (Murata GRM03)
2.8GHz – 5GHz	Anaren BD3150L50100AAF	L311: DNI C311: DNI	L309/310: 220nH (Murata Ind LQW18AN)	C309/ C310: 10nF (Murata GRM03)	L312/313: 1.2nH AVX Ind (L0201 Series) C312/313: DNI R312/313: DNI	L339: DNI C339: DNI	C346/347: DNI R367/368: 20pF Cap (Murata GJM03)	R361: 0Ω	C333: DNI C334: DNI C335: DNI	L341: DNI C341: DNI R341: 0Ω	L348: DNI C348: DNI R348: DNI	L314: DNI C314: 0.2pF (Murata GJM03)	L315: 0.68nH AVX Ind (L0201 Series) C315: DNI R315: DNI	L316: DNI C316: 0.2pF (Murata GJM03)
4.5GHz– 6GHz	Johanson 5400BL15 B0050	L311: DNI C311: 0.1pF (Murata GRM03)	L309/310: 220nH (Murata Ind LQW18AN)	C309/ C310: 10nF (Murata GRM03)	L312/313: 0.82nH (Murata LPQ03HQ) C312/313: DNI R312/313: DNI	L339: DNI C339: 0.1pF (Murata GRM03)	C346/347: DNI R367/368: 20pF Cap (Murata GJM03)	R361: 0Ω	C333: DNI C334: DNI C335: DNI	L341: DNI C341: DNI R341: 0Ω	L348: DNI C348: DNI R348: DNI	L314: DNI C314: 0.3pF (Murata GJM03)	L315: 0.82nH AVX Ind (L0201 Series) C315: DNI R315: DNI	L316: DNI C316: 0.1pF (Murata GRM03)

TRANSMITTER RF PORT IMPEDANCE MATCH MEASUREMENT DATA

Data for Transmitter RF Ports for 30 MHz to 3 GHz Band Match

Return loss is measured on the transmitter RF ports of evaluation boards and plotted in Figure 266. The blue and pink curves represent four different return loss measurements, and the black dotted line represents the simulated return loss curve. Simulated Insertion loss including balun loss is plotted in Figure 267.

RF PORT INTERFACE INFORMATION

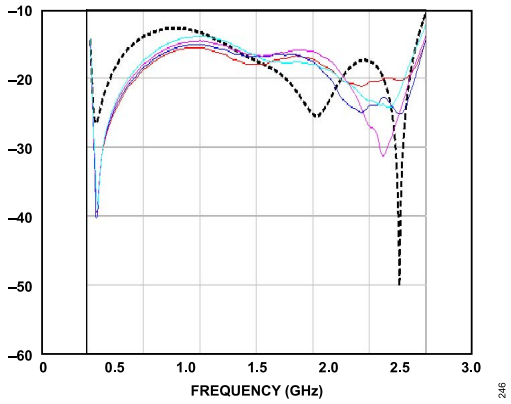


Figure 266. Tx1/Tx2 Return Loss

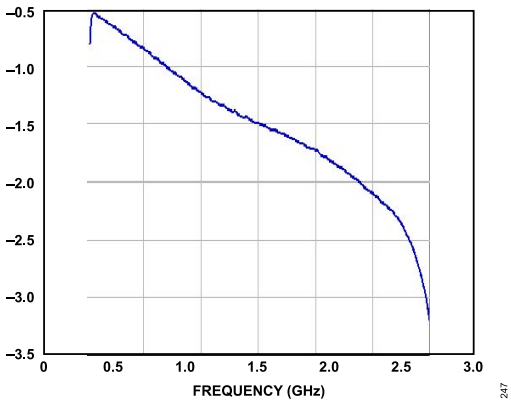


Figure 267. Tx1/Tx2 Insertion Loss, Simulated

EXTERNAL LO PORT IMPEDANCE MATCHING NETWORK

The external LO1 and LO2 PORT are used to inject LO signals with very high spectral purity for internal receivers and transmitters. Implement the RF matching network for these ports on single-ended and differential sides of the balun to reduce insertion loss due to reflections at the desired LO frequency. The method of obtaining a matching network is similar to the receiver and transmitter port matching. Depending on the selected divide ratio of the ADRV9001 external LO input frequency divider SPI register setting, a band of frequency must operate in the external LO matching networks, and the chosen division ratio must derive these correctly.

External LO Inputs

Unlike the internal synthesizers that always operate from 6 GHz to 12 GHz regardless of the RF-tuned frequency, when using external LO pins, apply a frequency that is a multiple of two times (i.e., 2x, 4x, 8x, and so on) of the desired RF signal channel frequency. The LO input signal is internally divided by a series of dividers to generate the required LO quadrature relationship at the desired RF frequency of upconversion or downconversion for internal transmitters and receivers. Table 115 describes the specifications for the external LO input pins when the input pins are driven by a differential signal using a balun.

Alternatively, use a single-ended external LO signal source for the positive side of the external LO pin, with the negative side of the input pin terminated with a capacitor to provide AC ground. Set the frequency of the external LO source to 4x of the desired receiver or transmitter frequency, with the external LO divider configured to the division of four for the best in-phase and quadrature generation of the LO necessary for internal receivers and transmitters. Table 116 describes the specifications for the single-ended external LO input source.

In general, the higher power level of the applied external LO signal gives better phase noise to some extent. Use the minimum input power level that satisfies the receiver/transmitter phase noise requirements with some margin. See Table 116 for power level recommendations.

RF PORT INTERFACE INFORMATION

Table 115. Specifications for ADRV9001 RF EXT LO Differential Input Pins

Parameter	Note	Min	Typical	Max	Unit
External LO Frequency	FEXTLO	60		12000	MHz
RF Channel Frequency	FCHANNEL	30		6000	MHz
External LO Power	100 Ω matching Signal amplitude depends on FEXTLO frequency. Typical = 0 dBm for FEXTLO ≤ 2 GHz. From 2 GHz, add 0.5 dB/GHz. For example, Typical = +3 dBm for FEXTLO = 8 GHz.	−6	−3 ~ +3	+6	dBm
Input Impedance	Nominal, small-signal input. Note the following.		100		Ω
Differential Phase Error	Combined Differential-Phase Error, Differential Amplitude Error, Duty-Cycle Error, and Even Order Harmonic Content			±5	Degrees
Differential Amplitude Error				1	dB
Duty-Cycle Error				2	%
Even Order Harmonic Content				−50	dBc
External LO Source Modulus	Must match the internal modulus on the ADRV9001.	83865 60			

Table 116. Specifications for ADRV9001 RF EXT LO Single-Ended Input Pin

Parameter	Note	Min	Typical	Max	Unit
External LO Frequency	FEXTLO	60		2000	MHz
RF Channel Frequency	FCHANNEL	30		1000	MHz
External LO Power	50 Ω matching	0	+3	+6	dBm
Input Impedance	Nominal, small-signal input. Note the following.		50		Ω
External LO Source Modulus	Must match the internal modulus on the ADRV9001.	8386560			

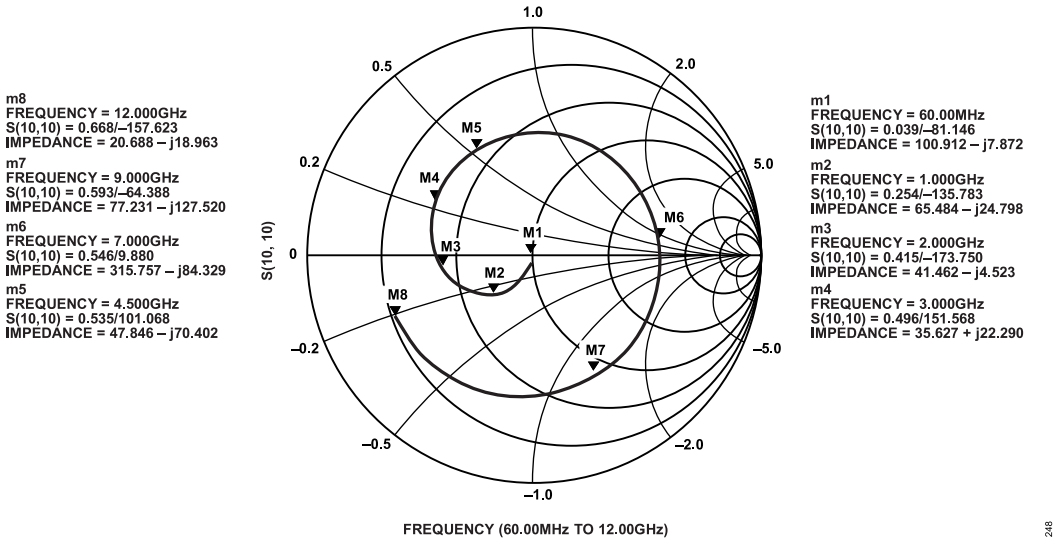


Figure 268. External LO Series Equivalent Differential Input Impedance

Ensure care when selecting an on-board balun for this application. The combination of amplitude and phase balance performance of the balun can affect quadrature error performance. Additionally, duty cycle and differential second-order harmonic distortion impact the ability to correct quadrature error. The recommended minimum requirement for external LO input pins is a combination of no more than 5° differential phase error, 1 dB differential amplitude error, 2% duty-cycle error, and less than −50 dBc even order harmonics (primarily second order).

The ADRV9001 provides a special mode of operation for external LO in the range from 500 MHz to 1000 MHz. In that region, it is possible to inject external LO that produces RF channel frequency with an x1 multiplier.

RF PORT INTERFACE INFORMATION

For example:

For FEXTLO = 500 MHz, the FCHANNEL = 500 MHz

For FEXTLO = 1000 MHz, the FCHANNEL = 1000 MHz

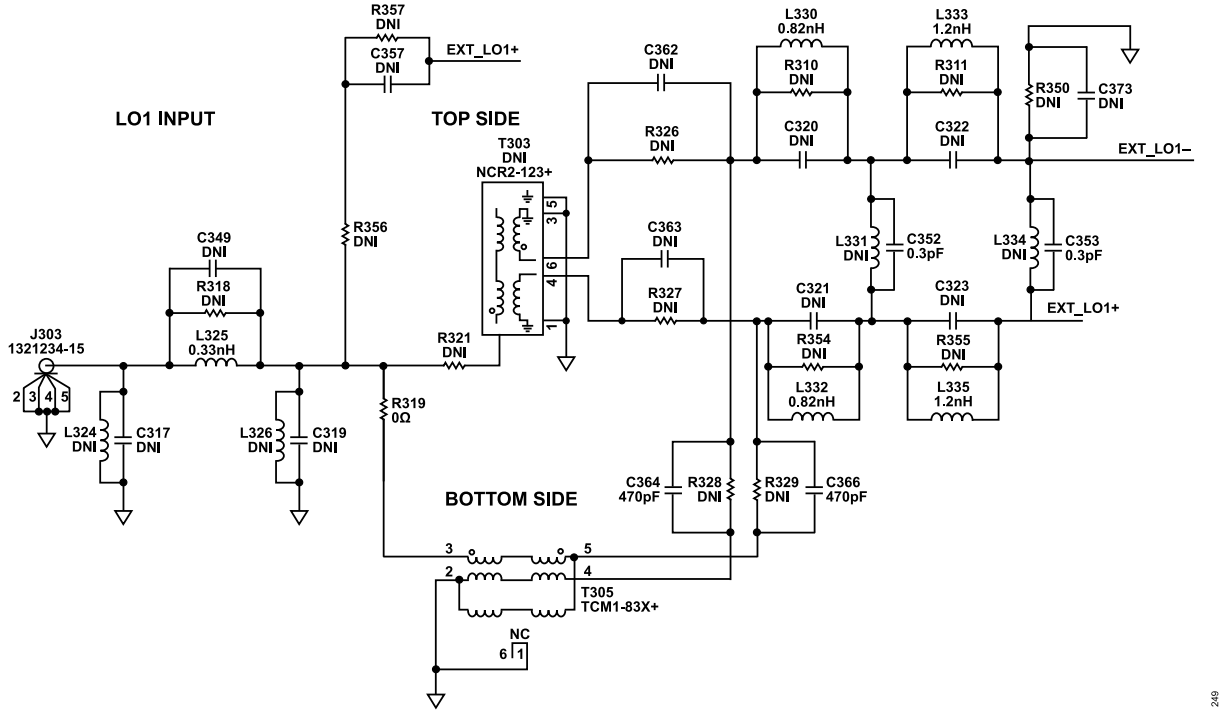


Figure 269. External LO Impedance Matching Network

Table 117. EXTLO1± and EXTLO2± Impedance Matching Network

Frequency	Balun	L324 C317	C349 L325 R318	L326 C319	R356 R321 R319	C357 R357	C362/363 R326/327	C364/366 R328/329	L330/332 C320/321 R310/354	L331 C352	L333/335 C322/323 R311/355	L334 C353	R350 C373
60MHz to 6GHz	MiniCircuits TCM1-83X+	L324: DNI C317: DNI	L325: 0.33nH (AVX L0201) C349: DNI R318: DNI	L326: DNI C319: DNI	R356: DNI R321: DNI R319: 0Ω	C357: DNI R357: DNI	C362/363: DNI R326/327: DNI	R328/329: DNI C364/366: 470pF (Murata GRM03)	L330/332: 0.82nH (AVX L0201) C320/321: DNI R310/354: DNI	L331: DNI C352: 0.3pF (Murata GJM03)	L333/335: 1.2nH (AVX L0201) C322/323: DNI R311/355: DNI	L334: DNI C353: 0.3pF (Murata GJM03)	R350: DNI C373: DNI
6GHz to 12GHz	MiniCircuits NCR2-123+	L324: DNI C317: DNI	L325: DNI C349: DNI R318: 0Ω	L326: DNI C319: DNI	R356: DNI R321: 0Ω R319: DNI	C357: DNI R357: DNI	C362/363: : 470pF (Murata GRM03) R326/327: DNI	R328/329: DNI C364/366DNI	L330/332: DNI C320/321: DNI R310/354: 0Ω	L331: DNI C352: DNI	L333/335: DNI C322/323: DNI R311/355: 0Ω	L334: DNI C353: DNI	R350: DNI C373: DNI

Apply a single-ended external LO signal by bypassing the balun interface and installing an appropriate impedance matching network comprising L324/C317, C349/L325/R318, L326/C319, and an AC-coupling cap of C357. Additionally, install a large cap for C373 to provide an AC-ground for the negative side of the input pins to the internal buffer circuitry.

RF PORT INTERFACE INFORMATION

EXTERNAL LO IMPEDANCE MATCH MEASUREMENT DATA

External RF Port Impedance Match Measurement Data for 60 MHz to 6 GHz Band Match

Return loss is measured on the EXT LO RF ports of evaluation boards and plotted in Figure 270. The blue and pink curves represent three different return loss measurements, and the black dotted line represents the simulated return loss curve. Simulated Insertion loss including balun loss is plotted in Figure 271.

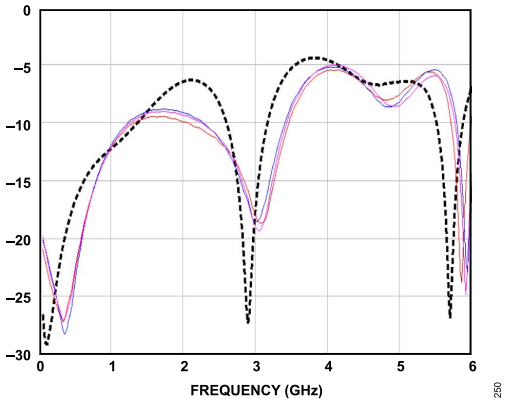


Figure 270. External LO1/ External LO2 Return Loss of External LO Port

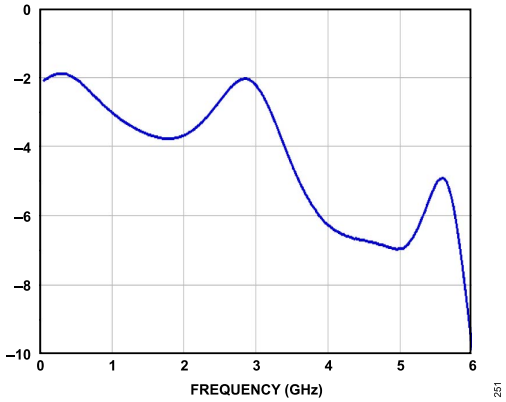


Figure 271. External LO1/ External LO2 Insertion Loss, Simulated

CONNECTION FOR EXTERNAL DEVICE CLOCK (DEV_CLK_IN)

The ADRV9001 can accommodate three different types of external clock signals applied at the device clock input pins. Apply a differential LVDS clock signal or a single-ended clipped-sinewave clock signal from a TCXO to the device input pins. Furthermore, connect a crystal to the device clock input pins to configure it as a crystal oscillator/driver by applying the DC voltage to the MODEA pin, as shown in Table 118.

Table 118. Device Clock Input Interface Modes Description

Voltage Applied at MODEA Pin	Device Clock Input Electrical Interface	DEV_CLK_OUT Divider Value Applied to DEV_CLK_IN Signal	Note
0 V (grounded)	LVDS	/16	Up to 1GHz clock
0.45 V	CMOS or XTAL	/2	CMOS (10MHz to 80MHz) /XTAL (20MHz to 80MHz) with Nominal Gm multiplier = x8
0.9 V	CMOS or XTAL	/2	CMOS (10MHz to 80MHz) /XTAL (20MHz to 80MHz) with Nominal Gm multiplier = x6
1.35 V	CMOS or XTAL	/2	CMOS (10MHz to 80MHz)

RF PORT INTERFACE INFORMATION

Table 118. Device Clock Input Interface Modes Description (Continued)

Voltage Applied at MODEA Pin	Device Clock Input Electrical Interface	DEV_CLK_OUT Divider Value Applied to DEV_CLK_IN Signal	Note
			/XTAL (20MHz to 80MHz) with Nominal Gm multiplier = x2
1.8 V	CMOS or XTAL	/2	CMOS (10MHz to 80MHz) /XTAL (20MHz to 80MHz) with Nominal Gm multiplier = x4

By applying 1.8 V to the MODEA pin A for CMOS interface mode, apply a clipped sinewave clock signal from a TCXO to the pin DEV_CLK_IN+ (E7) through an AC-coupling capacitor, and leave the pin DEV_CLK_IN-(E8) unconnected, as shown in Figure 273.

Connect an Xtal to both DEV_CLK_IN+ and DEV_CLK_IN- pins with a DC voltage between 0.45 V and 1.8 V applied to the MODEA pin, as shown in Figure 274.

When selecting the LVDS mode input clock interface with the MODEA pin grounded, use an external clock as the reference clock for the RFPLL and the clocking PLL on the device, and thus it must be a very clean clock source. Connect the external clock inputs to the DEV_CLK_IN+ (E7) and DEV_CLK_IN- (E8) balls through AC-coupling capacitors and terminate with 100 Ω, as shown in Figure 272. Bias the inputs on the device to a 200 mV voltage level. Figure 275 shows the input impedance plot over the operating frequency. The operational frequency range of the DEV_CLK signal is between 10 MHz and 1000 MHz. Ensure that the external clock peak-to-peak amplitude does not exceed 800mV (note: the positive and negative side of the differential input pins should not exceed 400mV peak-to-peak.). For best synthesizer performance, a high slew-rate signal is best with fast rise and fall times.

Device Clock Interface Modes

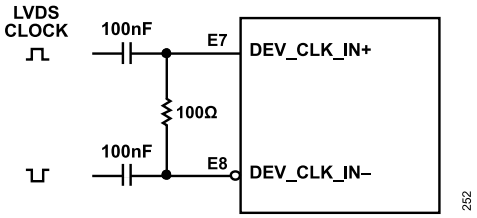


Figure 272. LVDS Interface Mode

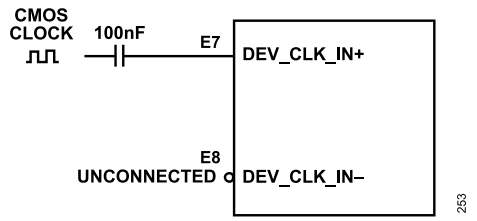


Figure 273. CMOS Interface Mode

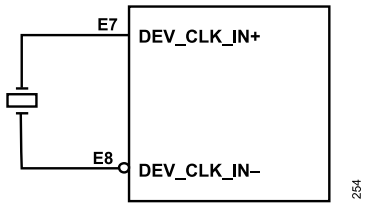


Figure 274. Crystal (XTAL) Interface Mode

RF PORT INTERFACE INFORMATION

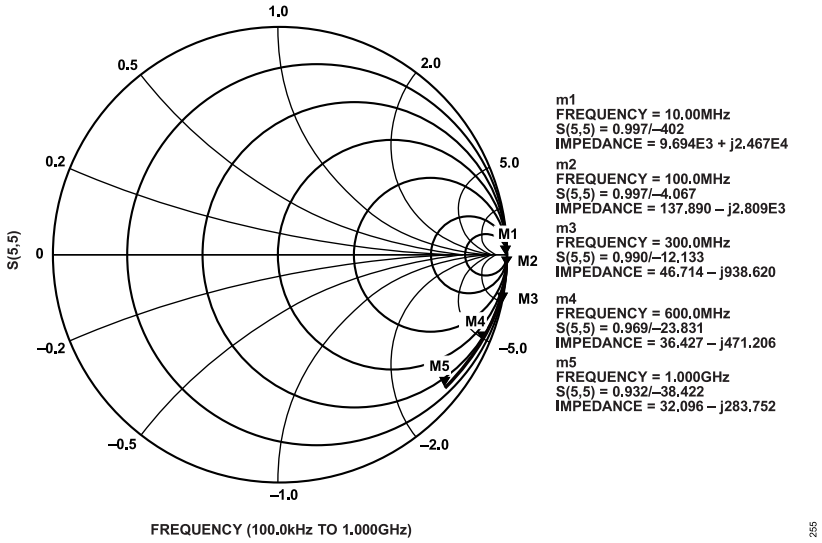


Figure 275. Device Clock Input Series Equivalent Differential Impedance

Implement the device clock input board traces connected to the device clock inputs balls with stripline transmission lines using inner copper layers in the PCB stackup. The frequency of the device clock input signal can go as high as 1 GHz, and the stripline transmission line approach provides better signal integrity of the clock signal, especially at higher frequency, as well as superior shielding of the RF emission of the device clock signal.

The DEV_CLK_IN signal is available on the DEV_CLK_OUT pin. Table 118 describes the default division applied to the DEV_CLK_IN signal after power up. Change this divider later on using an API command. Note that the DEV_CLK_OUT pin is a CMOS type pin with 80 MHz as its maximum frequency of operation. It is intended to provide clock to the BBIC, on-board microcontroller, or audio CODEC type devices. It is not intended for use by another RF sensitive IC.

For applications that use internal RF LOs, it is recommended to use 40 MHz or above as a DEV_CLK frequency to get the best in-band phase noise.

There is a known issue with the DEV_CLK input working in the CMOS mode. When the applied DEV_CLK input signal is in range between 10 MHz to 30 MHz, internal RF LOs exhibit in-band phase noise degradation of around 10 dB.

If this phase noise degradation is not acceptable and it is mandatory to use the DEV_CLK below 30 MHz in the end application:

- Connect the MODEA pin to GND, which enables the differential mode of operation for the DEV_CLK input circuitry.
- Apply DEV_CLK as single-ended to DEV_CLK_IN + (E7 ball) and leave DEV_CLK_IN − (E8 ball) unconnected. Basically, use the same hardware configuration as in the CMOS mode, outlined in Figure 273.
- Ensure the amplitude of the applied signal does not exceed 1 V peak-to-peak. It is recommended to use a signal in the range of 400 mW peak-to-peak. The DEV_CLK_IN + (E7 ball) inputs, when operating in the LVDS mode, is biased on the device to around 200 mV voltage level. A maximum of 400mV peak-to-peak amplitude ensures that the external clock stays compliant with the electrical specification of DEV_CLK_IN + pin.

DEV_CLK_IN PHASE NOISE REQUIREMENTS

To prevent performance degradation, the DEV_CLK reference must be a very clean signal. The synthesizer provides best performance if the applied reference is ideal. However, that is unrealistic. Table 119 lists the required phase noise of the DEV_CLK signal for a 1dB system PN degradation compared to an ideal DEVICE CLOCK. For different DEV_CLK frequencies, the table can be scaled appropriately. Clock source with phase noise performance outlined in Table 120 (or better) allows the ADRV9001 to deliver data sheet performance. Note that Table 119 provides reference information for the ADRV9001 operating with LTE type standards. Each standard determines its own DEV_CLK phase noise requirements. As an example, Table 120 provides recommendations for the DEV_CLK when the ADRV9001 is intended to operate with LMR type standards. Ideally, derive the DEV_CLK phase noise requirement from the user-specific application and its requirements set for the adjacent channel rejection.

In general, using a higher phase noise source can degrade performance delivered by the ADRV9001 transceiver.

RF PORT INTERFACE INFORMATION

Table 119. DEV_CLK_IN Phase Noise Requirements for 1dB System PN Degradation Compared to an Ideal DEVICE CLOCK

Frequency Offset From Carrier	Narrow PLL Loop Bandwidth (Approximately 50 kHz) (Default, Typically <3 GHz)			Wide PLL Loop Bandwidth (Approximately 300 kHz) (User Configured, Typically >3 GHz)		
	122.88 MHz (dBc/Hz)	153.6 MHz (dBc/Hz)	245.76 MHz (dBc/Hz)	122.88 MHz (dBc/Hz)	153.6 MHz (dBc/Hz)	245.76 MHz (dBc/Hz)
100 Hz	−113.02	−111.08	−107.00	−114.02	−112.08	−108.00
1000 Hz	−125.02	−123.08	−119.00	−127.02	−125.08	−121.00
10 kHz	−133.02	−131.08	−127.00	−138.02	−136.08	−132.00
100 kHz	−137.02	−135.08	−131.00	−146.02	−144.08	−140.00
1 MHz	−133.02	−131.08	−127.00	−147.02	−145.08	−141.00
10 MHz	−104.02	−102.08	−98.00	−118.02	−116.08	−112.00

Table 120. DEV_CLK_IN Phase Noise Requirements for LMR Type Applications

Frequency Offset From Carrier	PLL Loop Bandwidth Optimized for LMR Type Applications, 38.4 MHz (dBc/Hz)
100 Hz	−106
1000 Hz	−151
10 kHz	−151
100 kHz	−151
10 MHz	−151

CONNECTION FOR MULTICHIP SYNCHRONIZATION (MCS) INPUT

An LVDS type MCS signal applied between MCS + (D7) and MCS − (D8) pins is used to provide time alignment synchronization for both the RF and datalink systems. Similar to the device clock input signal, use a clock source with fast rise and fall times as an MCS input signal. Implement PCB traces for routing MCS signals by following the guidelines similar to the LVDS mode device clock input trace.

Note: The CMOS type of MCS signal applied only on the D7 pin is also supported.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

The ADRV9001 is a highly integrated RF agile transceiver with significant signal conditioning integrated onto one chip. Due to the integration complexity of the ADRV9001 and its high pin count, a carefully printed circuit board (PCB) layout is important to optimize performance. This section provides a checklist of issues to look for and guidelines on how to optimize the PCB to mitigate performance issues. The goal of this document is to help achieve the best possible performance from the ADRV9001 while reducing board layout effort. It is assumed that the reader is an experienced analog/RF engineer who understands RF PCB layouts and has an understanding of RF transmission lines as well as low-noise analog design techniques. The ADRV9001 evaluation card is used as the reference for this information, but all guidelines are best practices that can be applied to other reference designs. This document provides guidelines for system designers and discusses the following issues relative to layout and power management.

- Selecting the PCB material and stackup
- Fanout and layout guidelines relative to trace widths and spacing
- Component placement and routing guidelines
- RF and data port transmission line layout
- Isolation techniques used on the ADRV9001 customer evaluation board

SELECTING THE PCB MATERIAL AND STACKUP

Figure 276 shows the PCB stackup used for the ADRV9001 customer evaluation boards. These boards employ 12 layers to achieve proper routing and isolation to best demonstrate all device functionality. The dielectric material used is I-SPEED with a thickness of 7 mil on outer layers. The board design uses the I-SPEED laminate for its low loss tangent at high frequencies. The ground planes under the I-SPEED laminate (layers 2 and 11) are the reference planes for the transmission lines routed on the outer surfaces. These layers are solid copper planes under the RF traces with no discontinuities. Layers 2 and 11 are crucial to maintaining the RF signal integrity.

RF traces on the outer layers must be a controlled impedance to get the best performance. These outer layers use 0.5 ounce copper. The 0.5- and 1-ounce copper thicknesses are used for all the inner layers in this board. All ground planes on this board are full copper floods with no splits except vias, through-hole components, and isolation structures.

Layers 3, 5, 7, and 9 are mainly used to route power-supply domains. The data port interface lines are routed on layers 1, 7, and 12. Those layers have impedance control set to 100 Ω differential for the differential LVDS pairs. The remaining digital signals are routed on inner layers 3, 5, 9, and 10. Table 121 describes the details of the trace impedance controls used on different layers.

There are no buried-in or blind vias used in this PCB design. All vias used in the PCB design are through-hole type. For vias carrying high frequency or RF sensitive signals, apply the back drilling technique.

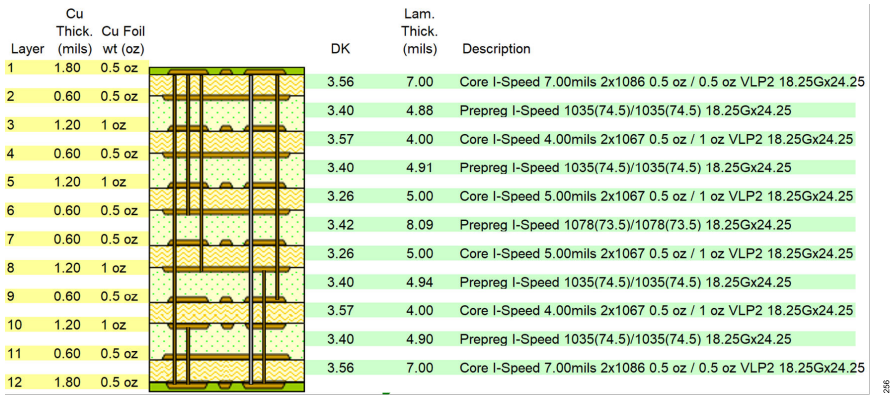


Figure 276. ADRV9001 Customer Evaluation Card Stackup

Table 121. Impedance Table

Layer	Structure Type	Coated Microstrip[1]	Target Impedance (Ω)	Impedance Tolerance (Ω)	Target Linewidth (mils)	Edge-Coupled Pitch (mils)	Reference Layers	Modeled Linewidth (mils)	Modeled Impedance (Ω)	Coplaner Space (mils)
1	Single-Ended	N/A	50.00	±5	12.00	0.00	(2)	13.00	50.87	9.50
1	Single-Ended	Yes	50.00	±5	13.50	0.00	(2)	12.00	50.42	10.75

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

Table 121. Impedance Table (Continued)

Layer	Structure Type	Coated Microstrip[1]	Target Impedance (Ω)	Impedance Tolerance (Ω)	Target Linewidth (mils)	Edge-Coupled Pitch (mils)	Reference Layers	Modeled Linewidth (mils)	Modeled Impedance (Ω)	Coplaner Space (mils)
1	Edge-Coupled Differential	Yes	100.00	±10	8.25	15.25	(2)	8.00	100.43	9.15
1	Edge-Coupled Differential	N/A	100.00	±10	7.50	14.50	(2)	9.00	100.55	9.25
3	Single-Ended	N/A	50.00	±5	4.00	0.00	(2, 4)	4.25	49.53	17.88
3	Edge-Coupled Differential	N/A	100.00	±10	3.75	10.75	(2, 4)	3.75	100.86	12.02
7	Edge-Coupled Differential	N/A	100.00	±10	6.00	14.25	(6, 8)	6.00	99.75	12.02
9	Edge-Coupled Differential	N/A	100.00	±10	6.25	15.00	(8, 11)	6.00	100.68	12.14
10	Edge-Coupled Differential	N/A	100.00	±10	4.25	9.50	(11, 8)	4.50	100.23	11.89
12	Edge-Coupled Differential	Yes	100.00	±10	8.00	15.25	(11)	8.00	100.80	10.00
12	Single-Ended	Yes	50.00	±5	12.00	0.00	(11)	12.00	50.31	10.00
12	Edge-Coupled Differential	N/A	100.00	±10	7.50	14.50	(11)	9.00	100.55	9.25
12	Edge-Coupled Differential	N/A	100.00	±10	8.25	15.50	(11)	8.25	99.64	10.02

[1] N/A means not applicable.

FANOUT AND TRACE SPACE GUIDELINES

The ADRV9001 device family uses a 196-pin ball grid array (BGA) 12 mm × 12 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. There are RF pins on the outer edges of the ADRV9001 package. This helps in routing the critical signals without a fanout via. Each digital signal is routed from the BGA pad using a 4.5 mil trace. The trace is connected to the BGA using the via-in-the-pad structure. The signals are buried in the inner layers of the board for routing to other parts of the system.

Take extra care to ensure that the DEV_CLK signal is shielded from any potential source of noise. The recommended approach is to use differential signaling for the DEV_CLK clock. Route the data port interface signals as 100 Ω differential pairs when used in the LVDS-SSI mode . Figure 277 shows the fan-out scheme of the ADRV9001 evaluation card. There are no traces routed between the BGA pads on the top layer. As mentioned, the ADRV9001 evaluation card uses the via-in-the-pad technique. Use this routing approach for the ADRV9001 if there are no issues with manufacturing capabilities.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

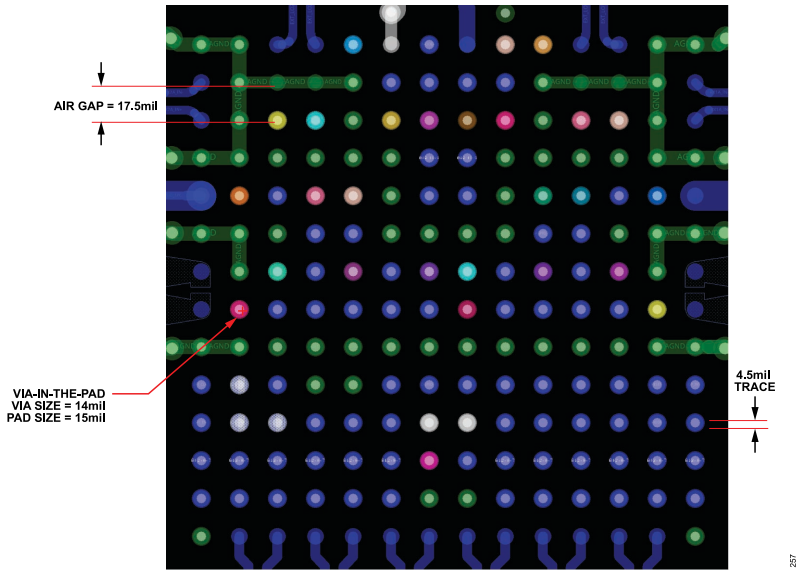


Figure 277. Trace Fan-Out Scheme on ADRV9001 Evaluation Card (PCB Layer TOP and Layer 8 Enabled)

COMPONENT PLACEMENT AND ROUTING PRIORITIES

The ADRV9001 transceiver requires a few external components to function, and the ones needed require careful placement and routing to optimize performance. This section provides a priority order and checklist to properly place and route critical signals and components, as well as those whose location and isolation are not as critical.

Board layout design involves compromise. The recommendations in this section are intended for wide RF bandwidth applications. For narrow RF bandwidth applications, the board line impedance parameters within this document may not be optimal.

The following list provides general suggestions for board design:

- Match the customer board design as close as possible to the ADRV9001 board design.
- Be attentive to power distribution and power ground return methodology.
- Do not run high-speed digital lines close to the DC power distribution routes or RF line routes.

Signals with Highest Routing Priority

RF lines and DEV_CLK clock are the most critical signals. Route these with the highest priority. Figure 278 shows the general directions to route each signal so that are properly isolated from noisy signals.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

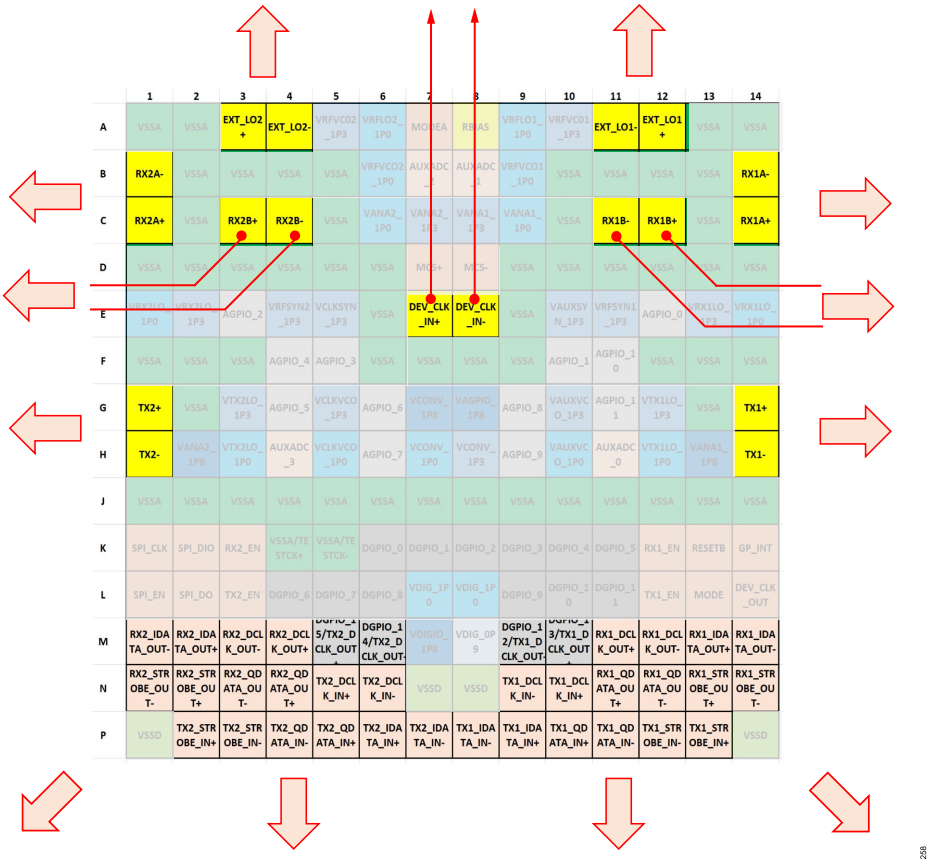


Figure 278. RF I/O, DEV_CLK, and Data Port Signal Routing Guidelines

- RF baluns are typically used to interface single-ended signals to the differential receiver and transmitter ports. These baluns and their associated matching circuits affect the overall RF performance. Make every effort to optimize the component selection and placement to avoid performance degradation. The section describes the proper matching circuit placement and routing in more detail. Please see the respective paragraph for more information.
- Use microstrip or coplanar waveguides (CPWG) for transmission lines. These structures do not require via structures that cause additional impedance discontinuities that vary across frequency. For Rx1B and Rx2B receiver ports, which do not have balls on the perimeter of the BGA, a via structure such as stripline is necessary.
- Design the RF line systems between the device ball pad reference plane and the balun/filter reference plane for a differential impedance (ZDIFF) of 100 Ω for the receivers and 50 Ω for the transmitters. This is a compromise impedance with respect to frequency and a good starting point for design. Optimize the ZDIFF to fit a narrower frequency range. It is desirable to design the lines for reasonable coupling (−10 dB to −15 dB) to promote adequate electromagnetic interference (EMI) suppression performance.
- In most cases, the required board artwork stackup is going to be different than the ADRV9001 evaluation board stackup. Optimization of RF transmission lines specific to the desired board environment is essential to the design and layout process.
- The ADRV9001 evaluation board uses microstrip lines for receiver and transmitter RF traces. Some data port signal are routed using a combination of microstrip lines at the bottom of the PCB and stripline traces on the internal layers due to board complexity. In general, RF traces should avoid using vias if a direct line route is possible.
- Differential lines from the balun to the RX and TX pins must be as short as possible. The length of the single-ended transmission line should also be short to minimize the effects of parasitic coupling.
- The system designer can optimize the RF performance with the proper selection of balun, matching components, and AC-coupling capacitors. The external LO traces and the DEV_CLK_IN traces can require matching components as well to ensure optimal performance. The matching network design is explained in greater detail in the respective section of this document.
- RF signal path isolation is critical to achieving the level of isolation specified in the ADRV9001 data sheet. The Isolation Techniques Used on the ADRV9001 Evaluation Card section provides more details on proper isolation.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

- For each RF transmitter output, install a 10 µF capacitor near the balun power-supply pin connected to the VANA1_1P8, VANA2_1P8 supplies. If baluns with no DC supply connection are used, supply power to the transmitter outputs using RF chokes. Connect chokes between the VANA1_1P8 and Tx1 output and VANA2_1P8 and Tx2 output, respectively. In both cases, the 10 µF capacitor acts as a reservoir for transmitter supply current. The Transmitter Balun DC Supply Options section describes the transmitter output power supply configuration in more detail.
- Connect the external clock inputs to the DEV_CLK_IN + (E7) and DEV_CLK_IN − (E8) pins using AC-coupling capacitors. Use a 100 Ω termination at the input to the device. Figure 279 illustrates the recommended placement for termination resistor near the DEV_CLK_IN pins. Traces are shielded by surrounding ground with vias staggered along the edge of the differential trace pair. This arrangement creates a shielded channel that protects the reference clock from any interference from other signals. Refer to the ADRV9001 evaluation board layout for exact details.

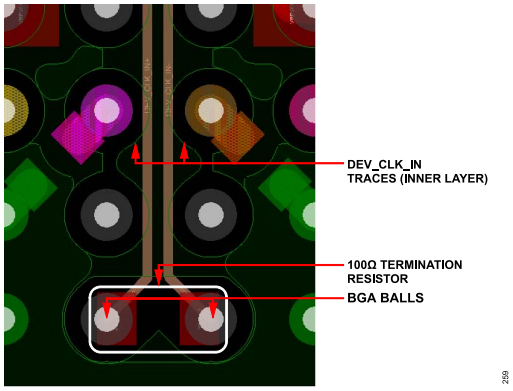


Figure 279. DEV_CLK_IN Signal Routing Recommendations

- The EXT_LO1+ (A12), EXT_LO1- (A11), EXT_LO2+ (A3), EXT_LO2- (A4) pins are internally DC-biased. If an external LO is used, connect it through AC-coupling capacitors.
- The data port interface is routed at the beginning of the PCB design and with the same priority as RF signals. This is especially important if data port runs in the LVDS configuration. Pay attention to provide appropriate isolation between the data port differential pairs.

Signals with Secondary Routing Priority

Power-supply quality has a direct impact on the overall system performance. To achieve optimal performance, follow the recommendations for power-supply routing. The following recommendations outline how to route different power domains and which supplies to connect to the same supply but separated by a ferrite bead.

A general recommendation for power-supply routing is to follow the star methodology, in which each power domain is delivered by a separate trace from the source supply. Make sure that each power trace is surrounded by ground. Figure 280 shows an example of such traces routed on the evaluation card on Layer 3. Each trace is separated from any other signal by ground plane fill and vias. This approach is essential to provide necessary isolation between power domains.

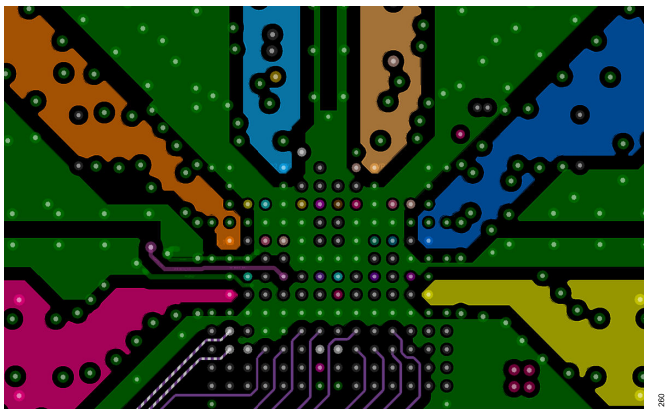


Figure 280. Layout Example of Power-Supply Connections Routed with Ground Shielding (Layer 3)

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

Figure 281 shows an example of how to place the ferrite beads, reservoir capacitors, and decoupling capacitors. The recommendation is to connect a ferrite bead between a power plane and the ADRV9001 at a distance from the ADRV9001. The ferrite bead supplies a trace with a reservoir capacitor connected to it. That trace is then shielded with ground and provides power to the ADRV9001 power pin. Place a 1 µF capacitor near the power-supply pin with the ground side of the bypass capacitor placed so that ground currents flow away from other power pins and their bypass capacitors.

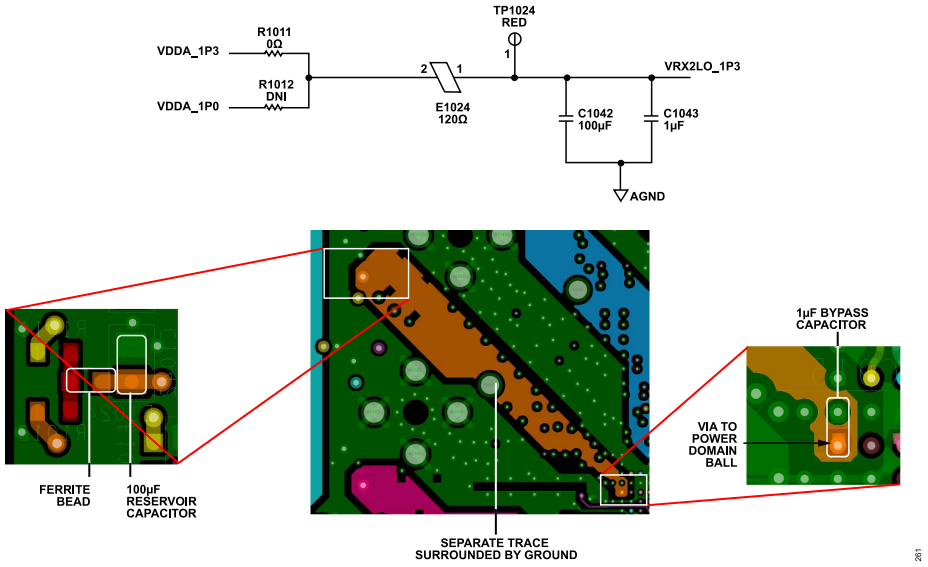


Figure 281. Placement Example of Ferrite Beads, Reservoir, and Bypass Capacitors on ADRV9001 Customer Card (Layers: TOP, 3-Power, and BOTTOM)

There are two possible power-supply architectures for ADRV9001 transceivers.

- High-performance, low-risk four power domains:
  - 1.8 V digital
  - 1.8 V analog
  - 1.3 V analog
  - 1.0 V digital

This approach uses the ADRV9001 internal LDOs to generate 1.0 V for all internal blocks. Figure 282 outlines the power-supply routing recommendations for this architecture.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

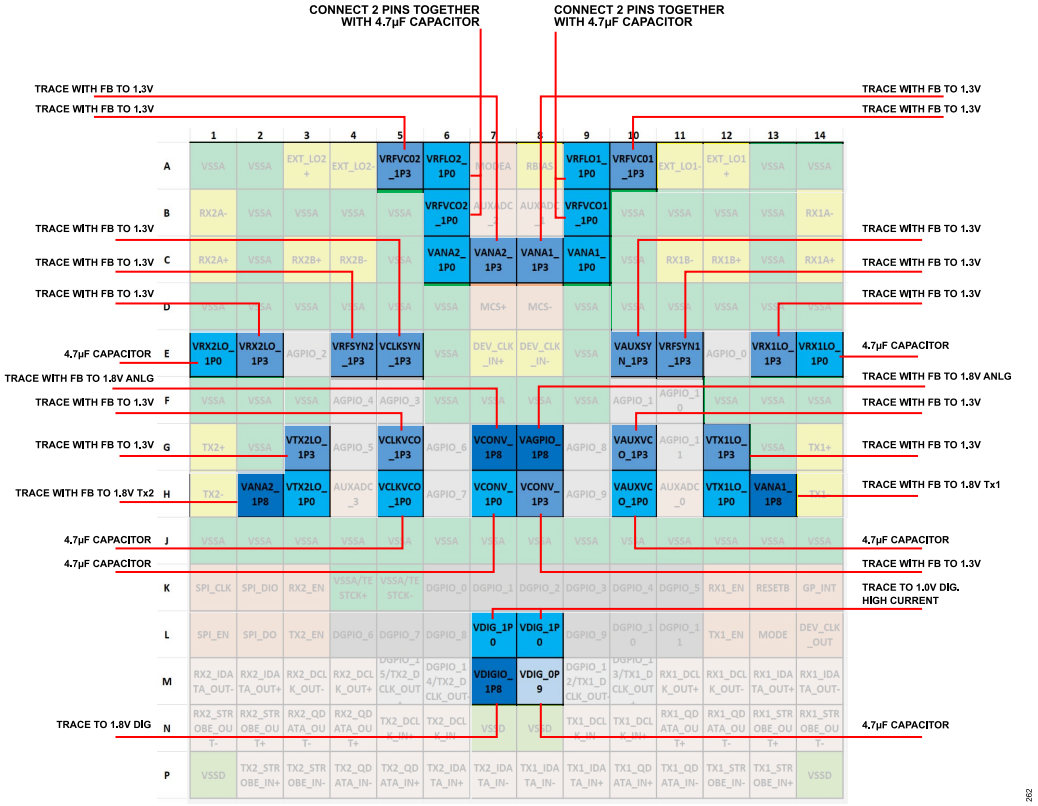


Figure 282. ADRV9001 Power-Supply Domains with Connection Guidelines, All Internal LDOs in Use

- Power-supply optimization, higher risk (use noise sensitive 1.0 V analog) five power domains:
  - 1.8 V digital
  - 1.8 V analog
  - 1.3 V analog
  - 1.0 V digital
  - 1.0 V analog
-

This approach uses some of the ADRV9001 internal LDOs to generate 1.0 V for internal blocks. For the remaining blocks, it expects the 1.0 V to be delivered from an external power source. Figure 283 outlines the power-supply routing recommendations for this architecture.

- For domains shown in Figure 283 that should be powered through a ferrite bead (FB), take care to place the FBs near the ADRV9001 supply pins. Also space the FBs to ensure their electric fields do not influence each other. The FB supplies a trace with a reservoir capacitor connected to it. That trace is then shielded with ground and provides power to the input power pin.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

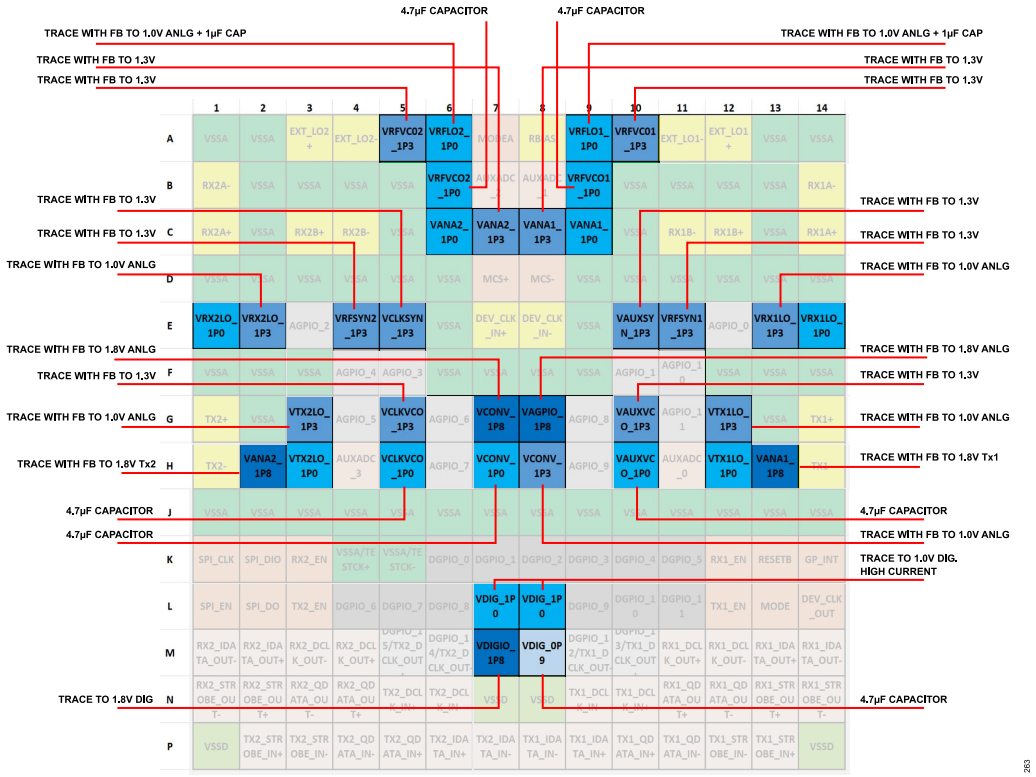


Figure 283. ADRV9001 Power-Supply Domains with Connection Guidelines, Some Internal LDOs Bypassed, 1.0V Analog Domain Required

Place ceramic 4.7 µF bypass capacitors at the VRFVCO2_1P0, VRFVCO1_1P0, VRX2LO_1P0, VRX1LO_1P0, VCLKVCO_1P0, VAUXVCO_1P0, VCONV_1P0, and VDIG_0P9 pins. Place these capacitors as close as possible to the device with the ground side of the bypass capacitor placed so that ground currents flow away from other power pins and their bypass capacitors, if at all possible.

In a scenario when power supply follows the recommendations in Figure 283 (some internal LDOs bypassed, external 1.0 V analog domain in use), 4.7 µF capacitors at VRX2LO_1P0 and VRX1LO_1P0 pins are not necessary. The 1.0 V domains connected to the VRFLO1_1P0 and VRFLO2_1P0 pins require 1 µF capacitors.

Signals with Lowest Routing Priority

The following guidelines govern the signals with the lowest signal routing priority. Route these after all critical signal routes are completed so they do not interfere with the critical component placement and routing. Route the signals shown in Figure 284 with the lowest priority.

- Connect a 4.99 kΩ resistor to the RBIAS pin (C14). This resistor must have a 1% tolerance or better.
- The device supports joint test action group (JTAG) boundary scan, and the MODE pin is used to access the function. Connect the MODE pin (L13) to ground for normal operation. Refer to the data sheet for JTAG boundary scan information.
- Connect the RESETB pin (K13) to VDIGIO_1P8 with a 10 kΩ resistor for normal operation. Reset the device by driving this pin low.
- When routing digital signals from rows K and below, it is important to route them away from the analog section (rows A through H). The digital-signal routing should not pass above the red dotted line highlighted in Figure 284.
- Route the AGPIO_N signals using inner PCB layers. These signals control the analog blocks such as power amplifiers or low-noise amplifiers. Also use the AGPIO_0 through AGPIO_3 as general-purpose analog outputs when muxed with the internal AUXDAC outputs. To prevent noise coupling into these signals, route them away from the digital region (above the red dotted line highlighted in Figure 284).
- Route the AuxADC_N signals using inner PCB layers. These signals sense the analog voltage levels such as temperature sensors. To prevent noise coupling into these signals, route them away from the digital region (above the red dotted line highlighted in Figure 284).
- The MODEA signal sets up the operation of the DEV_CLK_IN ± pins (LVDS differential, CMOS single-ended, XTAL with different bias voltage). Follow the recommendations in the Connection for External Device Clock (DEV_CLK_IN) section when controlling this pin.
- The MCS± signals are treated as differential. If using the multichip synchronization feature in the end application, route these signals with traces matching the length of the DEV_CLK_IN ± traces.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

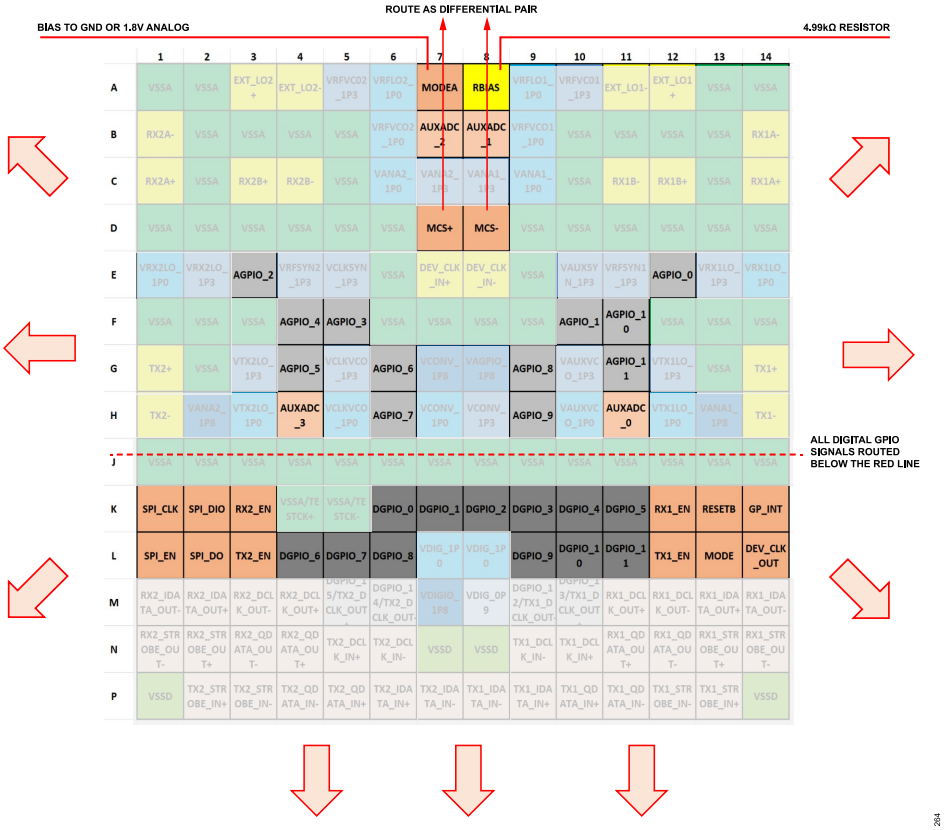


Figure 284. ADRV9001 AuxADC, SPI, Analog GPIO/AuxDAC, MCS±, and Digital GPIO Signal Routing Guidelines

RF AND DATA PORT TRANSMISSION LINE LAYOUT

RF Line Design Summary

The RF line design is a compromise among many variables. Line impedance, line-to-line coupling, and physical size represent the parameters subject to compromise. Smallest physical size is in direct opposition to the ZCM of the line, which is directly opposed to the line EMI performance. In addition, the interface between the RF line width and the device ball pad diameter on the PCB represents a potential discontinuity. As the RF line width approaches the ball pad diameter, the risk associated with potential interface discontinuity reduces.

Balanced lines for differential mode signaling used between the device and the RF balun should be as short as possible. The length of the single-ended transmission lines for RF signals should also be as short as possible. Keeping signal paths as short as possible reduces susceptibility to undesired signal coupling and 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, route these prior to any other signals and maintain the highest priority in the PCB layout process.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

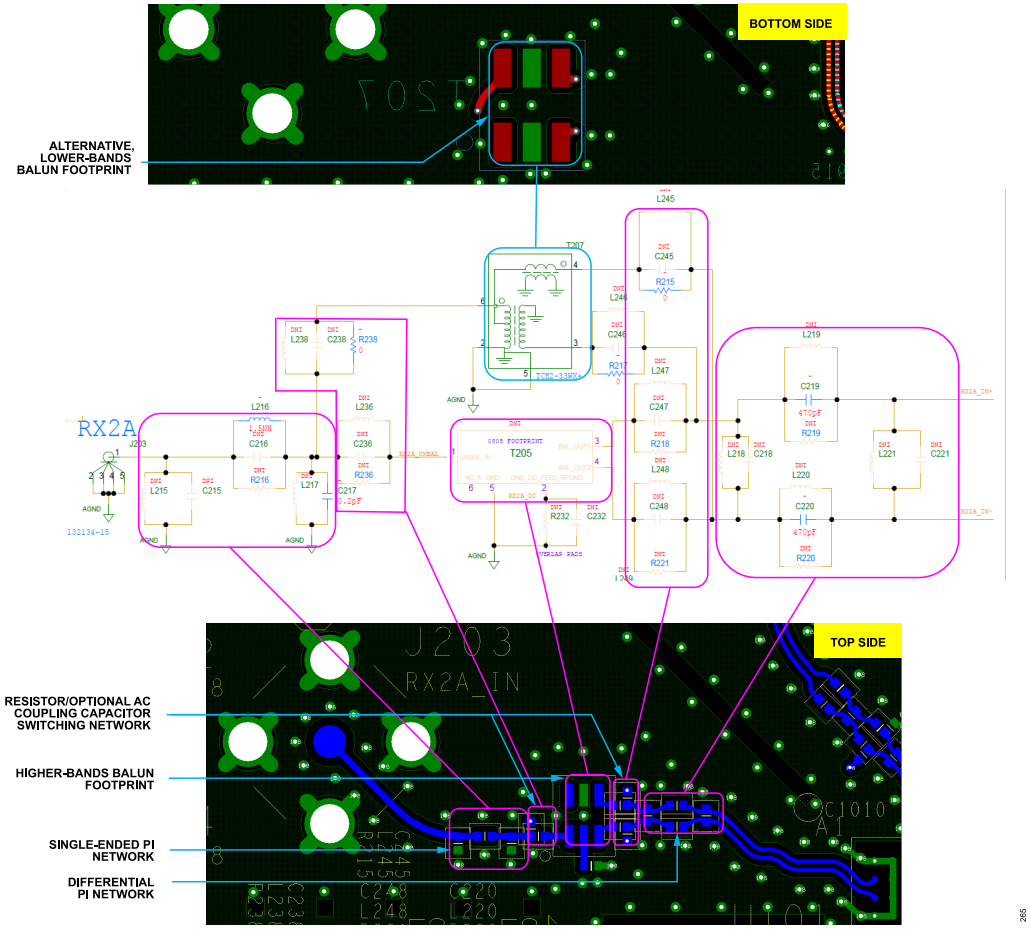


Figure 285. Receiver Matching Network on ADRV9001 Evaluation Board

The circuit in Figure 285 shows the layout topology for the chosen receiver matching network. Note the location and orientation of each component (placement is critical for expected performance). Similarly, the circuit in Figure 286 shows the layout topology used for the transmitter matching network (see the section for circuit details). The Transmitter Bias and Port Interface section provides more details concerning the DC supply to the transmitter.

All the RF signals must have a solid ground reference under each path to maintain the desired impedance. None of the critical traces should run over a discontinuity in the ground reference.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

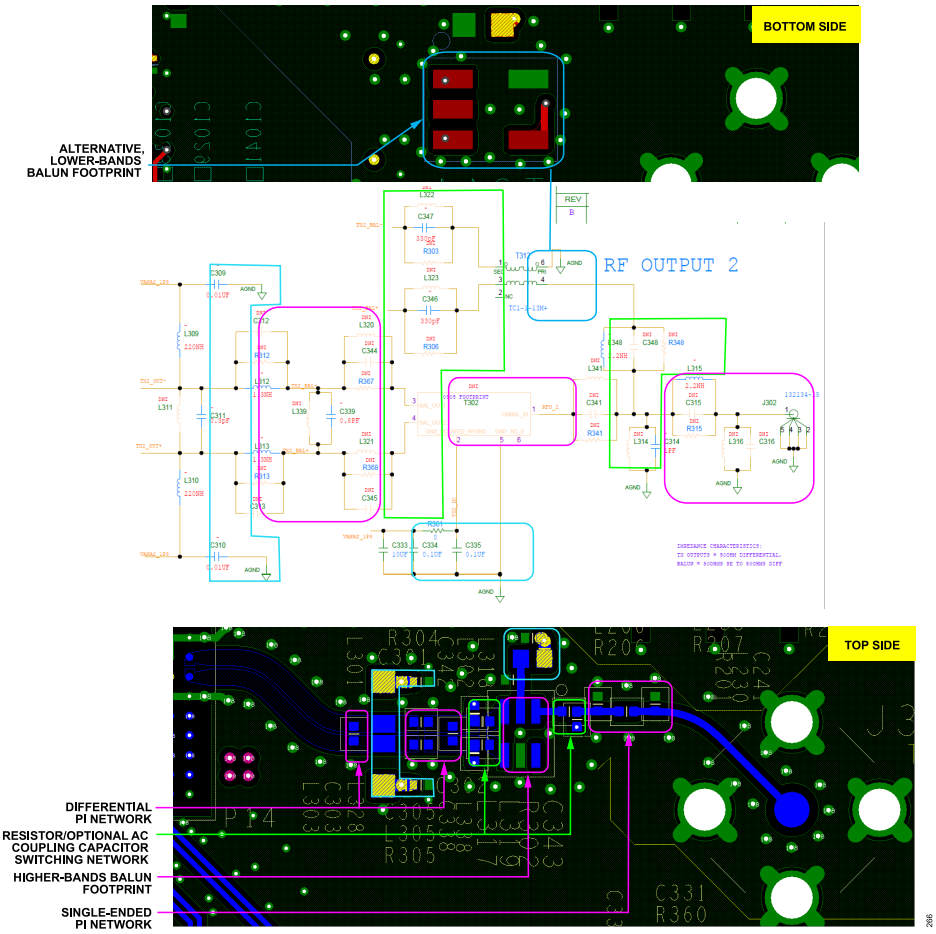


Figure 286. Transmitter Matching Network on ADRV9001 Evaluation Board

Transmitter Bias and Port Interface

This section considers the DC biasing of the ADRV9001 transmitter outputs and how to interface to each transmitter port. At full output power, each differential output side draws approximately 100 mA of DC bias current. The transmitter outputs are DC biased to a 1.8 V supply voltage using either RF chokes (wire-wound inductors) or a transformer (balun) center tap connection.

Careful design of the DC bias network ensures optimal RF performance levels. When designing the DC bias network, select components with low DC resistance (RDCR) to minimize the voltage drop across the series parasitic resistance element with either of the DC bias schemes suggested in Figure 287 and Figure 288. The red resistors (R_DCR) indicate the parasitic elements. As the impedance of the parasitic elements increases, the voltage drop (ΔV) across the parasitic element increases, which causes the transmitter RF performance (such as PO,1dB, PO,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. If using chokes, they should be very well matched (including PCB traces). Uneven matching of chokes design can cause unwanted emission of spikes at the transmitter output. This emission can affect components connected to the transmitter output.

The recommended DC bias network uses the center tap balun, as shown in Figure 288. This network has fewer parasitic elements and fewer total components.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

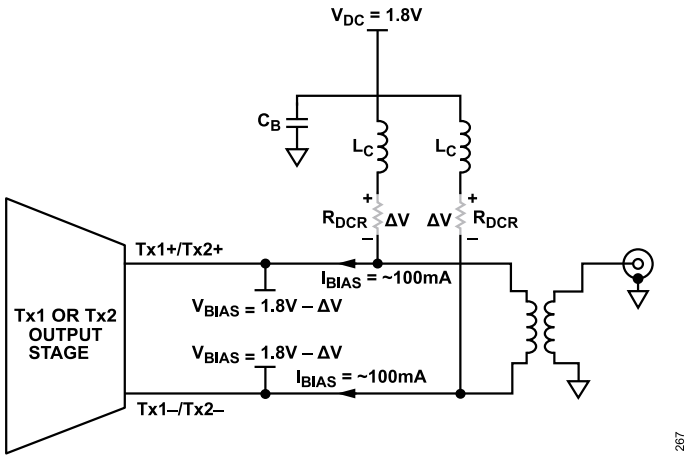


Figure 287. ADRV9001 DC Bias Configuration for the Transmitter Output Using Wire-Wound Chokes

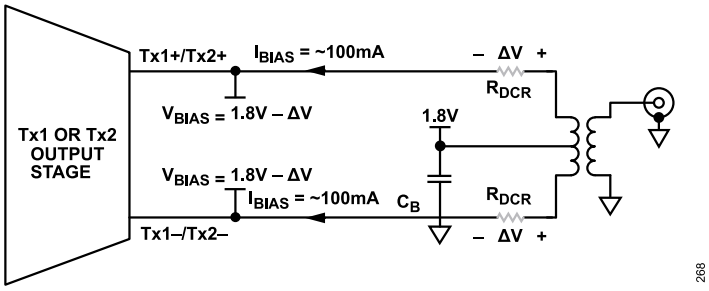


Figure 288. ADRV9001 DC Bias Configuration for the Transmitter Output Using a Center-Tapped Transformer

The ADRV9001 evaluation board provides flexibility to configure each transmitter output to work with either a center-tapped transformer (balun) or a set of two closely matched wire-wounded chokes. The center-tapped transformer passes the bias voltage directly to the transmitter outputs through each differential input. This configuration offers the lowest component count.

In some cases, the desired balun does not provide a DC connection to the transmitter output lines. To support this situation, the ADRV9001 evaluation board provides the placeholders for RF chokes connected to the VANA1_1P8 (for Tx1 output) and VANA2_1P8 (for Tx2 output) supply. It also provides the placeholders for AC-coupling capacitors to prevent creating a DC short through the balun to ground.

Impedance matching networks on the balun single-ended port are usually required to achieve optimum performance. In addition, AC-coupling is often required on the single-ended side if the balun contains a DC path from one of the transmitter's differential outputs to the single-ended port.

Careful planning is required to select the transmitter balun. If a transmitter balun is selected that requires a set of external DC bias chokes, it is necessary to find the optimum compromise between the choke physical size, choke DC resistance (RDCR), and the balun pass band insertion loss. Refer to the relevant section of this document for more information on selecting the transmitter output balun and RF choke as well as recommendations for matching circuit.

Transmitter Balun DC Supply Options

Each transmitter requires approximately 200 mA supplied through an external connection. The PCB layout of the ADRV9001 board allows the use of external chokes to provide 1.8 V power domain to the ADRV9001 outputs. This allows users to try different baluns that do not have a DC center tap pin to supply the bias voltage to the transmitter outputs.

To reduce switching transients when attenuation settings change, the balun DC feed is powered directly by the 1.8 V plane. The geometry of the 1.8 V plane is designed so that each balun or each pair of chokes is associated with its transmitter output. The VANA1_1P8 is used to power the Tx1 output and VANA2_1P8 is used to power the Tx2 output.

If careful layout and isolation of the DC supply is not followed, it can adversely affect transmitter-transmitter isolation. Figure 289 shows the power-supply layout configuration used on the ADRV9001 board to achieve the desired transmitter-transmitter isolation performance. This image illustrates star connection from the common 1.8 V analog power plane.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

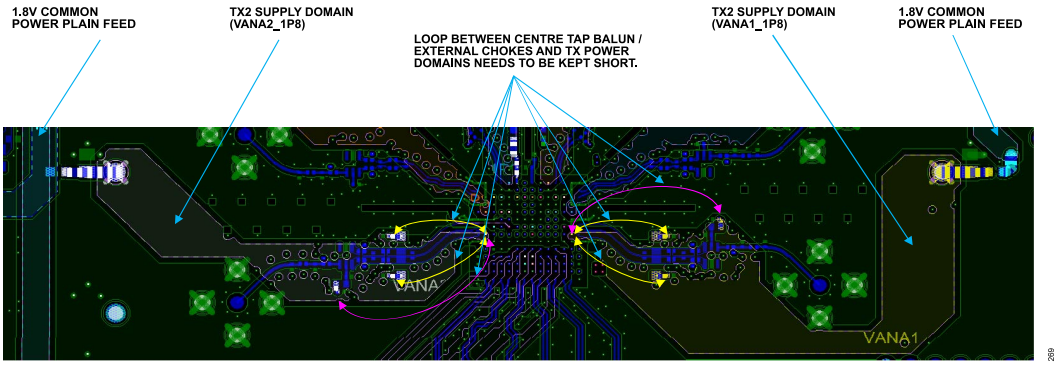


Figure 289. 1.8V Transmitter Power-Supply Routing on the ADRV9001 Evaluation Board

Figure 290 and Figure 291 show an example of the balun feed supply designed to achieve the isolation managed in the evaluation board.

DC Balun

When using a transmitter balun able to conduct DC, use the system shown in Figure 290. Place the decoupling cap near the transmitter balun as close as possible to the balun's DC feed pin. Its orientation is perpendicular to the ADRV9001 device so the return current avoids a ground loop with the ground pins surrounding the receiver input. The customer card provides an option to install an RF isolation inductor, which can provide extra isolation between the Tx1 and Tx2 balun supply feeds. A 10 µF capacitor and a 0.1 µF capacitor are helpful on the DC feed pin to eliminate transmitter spectrum spurs and dampen the transients. Note that when using this supply approach, the series matching components must be DC shorts. It is preferred to use 0 Ω if an inductor is not needed to match the balun impedance to the transmitter output impedance.

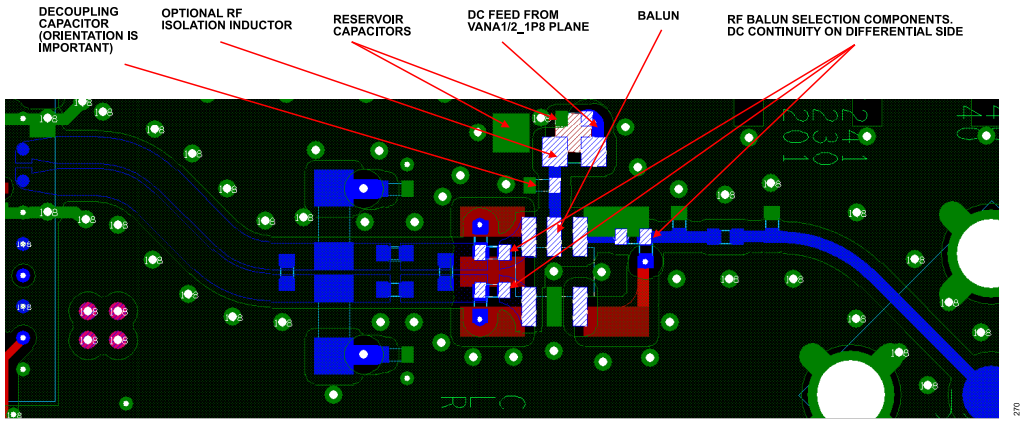


Figure 290. Transmitter Power Supply for a Balun with a Center Tap

Chokes

The ADRV9001 evaluation board can be configured to use a transmitter balun not capable of conducting DC current. In such a scenario, install DC chokes as well as their decoupling capacitors, as highlighted in Figure 291. Take care to match both the chokes to avoid potential current spikes. Difference in parameters between both the chokes can cause unwanted emission at transmitter outputs. Note that if the differential input to the balun can create a DC short to ground (through the balun), the series matching components must be capacitors. If a short can form on the single-ended side, the single-end series blocking element must be a capacitor.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

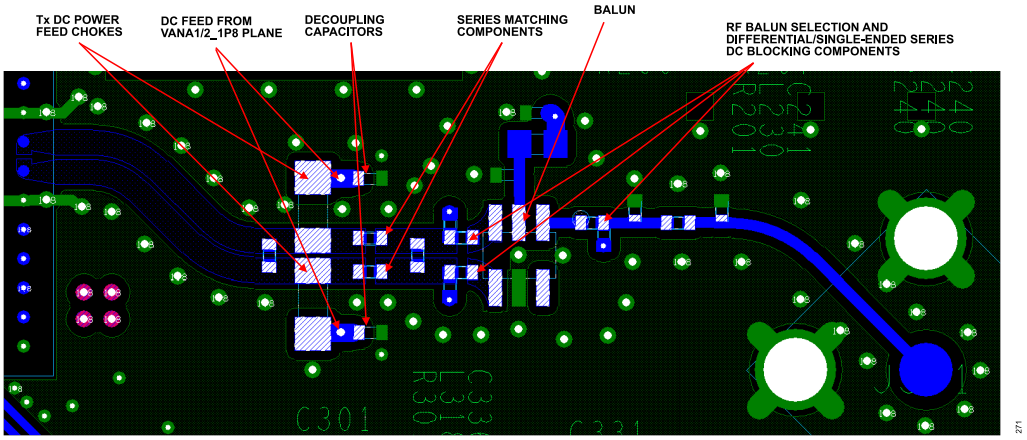


Figure 291. Transmitter Power Supply Using RF Chokes

SSI Data Port Trace Routing Recommendations

The data port interface transfers I/Q data between the BBIC/FPGA and ADRV9001 transmitter and receiver datapaths. There are two possible modes of operation for the SSI data port:

- CMOS-SSI mode (single-ended), with clock rate for data transfer up to 80 MHz.
- LVDS-SSI mode (differential), with clock rate for data transfer up to 500 MHz DDR (1000 MHz data rate).

Follow the correct layout practice while routing the SSI signals.

If selecting the CMOS-SSI mode, single-ended signal lines between the ADRV9001 and BBIC/FPGA must be as short as possible. Also reduce the trace capacitance to minimize the current needed by the ADRV9001 to drive the line. Refer to the ADRV9001 data sheet for details on pin drive capabilities.

If selecting the LVDS-SSI mode, route all the LVDS signals as 100 Ω differential pairs.

- When routing the PCB layout for the LVDS-SSI data lines, the designer must route the signals using stripline or microstrip traces. There are positives and negatives for each.
- Stripline has less loss and emits less EMI than microstrip lines, but stripline traces require the use of vias that can add complexity to the task of controlling the impedance by adding line inductance.

Microstrip is easier to implement if the component placement and density allow for routing on the top layer, simplifying the task of controlling the impedance.

If using the top layer of the PCB is problematic or the advantages of stripline are desirable, then follow these recommendations:

- Minimize the number of vias.
- Use blind vias wherever possible to eliminate via stub effects, and use micro vias to minimize via inductance.
- If using standard vias, use maximum via length to minimize the stub size. For example, on an eight-layer board, use Layer 7 for the stripline pair.
- For each via pair, place a pair of ground vias close to them to minimize the impedance discontinuity.

In the LVDS-SSI mode:

- For transmitter data port inputs, implement 100 Ω termination inside the ADRV9001.
- For receiver data port outputs, implement 100 Ω termination at the receiver end.

Evaluation Board FMC Connector Signals Mapping

The ADRV9001 evaluation board uses the FPGA mezzanine card (FMC) standard connector as an interface to carrier boards. Table 122 outlines signal mapping used on the FMC connector implemented on the ADRV9001 evaluation board. The second column refers to the FMC standard pinout names. For more information, refer to the ADRV9001 EVB schematic.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

Table 122. FMC Pinout Mapping Used by ADRV9001 Evaluation Board

Schematic Net Name	FMC Connector Mappings
FPGA_REF_CLK+	G02-FMC_CLK1_M2C_P
FPGA_REF_CLK-	G03-FMC_CLK1_M2C_N
DEV_CLK_OUT	H04-FMC_CLK0_M2C_P
SM_FAN_TACH	H05-FMC_CLK0_M2C_N
RX1_DCLK_OUT+	G06-FMC_LA00_CC_P
RX1_DCLK_OUT-	G07-FMC_LA00_CC_N
RX1_STROBE_OUT+	H07-FMC_LA02_P
RX1_STROBE_OUT-	H08-FMC_LA02_N
RX1_IDATA_OUT+	G09-FMC_LA03_P
RX1_IDATA_OUT-	G10-FMC_LA03_N
RX1_QDATA_OUT+	H10-FMC_LA04_P
RX1_QDATA_OUT-	H11-FMC_LA04_N
DGPIO_13_TX1_DCLK_OUT+	D08-FMC_LA01_CC_P
DGPIO_12_TX1_DCLK_OUT-	D09-FMC_LA01_CC_N
TX1_DCLK_IN+	H13-FMC_LA07_P
TX1_DCLK_IN-	H14-FMC_LA07_N
TX1_STROBE_IN+	C10-FMC_LA06_P
TX1_STROBE_IN-	C11-FMC_LA06_N
TX1_IDATA_IN+	G12-FMC_LA08_P
TX1_IDATA_IN-	G13-FMC_LA08_N
TX1_QDATA_IN+	D11-FMC_LA05_P
TX1_QDATA_IN-	D12-FMC_LA05_N
RX2_DCLK_OUT+	D20-FMC_LA17_CC_P
RX2_DCLK_OUT-	D21-FMC_LA17_CC_N
RX2_STROBE_OUT+	H25-FMC_LA21_P
RX2_STROBE_OUT-	H26-FMC_LA21_N
RX2_IDATA_OUT+	G21-FMC_LA20_P
RX2_IDATA_OUT-	G22-FMC_LA20_N
RX2_QDATA_OUT+	H22-FMC_LA19_P
RX2_QDATA_OUT-	H23-FMC_LA19_N
DGPIO_15_TX2_DCLK_OUT+	C22-FMC_LA18_CC_P
DGPIO_14_TX2_DCLK_OUT-	C23-FMC_LA18_CC_N
TX2_DCLK_IN+	G24-FMC_LA22_P
TX2_DCLK_IN-	G25-FMC_LA22_N
TX2_STROBE_IN+	H28-FMC_LA24_P
TX2_STROBE_IN-	H29-FMC_LA24_N
TX2_IDATA_IN+	D23-FMC_LA23_P
TX2_IDATA_IN-	D24-FMC_LA23_N
TX2_QDATA_IN+	G27-FMC_LA25_P
TX2_QDATA_IN-	G28-FMC_LA25_N
RX1_ENABLE	C14-FMC_LA10_P
RX2_ENABLE	D27-FMC_LA26_N
TX1_ENABLE	D14-FMC_LA09_P
TX2_ENABLE	G30-FMC_LA29_P
SPI_EN	H19-FMC_LA15_P
SPI_CLK	G15-FMC_LA12_P
SPI_DIO	G31-FMC_LA29_N
SPI_DO	G16-FMC_LA12_N
MODE	D17-FMC_LA13_P

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

Table 122. FMC Pinout Mapping Used by ADRV9001 Evaluation Board (Continued)

Schematic Net Name	FMC Connector Mappings
RESET_TRX	D18-FMC_LA13_N
DEV_MCS_FPGA_IN+	C18-FMC_LA14_P
DEV_MCS_FPGA_IN-	C19-FMC_LA14_N
DGPIO_0	G18-FMC_LA16_P
DGPIO_1	G19-FMC_LA16_N
DGPIO_2	H20-FMC_LA15_N
DGPIO_3	H17-FMC_LA11_N
DGPIO_4	D15-FMC_LA09_N
DGPIO_5	C15-FMC_LA10_N
DGPIO_6	C26-FMC_LA27_P
DGPIO_7	D26-FMC_LA26_P
DGPIO_8	H31-FMC_LA28_P
DGPIO_9	H32-FMC_LA28_N
DGPIO_10	H16-FMC_LA11_P
DGPIO_11	C27-FMC_LA27_N
GP_INT	H34-FMC_LA30_P
VADJ_TEST_1 (VADJ_ERR)	G33-FMC_LA31_P
VADJ_TEST_2 (PLATFORM_STATUS)	G34-FMC_LA31_N
FPGA_MCS_IN+	H37-FMC_LA32_P
FPGA_MCS_IN-	H38-FMC_LA32_N

ISOLATION TECHNIQUES USED ON THE ADRV9001 EVALUATION CARD

Given the density of sensitive and critical signals, there are significant isolation challenges when designing a PCB for the ADRV9001. Follow the isolation requirements listed in Table 123 to accurately evaluate the ADRV9001 device performance. Analytically determining aggressor-to-victim isolation in a system is very complex and involves considering vector combinations of aggressor signals and coupling mechanisms.

Isolation Goals

Table 123 lists the isolation targets for each RF channel-to-channel combination type. To meet these goals with significant margin, isolation structures are designed into the ADRV9001 evaluation board.

Table 123. Port to Port Isolation Goals

	30 MHz to 1 GHz	1 GHz to 6 GHz
Tx1 to Tx2	75 dB	70 dB
Tx1 to Rx1A/Rx1B	75 dB	70 dB
Tx1 to Rx2A/Rx2B	75 dB	70 dB
Rx1A/Rx1B to Rx2A/Rx2B	70 dB	65 dB
Rx1A to Rx1B	70 dB	65 dB

Isolation Between RF I/O Ports

The following are the primary coupling mechanisms between the RF I/O paths on the evaluation board:

- Magnetic field coupling
- Surface propagation
- Cross-domain coupling through ground

There are several strategies to reduce the impact of these coupling mechanisms on the ADRV9001 customer evaluation. Open large slots in the ground plane between the RF I/O paths. These discontinuities prevent surface propagation. These structures consist of a combination of slots and square apertures. Both structures are present on every copper layer of the PCB stack. The advantage of using square apertures is

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

that signals can be routed between the openings without disturbing the isolation benefits provided by the array of apertures. A careful designer notices various bends in the routing of differential paths. Develop and tune these routes through iterative electromagnetic simulation to minimize magnetic field coupling between differential paths. Figure 292 shows these techniques.

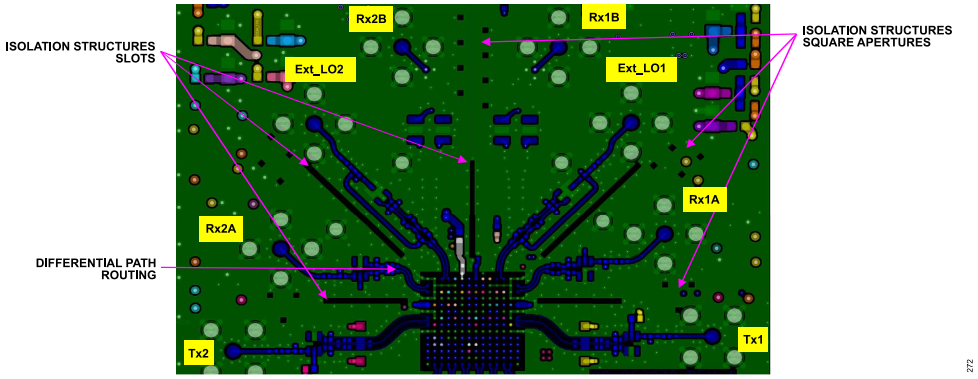


Figure 292. RF I/O Isolation Structures

When utilizing the proposed isolating structures, it is important to place ground vias around the slots and apertures. Figure 293 shows the methodology used on the ADRV9001 evaluation card. When using slots, place the ground vias at each end of the slots and along each side. When using square apertures, place at least one single ground via next to each square. These vias are through-hole vias connecting the top to the bottom layer and all layers in between. The function of these vias is to steer return current to the ground planes near the apertures.

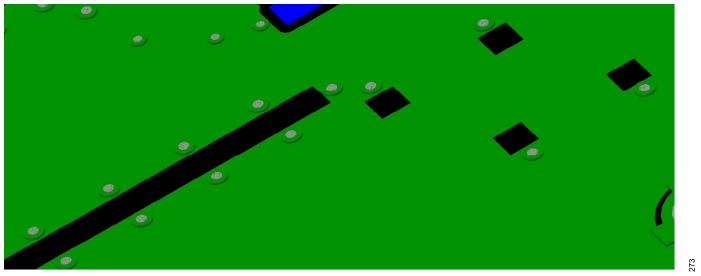


Figure 293. Current Steering Vias Placed Near Isolation Slots and Square Apertures

For accurate slot spacing and square apertures layout, use simulation software when designing a PCB for an ADRV9001 based transceiver. As a general rule, the spacing between square apertures should be no more than one-tenth of the shortest wavelength. Calculate the wavelength calculated using Equation 1.

$$Wave\ length\ [m] = \frac{300}{Frequency\ [MHz] \times \sqrt{\varepsilon_r}} \quad (1)$$

where:

ε_r is the dielectric constant of the isolator material.

For ISOLA I-speed material, ε_r = 3.56, and for FR4-408 HR material, ε_r = 3.77.

Example: Given a maximum RF signal frequency of 6 GHz, for ISOLA I-speed material, using microstrip structures and ε_r = 3.56, the minimum wavelength is approximately 26.4 mm. To fulfill the one-tenth of a wavelength rule, the square aperture spacing should be at a distance of 2.64 mm or closer.

Provide additional shielding by 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. Figure 294 shows how this is done for Rx1. Do the same for each set of sensitive RF I/O ports. Use ground vias along single-ended RF I/O traces. Optimal via spacing is one-tenth of a wavelength, but that spacing can vary somewhat due to practical layout considerations.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

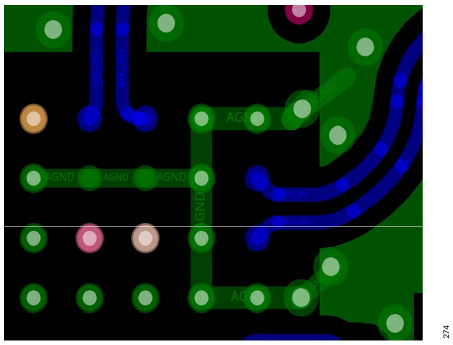


Figure 294. Shielding of Rx Launches

Space and align the RF I/O baluns to reduce magnetic coupling from the structures in the balun package. Take care to reduce the crosstalk over shared grounds between baluns. Another precaution is to place and orient the SMA connectors to minimize the connector-to-connector coupling between ports.

POWER-SUPPLY RECOMMENDATIONS

This section provides an overview of the ADRV9001 power-supply solution. The power supply solution for the ADRV9001 changes based on the desired mode of operation (FDD, TDD, DPD, transmitter tracking, number of active receiver inputs, number of active Tx outputs, internal LOs vs. external LOs, and 1.0 V power domain). Therefore, it is important to understand the available configurations and optimize the PCB layout based on the selected mode. This section uses power-supply solutions implemented on the EVB as a reference.

POWER MANAGEMENT CONSIDERATIONS

The ADRV9001 family of devices requires the following four or five different power-supply domains.

1. 1.0 V digital: Connect this supply to the device through the two VDIG_1P0 pins. This is the supply that feeds all digital blocks. Take care to properly isolate this supply from all analog signals on the PCB to avoid noise corruption. 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. In some modes of operation, this supply can draw high current. It is critical that the input traces for these two inputs be balanced (same impedance for inputs) and as thick as possible to minimize the I × R drop.
2. 1.0 V analog: These supplies are collectively referred to in the user guide as the VDDA_1P0 supply. This power domain is optional and is suggested for use only to achieve minimum power consumption. It requires a low noise 1.0 V power domain available in the end system. In modes of operation where VDDA_1P0 is not used, this 1.0 V power domain is created internally inside the ADRV9001 using internal LDOs. This power domain supplies voltage to noise sensitive blocks of the ADRV9001. To provide external 1.0 V, ensure very low noise level on this power domain. This supply input has a tolerance of ±2.5%.
3. 1.3 V analog: These supplies connect to all functional blocks in the device through a number of different input pins. They are collectively referred to in the user guide as the VDDA_1P3 supply. Each input is treated as a noise-susceptible input, which means proper decoupling and isolation techniques are followed to avoid crosstalk between channels. The tolerance on these supply inputs is ±2.5%. If VDDA_1P0 is used, some of VDDA_1P3 supply pins change their intended voltage input level from 1.3 V to 1.0 V. This chapter provides a detailed overview of these modifications.
4. 1.8 V analog: These supplies primarily supply the transmitter outputs. They also supply current for multiple transmitter, receiver, converter, and auxiliary converter blocks. They are collectively referred to in the user guide as the VDDA_1P8 supply. This supply has a tolerance of ±5%.
5. 1.8 V digital: This is an interface supply. The VDIGIO_1P8 supply is a separate power domain shared with the baseband processor (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 and SSI), DGPIO, and digital control inputs.

IMPORTANT

During operation, supply currents can vary significantly, especially if operating in the TDD mode. The supply must have adequate capacity to provide the necessary current (as indicated in the user guide) to maintain the performance criteria over all the process and temperature variations. Analog Devices recommends adding at least 15% margin to all supply maximums to ensure proper operation under all conditions.

Power-Supply Sequence

The ADRV9001 requires a specific power-up sequence to avoid undesired power-up currents. The optimal power-on sequence requires VDD_1P0 to power up first. The VDDA_1P3, VDDA_1P8, and VDD_1P8 supplies must then power up after the VDD_1P0 supply. If using VDDA_1P0, power it up after enabling VDDA_1P3 and VDDA_1P8. Toggle the RESET signal after the power is stabilized before configuration.

The power-down sequence recommendation is similar to power-up. Disable all supplies in any order (or all together) before disabling VDD_1P0. If such a sequence is not possible, then disable all the sources of the supplies simultaneously to ensure there are no back-feeding circuits to power down.

Power-Supply Domain Connections

Careful low-noise power management design is a key aspect to ensure good performance. Table 124 lists the pin number, pin type and name, expected voltage on that pin, and a brief description of routing technique together with the block it powers on the chip. Power supply to the ADRV9001 is delivered after the star configuration, where a separate trace from a common power plane is used to power each power-supply pin. Refer to the Printed Circuit Board Layout Recommendations section and an evaluation board CAD layout file for details.

POWER-SUPPLY RECOMMENDATIONS

Table 124. Power-Supply Pins and Functions

Pin No.	Type	Pin Name	Voltage [V]	Description
A1, A2, A13, A14, B2 to B5, B10 to B13, C2, C5, C10, C13, D1 to D6, D9 to D14, E6, E9, F1 to F3, F6 to F9, F12 to F14, G2, G13, J1 to J14	ANALOG	VSSA	0	Analog supply voltage (V_{SS}).
A5	ANALOG	VRFVCO2_1P3	1.3	1.3 V internal LDO input supply for RF LO2 VCO and LO generation circuitry. This pin is sensitive to supply noise.
A6	ANALOG	VRFLO2_1P0	1.0	1.0 V internal supply node for RF LO2 LO generation circuitry. Connect this pin together with B6 and bypass with a 4.7 µF capacitor, when internal LDO operated from A5 input is in use. Provide 1.0 V supply to this pin when internal LDO operated from A5 is not in use.
A9	ANALOG	VRFLO1_1P0	1.0	1.0 V internal supply node for RF LO1 LO generation circuitry. Connect this pin together with B9 and bypass with a 4.7 µF capacitor, when internal LDO operated from A10 input is in use. Provide 1.0 V supply to this pin when internal LDO operated from A10 is not in use.
A10	ANALOG	VRFVCO1_1P3	1.3	1.3 V internal LDO input supply for RF LO1 VCO and LO generation circuitry. This pin is sensitive to supply noise.
B6	ANALOG	VRFVCO2_1P0	1.0	1.0 V internal supply node for RF LO2 VCO circuitry. Connect this pin together with A6 and bypass with a 4.7 µF capacitor, when internal LDO operated from A5 input is in use.
B9	ANALOG	VRFVCO1_1P0	1.0	1.0 V internal supply node for RF LO1 VCO circuitry. Connect this pin together with A9 and bypass with a 4.7 µF capacitor, when internal LDO operated from A10 input is in use.
C6	ANALOG	VANA2_1P0	1.0	1.0 V internal supply node for Tx2/Rx2 baseband circuits, transimpedance amplifier (TIA), Tx transconductance (GM), baseband filters, and auxiliary DACs/ADCs. For normal operation, leave this pin unconnected.
C7	ANALOG	VANA2_1P3	1.3	1.3 V internal LDO input supply for Tx2/Rx2 baseband circuits, transimpedance amplifier (TIA), Tx transconductance (GM), baseband filters, and auxiliary DACs/ADCs. This pin is sensitive to supply noise.
C8	ANALOG	VANA1_1P3	1.3	1.3 V internal LDO input supply for Tx1/Rx1 baseband circuits, transimpedance amplifier (TIA), Tx transconductance (GM), and baseband filters. This pin is sensitive to supply noise.
C9	ANALOG	VANA1_1P0	1.0	1.0 V internal supply node for Tx1/Rx1 baseband circuits, transimpedance amplifier (TIA), Tx transconductance (GM), and baseband filters. For normal operation, leave this pin unconnected.
E1	ANALOG	VRX2LO_1P0	1.0	1.0 V internal supply node for Rx2 LO buffers and mixers. This pin is sensitive to supply noise. Bypass this pin with a 4.7 µF capacitor.
E2	ANALOG	VRX2LO_1P3	1.3/1.0	1.3 V internal LDO input supply for Rx2 LO buffers and mixers. Provide 1.0 V supply to this pin when internal LDO is not used. This pin is sensitive to supply noise.
E4	ANALOG	VRFSYN2_1P3	1.3	1.3 V supply for RF LO2 synthesizer. This pin is sensitive to supply noise.
E5	ANALOG	VCLKSYN_1P3	1.3	1.3 V supply for clock synthesizer. This pin is sensitive to supply noise.
E10	ANALOG	VAUXSYN_1P3	1.3	1.3 V supply for auxiliary synthesizer. This pin is sensitive to supply noise.
E11	ANALOG	VRFSYN1_1P3	1.3	1.3 V supply for RF LO1 synthesizer. This pin is sensitive to supply noise.
E13	ANALOG	VRX1LO_1P3	1.3/1.0	1.3 V internal LDO input supply for Rx1 LO buffers and mixers. Provide 1.0 V supply to this pin when internal LDO is not used. This pin is sensitive to supply noise
E14	ANALOG	VRX1LO_1P0	1.0	1.0 V internal supply node for Rx1 LO buffers and mixers. This pin is sensitive to supply noise. Bypass this pin with a 4.7 µF capacitor.
G3	ANALOG	VTX2LO_1P3	1.3/1.0	1.3 V supply for Tx2 LO buffers, upconverter, and LO delay. Provide 1.0 V supply to this pin when internal LDO is not used. This pin is sensitive to supply noise.
G5	ANALOG	VCLKVCO_1P3	1.3	1.3 V internal LDO input supply for clock LO VCO and LO generation circuitry. This pin is sensitive to supply noise.
G7	ANALOG	VCONV_1P8	1.8	1.8 V supply for Tx1/Tx2 DAC and Rx1/Rx2 ADC
G8	ANALOG	VAGPIO_1P8	1.8	1.8 V supply for AuxDACs, AuxADCx, and AGPIO signals.

POWER-SUPPLY RECOMMENDATIONS

Table 124. Power-Supply Pins and Functions (Continued)

Pin No.	Type	Pin Name	Voltage [V]	Description
G10	ANALOG	VAUXVCO_1P3	1.3	1.3 V internal LDO input supply for auxiliary LO VCO and LO generation circuitry. This pin is sensitive to supply noise.
G12	ANALOG	VTX1LO_1P3	1.3/1.0	1.3 V internal LDO input supply for Tx1 LO buffers, upconverter, and LO delay. Provide 1.0 V supply to this pin when internal LDO is not used. This pin is sensitive to supply noise.
H2	ANALOG	VANA2_1P8	1.8	1.8 V supply for Rx2 Mixer, Rx2 transimpedance amplifier (TIA), Tx2 Low-Pass Filter (LPF) and Internal References.
H3	ANALOG	VTX2LO_1P0	1.0	1.0 V internal supply node for Tx2 LO buffers, upconverter, and LO delay. For normal operation, leave this pin unconnected.
H5	ANALOG	VCLKVCO_1P0	1.0	1.0 V internal supply node for clock LO VCO and LO generation circuitry. Bypass this pin with a 4.7 µF capacitor.
H7	ANALOG	VCONV_1P0	1.0	1.0 V internal supply node for Rx ADCs and Tx DACs. Bypass this pin with a 4.7 µF capacitor.
H8	ANALOG	VCONV_1P3	1.3/1.0	1.3 V internal LDO input supply for Rx ADCs and Tx DACs. Provide 1.0 V supply to this pin when internal LDO is not used. This pin is sensitive to supply noise.
H10	ANALOG	VAUXVCO_1P0	1.0	1.0 V internal supply node for auxiliary LO VCO and LO generation circuitry. Bypass this pin with a 4.7 µF capacitor.
H12	ANALOG	VTX1LO_1P0	1.0	1.0 V internal supply node for Tx1 LO buffers, upconverter, and LO delay. For normal operation, leave this pin unconnected.
H13	ANALOG	VANA1_1P8	1.8	1.8 V supply for Rx1 mixer, Rx1 transimpedance amplifier (TIA), Tx1 low-pass filter (LPF), Xtal oscillator, DEV_CLK circuitry, and internal references.
K4	ANALOG	VSSA/TESTCK+	0	Connect to VSSA for normal operation.
K5	ANALOG	VSSA/TESTCK-	0	Connect to VSSA for normal operation.
L7, L8	DIGITAL	VDIG_1P0	1.0	1.0 V digital core. Connect Pin L7 and Pin L8 together. Use a wide trace to connect to a separate power-supply domain. Provide reservoir capacitance close to the chip.
M7	DIGITAL	VDIGIO_1P8	1.8	1.8 V supply input for data port interface (CMOS-SSI/LVDS-SSI), SPI signals, control input/output signals, and DGPIO interface.
M8	DIGITAL	VDIG_0P9	0.9	0.9 V internal supply node for digital circuitry. Bypass this pin with a 4.7 µF capacitor.
N7, N8, P1, P14	DIGITAL	VSSD	0	Digital supply voltage (V_{SS})

Power-Supply Architecture

Figure 295 outlines the power-supply configuration used on the ADRV9001 evaluation board. This configuration follows the recommendations outlined in Table 124. This diagram includes the use of C/FB/C/FB cascaded filters and FBs for additional RF isolation. The use of FBs and 0 Ω resistors in the EVB power-supply solution accomplishes the following three goals.

- Serves as placeholders for FBs or other filter devices if RF noise problems are encountered in the application and for additional isolation. For more details to select the FB, see the RF and Clock Synthesizer Supplies section.
- Ensures to follow the power routing recommendations outlined in the Printed Circuit Board Layout Recommendations section. The FBs and resistor placeholders in series force the use of separate traces to deliver different power domains to the ADRV9001 device.
- Provides a place in the circuit to monitor and measure the current for debugging. For this case, replace the 0 Ω components or FB with very low-impedance shunt resistors and measure the voltage to determine current to the specified input ball.

For more details on exact power supply implementation, refer to the ADRV9001 customer evaluation board schematic supplied with an evaluation kit.

The ADRV9001 evaluation board also provides an on-board current and voltage sensor (ADM1293), which monitors and reports back the power consumed by the transceiver in the selected mode of operation. The intent is to provide live feedback from an evaluation system on the real-time power consumption. Current readbacks from these sensors are accurate within 2.5% tolerance. If using these readback numbers to estimate the overall power for the power supply design, add an extra power margin to accommodate dynamic conditions. The first paragraph in this section provides more suggestions.

POWER-SUPPLY RECOMMENDATIONS

Evaluation Board Power-Supply Overview

Figure 295 outlines the power-supply configuration used on the ADRV9001 evaluation board (EVB). This supply architecture follows a conservative approach in the power-supply design. A switch-mode regulator (ADP5056) is used to achieve power efficiency while generating domains that supply the ADRV9001. Remote sensing configuration is used to take into account the voltage drop in filters and ensure power domain accuracy at the ADRV9001 input pins.

The ADP5056 contains three switch-mode step-down regulators. Each regulator produces a different power domain that supplies power to the ADRV9001. They operate as follows:

- Channel 1 generates 1.3 V, which supplies voltage for the ADRV9001 1.3 V power domains. Sensing is done after C/FB/C/FB cascaded filters and current sensing shunt resistor, ensuring that the 1.3 V analog power domain voltage stays within the user guide specifications.
  - The ADRV9001 EVB also supports an optional 1.0 V power domain. Generate this domain using an on-board LDO (ADP1762). This LDO uses its own remote sensing scheme to ensure that the 1.0 V analog power domain voltage stays within the user guide specifications.
- Channel 2 generates 1.0 V, which supplies voltage for the ADRV9001 digital domain. Sensing is done after C/FB/C/FB cascaded filters and current sensing shunt resistor to ensure that the 1.0 V digital power domain voltage stays within data sheet specifications.
- Channel 3 generates 1.8 V, which supplies voltage for the ADRV9001 digital and analog power domains. The C/FB/C/FB cascaded filters are used to isolate the analog power domain from the digital power domain. As more than one power domain is produced from a single source, remote sensing is done before C/FB/C/FB cascaded filters.

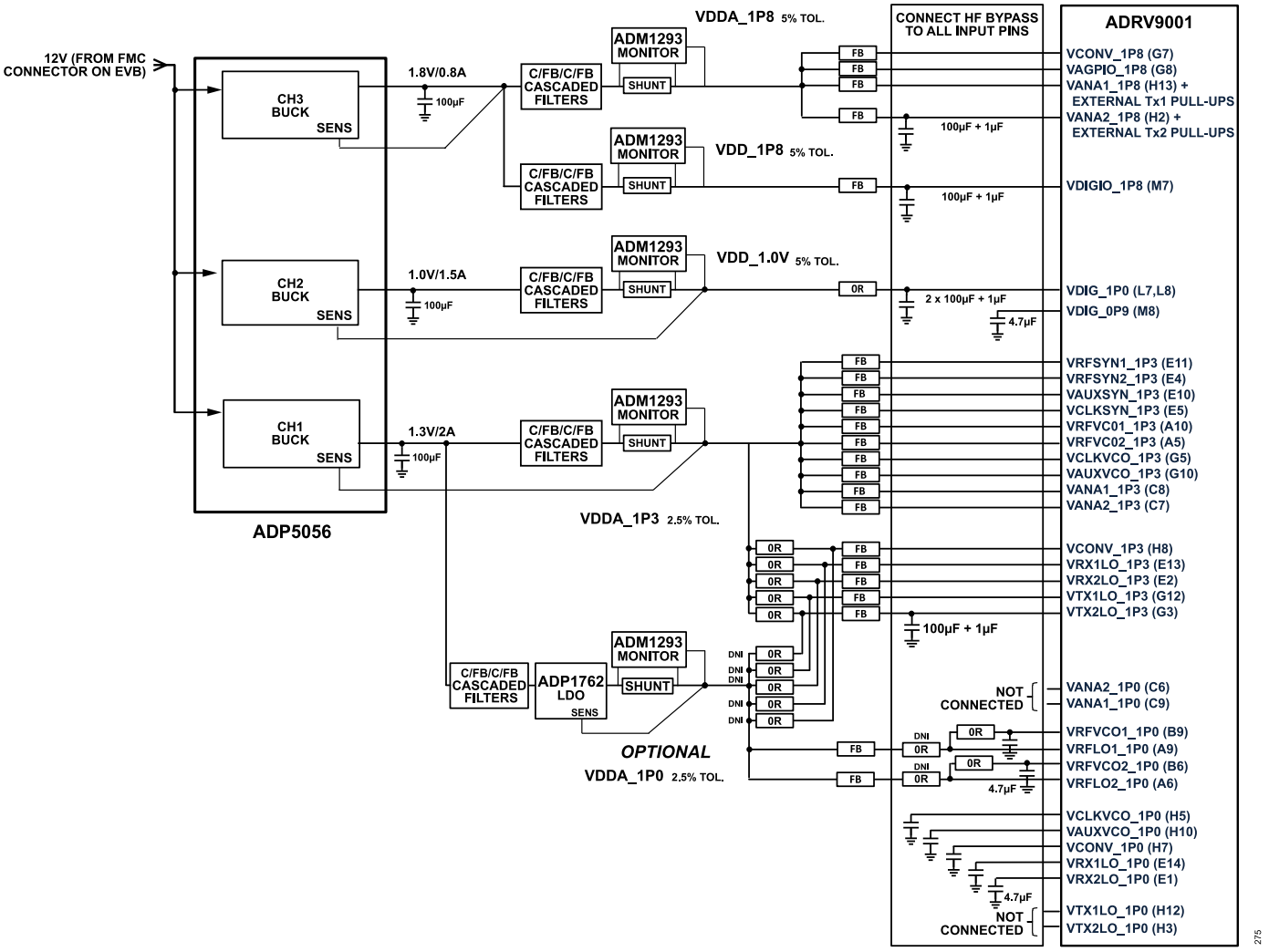


Figure 295. Power-Supply Connection Diagram

POWER-SUPPLY RECOMMENDATIONS

Power Domain Filtering

C/FB/C/FB cascaded filters followed by high-current FBs further isolate power signals to the ADRV9001 from each other. Figure 296 outlines a filtered approach implemented on an EVB.

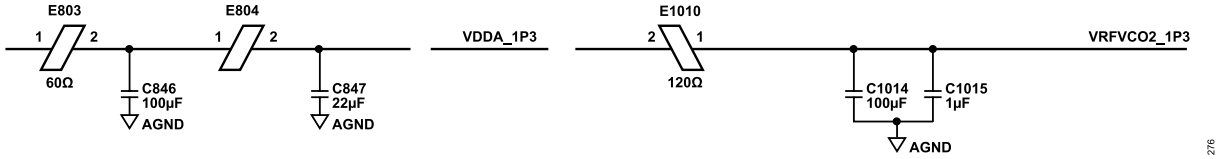


Figure 296. An Example of Power Domain Filtering Implemented on ADRV9001 EVB (1.3V -> VRFVCO2_1P3)

Figure 297 shows a simulation schematic with power-supply filter components used on the EVB power rails in the ADS environment. It attempts to simulate the frequency response of the filter outlined in Figure 296.

- Locate the E1, C1, E2, and C2 elements right at each rail input pin.
- Locate the E3 and C4 elements right at the trace that feeds the particular pin.
- Place the C4 element as the RF capacitor right at each ADRV9001 pin using the rail. For more information about placing these capacitors, see the PCB layout section and EVB PCB file itself.
- The combined DC resistance of the three ferrites is 0.106 Ω.

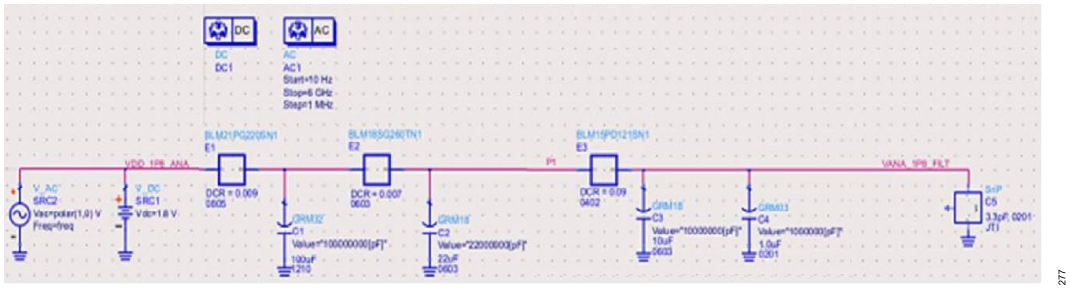


Figure 297. ADS Model of C/FB/C/FB Cascaded Filter Utilized on ADRV9001 EVBs

Finally, Figure 298 shows the results of the simulation described in Figure 297. The overall frequency response is better than a typical bank of capacitors. The region >1 GHz outlines the performance of the implemented cascaded C/FB/C/FB filtering approach.

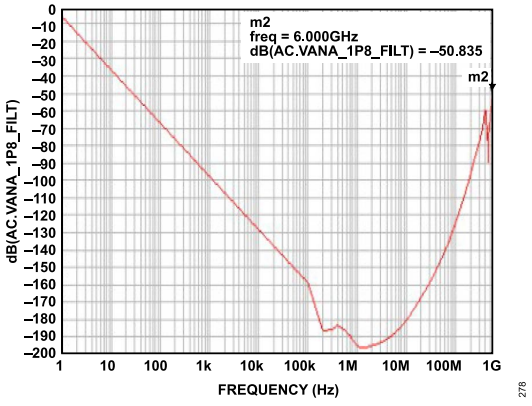


Figure 298. Response of C/FB/C/FB Cascaded Filter Utilized on ADRV9001 EVBs

Take care when introducing multiple inductors and FBs in series, as it creates issues with potential I x R voltage drops on them that can violate the recommended power domain voltage accuracy. The power-supply solution on the ADRV9001 EVB uses the sense line to monitor the voltage output after the FB (where possible). This approach ensures the voltage drop resulting from the cascaded filters and FB resistance is taken into account, and the voltage level delivered to the ADRV9001 is in line with the expected accuracy. In scenarios where single-power

domain powers multiple power pins, current tends to be distributed over multiple pins, and this helps to minimize I x R voltage drops on the FB components.

RF and Clock Synthesizer Supplies

The noise performance of the power domain used to power the RF blocks directly affects the overall transceiver phase noise. Power the power domains using separate traces with extra isolation using a low DCR FB such as the Murata BLM18KG121TN1D or similar device.

The power-supply noise rejection on the synthesizer power input pins is very low. This means that any noise ripple on these pins affects the synthesizer performance. The RF synthesizer requires more critical supply decoupling because any noise or variation in voltage during operation is directly imposed on the RF channel. See Figure 295 for an example of how the power-supply connections are made on the ADRV9001 evaluation card.

The synthesizers are more susceptible to low-frequency noise than other supplies because they have programmable loop filters. The loop filter bandwidth directly affects the supply noise rejection on the synthesizers. For example, if the loop filter bandwidth is 50 kHz, then any noise on the supply below 50 kHz is not filtered. The roll-off of the loop filter provides the noise rejection above 50 kHz.

For each power domain, an FB with high isolation at the frequency of operation is recommended to help isolate the pin from the supply source for best performance. This is especially important when operating in TDD mode. Such high isolation FBs tend to also have high DC resistance. This trade-off is acceptable for the synthesizer power inputs because their low-current draws result in relatively small voltage drops well within the supply tolerance range.

For domains that consume higher amount of current, ensure that voltage drops on in-series FBs do not exceed power domain voltage tolerances. PANASONIC EXC-ML20A390U 39OHM or MURATA, BLM15AX300SN1D or BLM18KG121TN1D with low DCR are good candidates for such FBs.

For power domains, which do not consume higher amount of current and simultaneously require higher level of isolation, use ferrite beads with higher DCR. Taiyo Yuden, BK1005LL470-T is an example of the recommended high DCR FB.

POWER-SUPPLY CONFIGURATIONS

From the power-supply implementation point of view, the ADRV9001 can work in multiple configurations. This section outlines them in detail. It depends on the final application of the ADRV9001 in the end system that provides the freedom to implement different ways to power the IC. The final solution depends on the following:

1. If the external 1.0 V power domain is used or not.
2. Type of application: FDD or TDD
    a. For FDD, if DPD and/or transmitter tracking calibrations are used or not.
3. Number of active receiver inputs and transmitter outputs.
    a. Note that in a scenario where Rx2/Tx2 are used and Rx1/Tx1 are not used (not recommended scenario from an optimal power savings point of view), there must be power supply to VANA1_1P3 (C8).
4. LO scheme.
    a. Internal LOs with internal PLL + VCO + LO_GEN powered by internal LDO.
    b. External LOs with internal LO_GEN powered by:
        1. Internal LDO
        2. External LDO
            - Note that even in scenarios when external LOs are used, power up the VRFVCO1_1P0 (B9), VRFVC01_1P3 (A10), and VRFSYN1_1P3 (E11). The RF PLL1 is used to generate test signals during the initialization calibration stage. Therefore, provide power supply to these blocks. After performing initial calibrations, power down these blocks internally.

This section outlines how decisions based on these descriptions impact the final power-supply interconnectivity.

Figure 299 outlines the recommended power supply interconnectivity in scenarios using the external 1.0 V analog power domain. In such modes, reconnect the number of power-supply input pins assigned to the 1.3 V analog power domain physically on the PCB to the new 1.0 V analog power domain. In cases where an external RF LO is used, save more power by physically disconnecting/grounding LO supplies powering up the internal LO generation blocks.

POWER-SUPPLY RECOMMENDATIONS

Figure 300 and Figure 301 provide recommendations on how to interconnect power supply when not all of the available RF IOs (transmitters and receivers) are used in the end application. Note that to perform Tx1 tracking calibration or DPD on Tx1, the Rx1 datapath must be available to observe the Tx output. The same relationship exists between the Tx2 and Rx2. It is not possible to perform Tx1 tracking calibration or DPD operation on the Tx1 output using Rx2 datapath (and vice versa, Tx2 using Rx1).

Figure 301 provides suggestions to configure 1T1R. In TDD applications, to achieve the lowest possible current consumption in deep sleep state, use the Rx1 and Tx1 datapaths and disable the Rx2 and Tx2 datapaths.

Once any of the configurations outlined in Figure 299, Figure 300, or Figure 301 is implemented, initialize the ADRV9001 device with the correct initialization settings described in the data structure.

POWER-SUPPLY RECOMMENDATIONS

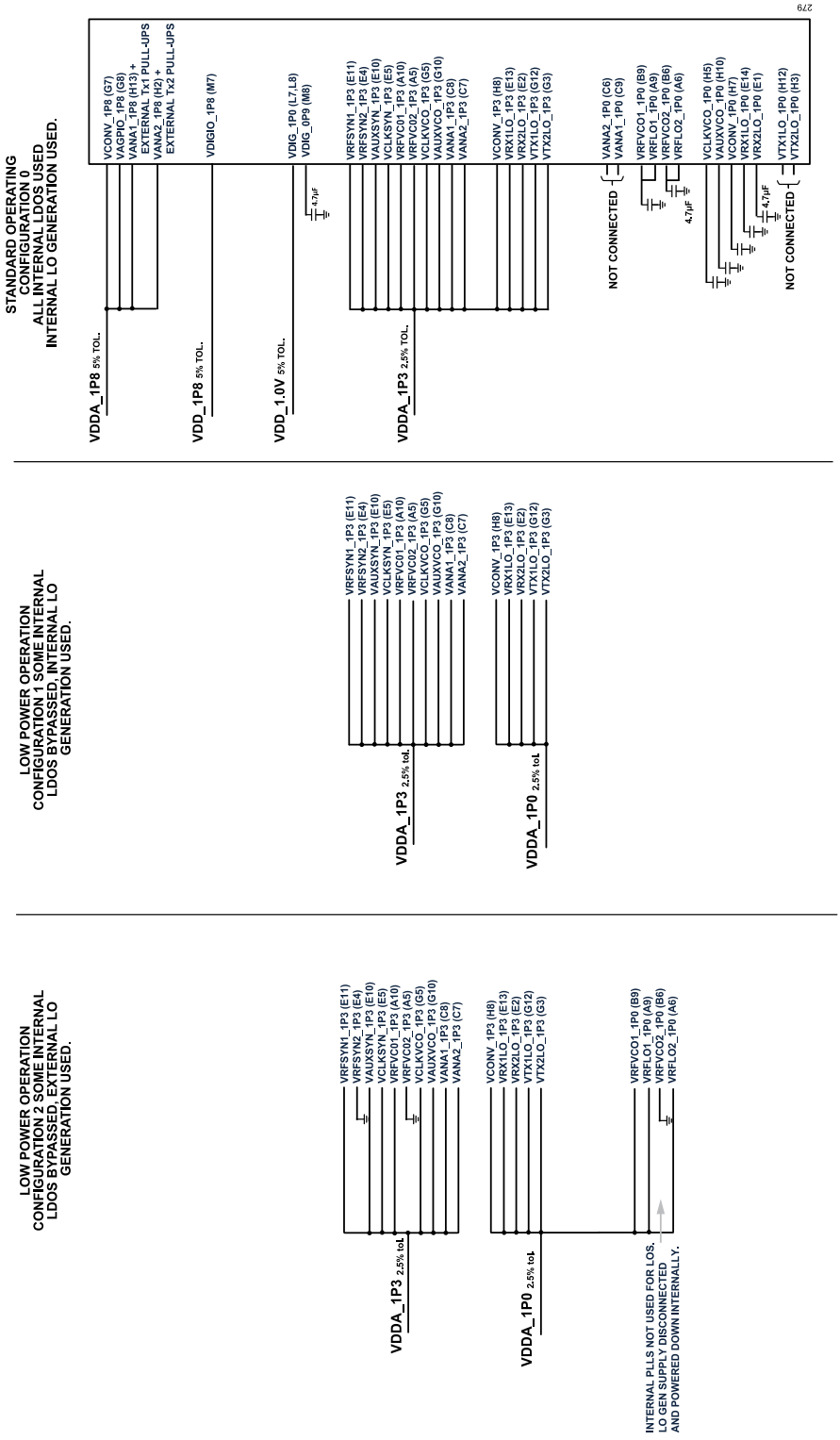


Figure 299. Available Modes for External 1.0V Power Domain and LO Power-Supply Configuration

POWER-SUPPLY RECOMMENDATIONS

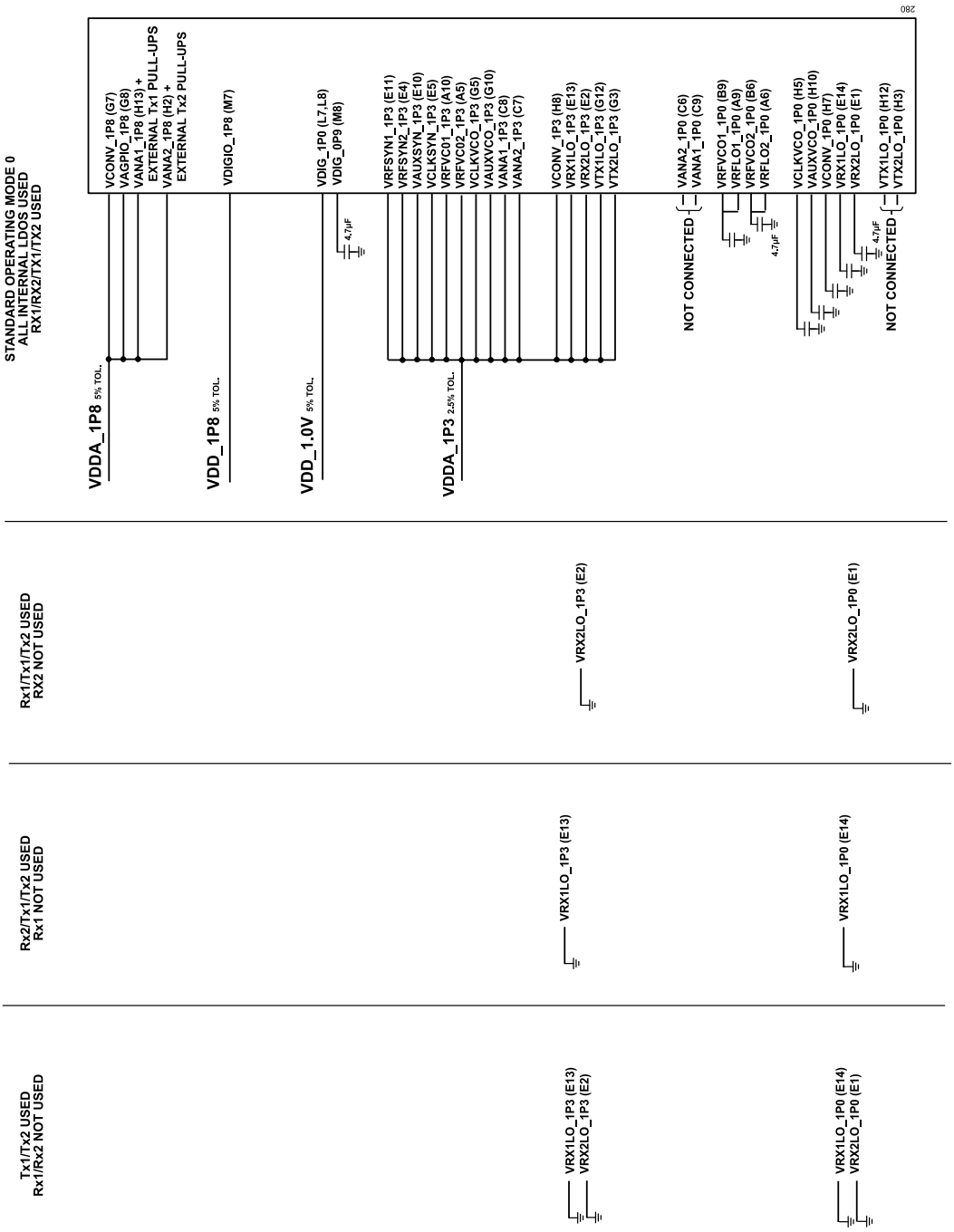


Figure 300. Power-Supply Modes for Different Number of Active Tx RF IOs

POWER-SUPPLY RECOMMENDATIONS

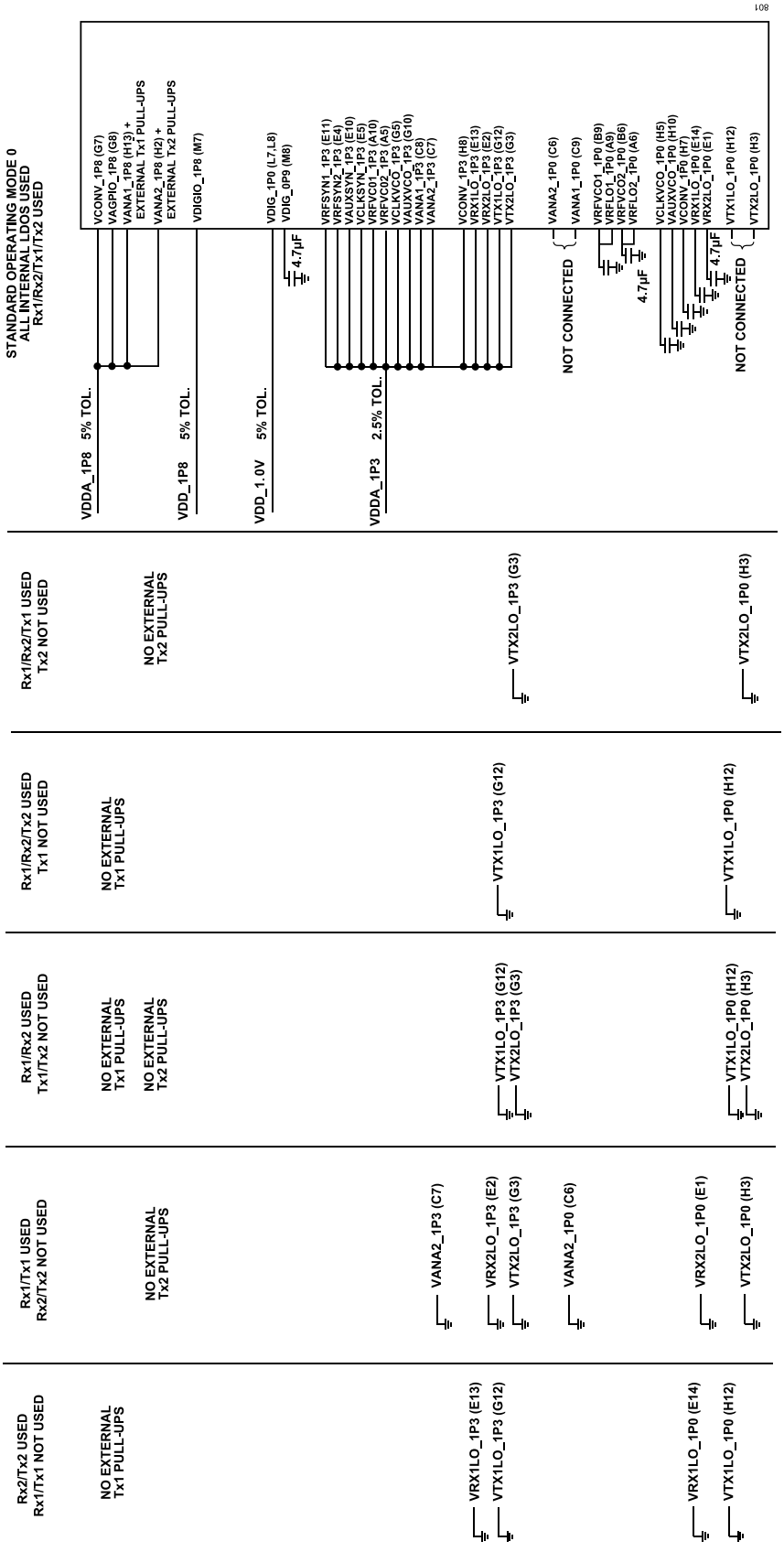


Figure 301. Power-Supply Modes for Different Number of Active Rx and Tx RF IOs

POWER-SUPPLY RECOMMENDATIONS

In cases where not all RF IOs or other interface pins are used in the end application, follow Table 125 for recommendations on what to do with unused pins.

Table 125. Instructions Explaining How to Handle ADRV9001 Unused Pins

Pin No.	Type	Mnemonic	Unused Instructions
A1, A2, A13, A14, B2 to B5, B10 to B13, C2, C5, C10, C13, D1 to D6, D9 to D14, E6, E9, F1 to F3, F6 to F9, F12 to F14, G2, G13, J1 to J14	Input	VSSA	Not applicable.
A3, A4	Input	EXT_LO2+, EXT_LO2−	Connect to VSSA.
A5	Input	VRFVCO2_1P3	Connect to VSSA when unused.
A6	Input/Output	VRFLO2_1P0	Not applicable.
A7	Input	MODEA	Not applicable.
A8	Input	RBIAS	Not applicable.
A9	Input/Output	VRFLO1_1P0	Not applicable.
A10	Input	VRFVCO1_1P3	Connect to VSSA when unused.
A11, A12	Input	EXT_LO1-, EXT_LO1+	Connect to VSSA.
B1, C1	Input	RX2A-, RX2A+	When accidentally enabled, bias voltage is present on those inputs. Connect to VSSA through capacitor.
B6	Output	VRFVCO2_1P0	Connect to VSSA when unused.
B7	Input	AUXADC_2	Do not connect.
B8	Input	AUXADC_1	Do not connect.
B9	Output	VRFVCO1_1P0	Connect to VSSA when unused
B14, C14	Input	RX1A-, RX1A+	When accidentally enabled, bias voltage is present on those inputs. Connect to VSSA through capacitor.
C3, C4	Input	RX2B+, RX2B-	When accidentally enabled, bias voltage is present on those inputs. Connect to VSSA through capacitor.
C6	Input/Output	VANA2_1P0	Connect to VSSA when unused.
C7	Input	VANA2_1P3	Connect to VSSA when unused.
C8	Input	VANA1_1P3	Not applicable.
C9	Input/Output	VANA1_1P0	Not applicable.
C11, C12	Input	RX1B-, RX1B+	When accidentally enabled, bias voltage is present on these inputs. Connect to VSSA through capacitor.
D7, D8	Input	MCS+, MCS-	Connect to VSSA.
E1	Output	VRX2LO_1P0	Connect to VSSA when unused.
E2	Input	VRX2LO_1P3	Connect to VSSA when unused.
E3, E12, F4, F5, F10, F11, G4, G6, G9, G11, H6, H9	Input/Output	AGPIO_xx	Do not connect.
E4	Input	VRFSYN2_1P3	Connect to VSSA when unused.
E5	Input	VCLKSYN_1P3	Not applicable.
E7, E8	Input	DEV_CLK_IN+, DEV_CLK_IN-	Not applicable.
E10	Input	VAUXSYN_1P3	Not applicable.
E11	Input	VRFSYN1_1P3	Connect to VSSA when unused.
E13	Input	VRX1LO_1P3	Not applicable.
E14	Output	VRX1LO_1P0	Not applicable.
G1, H1	Output	TX2+, TX2-	Do not connect.
G3	Input	VTX2LO_1P3	Connect to VSSA when unused.
G5	Input	VCLKVCO_1P3	Not applicable.
G7	Input	VCONV_1P8	Not applicable.
G8	Input	VAGPIO_1P8	Not applicable.
G10	Input	VAUXVCO_1P3	Not applicable.

POWER-SUPPLY RECOMMENDATIONS

Table 125. Instructions Explaining How to Handle ADRV9001 Unused Pins (Continued)

Pin No.	Type	Mnemonic	Unused Instructions
G12	Input	VTX1LO_1P3	Not applicable.
G14, H14	Output	TX1+, TX1-	Do not connect.
H2	Input	VANA2_1P8	Not applicable.
H3	Output	VTX2LO_1P0	Connect to VSSA when unused.
H4	Input	AUXADC_3	Do not connect.
H5	Output	VCLKVCO_1P0	Not applicable.
H7	Output	VCONV_1P0	Not applicable.
H8	Input	VCONV_1P3	Not applicable.
H10	Output	VAUXVCO_1P0	Not applicable.
H11	Input	AUXADC_0	Do not connect.
H12	Output	VTX1LO_1P0	Not applicable.
H13	Input	VANA1_1P8	Not applicable.
K1	Input	SPI_CLK	Not applicable.
K2	Input/Output	SPI_DIO	Not applicable.
K3	Input	RX2_EN	Do not connect.
K4	Input	VSSA/TESTCK+	Not applicable.
K5	Input	VSSA/TESTCK-	Not applicable.
K6 to K11, L4 to L6, L9 to L11	Input/Output	DGPIO_xx	Do not connect.
K12	Input	RX1_EN	Do not connect.
K13	Input	RESETB	Not applicable.
K14	Output	GP_INT	Do not connect.
L1	Input	SPI_EN	Not applicable.
L2	Output	SPI_DO	In SPI 3-wire mode, do not connect.
L3	Input	TX2_EN	Do not connect.
L7, L8	Input	VDIG_1P0	Not applicable.
L12	Input	TX1_EN	Do not connect.
L13	Input	MODE	Connect to VSSA.
L14	Output	DEV_CLK_OUT	Do not connect.
M1	Output	RX2_IDATA_OUT-	Do not connect.
M2	Output	RX2_IDATA_OUT+	Do not connect.
M3	Output	RX2_DCLK_OUT-	Do not connect.
M4	Output	RX2_DCLK_OUT+	Do not connect.
M5	Input/Output	DGPIO_15/TX2_DCLK_OUT+	Do not connect.
M6	Input/Output	DGPIO_14/TX2_DCLK_OUT-	Do not connect.
M7	Input	VDIGIO_1P8	Not applicable.
M8	Output	VDIG_0P9	Not applicable.
M9	Input/Output	DGPIO_12/TX1_DCLK_OUT-	Do not connect.
M10	Input/Output	DGPIO_13/TX1_DCLK_OUT+	Do not connect.
M11	Output	RX1_DCLK_OUT+	Do not connect.
M12	Output	RX1_DCLK_OUT-	Do not connect.
M13	Output	RX1_IDATA_OUT+	Do not connect.
M14	Output	RX1_IDATA_OUT-	Do not connect.
N1	Output	RX2_STROBE_OUT-	Do not connect.
N2	Output	RX2_STROBE_OUT+	Do not connect.
N3	Output	RX2_QDATA_OUT-	Do not connect.
N4	Output	RX2_QDATA_OUT+	Do not connect.
N5	Input	TX2_DCLK_IN+	Do not connect.

POWER-SUPPLY RECOMMENDATIONS

Table 125. Instructions Explaining How to Handle ADRV9001 Unused Pins (Continued)

Pin No.	Type	Mnemonic	Unused Instructions
N6	Input	TX2_DCLK_IN-	Do not connect.
N7, N8, P1, P14	Input	VSSD	Not applicable.
N9	Input	TX1_DCLK_IN-	Do not connect.
N10	Input	TX1_DCLK_IN+	Do not connect.
N11	Output	RX1_QDATA_OUT+	Do not connect.
N12	Output	RX1_QDATA_OUT-	Do not connect.
N13	Output	RX1_STROBE_OUT+	Do not connect.
N14	Output	RX1_STROBE_OUT-	Do not connect.
P2	Input	TX2_STROBE_IN+	Do not connect.
P3	Input/Output	TX2_STROBE_IN-	Do not connect.
P4	Input	TX2_QDATA_IN-	Do not connect.
P5	Input	TX2_QDATA_IN+	Do not connect.
P6	Input	TX2_IDATA_IN+	Do not connect.
P7	Input	TX2_IDATA_IN-	Do not connect.
P8	Input	TX1_IDATA_IN-	Do not connect.
P9	Input	TX1_IDATA_IN+	Do not connect.
P10	Input	TX1_QDATA_IN+	Do not connect.
P11	Input	TX1_QDATA_IN-	Do not connect.
P12	Input/Output	TX1_STROBE_IN-	Do not connect.
P13	Input	TX1_STROBE_IN+	Do not connect.

POWER SUPPLY OPTIMIZATION

Power Supply Optimization Options

Make optimizations in the power supply section to reduce the number of individual rails and filters to save on BOM and space. One possible optimization is the combination of different rails of the same voltage domain. In doing this, it is important to identify the aggressor and victim rails to decide what to combine and what to keep shielded with an FB filter.

Table 126. Victim and Aggressor Rails

Rail Name	Rail Type
VANNA1_1P3	Victim
VANNA2_1P3	Victim
VCONV_1P3	Aggressor

For this experiment, the combined rails are as follows. This reduces the number of caps and ferrite beads needed on the board.

POWER-SUPPLY RECOMMENDATIONS

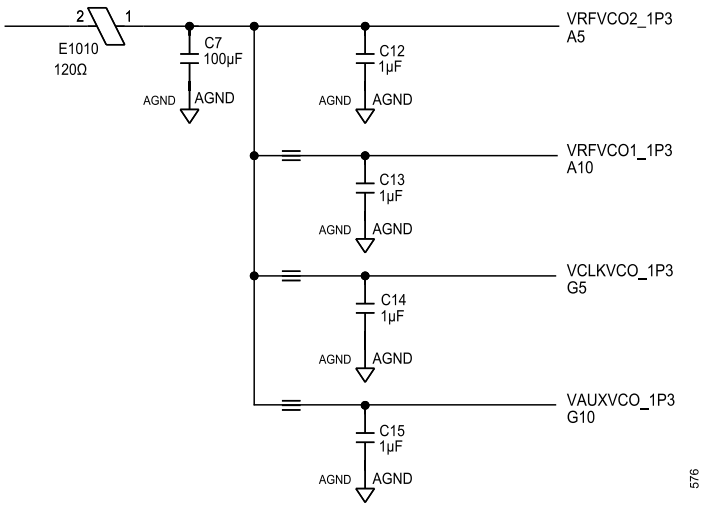


Figure 302. Merged VCO Supplies

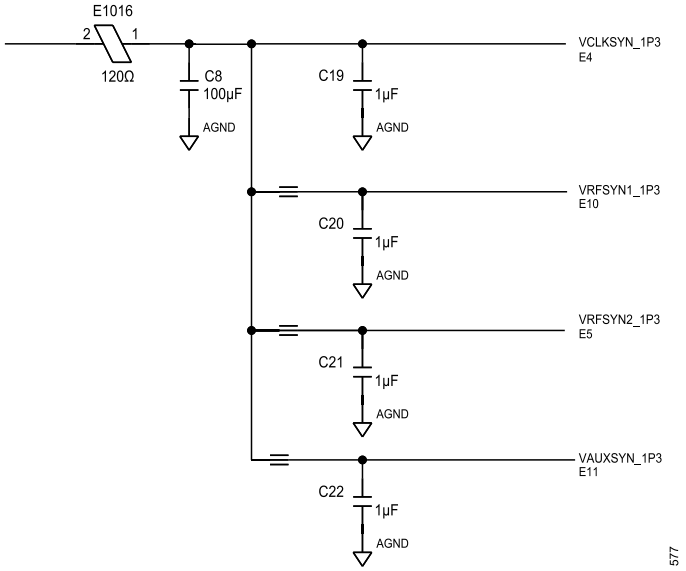


Figure 303. Merged SYNTH Supplies

POWER-SUPPLY RECOMMENDATIONS

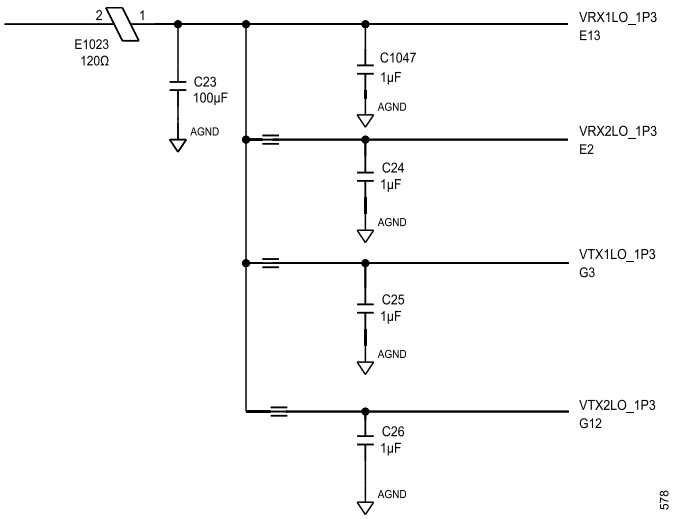


Figure 304. Merged LO Supplies

Note: is a schematic notation to merge nets with different names. There is no real component here. The rest of the power supplies need their own FB/cap filter network.

Implementing and testing this power solution shows no visible signs of RF performance degradation in phase noise or additional spurs, or any other system level issues. Any noise on the power supplies inadvertently affects the RF output of the part. So, be cautious when designing a power solution.

LDO CONFIGURATIONS

Note: Consider this section carefully. Failure to implement the LDO modes correctly can result in an incorrect voltage being applied and potentially damage the RF circuits.

The VDDA_1P0 is a power domain that powers all the transmitter and receiver LO circuits. Power this domain using internal LDOs that generate the 1 V needed from the 1.3 V supply. Alternatively, power it externally by bypassing some of the internal LDOs. Use this in cases requiring power savings or with an already available 1 V supply in the system. Save some power by turning off the LDOs and applying a higher efficiency external power source to the 1 V rails. The new 1 V external power domain is a low-noise supply with a tolerance of ±2.5%.

The LDO configurations on the ADRV9001 can affect the overall power consumption. On the ADRV9001 evaluation board, all the internal LDOs are used and they generate the 1 V needed for all the rails. This allows for different RF channels while evaluating. In an end system, the user knows what RF channels they are using and can tailor the RF channels to the system. Figure 305 shows three examples.

To implement an ideal LDO configuration for the end use case:

- Use the application to determine how many transmitter, receiver, and LO channels are needed.
- Design a power supply to implement the determined use case.
- Choose what LDO configuration is needed and set the LDO modes appropriately.

APPLICATION POWER NEEDS

Deciding what LDOs to use/bypass depends on the transmitter, receiver, and LO combinations going to be used. For example, in the 1T/2R FDD use case, do not use one of the transmit channels and then power down the internal LDOs for this channel. The Power-Supply Configurations section provides details on how to wire the power domains.

The following board power supply configurations are supported.

POWER-SUPPLY RECOMMENDATIONS

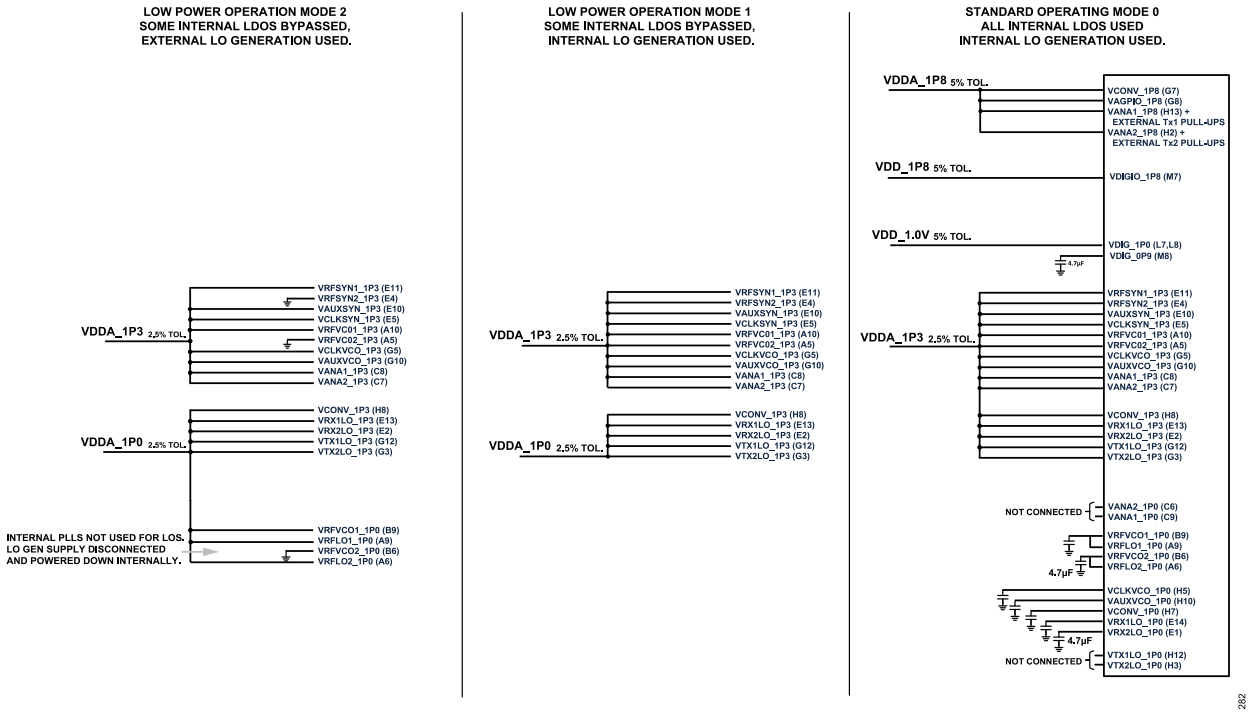


Figure 305. Power-Saving LDO Configurations

CHOOSING THE LDO CONFIGURATION

Before choosing the proper LDO configurations, it is important to understand the power domains used in the system. The LDO configurations vary depending on the use of RF channels, internal/external LO, or if an external VDDA_1P0 is supplied. Take care when setting the LDO modes to avoid any potential incorrect voltages getting supplied to the internal blocks.

Table 127 shows the LDO operating modes. Use the enumerators for each mode when setting up the adi_adrv9001_powermanagement_Configure function.

Table 127. LDO Mode for Power-Saving Configurations

Enumerator	
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1	Normal LDO operation.
ADI_ADRV9001_LDO_POWER_SAVING_MODE_2	LDO always off. Input and output of the LDO connected to ground. Used when a power domain is not required.
ADI_ADRV9001_LDO_POWER_SAVING_MODE_3	LDO always off through software.
ADI_ADRV9001_LDO_POWER_SAVING_MODE_4	Not used.
ADI_ADRV9001_LDO_POWER_SAVING_MODE_5	LDO bypassed. Supply an external source at the level expected at the LDO output (1 V, low noise, 2.5% tolerance).

- LDO mode 1 (normal operation): The LDO is in a normal mode of operation. It is used to generate an on-chip voltage.
- LDO mode 2 (LDO always off through PCB wiring): An LDO is not required and is permanently powered down through shorting its input pin, and its output pin (if available) to ground on the PCB. Generally, use this mode if the entire power domain is not required. No software control of the LDO is available.
- LDO mode 3 (LDO always off through software): Use this for the various LDOs that do not have a physical output pin. These LDOs cannot be physically tied to GND or bypassed as a result. use it if the LDOs with no output pin must be turned off. See Table 128 for the details of the LDOs.
- LDO mode 5 (LDO in bypass mode): An LDO is not required and is placed into bypass mode through software control. However, the power domain is still required. Apply a high-efficiency power source at the input pin. This mode allows to use higher efficiency power sources to supply the ADRV9001, as opposed to having the power overhead associated with an LDO.
- Table 128 lists each of the 19 LDOs as well as the constraints to explain when to power on/off, or bypass each LDO.

POWER-SUPPLY RECOMMENDATIONS

Table 128. LDO Constraints

Index	LDO	Constraints	Input Pin	Output Pin
0	GP_LDO_1	Do not power down or bypass.	VANA1_1P3	VANA1_1P0
1	DEV_CLK_LDO	Do not power down or bypass.	VANA2_1P3	VANA2_1P0
2	CONVERTER_LDO	Power down only if all ADCs and DACs must be powered down.	VCONV_1P3	VCONV_1P0
3	RX_1_LO_LDO	Power down if RX1 is not needed.	VRX1LO_1P3	VRX1LO_1P0
4	TX_1_LO_LDO	Power down if TX1 is not needed.	VTX1LO_1P3	VTX1LO_1P0
5	GP_LDO_2	Do not bypass.	VANA2_1P3	VANA2_1P0
6	RX_2_LO_LDO	Power down if RX2 is not needed.	VRX2LO_1P3	VRX2LO_1P0
7	TX_2_LO_LDO	Power down if TX2 is not needed.	VTX2LO_1P3	VTX2LO_1P0
8	CLK_PLL_SYNTH_LDO	Power down if the CLK_PLL_LP is in use.	VCLKSYN_1P3	N/A
9	CLK_PLL_VCO_LDO	Power down if the CLK_PLL_LP is in use.	VCLKVCO_1P3	VCLKVCO_1P0
10	CLK_PLL_LP_SYNTH_LDO	Power down if the CLK_PLL is in use.	VCLKSYN_1P3	N/A
11	CLK_PLL_LP_VCO_LDO	Power down if the CLK_PLL is in use.	VCLKVCO_1P3	VCLKVCO_1P0
12	LO1_PLL_SYNTH_LDO	Power down if LO1 is being externally supplied.	VRFSYN1_1P3	N/A
13	LO1_PLL_VCO_LDO	Power down if LO1 is being externally supplied.	VRFVCO1_1P3	VRFVCO1_1P0
14	LO2_PLL_SYNTH_LDO	Power down if LO2 is being externally supplied.	VRFSYN2_1P3	N/A
15	LO2_PLL_VCO_LDO	Power down if LO2 is being externally supplied.	VRFVCO2_1P3	VRFVCO2_1P0
16	AUX_PLL_SYNTH_LDO	Power down if the AUX_PLL is not required.	VAXUSYN_1P3	N/A
17	AUX_PLL_VCO_LDO	Power down if the AUX_PLL is not required.	VAUXVCO_1P3	VAUXVCO_1P0
18	SRAM_LDO	Do not power down or bypass.	VDIG_1P0	VDIG_0P9

Set up the three different configurations, as shown in Figure 305, in this function by setting the LDO modes per Table 129 and Table 128 .

Table 129. Power-Saving Configuration for Each LDO

Index	LDO	Configuration 2	Configuration 1	Configuration 0
0	GP_LDO_1	1	1	1
1	DEV_CLK_LDO	1	1	1
2	CONVERTER_LDO	5	5	1
3	RX_1_LO_LDO	5	5	1
4	TX_1_LO_LDO	5	5	1
5	GP_LDO_2	1	1	1
6	RX_2_LO_LDO	5	5	1
7	TX_2_LO_LDO	5	5	1
8	CLK_PLL_SYNTH_LDO	1	1	1
9	CLK_PLL_VCO_LDO	1	1	1
10	CLK_PLL_LP_SYNTH_LDO	1	1	1
11	CLK_PLL_LP_VCO_LDO	1	1	1
12	LO1_PLL_SYNTH_LDO	1	1	1
13	LO1_PLL_VCO_LDO	2	1	1
14	LO2_PLL_SYNTH_LDO	2	1	1
15	LO2_PLL_VCO_LDO	2	1	1
16	AUX_PLL_SYNTH_LDO	1	1	1
17	AUX_PLL_VCO_LDO	1	1	1
18	SRAM_LDO	1	1	1

Use the API function **adi_adrv9001_powermanagement_Configure()** to implement a device driver interface, allowing to set the ldoPowerSavingsModes.

Configure the **ldoPowerSavingModes** struct in **adi_adrv9001_PowerManagementSettings** (GUI generated code sets all modes = 1) to achieve the different power-saving configurations shown in Figure 305.

POWER-SUPPLY RECOMMENDATIONS

Configuration 0 is the default setup of the LDO array in the power management settings function, where all the LDOs are set to normal operation (mode 1) as follows.

```
adi_adrv9001_PowerManagementSettings_t initialize_powerManagementSettings_35 = {
.ldoPowerSavingModes = { ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1 }
```

For example in Configuration 2 of Figure 305, where some internal LDOs are bypassed and an external LO is supplied, the struct is set up as follows.

```
adi_adrv9001_PowerManagementSettings_t initialize_powerManagementSettings_35 = {
.ldoPowerSavingModes = { ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_5,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_5, ADI_ADRV9001_LDO_POWER_SAVING_MODE_5,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_5,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_5, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_2, ADI_ADRV9001_LDO_POWER_SAVING_MODE_2,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_2, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1 }
```

In this case, it is still assumed that 2T/2R channels are being used. If the user application only uses Tx1 and Rx1, then set the LDOs for these channels to a bypass or power-down mode along with hardware modifications. There is a relationship between the hardware power layout and the LDO mode. So, pay attention to correctly set the part to reflect the physical implementation. Attention is also needed when bypassing LDOs to make sure that internal circuits are not supplied with incorrect voltage levels.

EXAMPLE:

TDD operation using Tx1, Rx1, and external LOs.

The hardware power setup looks like Figure 306, where the pins for the Tx2 and Rx2 1 V and 1.3 V LO domains are tied to ground. The LDOs powering these RF blocks, VANA2_1P0, and VANA2_1P3, are also tied to ground because they are not needed.

POWER-SUPPLY RECOMMENDATIONS

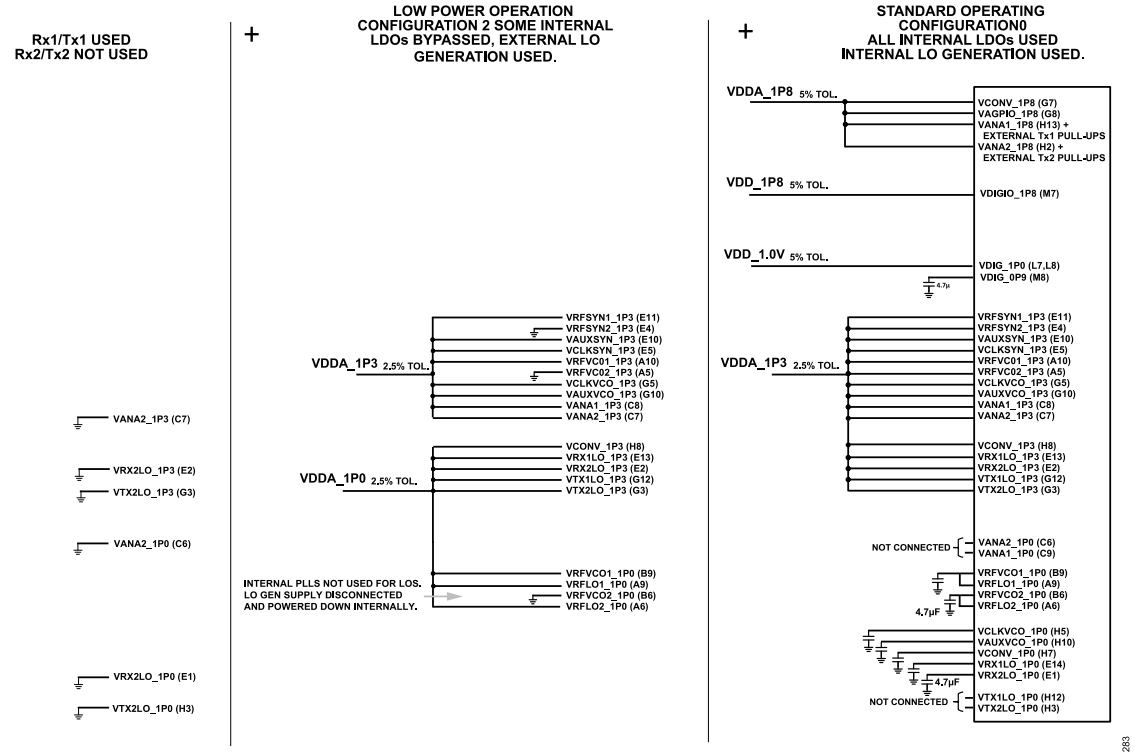


Figure 306. Standard Operating Config 0 and External LO Config and Rx1/Tx1 Power-Supply Config

When the external LO configuration and Rx1/Tx1 configuration are super-positioned onto the standard operating configuration, connect the power supplies per Figure 307.

POWER-SUPPLY RECOMMENDATIONS

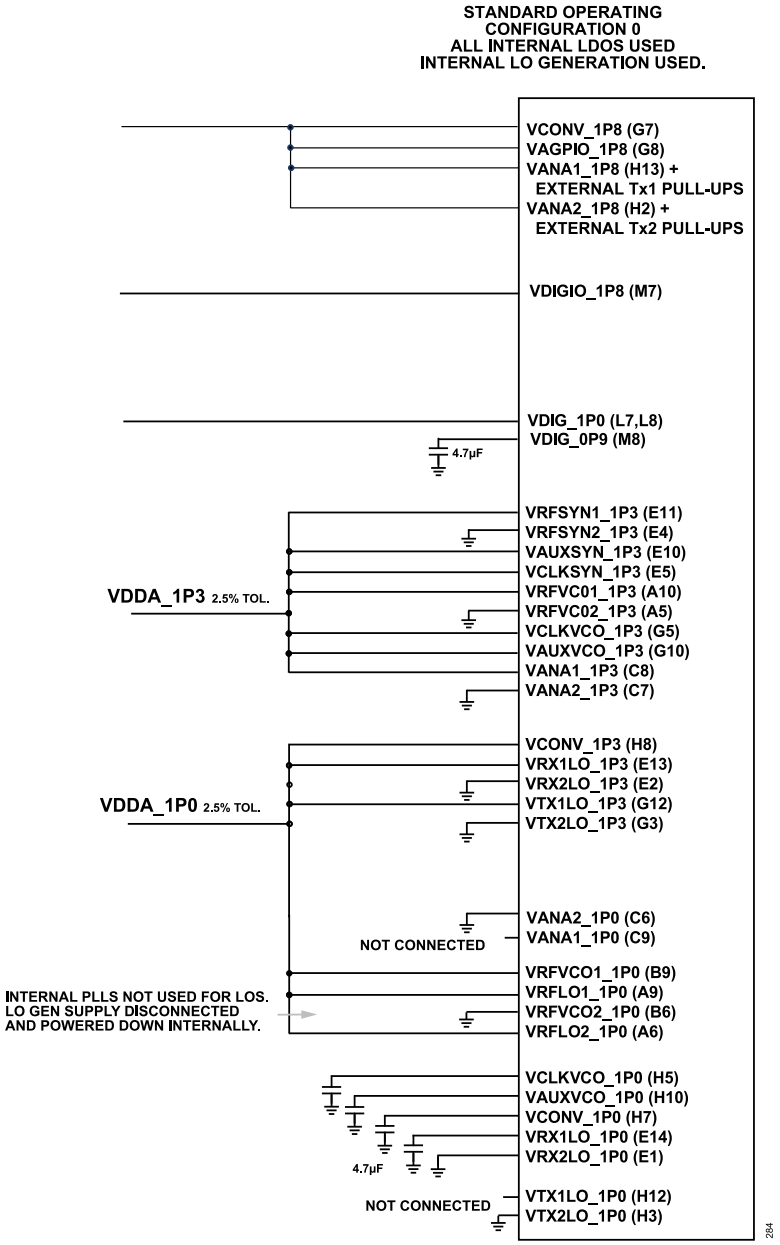


Figure 307. Tx1/Rx1 External LO Power Solution

Now that the hardware configuration is known, determine the LDO configuration and set each LDO to the correct mode. Table 130 outlines the mode needed for each LDO to prevent overvoltage failures or other unwanted errors.

Table 130. Example LDO Modes and Pin Mapping

Index	LDO	LDO Mode	Output Pin	Output Pin Number	Comments
0	GP_LDO_1	1	VANA1_1P0	C9	Must be left on.
1	DEV_CLK_LDO	1	N/A	N/A	Must be left on.
2	CONVERTER_LDO	5	VCONV_1P0	H7	Bypassed due to external 1.0 V.
3	RX_1_LO_LDO	5	VRX1LO_1P0	E14	Bypassed due to external 1.0 V.
4	TX_1_LO_LDO	5	VTX1LO_1P0	H12	Bypassed due to external 1.0 V.

POWER-SUPPLY RECOMMENDATIONS

Table 130. Example LDO Modes and Pin Mapping (Continued)

Index	LDO	LDO Mode	Output Pin	Output Pin Number	Comments
5	GP_LDO_2	2	VANA2_1P0	C6	Powered off, ch2 not required.
6	RX_2_LO_LDO	2	VRX2LO_1P0	E1	Powered off, ch2 not required.
7	TX_2_LO_LDO	2	VTX2LO_1P0	H3	Powered off, ch2 not required.
8	CLK_PLL_SYNTH_LDO	1	N/A	N/A	Left on.
9	CLK_PLL_VCO_LDO	1	VCLKVCO_1P0	H5	Left on.
10	CLK_PLL_LP_SYNTH_LDO	1	N/A	N/A	Left on.
11	CLK_PLL_LP_VCO_LDO	1	VCLKVCO_1P0	H5	Left on.
12	LO1_PLL_SYNTH_LDO	3	N/A	N/A	Powered off, using external LO.
13	LO1_PLL_VCO_LDO	2	VRFVCO1_1P0	B9	Powered off, using external LO.
14	LO2_PLL_SYNTH_LDO	3	N/A	N/A	Powered off, LO2 not required.
15	LO2_PLL_VCO_LDO	2	VRFVCO2_1P0	B6	Powered off, LO2 not required.
16	AUX_PLL_SYNTH_LDO	1	N/A	N/A	Left on.
17	AUX_PLL_VCO_LDO	1	VAUXVCO_1P0	H10	Left on.
18	SRAM_LDO	1	VDIG_0P9	M8	Left on.

The following is the C# implementation of this configuration.

```
adi_adrv9001_PowerManagementSettings_t initialize_powerManagementSettings_35 = {
.ldoPowerSavingModes = { ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_5,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_5, ADI_ADRV9001_LDO_POWER_SAVING_MODE_5,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_2, ADI_ADRV9001_LDO_POWER_SAVING_MODE_2,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_2, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_3,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_2, ADI_ADRV9001_LDO_POWER_SAVING_MODE_3,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_2, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1,
ADI_ADRV9001_LDO_POWER_SAVING_MODE_1, ADI_ADRV9001_LDO_POWER_SAVING_MODE_1 }
```

SUMMARY

The design specification for the ADRV9001 EVB shows the best performance of the part with little to no degradation from the PCB design. The Analog Devices applications engineering team recommends the circuit board layout and power-supply decoupling covered in this document. These recommendations are based on circuit analysis, simulation, best practice layout techniques, and experience with this and other transceiver devices. Designers are strongly encouraged to follow these guidelines carefully when developing their own PCB applications. Deviation from these recommendations are possible (but may result in substandard system performance that is very difficult to isolate after the system is committed to a PCB design). The ADRV9001 demo board layout files and schematics use many of the recommended techniques. Download the design file package from https://www.analog.com/en/products/adrv9002.html.

ADRV9001 EVALUATION SYSTEM

The ADRV9001 family demonstration system evaluates the device without having to develop custom software or hardware. The system comprises a radio daughtercard, a Xilinx ZYNQ ZC706 or ZCU102 motherboard, an SD card with operating system, a 12 V power supply for the ZYNQ ZC706 or ZCU102 that connects to a wall outlet, and a C#-based evaluation software application. The evaluation system uses an Ethernet interface to communicate with the PC.

INITIAL SETUP

The ADRV9001 transceiver evaluation software (TES) is the GUI to communicate with the evaluation platform. It can run with or without the evaluation hardware. When the TES runs without the hardware, configure it fully for a particular operating mode for demonstration. If the evaluation hardware is connected, set up the desired operating parameters with the TES, and then the software can program the evaluation hardware. After the device is configured, use the evaluation software to transmit waveforms using custom waveform files as well as observe signals received on one of the receiver input ports. The TES can be used to generate and execute an initialization sequence in the form of an IronPython script.

HARDWARE KIT

The ADRV9001 demonstration kit contains the following:

- The customer evaluation (CE) board as a daughter card with FPGA mezzanine card (FMC) connector.
- One SD card containing the image of the Linux operating system with the required evaluation software.
- The SD card type is 16 GB size, type 10.

Requirements

The hardware and software require the following:

- The ADRV9001 demonstration system kit.
- A Xilinx ZC706 ZYNQ (EK-Z7-ZC706) or ZCU102 evaluation platform (the ADRV9001 demonstration kit does not include the Xilinx platforms)*.
- One 12 V power supply to power the Xilinx platform.
- Microsoft Windows® 7 (×86 and ×64) or Windows 10 (×86 and ×64) operating system on the controlling PC
- The PC must have a free Ethernet port with the following constraints:
- If the Ethernet port is occupied by another LAN connection, use a USB-to-Ethernet adapter.
- The PC must be able to access the following ports over this dedicated Ethernet connection:
  - 22 (SSH protocol)
  - 55557 (access to the evaluation software on the Xilinx platform).
  - TES (contact an Analog Devices representative for access to this software).
- The user must have administrative privileges.

SD Card Imaging

To image the SD card properly for use in the Xilinx platform, download the ADRV9001 Disk Imaging Utility and the dotNet Disk Imager. Find these on the EngineerZone® support forum for the TES GUI & Software Support. Follow the instructions and check if there is any encryption when writing to an SD card that prevents the FPGA from reading the card.

Hardware Setup (ZYNQ ZC706)

The Xilinx ZYNQ ZC706 platform setup requires the following steps:

1. All jumpers are in the positions shown in Figure 308.
2. SW11 is in position, as shown in Figure 308 (1, 2, 5 = A position).
3. Place the SD card included with the evaluation kit in the J3 slot of the ZYNQ platform.

Figure 310 shows the evaluation hardware setup.

**All variants of the ZC706 evaluation kit are suitable to demonstrate the performance of the ADRV9001. Analog Devices does not guarantee that all future versions and derivatives of the ZC706 will work with the ADRV9001 daughter card and ADRV9001 TES to demonstrate*

ADRV9001 EVALUATION SYSTEM

the ADRV9001 performance. However, as of the date of the current release of this user guide, there are no known incompatibilities with any versions and derivatives of the ZC706. For example, the following versions have been successfully used for evaluation purposes: EK-Z7-ZC706-G and EVAL-TPG-ZYNQ3

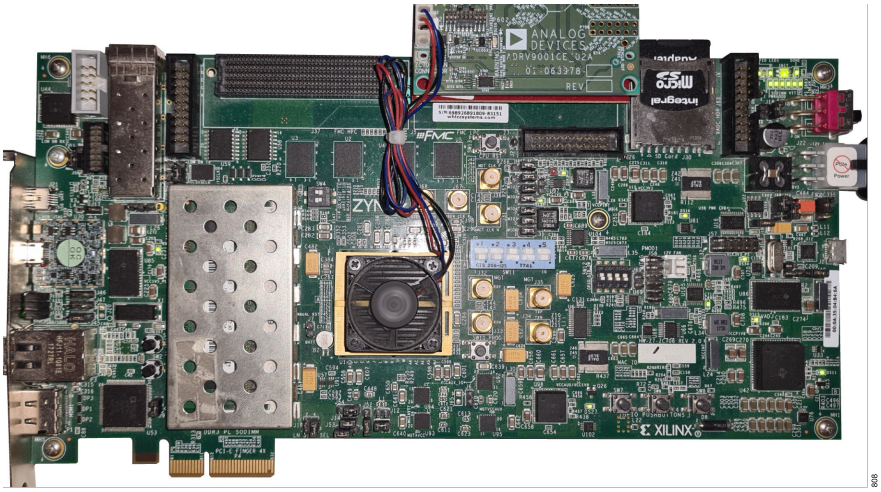


Figure 308. Xilinx Evaluation Board with Jumper Settings and Switch Position Configured to Work with ADRV9001 Evaluation Platform

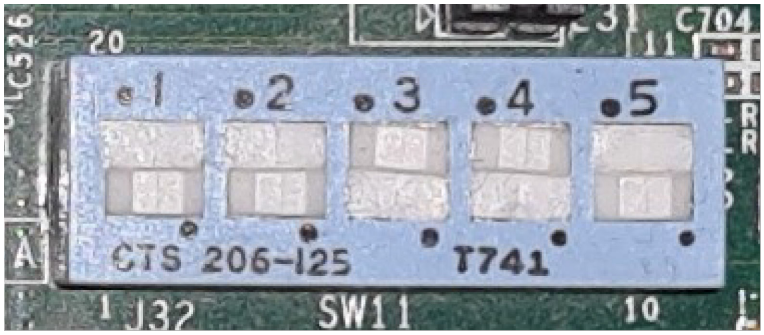


Figure 309. SW11 Positions

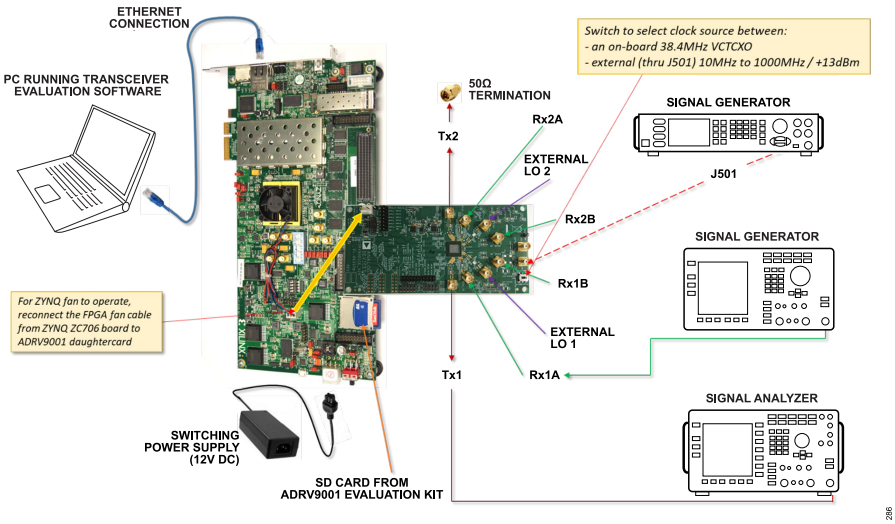


Figure 310. ADRV9001 Evaluation Card and ZYNQ ZC706 Evaluation Platform with Connections Required for Testing

The steps to set up the evaluation board for testing are as follows:

ADRV9001 EVALUATION SYSTEM

1. Connect the ADRV9001 evaluation card and the ZYNQ ZC706 evaluation platform together, as shown in Figure 310. Use the LPC FMC connector (J5). Take care to align the connectors properly.
2. Make sure that all jumpers on the ZYNQ ZC706 evaluation platform as well as the SW11 position (1, 2, 5 = “A” position) match the settings shown in Figure 308.
3. Insert the SD card that comes with the ADRV9001 evaluation kit into the ZYNQ ZC706 evaluation platform SD card slot (J30).
4. On the ADRV9001 evaluation card, provide a device clock (frequency must match the setting selected in the TES) at a +13 dBm power level to the J501 connector ((this signal drives the reference clock into the ADCLK944 clock distribution chip on the board (the Q1/Q1_N pins of the ADCLK944 generate the DEV_CLK for the ADRV9001 and REF_CLK for the Xilinx FPGA on the ZYNQ platform)).

   a. Note that the quality of the clock source used to generate the DEV_CLK directly impacts the overall system performance. Use a high-quality, stable, and low-phase noise clock source .

   Alternatively, switch the "DEV_CLK Source Sel." switch to 'Internal' to use the on-board VCTXO. The VCTXO only supports 38.4MHz clock frequency.
5. Connect a 12 V, 5 A power supply to the ZYNQ ZC706 evaluation platform at the J22 header.
6. Connect the ZYNQ evaluation platform to the PC with an Ethernet cable (connect to P3). Driver installation is not required.

   a. When the Ethernet port is already occupied by another connection, use a USB-to-Ethernet adapter.
   b. On an Ethernet connection dedicated to the ZYNQ evaluation platform, manually set the following:

      1. IPv4 Address to: 192.168.1.2
      2. IPv4 Subnet Mask to: 255.255.255.0

See Figure 311 for more details. Make sure the firewall on the PC does not block the following ports.

- 22 (SSH protocol)
- 55557 (access to the evaluation software on the ZYNQ platform).
- Note that the ZYNQ ZC706 evaluation platform IP address is set by default to: 192.168.1.10.

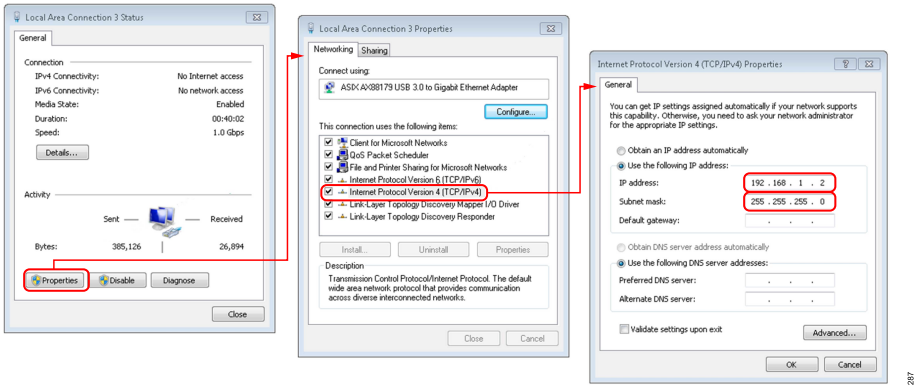


Figure 311. IP Settings for Ethernet Port Dedicated for ZYNQ ZC706 or ZCU102 Evaluation Platform

Hardware Setup (ZCU102)

The Xilinx ZCU102 platform setup requires the following steps:

1. All jumpers are in the positions shown in Figure 312.
2. The SW6 is in position, as shown in Figure 308 (2, 3, 4 = A position).
3. Place the SD card included with the evaluation kit in the J100 slot of the ZYNQ platform.

Figure 314 shows the evaluation hardware setup.

ADRV9001 EVALUATION SYSTEM

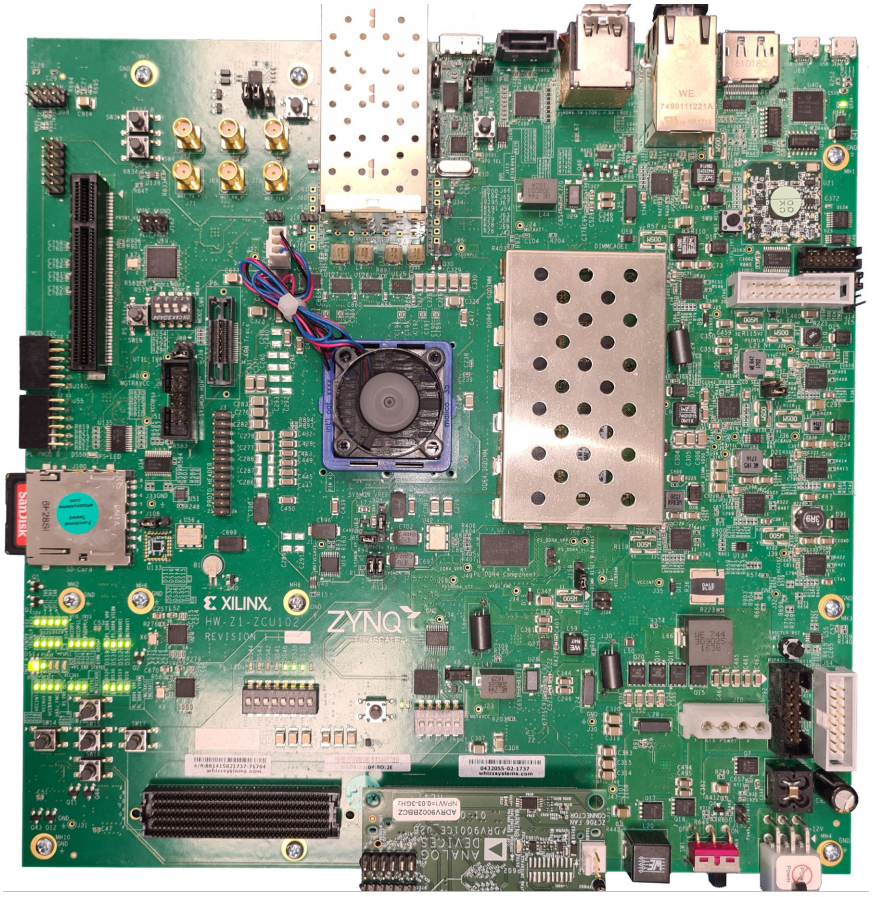


Figure 312. Xilinx ZCU102 Evaluation Board with Jumper Settings and Switch Position Configured to Work with ADRV9001 Evaluation Platform

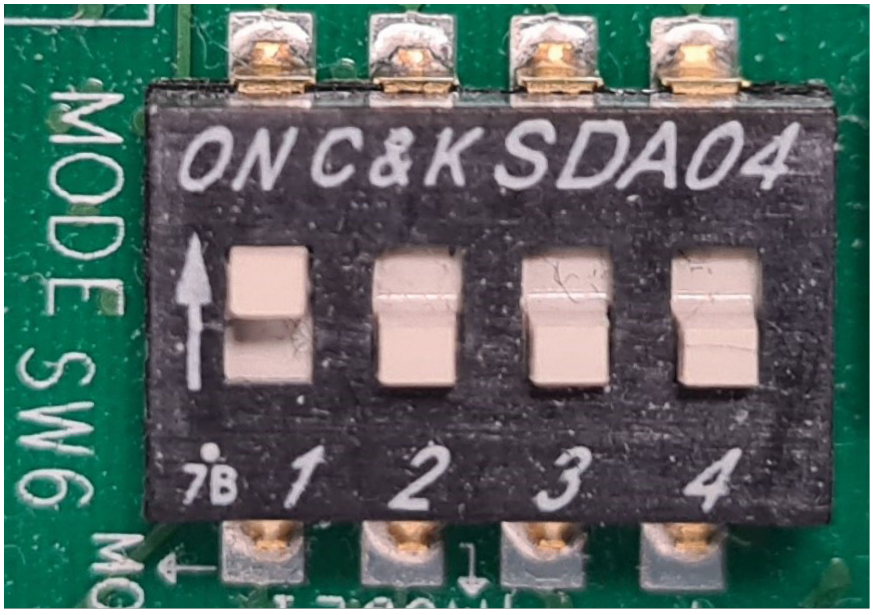


Figure 313. SW6 Positions

ADRV9001 EVALUATION SYSTEM

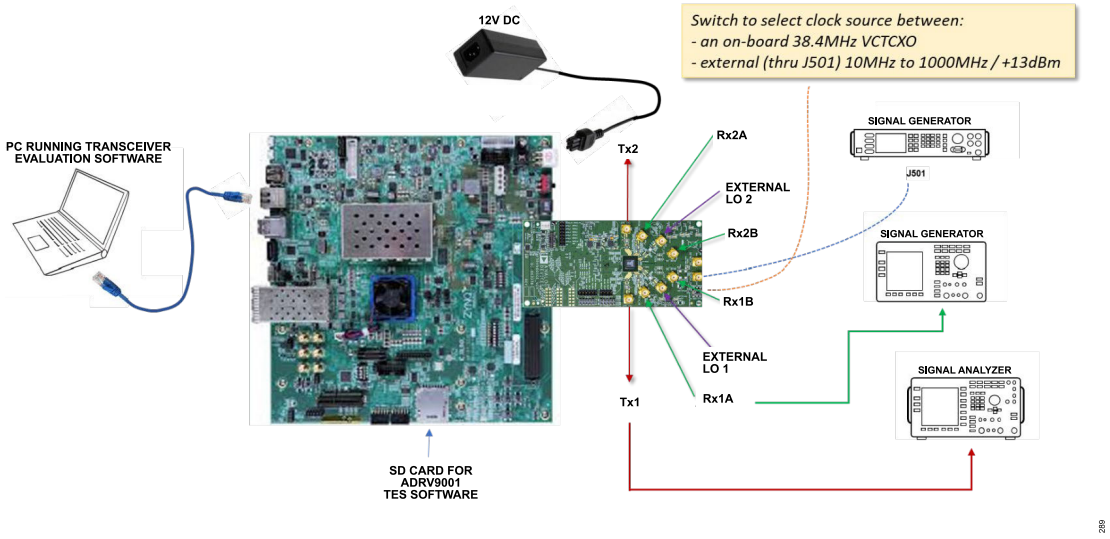


Figure 314. ADRV9001 Evaluation Card and ZCU102 Evaluation Platform with Connections Required for Testing

The steps to set up the evaluation board for testing are as follows:

1. Connect the ADRV9001 evaluation card and the ZCU102 evaluation platform together, as shown in Figure 314. Use the FMC connector (J5). Take care to align the connectors properly.
2. Make sure that all jumpers on the ZCU102 evaluation platform as well as the SW6 position (2, 3, 4 = “A” position) match the settings shown in Figure 312.
3. Insert the SD card that came with the ADRV9001 evaluation kit into the ZCU102 evaluation platform SD card slot (J100).
4. On the ADRV9001 evaluation card, provide a device clock (frequency must match the setting selected in the TES) at a +13 dBm power level to the J501 connector ((this signal drives the reference clock into the ADCLK944 clock distribution chip on the board (the Q1/Q1_N pins of the ADCLK944 generates the DEV_CLK for the ADRV9001 and REF_CLK for the Xilinx FPGA)).

   a. Note that the quality of the clock source used to generate the DEV_CLK directly impacts the overall system performance. Use a high-quality, stable, and low-phase noise clock source.

   Alternatively, switch the "DEV_CLK Source Sel." switch to 'Internal' to use the on-board VCTXO. The VCTXO only supports 38.4MHz clock frequency.
5. Connect a 12 V, 5 A power supply to the Xilinx platform at the J52 header.
6. Connect the Xilinx platform to the PC with an Ethernet cable (connect to P12). Driver installation is not required.

   a. When the Ethernet port is already occupied by another connection, use a USB-to-Ethernet adapter.
   b. On an Ethernet connection dedicated to the Xilinx platform, manually set the following:

      1. IPv4 Address to: 192.168.1.2
      2. IPv4 Subnet Mask to: 255.255.255.0

See Figure 311 for more details. Make sure the firewall on the PC does not block the following ports.

- 22 (SSH protocol)
- 55557 (access to the evaluation software on Xilinx platform).
- Note that the ZYNQ ZCU102 evaluation platform IP address is set by default to: 192.168.1.10.

HARDWARE OPERATION

ZYNQ ZC706

1. Turn on the evaluation system by switching the ZYNQ ZC706 evaluation platform power switch (SW1) to 'on'. If the hardware is connected correctly, two green LEDs (D801 and DS901) on the ADRV9001 evaluation card must be on.

ADRV9001 EVALUATION SYSTEM

2. The ZYNQ ZC706 evaluation platform uses a Linux operating system. It takes approximately 30 seconds before the system is ready for operation and can accept commands from the PC software. Observe the boot status on the ZYNQ ZC706 evaluation platform GPIO LEDs (L, C, R, O). The correct sequence has the following description:
    a. After the SW1 is turned on, all four LEDs start flashing (moving single OFF light).
    b. On the ADRV9001 evaluation board, the "System OK" LED also begins to flash.
    c. These LEDs continue to flash in the same sequence through the FPGA boot up, TES connection, and TES program.
3. The reference clock signal (in the 10 MHz to 1000 MHz range, CW tone, +13 dBm max) is connected to the J501.
    a. Note that the quality of the clock source used to generate the DEV_CLK directly impacts the overall system performance. Use a high-quality, stable, and low-phase noise clock source.
4. To test the receiver on the ADRV9001 evaluation card, use a clean signal generator with low-phase noise to provide an input signal to the selected receiver RF input. Use a shielded RG-58, 50 Ω coaxial cable (1 m or shorter) to connect the signal generator.
5. To set the input level near the receiver's full scale, set the generator level (for a single tone signal) to approximately -15 dBm. This level depends on the input frequency and gain settings through the path.
    a. Note that no input signal is applied to the receiver input when performing an "init" calibration.
6. To test the transmitter, connect a spectrum analyzer to either transmitter output on the ADRV9001 evaluation card. Use a shielded RG-58, 50 Ω coaxial cable (1 m or shorter) to connect the spectrum analyzer.
    a. Terminate both transmitter outputs, either into spectrum analyzers or into 50 Ω, if unused. This is because the initial calibrations run on both channels and can take a long time to complete if a transmitter channel is not correctly terminated.
7. Execute power-off using the TES or power down the ZYNQ ZC706 evaluation platform using the SW9 push button before powering off the evaluation system by switching SW1 to "off".

ZYNQ ZCU102

1. Turn on the evaluation system by switching the ZCU102 evaluation platform power switch (SW1) to "on". If the hardware is connected correctly, two green LEDs (D801 and DS901) on the ADRV9001 evaluation card must be on.
2. The ZCU102 evaluation platform uses a Linux operating system. It takes approximately 30 seconds before the system is ready for operation and can accept commands from the PC software. Observe the boot status on the ZCU102 evaluation platform GPIO LEDs (DS37–DS40). The correct sequence has the following description:
    a. After the SW1 is turned on, all four LEDs start flashing (moving single OFF light).
    b. On the ADRV9001 evaluation board the "System OK" LED also begins to flash.
    c. These LEDs continue to flash in the same sequence through the FPGA boot up, TES connection, and TES program.
    d. When boot is done, the "Init" LED is green and "Init Done" LED is "on" in the "Power" and "Status" LEDs.
3. The ADRV9001 reference clock signal (in the 10 MHz to 1000MHz range, CWtone, +13 dBm max) is connected to the J501.
    a. Note that the quality of clock source used to generate the DEV_CLK directly impacts the overall system performance. Use a high-quality, stable, and low-phase noise clock source.
4. To test the receiver on the ADRV9001 evaluation card, use a clean signal generator with low phase noise to provide an input signal to the selected receiver RF input. Use a shielded RG-58, 50 Ω coaxial cable (1 m or shorter) to connect the signal generator.
5. To set the input level near the receiver's full scale, set the generator level (for a single-tone signal) to approximately -18 dBm. This level depends on the input frequency and gain settings through the path.
    a. Note that no input signal is applied to the receiver input when performing an "init" calibration.
6. To test the transmitter, connect a spectrum analyzer to the transmitter output on the ADRV9001 evaluation card. Use a shielded RG-58, 50 Ω coaxial cable (1 m or shorter) to connect the spectrum analyzer.
    a. Terminate both transmitter outputs, either into spectrum analyzers or into 50 Ω, if unused. This is because the initial calibrations run on both channels and can take a long time to complete if a transmitter channel is not correctly terminated.
7. Execute power-off using the TES before powering off the evaluation system by switching SW1 to "off".

ADRV9001 EVALUATION SYSTEM

IMPORTANT NOTES

The ADRV9001 evaluation system uses a Linux operating system. Linux requires time to boot up as well as for a soft shutdown before the hardware powers off. Use the software power-off feature or press the SW9 button on the ZYNQ ZC706 or ZYNQ ZCU102 evaluation platform before physically switching the power to 'off' using SW1. If this advice is not followed, the file system on the SD card can get corrupted and the ADRV9001 evaluation system can stop operating.

Perform correct shutdown by executing one of the following options:

- Selecting *"File"* → *"Shutdown Zynq Platform"* in the TES.
- Pressing the SW9 push button on the ZYNQ platform.

After a few seconds, when all four GPIO LEDs on the ZYNQ platform blink together, safely power off the system using SW1 on the ZYNQ platform.

TRANSCEIVER EVALUATION SOFTWARE

Installation and Configuration

Contact an Analog Devices representative for access to the TES or to download it from the Product Page on the Analog Devices website. After the initial software download, copy the software to the target system, and unzip the files (unless it is already unzipped). The downloaded zip container has an executable file that installs the SDK, and then the evaluation software.

The TES installer does not demand administrator privileges by default. However, to install the TES to a folder that requires administrative privileges, run the installation process with the requisite administrator privileges.

After running an executable file, a standard installation process follows. Figure 315 shows the recommended configuration. Microsoft .NET Framework 4.5 or newer is necessary for the TES to operate. During installation process, the TES looks for the Microsoft .NET Framework, and if not available on the PC, it tries to download it from the Microsoft server. When the Microsoft .NET Framework installation is selected, the installer checks and informs the operator if the newer version is already installed. If so, it skips the .NET Framework installation.

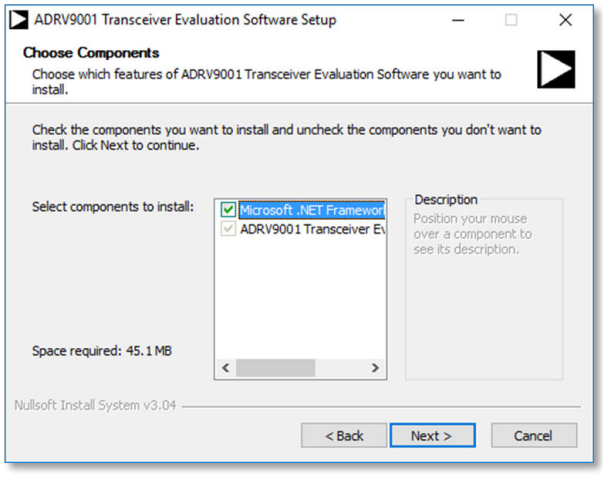


Figure 315. Software Installation Components

During installation process, the TES asks for the destination folder to install the files (Figure 316). Stay with the default location **C:\Program Files (x86)\Analog Devices\ADRV9002 Transceiver Evaluation Software**. If this is not possible, install the TES into any other location with write access. Select the shortcut configuration as the last step of the installation process.

ADRV9001 EVALUATION SYSTEM

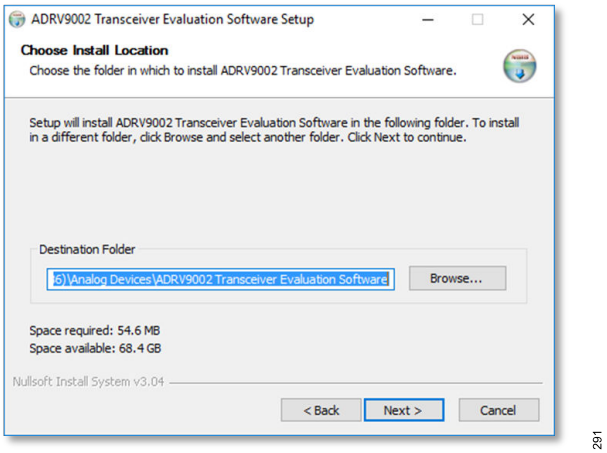


Figure 316. Software Installation Directory

Starting the Transceiver Evaluation Software

Start the TES by clicking "Start → ADRV9001 Transceiver Evaluation Software". Figure 317 shows the opening page of the TES after activation.

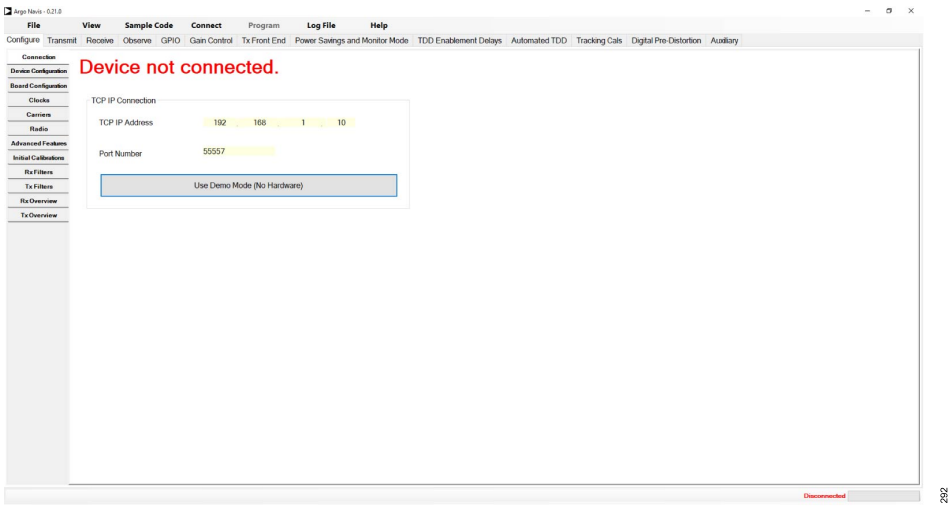


Figure 317. Main Interface of Transceiver Evaluation Software Bridge

When the evaluation hardware is connected to a PC, and to start using the complete evaluation system, the TES establishes a connection with the Xilinx platform system through the Ethernet connection upon clicking **Connect**. Upon establishing proper connection, configure an evaluation hardware. Upon selecting the **Connection** tab, the top part of that window shows the TCP IP address (default 192.168.1.10) and Port Number (default 55557), where the bottom part of the window displays the information about the connected hardware and revisions of different software setup blocks. DHCP is enabled by default in Firmware version 0.14.5.5. When connecting the evaluation platform directly to a PC, the default IP address works. If connecting to the platform through a router, then identify and populate the IP address into the TES before connecting. Contact the Analog Devices applications engineering team if the ADRV9001 evaluation system must operate over a remote connection and a different IP address is required for the Xilinx platform. For common connection issues, reference the debug section in the Engineer Zone.

Figure 318 shows an example of the correct connection between a PC and a Xilinx platform with an ADRV9001 daughter card connected to it. In this window, check both the hardware version and all software component versions used by the system in the current TES revision.

ADRV9001 EVALUATION SYSTEM

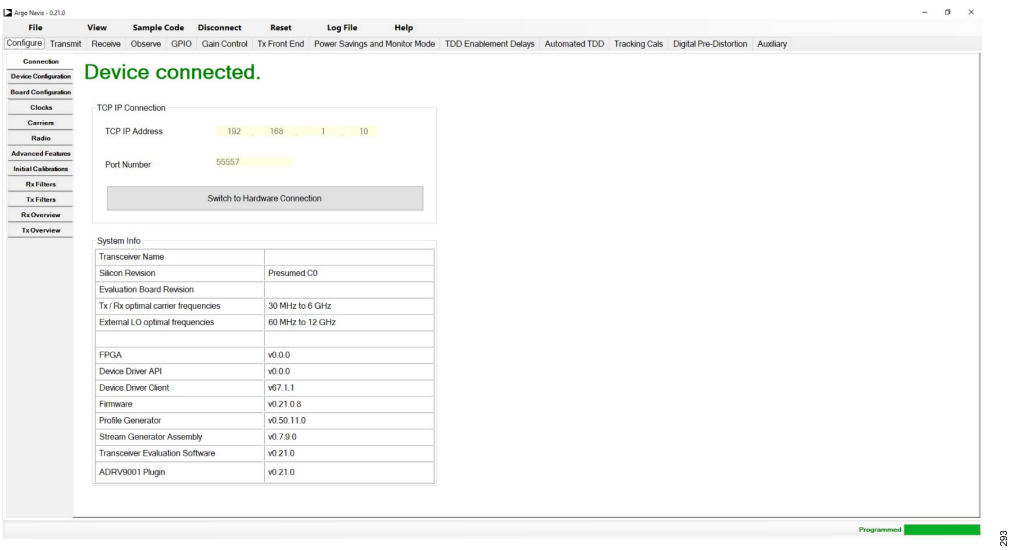


Figure 318. Setup Revision Information

Configuring the Device and Board

- The **Device Configuration** tab has the setup options for the device. In this page, select the following:
- Product
  - Supports ADRV9002, ADRV9003, and ADRV9004.
- System
- Supports TDD, FDD and TDM_FDD.
  - Under TDD:
    - Supports DMR setup.
    - Supports Analog FM setup.
    - Supports LTE setup.
    - Supports Configuration 1, 3, and 4 setup.
    - Custom
    - Custom Extended (allows the "System Clock" to run as slow as 150 MHz).
  - Under FDD:
    - Supports Analog FM setup.
    - Supports LTE setup: This mode allows to configure different channel configurations such as Rx2/Tx1.
    - Supports Configuration 2 setup.
    - Custom
    - Custom Extended (allows the "System Clock" to run as slow as 150 MHz).
  - Under TDM_FDD:
    - Supports Tetra.
    - Supports custom configuration.
    - Custom Extended (allows the "System Clock" to run as slow as 150 MHz).
- Set SSI to CMOS or LVDS.
  - 1 or 4 lane option for CMOS.
  - 2 lanes for LVDS.
  - Set SSI Strobe to long or short.
- Signal type (depends on the selected system and setup).
  - RX supports I/Q and frequency deviation types and bit lengths.

ADRV9001 EVALUATION SYSTEM

- TX supports I/Q, I/Q FM/FSK, direct FM/FSK types.
- Frequency Deviation
  - This option is available only for the TX FM type setups.
- Enable ORx1 and ORx2 for IQ input.
- Interface Rate allows to select the rate for the interface. Use this to over or under sample from the sample rate. Provide own PFIR to account for this.

There is a traffic light indicator that checks the selected settings and indicates they are acceptable or not.

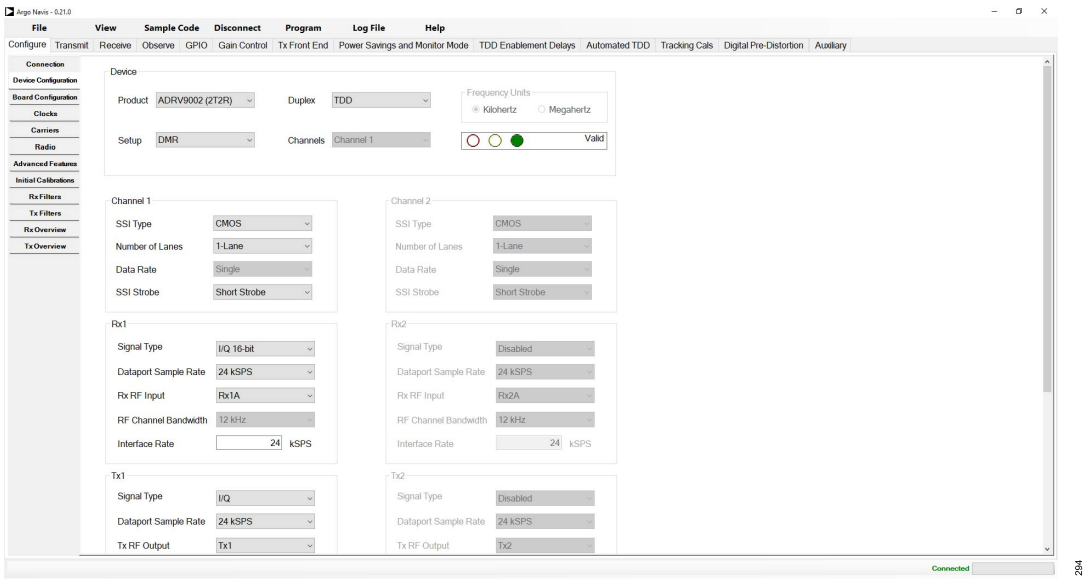


Figure 319. Device Configuration Tab

The **Board Configuration** tab has settings for the transmitter **External Loopback** typically used for DPD-type applications. Enable or disable the external loopback after power amplifier. If it is enabled, enter the expected loopback peak power in the **Peak Power** entry. Default peak power is −18 dBm. These are associated with RX1/2B ports.

Measure the external loopback path using an API, sending a low-level wideband signal in the datapath for delay measurement. This action disrupts the transmit signal in the air. Do this in a test environment and before the power amplifier is transmitting real data. Use the **adi_adrv9001_cals_ExternalPathDelay_Calibrate()** and **adi_adrv9001_cals_ExternalPathDelay_Get()** to retrieve the external loopback path delay in "ns". The SDK has an IronPython example of this. For details, see the IronPython Scripting section. Note that for this measurement, there is a limitation with the external delay measurement used by the DPD.

1. Measure the LTE10 profiles to obtain the highest possible measurement accuracy.
2. Measure upon ADRV9001 entering the CALIBRATED state for the first time.

Obtain **SSI Ref Clock** from the transmitter or receiver channel. When using the clock from the transmitter channel, push this to two GPIO pins. Using the receiver SSI clock releases these two pins for other uses.

Use **External LNA** to control the analog output from the GPIOs to control the gain of the system LNA. The pin settings currently are locked to one nibble. Filling in the LNA gain steps allows the API functions to calculate the extra gain table settings. View this in the **Gain Control** tab after programming the part.

Extend the gain table with gain settings that include the user-defined LNA configuration. The "Extended Gain Control" column specifies the control word for the LNA. and this is shown with colored backgrounds. The "Duplicates" column shows the row number from the table copied and appended with the "External Gain Control" word to get the desired gain. Figure 321 shows that the top row is a copy of row 196 with just the external gain control changed.

ADRV9001 EVALUATION SYSTEM

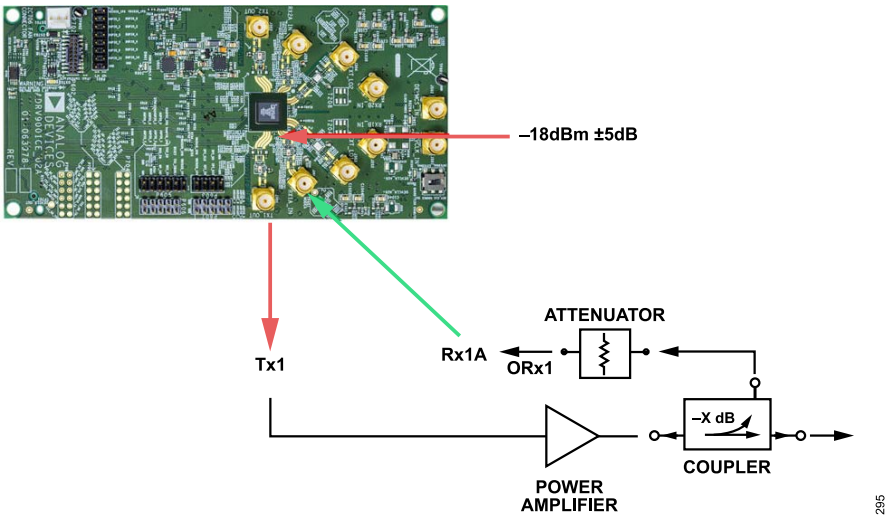
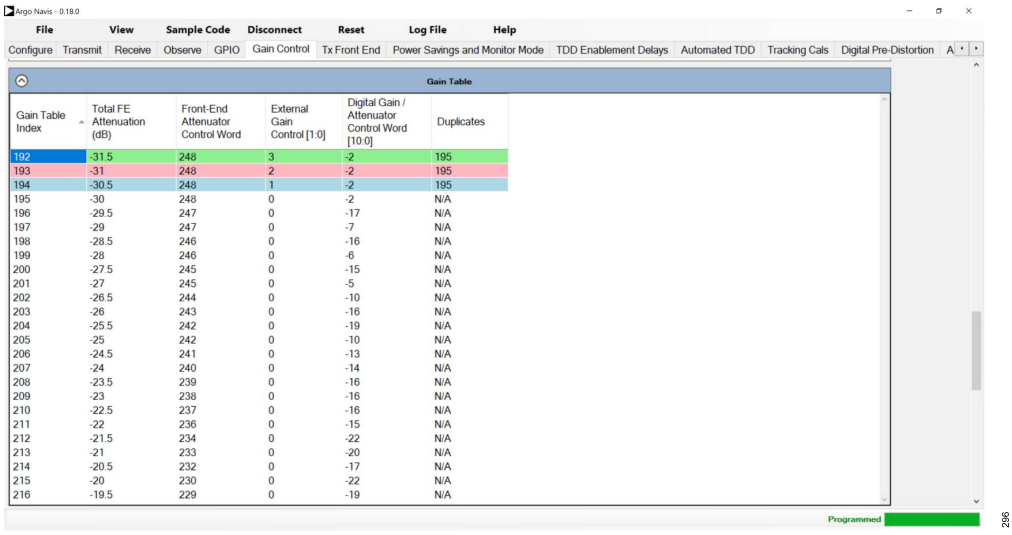


Figure 320. Receiver/Observation Receiver Loopback Diagram



Gain Table Index	Total FE Attenuation (dB)	Front-End Attenuator Control Word	External Gain Control [1:0]	Digital Gain / Attenuator Control Word [10:0]	Duplicates
192	-31.5	248	3	-2	195
193	-31	248	2	-2	195
194	-30.5	248	1	-2	195
195	-30	248	0	-2	N/A
196	-29.5	247	0	-17	N/A
197	-29	247	0	-7	N/A
198	-28.5	246	0	-16	N/A
199	-28	246	0	-6	N/A
200	-27.5	245	0	-15	N/A
201	-27	245	0	-5	N/A
202	-26.5	244	0	-10	N/A
203	-26	243	0	-16	N/A
204	-25.5	242	0	-19	N/A
205	-25	242	0	-10	N/A
206	-24.5	241	0	-13	N/A
207	-24	240	0	-14	N/A
208	-23.5	239	0	-16	N/A
209	-23	238	0	-16	N/A
210	-22.5	237	0	-16	N/A
211	-22	236	0	-15	N/A
212	-21.5	234	0	-22	N/A
213	-21	233	0	-20	N/A
214	-20.5	232	0	-17	N/A
215	-20	230	0	-22	N/A
216	-19.5	229	0	-19	N/A

Figure 321. Extended Gain Table Example

ADRV9001 EVALUATION SYSTEM

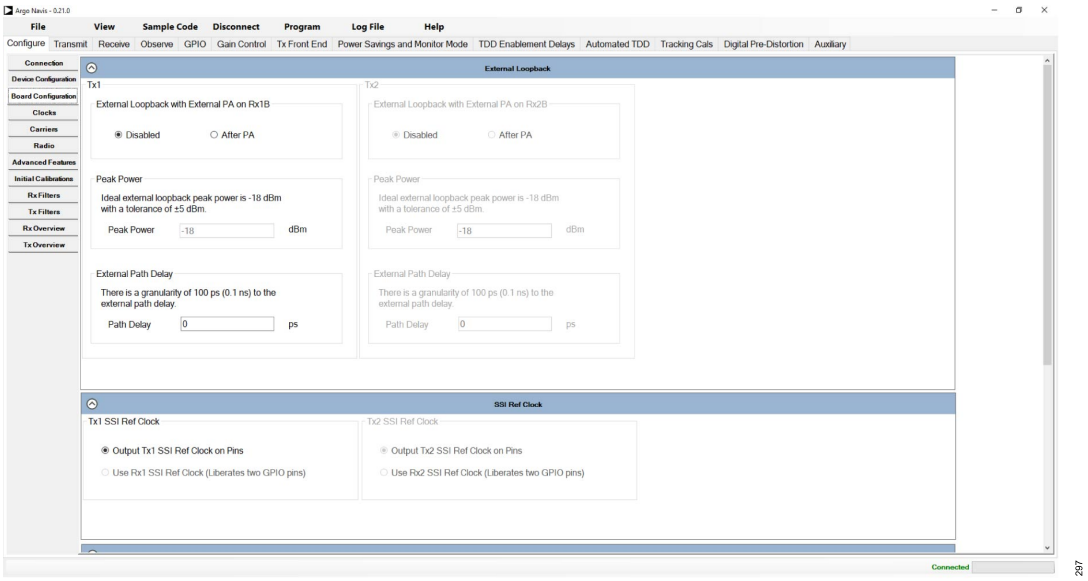


Figure 322. Board Configuration Tab

Clocks

The **Clocks** tab (Figure 323) provides access to the settings that determine the device clock configuration. This page allows to:

- Set the device clock.
  - Set the device clock frequency.
  - Set the divisor value applied to the frequency at DEV_CLK_OUT.
  - Enable/disable the DEV_CLK_OUT signal.
  - Select the clock PLL type to be either high performance or low-power. Note that low-power PLL supports only certain sampling rates. See the Clock Generation section for limitations.
  - Select the "Processor Clock Divisor" value from 1, 2, and 4. Lower clock rate saves power. Changing the "Processor Clock Divisor" value affects the whole system from changing the power-up time to tracking calibration times.

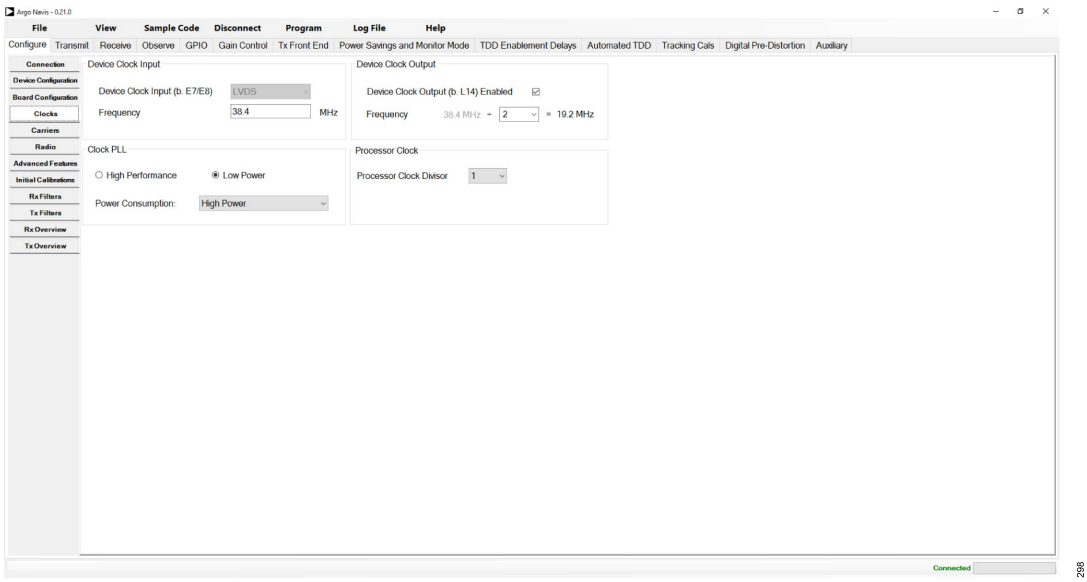


Figure 323. Clocks Configuration Tab

ADRV9001 EVALUATION SYSTEM

Carriers

The **Carriers** tab Figure 324 provides access to the settings that determine the device LO configuration. This page allows to:

1. **RX1 Port Switching** allows to set up port switching depending on the receiver LO frequency. In this section, set the maximum and minimum values to set up the switching for both the receiver channels. Note that this prevents the use of ORx channels.
2. **Define Carrier Frequencies** (default selection in the "Carrier Configuration Mode" dropdown).
    a. Configure the LO.
        1. Set **PLL Retuning** to allow or disallow PLL retuning when switching between the transmitter and receiver.
            a. When the transmitter and receiver are using the same LO but different frequency in the case of low IF mode, for example, when switching between the transmitter and receiver, retune the PLL to lock. If the transmitter and receiver are using different LOs, do not retune the PLL.
            b. In some configuration modes, such as TDD LTE, an "Antenna Diversity" checkbox appears. Use this checkbox to change the routing of the LOs from Rx1/Tx1 and Rx2/Tx2 to all on the same LO.
        2. Set the carrier frequency.
        3. Intermediate frequency is supported for RX. Enable it by ticking the NCO box. Recommended range is from 490 kHz to 20 MHz.
        4. Set the Rx1/Rx2/Tx1/Tx2 carrier source (internal or external, options vary depending on the selected setup).
        5. If external LO is used:
            a. Set the divisor value.
            b. The TES informs about the external LO frequency to provide to the ADRV9001 transceiver at the external LO input.
        6. If Internal LO is used, then select **Best Phase Noise** or **Best Power Saving** for the application. Note that the **Best Phase Noise** option supports only Sub-1 GHz transmitter frequencies. Configure PLL power consumption to low, medium or high power. This option changes some of the analog DC biases to reduce the VCO voltage swing, and as a result the power consumption. Set the PLL calibration to normal or fast mode, and also set the PLL bandwidth from 50 kHz to 1200 kHz.

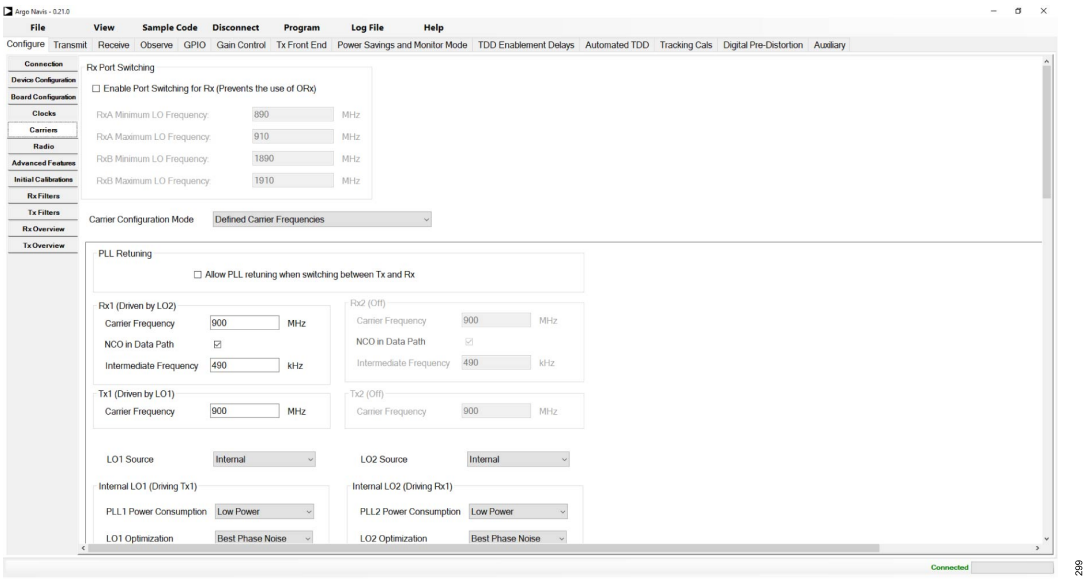


Figure 324. Carriers Configuration Tab

3. **Frequency Hopping** (select from the "Carrier Configuration Mode" dropdown).
    a. To use the frequency hopping settings in the TES, see the Frequency Hopping section.
    b. A prompt window pops up to use the "Frequency Hopping Wizard" on selecting this option. This wizard helps set up the FH configuration in a step-by-step manner.

ADRV9001 EVALUATION SYSTEM

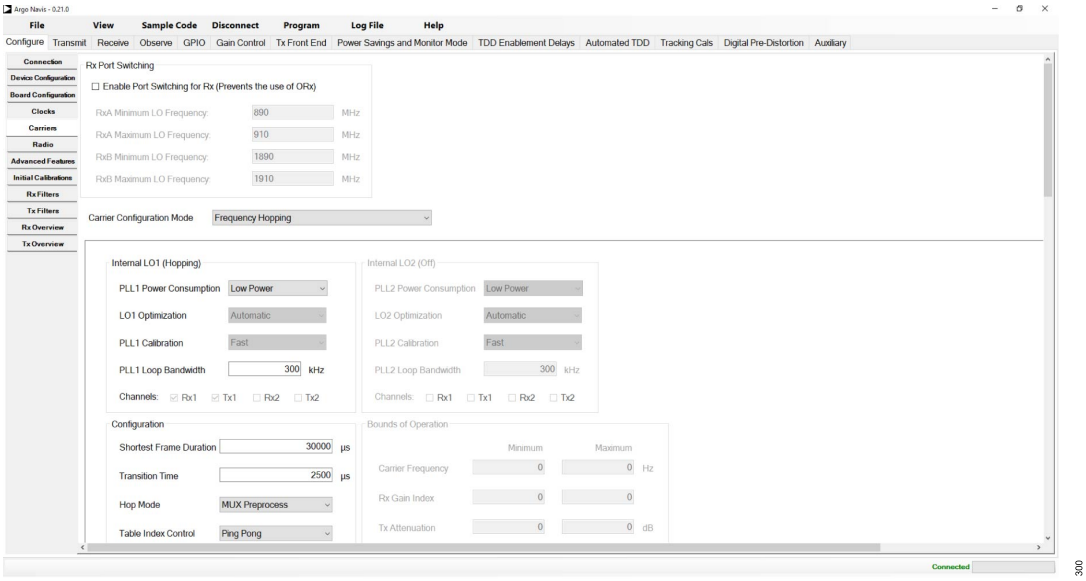


Figure 325. Carriers Configuration Tab (Frequency Hopping)

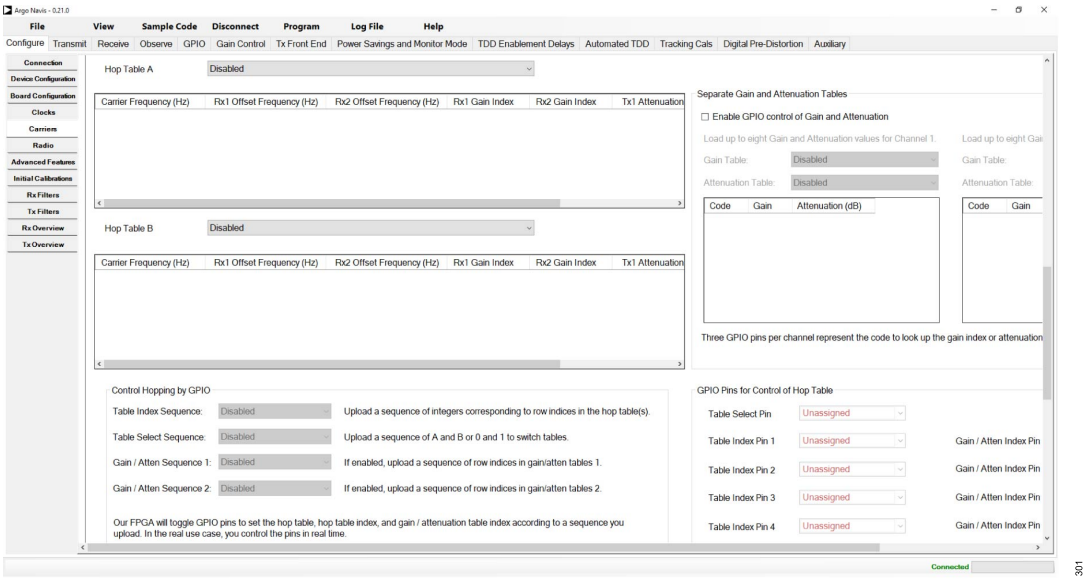


Figure 326. Carriers Configuration Tab (Frequency Hopping Tables)

Radio

The **Radio** tab (Figure 327) configures the channel enablement, and transmitter and receiver characteristics.

- Select the channel control mode (hardware enable signals or API command).
- Select the **HIGH**, **MED**, or **LOW** receiver ADC rate.
- Select the active receiver ADC from high-performance or low-power types.
- Determine the analog low-pass filter frequency response.
- Select the receiver frequency offset correction.
- Select the DAC, 3 dB boost mode.
- Select the transmitter frequency offset correction.
- Select the LPF power consumption for the transmitter and receiver.

ADRV9001 EVALUATION SYSTEM

- Use "Transmit Data Source" to send data from the FPGA or NCO internal signal source.

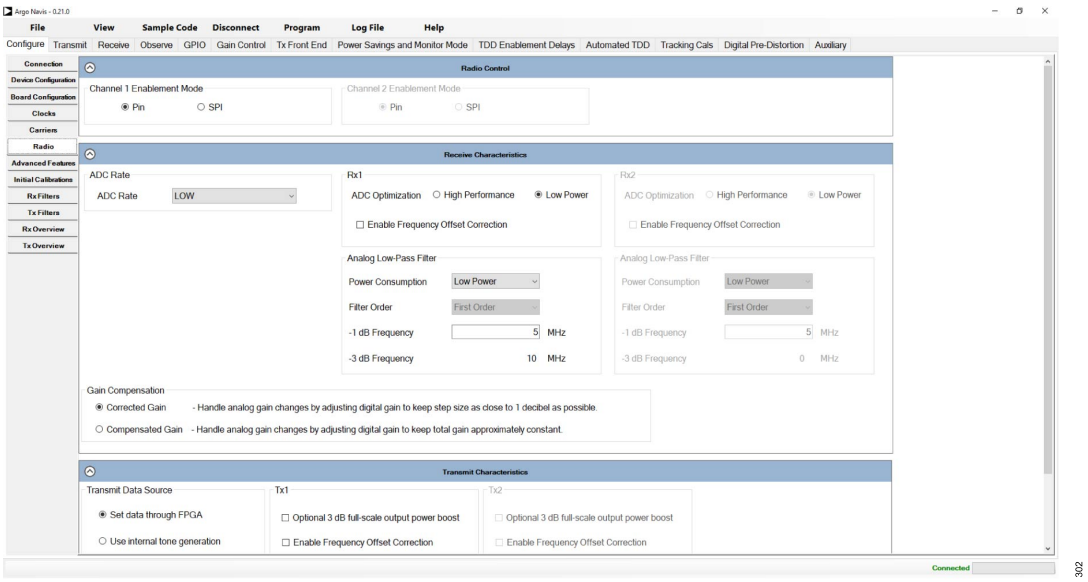


Figure 327. Radio Configuration Tab

Advanced Features

The **Advanced Features** tab (Figure 328) provides access to the settings that enable a set of advanced features the device supports. This page allows to:

- Multichip Sync
  - Select from "Enabled" and "Enabled with RF PLL Phase Sync" options.
  - Input sample and read delays for each channel in this section. For more details, see the Multichip Synchronization section.
- SSI Power-Down
  - Allows to power down either of the two SSI channels.
- Loopback
  - Internal loopback from the transmitter SSI or datapath to the receiver SSI or datapath.
- Enable or disable DPD.
  - Select the DPD tap polynomial terms.
    - There is a default configuration.
    - There is freedom to configure individual tabs.
- Enable or disable the closed-loop gain control (CLGC).
  - This enables the CLGC setting in the **Digital Pre-Distortion** tab.
- Monitor the mode RSSI configuration.
  - Set the measurement parameters for the monitor mode including the following:
    - Number of measurements to average
    - Measurement duration
    - Start period
    - Detection threshold
  - Sync Fast Buffer Read
    - In the monitor mode, when the device detects a signal, it waits to bring the Rx_EN pin high and it increases the interface rate X4 to push the data out of the buffer until the input matches the output and it reverts to the normal rate.
    - The device configuration must be in DMR, has frequency deviation enabled, and the ADCs are in high-performance operation.
- Stream Status Output over GPIO

ADRV9001 EVALUATION SYSTEM

- This can enable the stream status to be seen on the GPIOs to measure the rise-to-analog-on time.

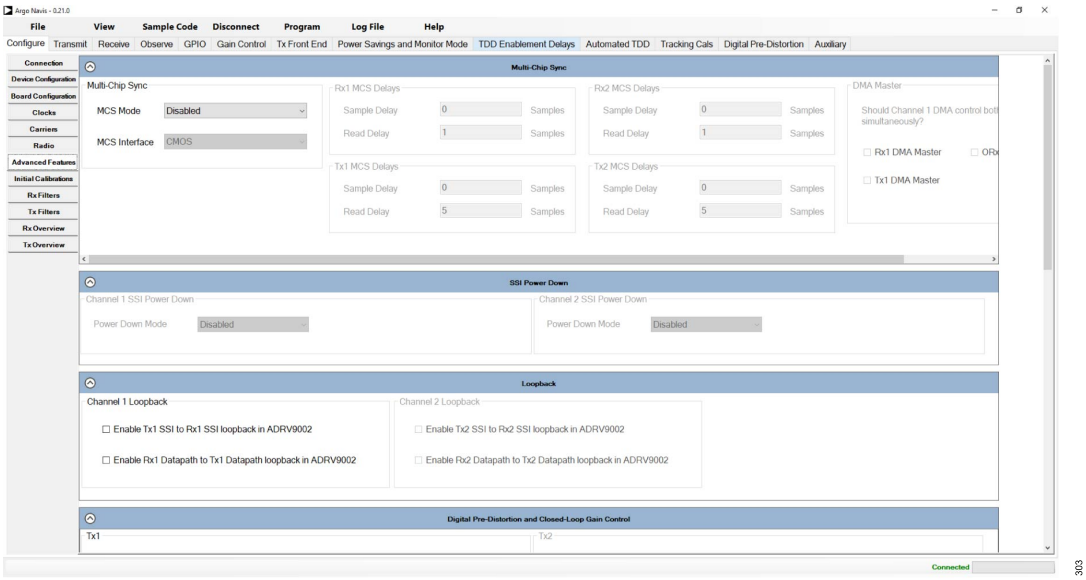


Figure 328. Advanced Features Tab

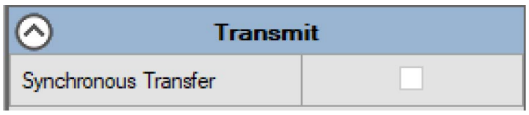


Figure 329. Synchronous Transfer Option in the Transmit Tab

Note that the evaluation software has the synchronous transfer option. This option allows to begin the transfer of data (through the FPGA DMA) on both channels (Tx1 and Tx2 or Rx1 and Rx2) at the same point in time. It tests the MCS to ensure that both channels have the same sample at the same point in time. Set the trigger source for the DMA on both channels to **adi_fpga9001_DmaTrigger_e.ADI_FPGA9001_DMA_TRIGGER_SYNC**. The FPGA generates a pulse (using the FPGA-generated MCS signal), which triggers simultaneous transfer on both the ports.

Initial Calibrations

The "Initial Calibrations" (Figure 330) shows a list of the calibrations that run during the initialization of the part. Unselect some of the calibrations for measurement comparisons. However, it is recommended to use all the calibrations for optimal performance.

Warm boot is used to store and reload initial calibration coefficients to speed up subsequent initializations. The Warm Boot section provides more details.

ADRV9001 EVALUATION SYSTEM

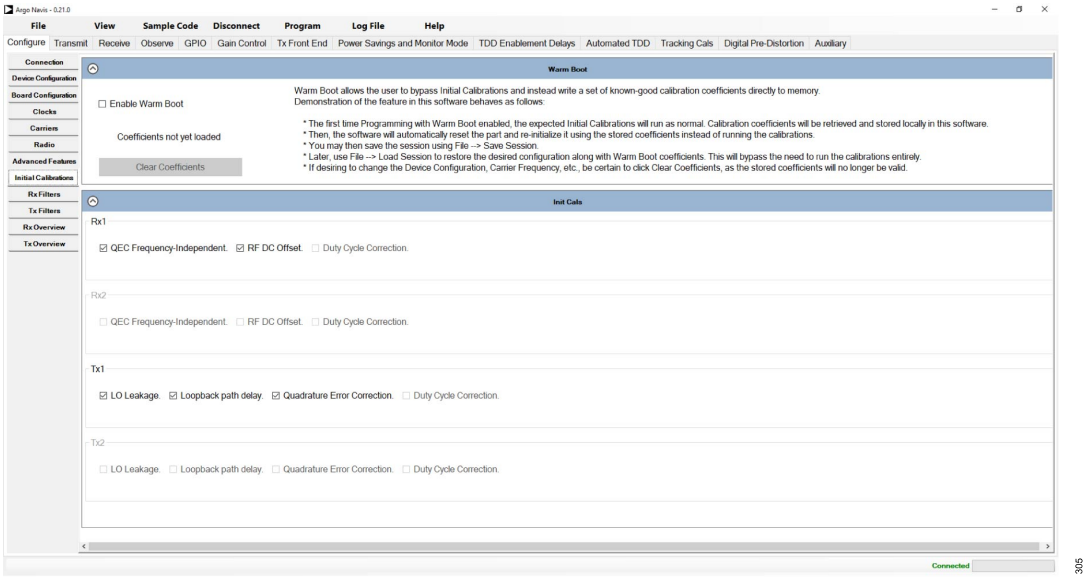


Figure 330. Initial Calibrations

Receiver and Transmitter Filters

The ADRV9001 evaluation software specifies the custom programmable filter for each channel. These filters are up to 128 taps. The custom filter must be in the .csv or .txt file format and the coefficients must be 24-bit signed integers with no carriage returns. The SDK has an example filter .txt file. There is also the option to bypass this filter entirely.

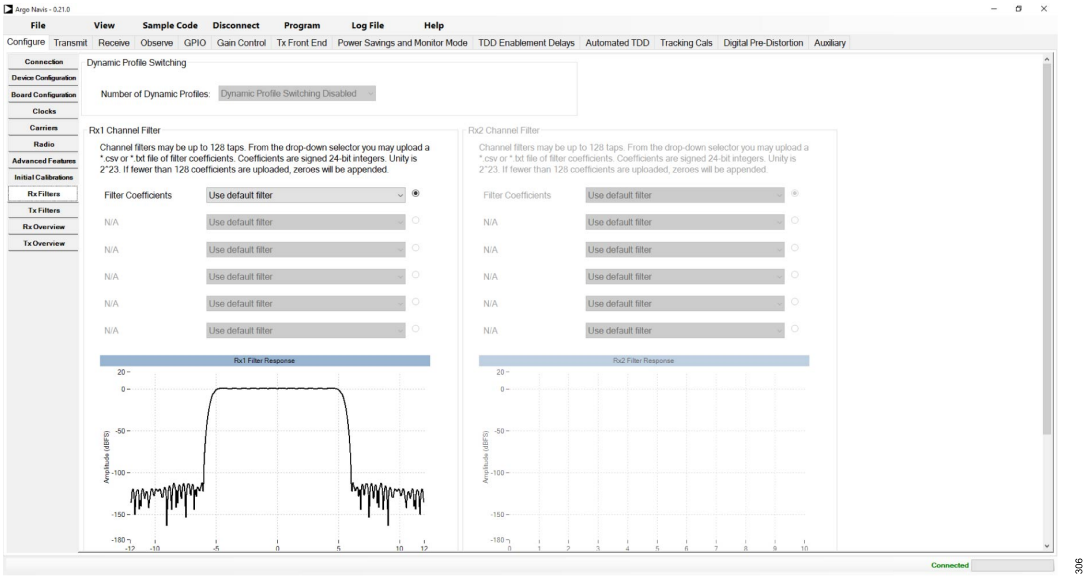
An important thing to note when programming the part is that the calibrations use the carrier bandwidth set by the user. If the bandwidth of the loaded filter is smaller than the bandwidth of the carrier, then the calibration fails. This is due to the signal being attenuated by the filter. If a narrower filter is required, it is best to use a wide enough filter when initializing and calibrating, then set the filter to the narrower profile.

The **Rx Filters** (Figure 331) and **Tx Filters** (Figure 332 and Figure 334) tabs allow to input programmable FIR filters on the chosen channels. In the Tx Filters tab, set the interpolations factor.

The **Rx Filter** tab also shows controls for the "Dynamic Profile Switching" (currently available only on LTE profiles). For more details, see the Dynamic Profile Switching (DPS) section.

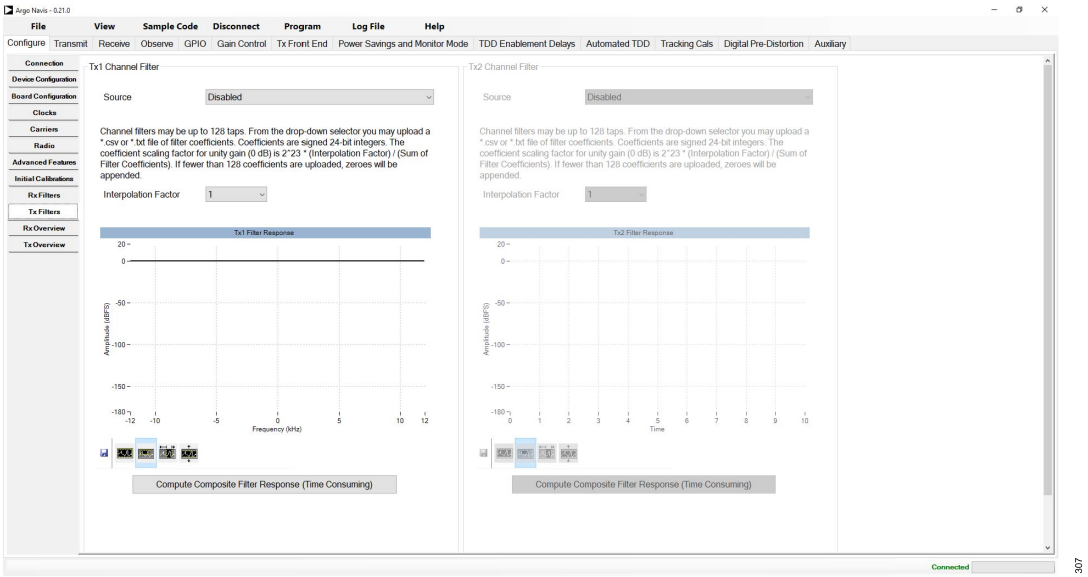
The compute composition filter response shows how the overall filter response is made. It takes some time to compute but gives more information on how the filter is made.

ADRV9001 EVALUATION SYSTEM



306

Figure 331. Rx Filters



307

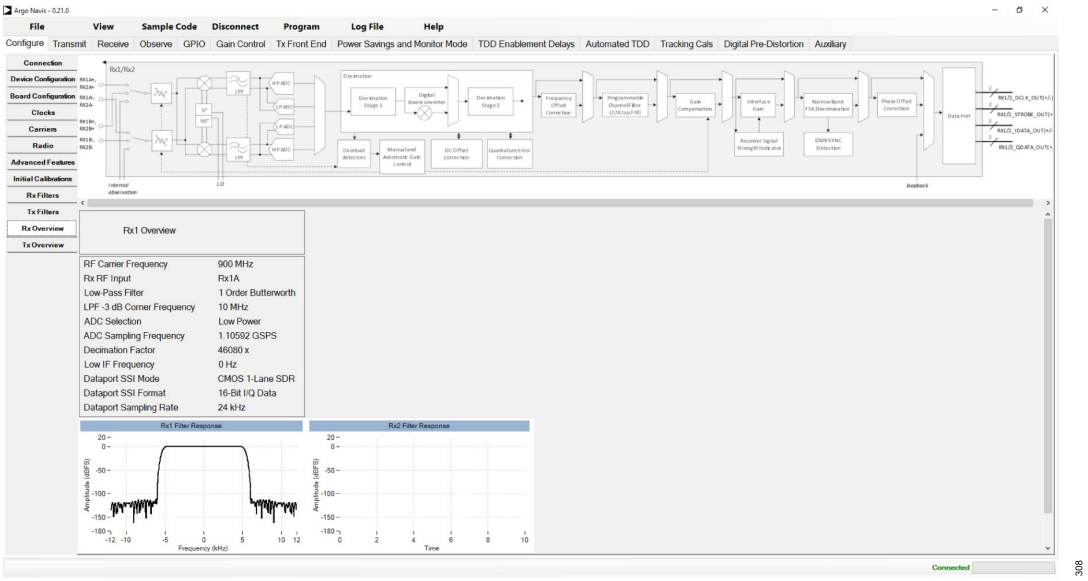
Figure 332. Tx Filters

Receiver and Transmitter Overview

The **Rx Overview** (Figure 333) and **Tx Overview** (Figure 334) tabs provide more details on the ADRV9001's selected mode of operation using the **Device Configuration** tab (Figure 319). Each tab provides the receiver and transmitter datapath overview diagrams. These tabs provide the readback of ADC/DAC sampling frequencies, analog filtering configuration, datapath sampling rate, data port format, mode of operation, and sampling rate.

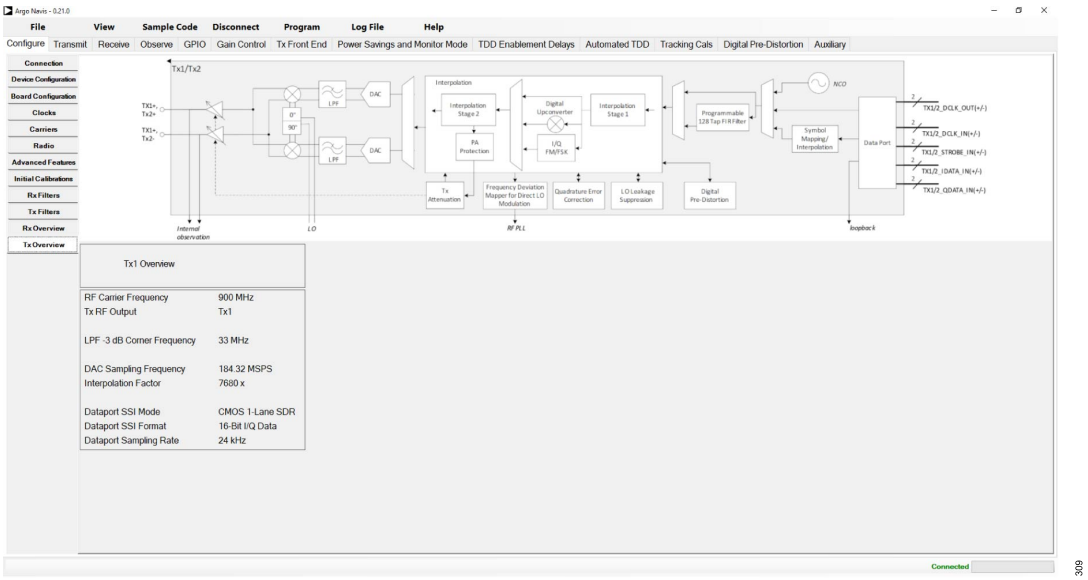
In the **Rx Overview** tab, it is also possible to readback the IF frequency and observe the pFIR channel filtering characteristics and their pass-band flatness. Quick zooming capability zooms the pass-band response using the mouse cursor as well as restores it to the full-scale plot. The TES also exports the data plotted on the graphs to an external file. Do this by right-clicking on the graph area and selecting "Export Data to File". Then save the data to the file for later analysis.

ADRV9001 EVALUATION SYSTEM



308

Figure 333. Rx Overview Tab



309

Figure 334. Tx Overview Tab

Transmitter Operation

Select the ***Transmit*** tab to open a page, as shown in Figure 335. The upper plot displays the fast fourier transform (FFT) of the digital data and the lower plot shows its time domain waveform. When multiple transmitter outputs are enabled, select the desired data to display in the "Spectrum" plot using the checkboxes below it.

Once the **Transmit** tab is open, do the following:

- Check the RF transmitter carrier frequency in MHz.
- Change the transmitter attenuation level in 0.05 dB steps.
- Continuous transfer selects by default and repeats the chosen transmitter signal on loop. To transmit just once, unselect this.
- Transmit the content of the selected file. The TES comes with some example files. Assuming the default TES installation process, the example files are located in **C:\Program Files (x86)\Analog Devices\ADRV9002 Transceiver Evaluation Software\Example**.
- Transmit a single-tone, two tones, and zeros. Adjust the digital power of the single- or dual-tone signal as well as the frequency offsets.

ADRV9001 EVALUATION SYSTEM

- There is a frequency offset correction option to change the frequency on the air without reprogramming the chip.

Pressing the play symbol moves the ADRV9001 to the transmit state and starts a process where selected data files for the Tx1 and Tx2 are sent to the ADRV9001. The data is then stored on the Xilinx platform motherboard RAM and the RAM pointer loops through the data continuously until **Stop** is pressed.

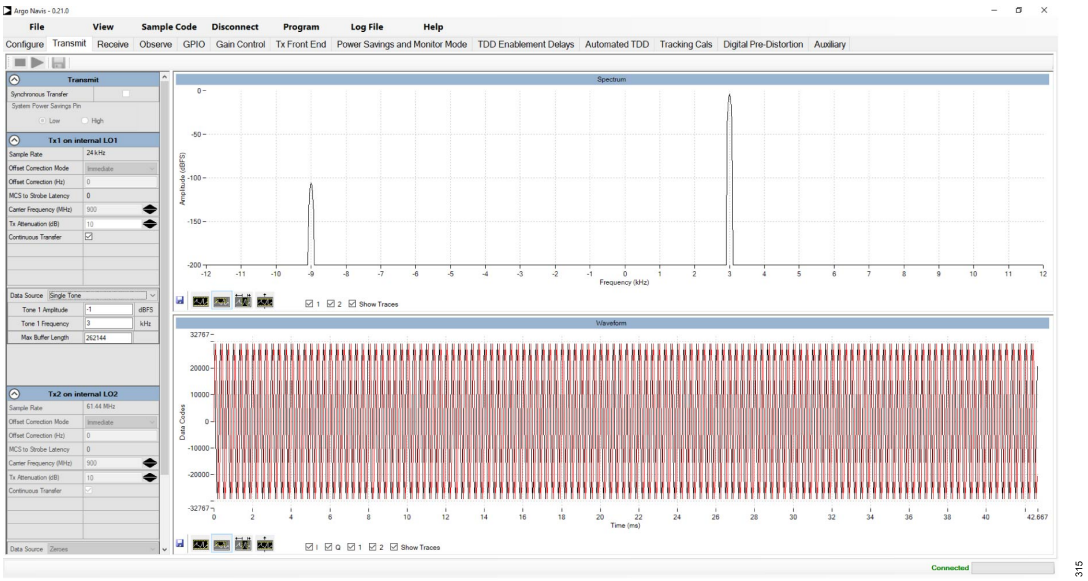


Figure 335. Transmit Data Tab

To use the transmit data file feature, save the transmit data in a file with the .txt or .csv extension. The data samples must be either complex (real and imaginary) or real-only, and must be Q1.15 fixed point integers. Note that in the "TX Direct Modulation" mode, the data sample must be Q4.12 fixed point integers. The data samples must have the following format.

If the data is only real, remove the imaginary column, with only one column of real samples.

For example, in the case of DMR FM/FSK direct modulation, use only real data samples, in which case, the data has only one column. The maximum number of samples that can be transmitted from the evaluation platform FPGA is 2097088.The length of the TX transmit data must be multiples of 64. The data file is played continuously. Therefore, the data is phase-continuous.

```
Real      | Imaginary
----------|-----------
I1        | Q1
I2        | Q2
I3        | Q3
I4        | Q4
.         | .
.         | .
```

Receiver Operation

The **Receive** tab opens a window, as shown in Figure 336. The upper plot displays the FFT of the received input data and the lower plot shows its time domain waveform. When the multiple receiver inputs are enabled, select the desired data to display in the "Spectrum" plot using the checkboxes below it.

In the TDD operation, the receiver data displays only when the enabled receiver is high. It does not display the data gap between the TDD time slots.

Once the **Receive** tab is open, do the following:

- Check the RF receiver carrier frequency in MHz.

ADRV9001 EVALUATION SYSTEM

- Change the capture length in number of samples.
- Change the receiver gain level (gain table index).
- Change the receiver interface gain (in four steps).
- Enable/disable the "Baseband DC Rejection" tracking calibration.
- Change the frequency offset in Hz.
- Read back the main parameters measured in received signals, such as the fundamental frequency, its amplitude, and DC offset.
- Plot and save the received data by clicking the floppy disk icon in the bottom left corner.
- Save the receiver-captured data (specified in the "Capture Length" window) as a *.csv or .tsv file.
  - There is an interleaving option pop-up window displayed when saving as a .tsv file.

Pressing the play symbols enables the selected receivers and displays the received data continuously until **Stop** is pressed.

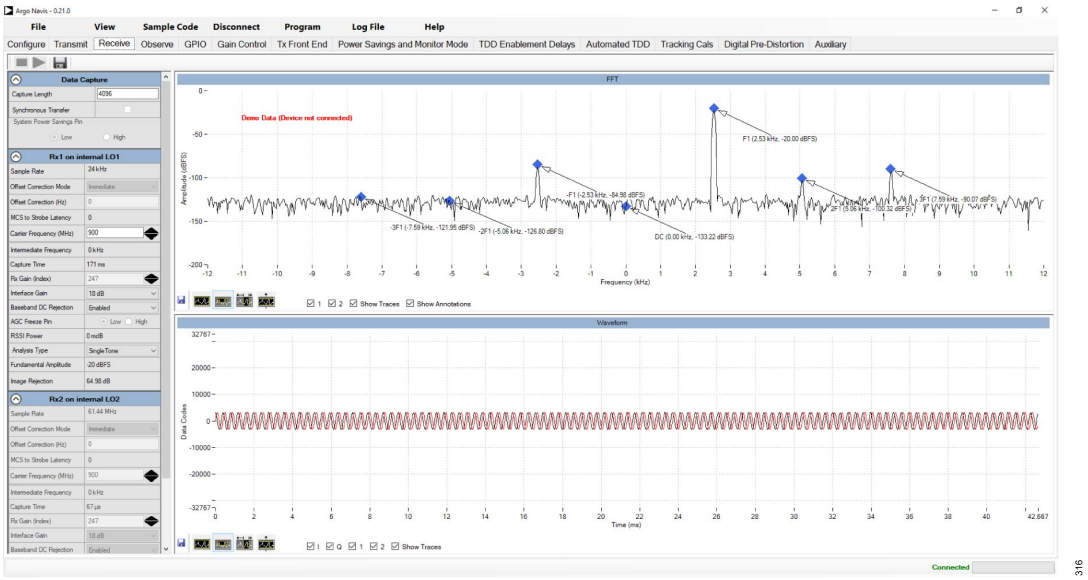


Figure 336. Receive Data Tab

If the receiver frequency deviation is enabled in the configuration step, the receiver input signal is demodulated. For example, if a continuous wave of 900.003 MHz is sent to the RX port with an LO of 900 MHz, there is a tone of 3 kHz offset from LO in the baseband, and on the RX tab it is expected to see a constant of 3000 in the time domain plot.

See Figure 337 for options on the captured data format.

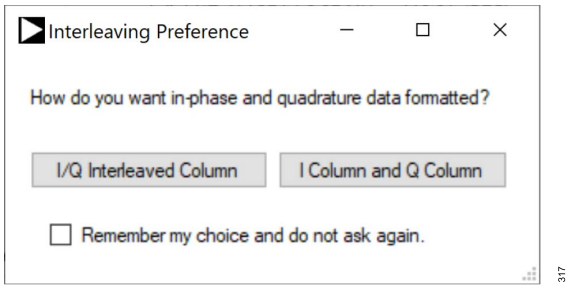


Figure 337. Interleaving Option

Save the receiver-captured data using **Save** next to **Play**. Save the data in either the .txt or .csv format. Each column corresponds to one channel. The data samples follow the Q1.15 fixed point format upon selecting the interleaved option. The following shows this.

```
Channel 1 | Channel 2
----------|-----------
```

ADRV9001 EVALUATION SYSTEM

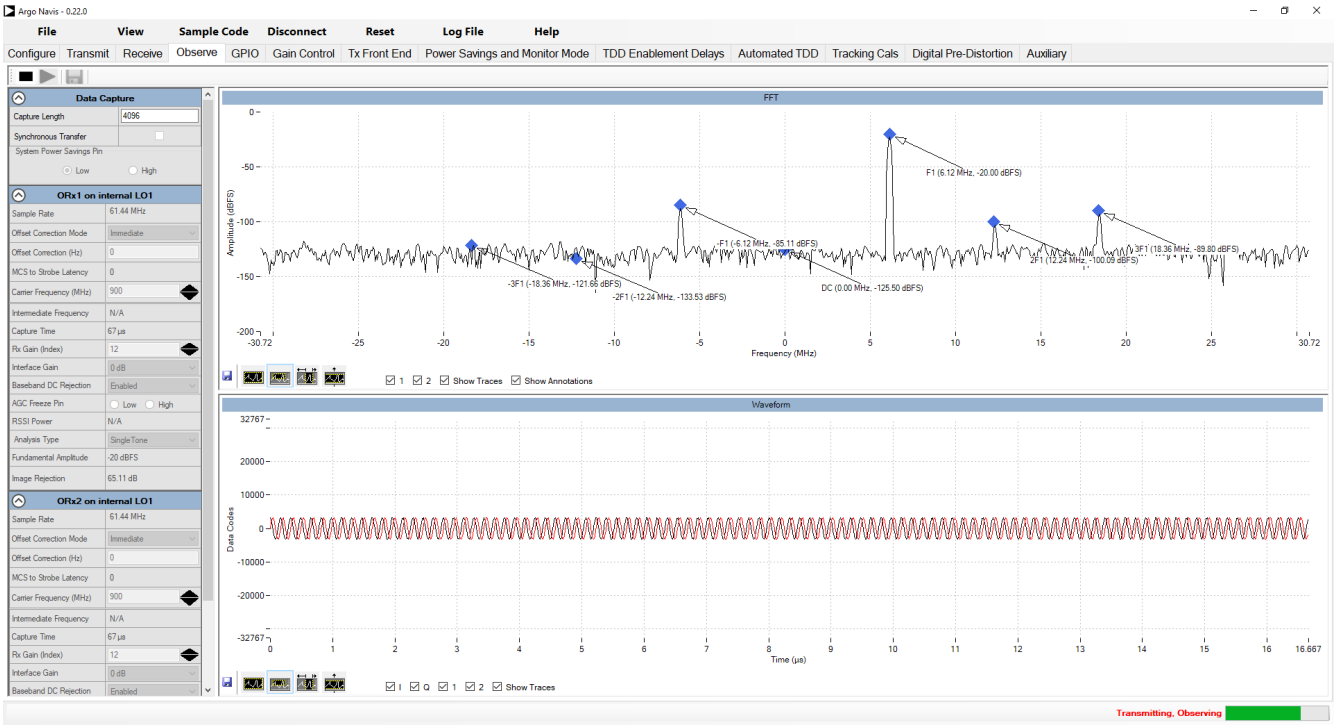
```
I1          | I1
Q1          | Q1
I2          | I2
Q2          | Q2
.           | .
.           | .
.           | .

When the I and Q column option is selected the data is formatted as follows:

Channel 1I | Channel 1Q
-----------|-----------
I1          | Q1
I2          | Q2
I3          | Q3
I4          | Q4
.           | .
.           | .
.           | .
```

For receiver frequency deviation, only I samples are shown and all Q samples are 0. The maximum number of samples that can be captured on the evaluation platform FPGA is 2097088.

Observer Operation

The **Observe** tab opens a window, as shown in Figure 338. The upper plot displays the FFT of the received input data and the lower plot shows its time domain waveform. This tab operates same as the **Receive** tab. It is used to observe the transmitted signals.

Enable this tab by selecting the "I/Q Signal Type" in the **Device Configuration** tab when setting up the device parameters. Start transmitting from the **Transmit** tab before pressing **Play** in the **Observe** tab.



410

Figure 338. Observe Tab

ADRV9001 EVALUATION SYSTEM

GPIO Configuration

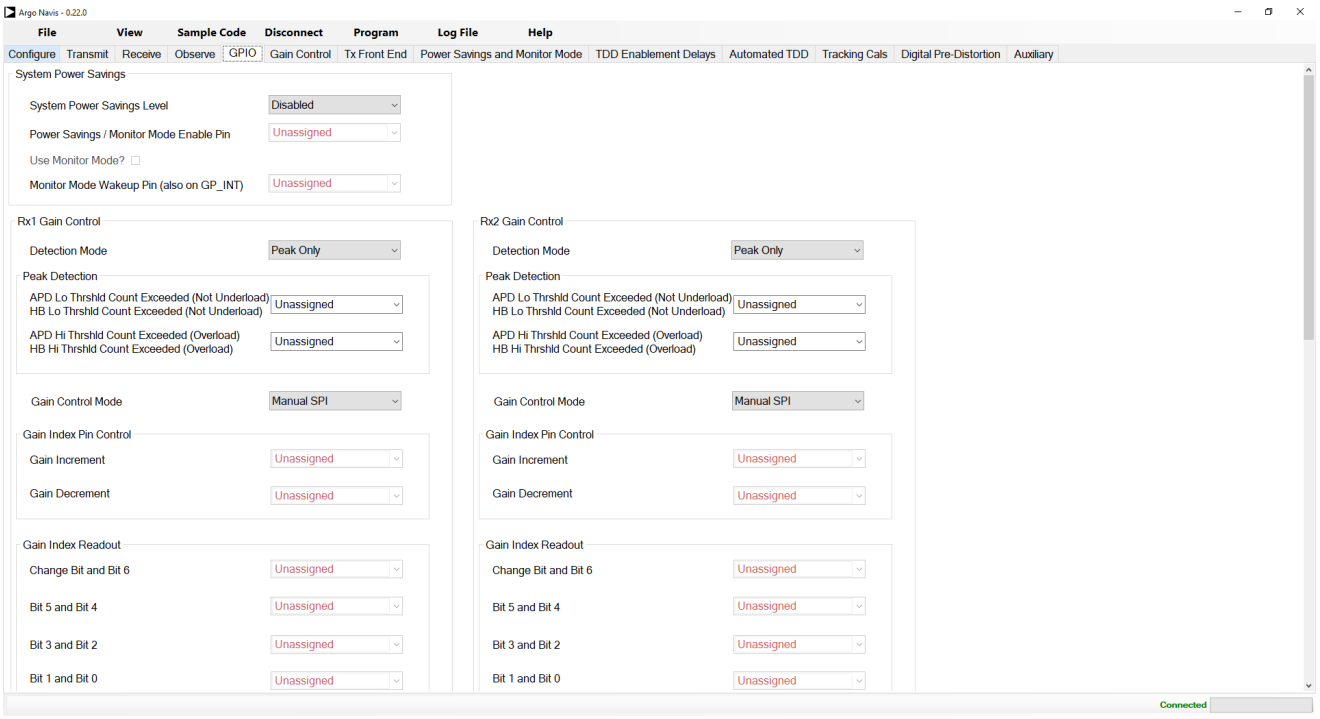
Use this tab to configure the various GPIO pins for multiple functions, such as power savings, attenuation control, gain control, and hop control pins.

Set the **System Power Savings** in this tab to configure the monitor mode saving level to apply to the part. This section also allows to assign the enable pin for the power saving mode.

Use the DGPIOs on the evaluation board from DGPIO_0 to DGPIO_11 for feedback signals as well as to set the gain index for **Rx Gain Control**.

Use the **Tx Attenuation Control** to assign the pins to increment and decrement the gain using the pin mode.

Configure the GPIO pins to control the hop tables, table index, and gain/attenuation pins.



411

Figure 339. GPIO Tab

Receiver Gain Control

The receiver **Gain Control** tab (Figure 340) configures the receiver gain control mode per channel. Apply the configuration selected in that tab to the ADRV9001 during initialization. During run time, change the interface gain as well as if the manual mode is enabled, then change the receiver gain.

Interface Gain selects MSBs or LSBs 16 bits from the data bus. This operation can be interpreted as a signal gain. In TDD operation, there is the option to update the interface gain **Now** or in the **Next Frame**. For more details, see the Receiver Gain Control section.

Select the **Manual** radio button in the **Gain Control Mode** to select the initial gain value. Change the receiver gain dynamically during the receiver capture operation.

Select the **Automatic** radio button in the **Gain Control Mode** to configure the basic ADRV9001 internal AGC parameters. The AGC becomes operational and automatically adjusts the receiver gain level when the ADRV9001 starts to receive data in the **Receive** tab. See the Receiver Gain Control section for more information about the AGC operation.

Select Digital **Correction** or Digital **Compensation** for the **Digital Gain** operation.

ADRV9001 EVALUATION SYSTEM

- Compensation: It is the process of compensating for the analog attenuation in the device (prior to the ADC) with a corresponding amount of digital gain before sending the digital signal to the user. Gain compensation uses the digital gain to effectively undo the analog gain so that the recipient signal from the receiver data stays constant. The digital gain effectively compensates for the analog attenuation.
- Correction: It is the process of correction that uses the digital gain to make the gain steps more accurate. This ensures that the receiver gain steps are accurate.

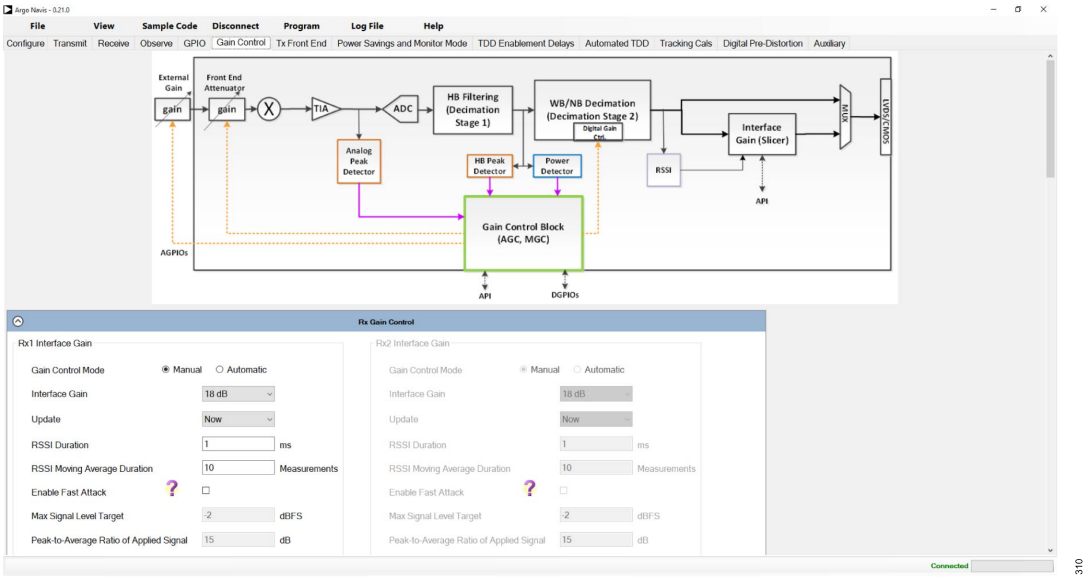


Figure 340. Gain Control Tab

Transmitter Front End

Transmitter Attenuation

- Use the DGPIO pins for transmitter attenuation control. Assign the DGPIO pins for attenuation increment and decrement. Specify the step size in the **Attenuation Control** tab. Set the default step size to 0.05 dB.

Transmitter PA Ramp

- Use one of the AuxDACs to output a ramp to control an external power amplifier.

ADRV9001 EVALUATION SYSTEM

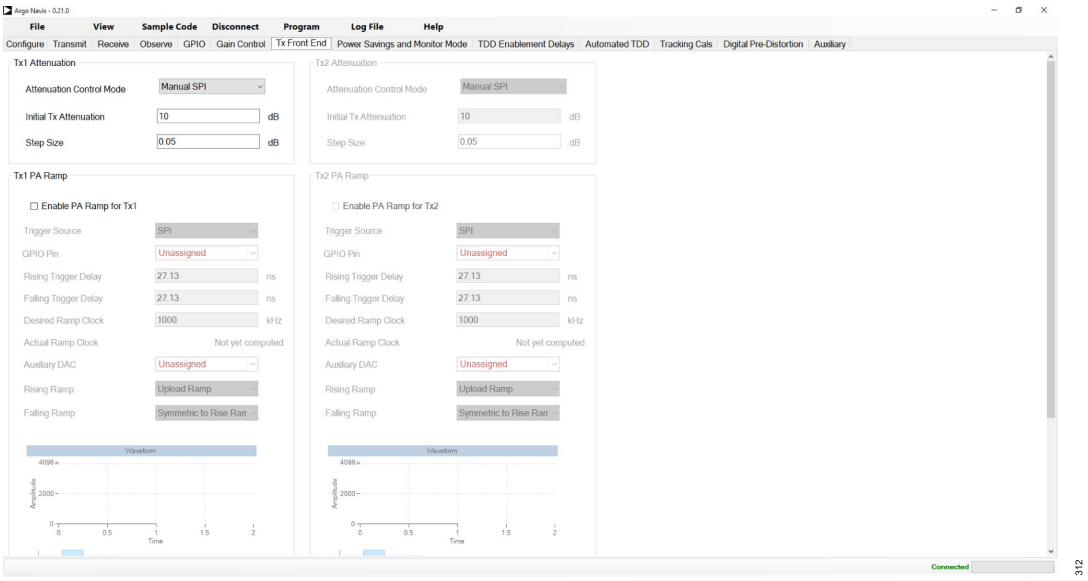


Figure 341. Transmitter Front End

Once the pins are assigned, go to the **Transmit** tab and start normal playback. Then adjust the transmitter attenuation level using the up and down arrows, and this adjusts the transmitter attenuation value by the specified step size.

Power Savings and Monitor Mode

Specify the certain power-saving mode in this tab (Figure 343). The power-saving modes are divided into two categories: system power savings (SPS) and channel power savings (CPS). The SPS category includes CLKPLL, LDO, and ARM power-down. Control these through the DGPIO pins. The CPS category includes RF PLL and LDO power-down. Control these through the DGPIO pins as well as the transmitter/receiver enables.

Enable the monitor mode by setting the "Monitor Mode Wakeup" pin. **View-->Power Savings** brings up the monitor mode window. If the "Monitor Mode Wakeup" pin is unassigned, the monitor mode is not enabled. Also, note to dock this window and all pop-up windows to the main GUI by clicking **Dock** in the menu bar of the pop-up. Note that "RSSI" can be used for any profile and "DMR Sync Detection" is only for DRM profiles, and requires the high performance ADC to be enabled along with a "Frequency Deviation" signal type selected in the **Device Configuration** tab.

Select the "Use Monitor Mode" checkbox to set up timing for the monitor mode pin to go high, giving the BBIC more time to see the pin change. Without ticking this checkbox, the pin just toggles.

ADRV9001 EVALUATION SYSTEM

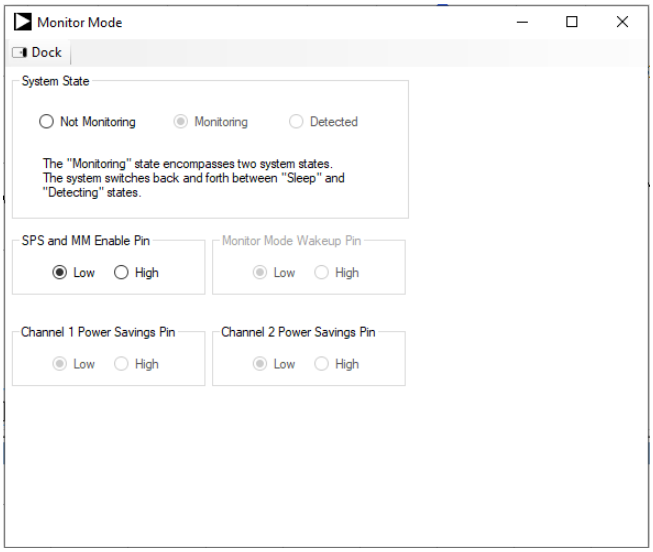


Figure 342. Monitor Mode Window

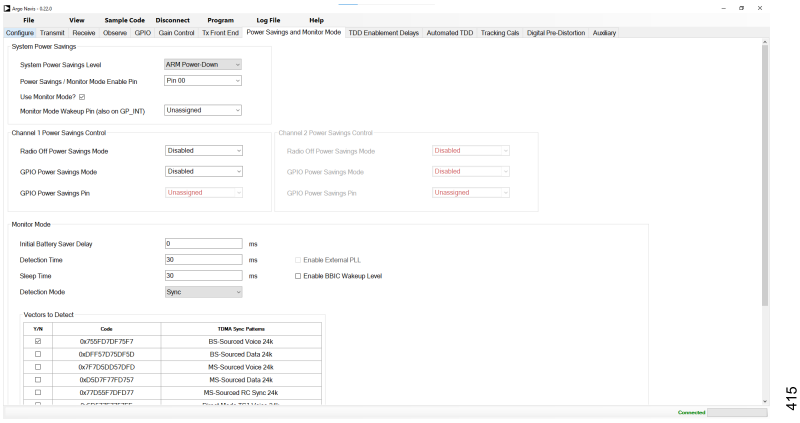


Figure 343. Power Savings and Monitor Mode

TDD Enablement Delays

For more detailed information on TDD enablement delays, see the Timing Parameters Control section.

Automated Time Division Duplexing

The ADRV9001 supports the automatic time division duplexing (TDD) operation. Send and receive TDD-framed data by configuring this tab (Figure 344). This of course depends on how the system and setup is selected. The ADRV9001 comes with predefined timing configurations by default. However, configure the timing as needed.

In the **Automated TDD** tab, configure the parameters using TDD configuration files:

- Frame and Sequences
- Specify the duration of a frame.
- Select from sequencing of frames.
- Specify the number of frames in a sequence.

Select the "Enable Automated TDD" checkbox. A prompt displays a timing diagram. This diagram updates every time one of the parameters in the TDD table changes. This timing diagram, as shown in Figure 345, is based off the TDD table in Figure 344.

ADRV9001 EVALUATION SYSTEM

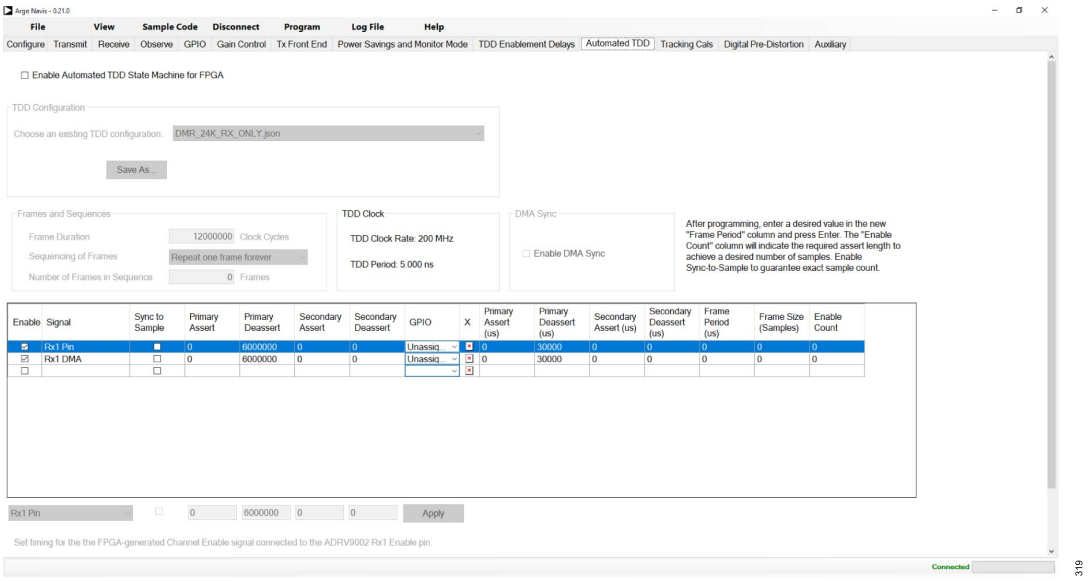


Figure 344. Automated TDD Configuration Tab

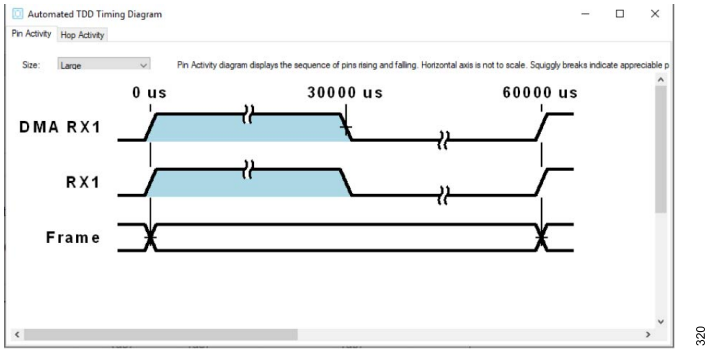


Figure 345. Automated TDD Timing Diagram

TDD Parameter Table

The table auto populates the TES based on the chosen configuration file.

- Enable Column
  - Enable/disable the receiver/transmitter channel.
- Signal Column
  - This displays the signal name attributed to that row.
- Frame Timing Columns
  - The TES generates a predefined timing based on the selected profile.
  - Modify the timing by entering **Primary Assert/Deassert**, **Secondary Assert/Deassert**.
  - Assert/deassert entries are frame locations and these are not durations. For example, if RX1 primary assert is 0 and primary deassert is 10000 µs, within the specified frame, the RX1 enable is "on" from 0 to 10000 µs and "off" for the rest of the frame.
  - The Figure 346 shows visually what primary/secondary assert/deassert mean. Black and gray indicates transmitter and receiver subframe data.
- Signal Input
  - Change the values of a signal already in the table or add a new signal to the table. Do this by selecting the row to edit in the table, filling in the parameters in the row underneath the table and clicking **Apply**.
  - To add a new row, select the blank row at the bottom of the table and click **Apply** with the parameters for the new row.

ADRV9001 EVALUATION SYSTEM

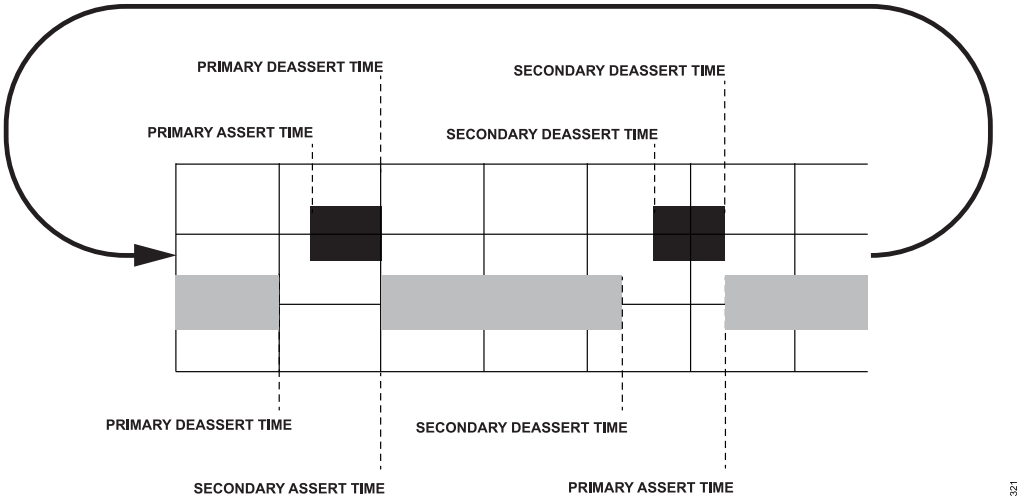


Figure 346. TDD Frame Timing Illustration

Table 131. TDD Signals

Signal	Description
RX1 Pin TX1 Pin RX2 Pin TX2 Pin	These signals are hardwired to the RX/TX ENABLE pins (used as SETUP signals in FH).
ORX1 Pin ORX2 Pin	Use as the ORX enable signal when routed to the GPIO assigned as ORX control.
RX1 DMA RX2 DMA TX1 DMA TX2 DMA ORX1 DMA ORX2 DMA	The DMA enables the gate data transfer for each channel.
RX1 DMA Trigger RX2 DMA Trigger TX1 DMA Trigger TX2 DMA Trigger ORX1 DMA Trigger ORX2 DMA Trigger	Configure these signals as triggers for the DMAs to signify that data transfer on the channel must only occur when the trigger signal has pulsed high followed by the DMA signal for the channel going high.
SMA1 Trigger	Hardwired to dedicated SMA (J67) on the ZC706 board, use this signal to trigger the external equipment. No option on the ZCU102 currently.
SMA2 Trigger	Hardwired to dedicated SMA (J68) on the ZC706 board, use this signal to trigger the external equipment. No option on the ZCU102 currently.
General-Purpose 1/ Hop Pin	General purpose signal that can be routed to GPIO pins. Also used as the Hop Pin in the FH mode.
General-Purpose 2 to 6	General purpose signals that can be routed to GPIO pins as needed.

Figure 344 shows a loaded sample JSON file with DMR settings. The parameters appear in the table for the receiver enable (Rx1 Pin) and DMA signals. In this example, the frame length is 12000000 clock cycles (60 ms). Both signals' primary assert happens at the beginning of the frame and then drops on the 6000000 clock cycle (30 ms). The frame then repeats itself on selecting the Repeat one frame forever.

Enabling the Tx1 DMA sends data from the FPGA to SSI. Disabling the Tx1 DMA stops sending data from the FPGA to SSI. It works together with the Tx_interface enabling/disabling (accepting data from SSI at the ADRV9001) to provide more flexibility to control what data to transmit. For example, there are four different scenarios:

- DMA disabled, Tx_interface disabled: nothing is transmitted.

ADRV9001 EVALUATION SYSTEM

- DMA disabled, Tx_interface enabled: 0s are transmitted.
- DMA enabled, Tx_interface disabled: data in DMA is not transmitted and is lost.
- DMA enabled, Tx_interface enabled: data in DMA is transmitted.

The following enum defines the Tx1 DMA trigger.

```
typedef enum adi_fpga9001_DmaTrigger
{
    ADI_FPGA9001_DMA_TRIGGER_SMA_1      = 0,
    ADI_FPGA9001_DMA_TRIGGER_SMA_2      = 1,
    ADI_FPGA9001_DMA_TRIGGER_MCS        = 2,
    ADI_FPGA9001_DMA_TRIGGER_GPIO       = 3,
    ADI_FPGA9001_DMA_TRIGGER_TDD_ENABLE = 4,
    ADI_FPGA9001_DMA_TRIGGER_IMMEDIATE  = 5
} adi_fpga9001_DmaTrigger_e;
```

- If using **ADI_FPGA9001_DMA_TRIGGER_IMMEDIATE**, the DMA transfers data whenever the DMA_Enable goes high (and has valid data from the SSI).
- If using **ADI_FPGA9001_DMA_TRIGGER_TDD_ENABLE**, the adi_fpga9001_TddConfig_t->**ADI_FPGA9001_TDDSELECT_RX1_DMA_TRIG** signal is the trigger. When this signal goes high (programmed in the TDD config), the DMA is triggered and transfers data whenever the DMA_Enable goes high (and has valid data from the SSI).
- If using **ADI_FPGA9001_DMA_TRIGGER_GPIO/MCS/SMA_1/2**, the **GPIO/MCS/SMA** signal is the trigger. When this signal goes high, the DMA is triggered and transfers data whenever the DMA_Enable goes high (and has valid data from the SSI).

The latest release uses the **ADI_FPGA9001_DMA_TRIGGER_IMMEDIATE**.

Tracking Calibrations

Enable/disable tracking calibration algorithms in the **tracking cals** tab. Certain algorithms can only be enabled in certain profiles. For example, RX harmonic distortion can only be enabled if configured in the DMR, Analog FM, and Tetra profiles. It greys out and is disabled in the LTE and custom profiles.

Enable and disable an individual tracking algorithm to observe its effect while transmitting and receiving a signal.

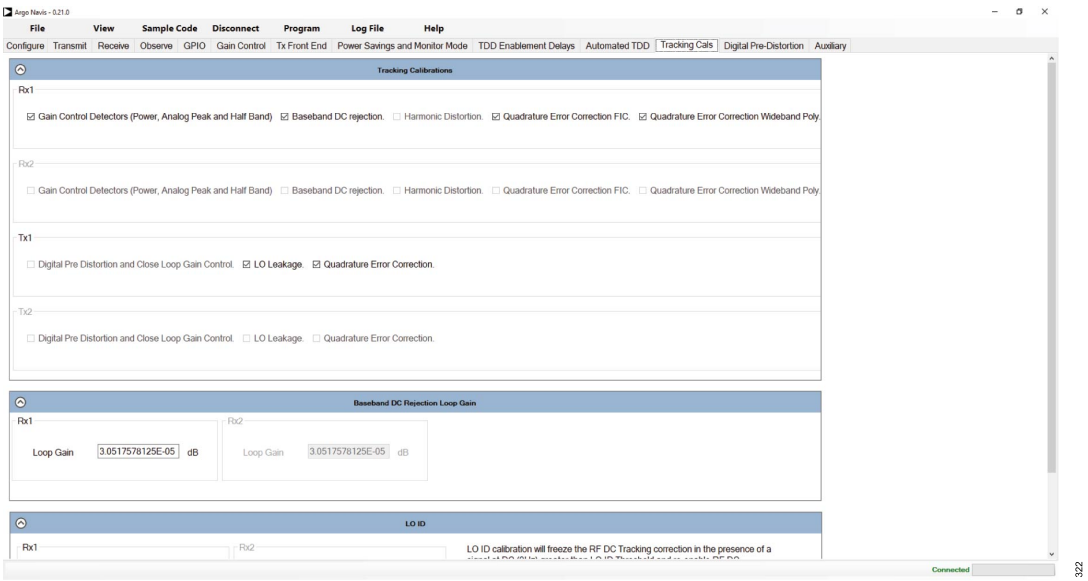


Figure 347. Tracking Calibration Tab

ADRV9001 EVALUATION SYSTEM

Digital Predistortion

For more detailed information on DPD, see the Digital Predistortion (DPD) section. This tab also contains the CLGC settings. It uses the same processing engine as the DPD and keeps the gain from the output of the power amplifier at a constant level.

To use the CLGC, use it in the open-loop configuration at first and read the "CLGC Last Gain" and "CLGC Filtered Gain" from the **Transmit** tab. Then input these readings into the "CLGC Gain Target" and "CLGC Filter Alpha" fields in the **Digital Predistortion** tab, and close the loop by unchecking the open-loop tickbox. The "CLGC Target Gain (dB)'" field controls the transmitter output power in the **Transmit** tab.

To use either of these functions, turn on the DPD tracking calibration as the two functions use the same tracking engine.

Auxiliary DAC/ADC

The ADRV9001 evaluation software allows to set the Auxiliary ADC/DAC for different control or monitoring purposes. Go to the **Auxiliary** tab and enable "Aux DAC/ADC". For Aux DACs, specify a DAC code, valued from 0 ~ 4095. This effectively sets the voltage level for that Aux DAC pin. For Aux ADCs, press **Capture Again** to observe the one-time voltage value on the Aux DAC pin. Enable this after the part is programmed.

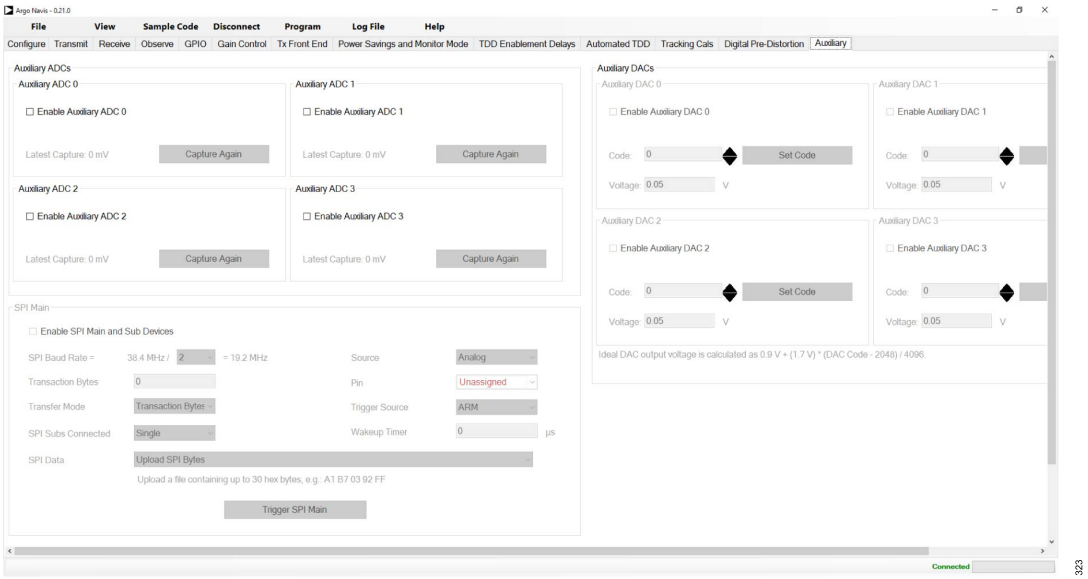


Figure 348. Auxiliary DACs and ADCs

File Menu

The **File** menu has the following options:

- **Save Session** and **Load Session**, which save and restore the TES configuration parameters.
- **Reset Preferences** resets the relative links to folders from the TES.
- **Generate Profile File** creates the JSON type profile configuration file.
- **Generate Stream Image** generates a .bin file of the stream images.
- **Force Update Platform** forcibly cleans up the resources in the SD card in case of errors.
- **Force Update Boot** cleans the boot files and reboots the platform.
- **Shut Down Platform** safely powers down the Xilinx platform.
- **Reboot Platform** reboots the platform.
- **Exit** exits the TES.

In case of some erroneous operation, the TES is capable of capturing its state with the *Log File* functionality that captures the steps that lead to error operation. Send the file created using the *Log File* function back to the Analog Devices support team for further debug.

ADRV9001 EVALUATION SYSTEM

Programming the Evaluation System

After configuring all tabs, press ***Program***. This starts the programming and initialization of an evaluation hardware. The TES sends a series of API commands executed by a dedicated Linux application that runs on the Xilinx platform. See a progress bar at the bottom of the window. When programming is complete, the system is ready to operate.

View Menu

IronPython Scripting

IronPython is an implementation of the Python programming language targeting the .NET Framework. The **IronPython** editor is in the **View** menu and allows the use of IronPython to write a unique sequence of events and then execute them using the ADRV9001 evaluation system.

For the **IronPython** scripting tab to operate, download the Iron Python 2.7 environment. Download the latest version from the Iron Python website (https://ironpython.net/). After installing IronPython, convey the installation library path to the TES. To set this, open the IronPython editor, then select **File** and select **Set IronPython path**. For the default Iron Python installation, set this path to **C:\Program Files (x86)\IronPython 2.7\Lib.**

Figure 349 shows the IronPython editor after executing the **File > New function** in the IronPython Script tab. The top portion of the window contains IronPython script commands, whereas the bottom portion displays the script output.

To use this tab, follow these steps:

1. Scroll to the bottom of the file with the text **#### YOUR CODE GOES HERE ####**.
2. This editor brings up suggestions of all the API functions available upon typing “Adrv9001”.
3. Find the information on the parameters for different API calls in the oxygen file (*ADRV9001_API.chm*) in the SDK.
4. Go to **IronPython**, select **Build**, and then select **Run**. This function executes the IronPython script open in the currently active script tab using the ADRV9001 evaluation hardware. The bottom side of the Iron Python script tab displays the script output.

For this example, the transmitter attenuation for the selected channel changes.

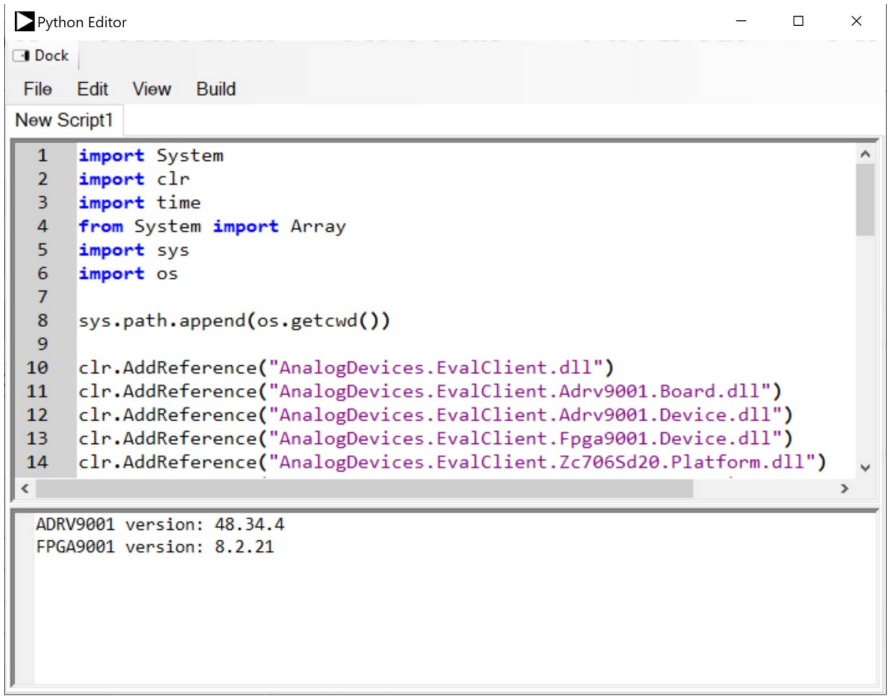


Figure 349. IronPython Scripting Window

The SDK has examples of IronPython scripts to run API functions that do not appear in the GUI. Find these files in the ***ADRV9001 Transceiver Evaluation Software\IronPython*** folder, and load these through the ***File*** menu and ***Load***.

ADRV9001 EVALUATION SYSTEM

External Path Delay Measurement (for DPD)

Connect the Tx1 output, through the chosen power amplifier, to the Rx1B input with a cable (add an optional step attenuator to the loop). Configure the TES to indicate there is an external path after the power amplifier in the **Board Configuration** tab. Enable the DPD from the **Advanced Features** tab. Run the script immediately after programming to make sure the device is in the “calibrated” state. Check and change the state of the part from the **View** menu under "Radio State".

This script runs the **adi_adrv9001_cals_ExternalPathDelay_Calibrate()** "cal" to get the path delay. It then sets the path delay value and checks it to make sure it is written correctly. The "Delay" displays in the "Output" window in "'ps". Run this script without the power amplifier using just a cable or cable and attenuator to test the script. For proper use case results, use the power amplifier.

Driver Debugger

A driver debugger is available from the **View** menu. It is a live window that captures all the driver calls used by the TES. It can be used to debug issues or understand the needed driver calls.

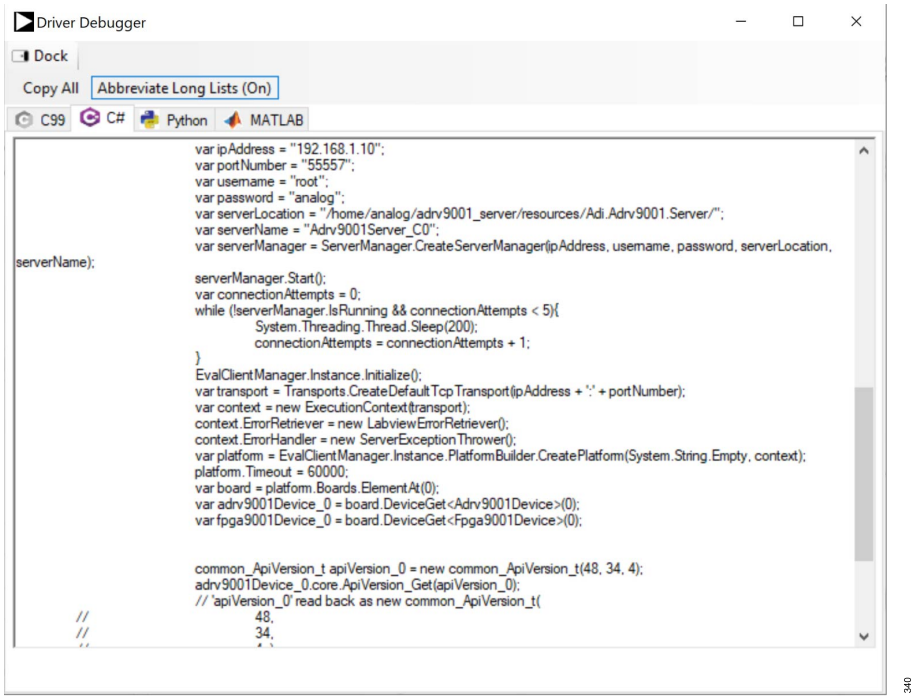


Figure 350. Driver Debugger Window

Note that all of these **View** menu pop-ups can be docked to the main GUI using **Dock** in the menu bar of the pop-up. Dock multiple pop-ups at once on the GUI, and they appear in a tabbed format on the right side of the TES, as shown in Figure 351.

ADRV9001 EVALUATION SYSTEM

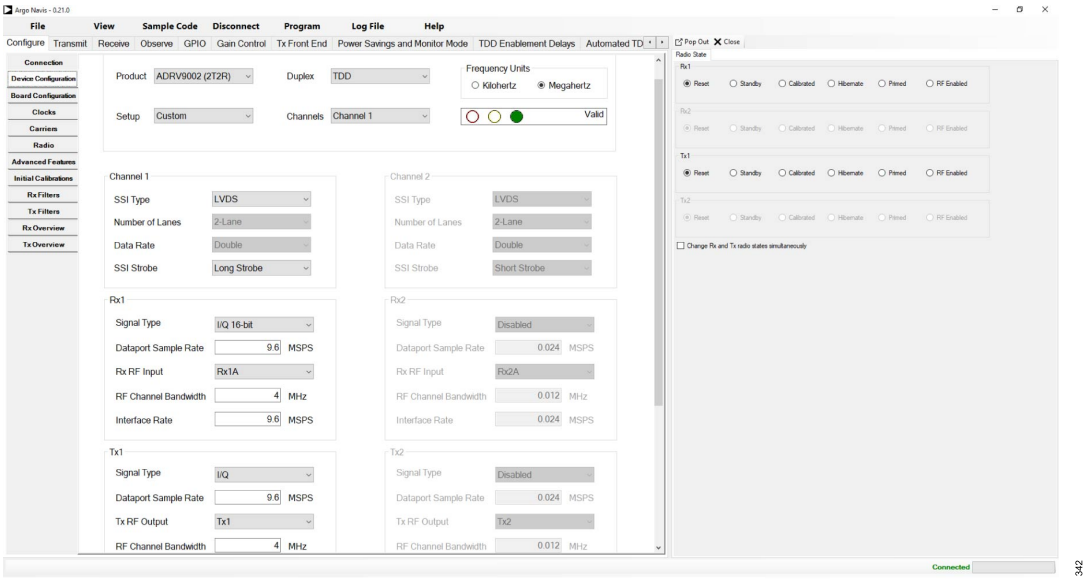


Figure 351. Pop-Up Options Docked in TES

Radio State

Once the board is connected, view/set the radio state under **View->Radio State**. Certain operations in the GUI can set the radio to certain states. Be aware of the state the radio is operating on and control the GUI accordingly. Also set the radio to a certain state from this window.

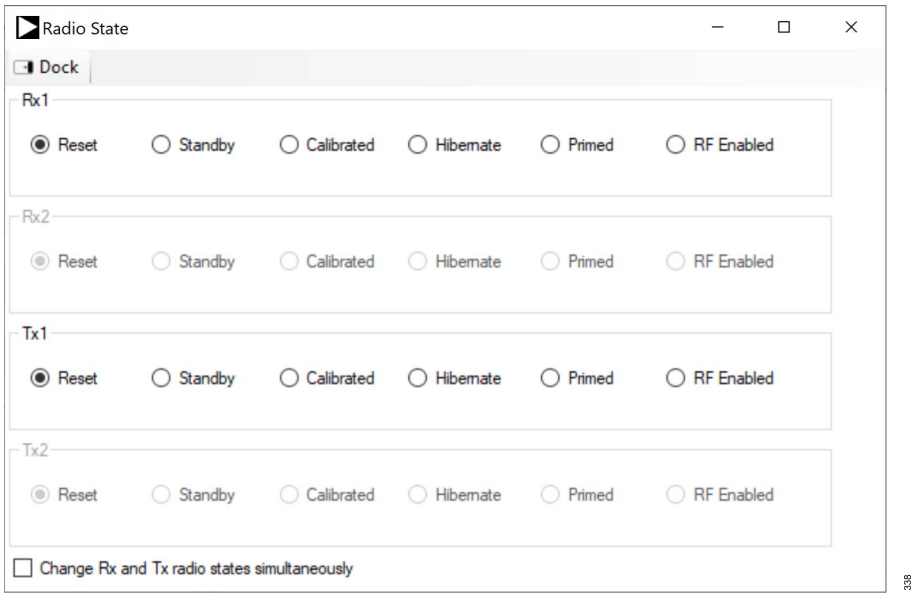


Figure 352. Radio State Window

Power/Temperature Monitoring

The ADRV9001 evaluation software allows to monitor the power usage of the system. On top, **Power/Temp Monitoring** shows the detailed voltage, current, and power status of each power domain. It also shows the temperature from an internal temperature sensor. Figure 353 is a screenshot of the power monitor window.

Note that the ADRV9001 does not currently support the VDDA_1P0 power domain. So, ignore its read back.

ADRV9001 EVALUATION SYSTEM

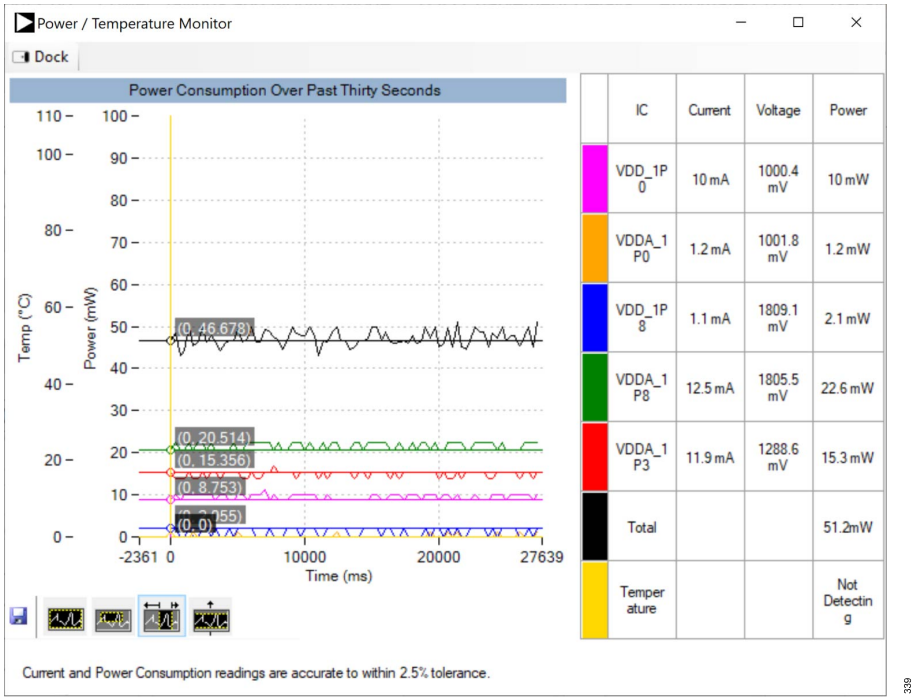


Figure 353. Power / Temperature Monitoring Window

Power Savings

This option opens a window with radio button options to change the system state for monitor mode. Select the "Monitoring" button to force the state into monitor mode. Similarly select "Detected" to push the state to detected. There are options to toggle monitor mode enable and wake up pins, also options to toggle both channels power saving pins. For more information on this see Power Saving and Monitor Mode and Power Savings and Monitor Mode

Timing Diagram

The timing diagram window will show when the "Enable Automated TDD State Machine for FPGA" option in the **Automated TDD** tab is selected.

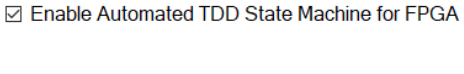


Figure 354. Enable Auto TDD option in TDD Tab

This window has two tabs, the first tab shows a visual representation of the timing diagram of all the TDD signals that have been input into the **Automated TDD** signal table. The second shows a visual representation of the hopping signals from the hop tables in the **Carriers** tab and needs frequency hopping enabled to display the signals.

ADRV9001 EVALUATION SYSTEM

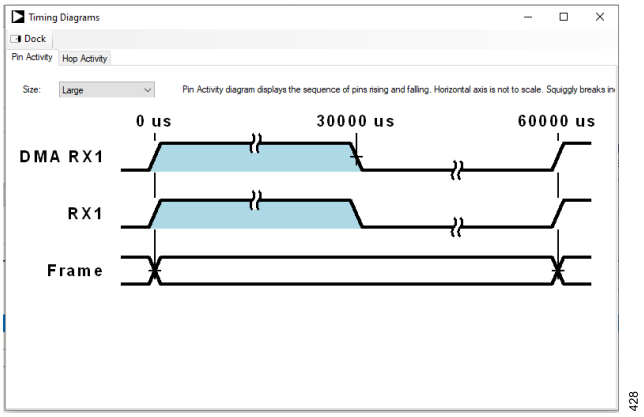


Figure 355. Timing Diagrams

Frequency Hopping

Frequency hopping displays the frequency hopping wizard window that assists in setting up the frequency hopping tables in a step-by-step format. For more details, see the Frequency Hopping section.

ADRV9001 EVALUATION SYSTEM

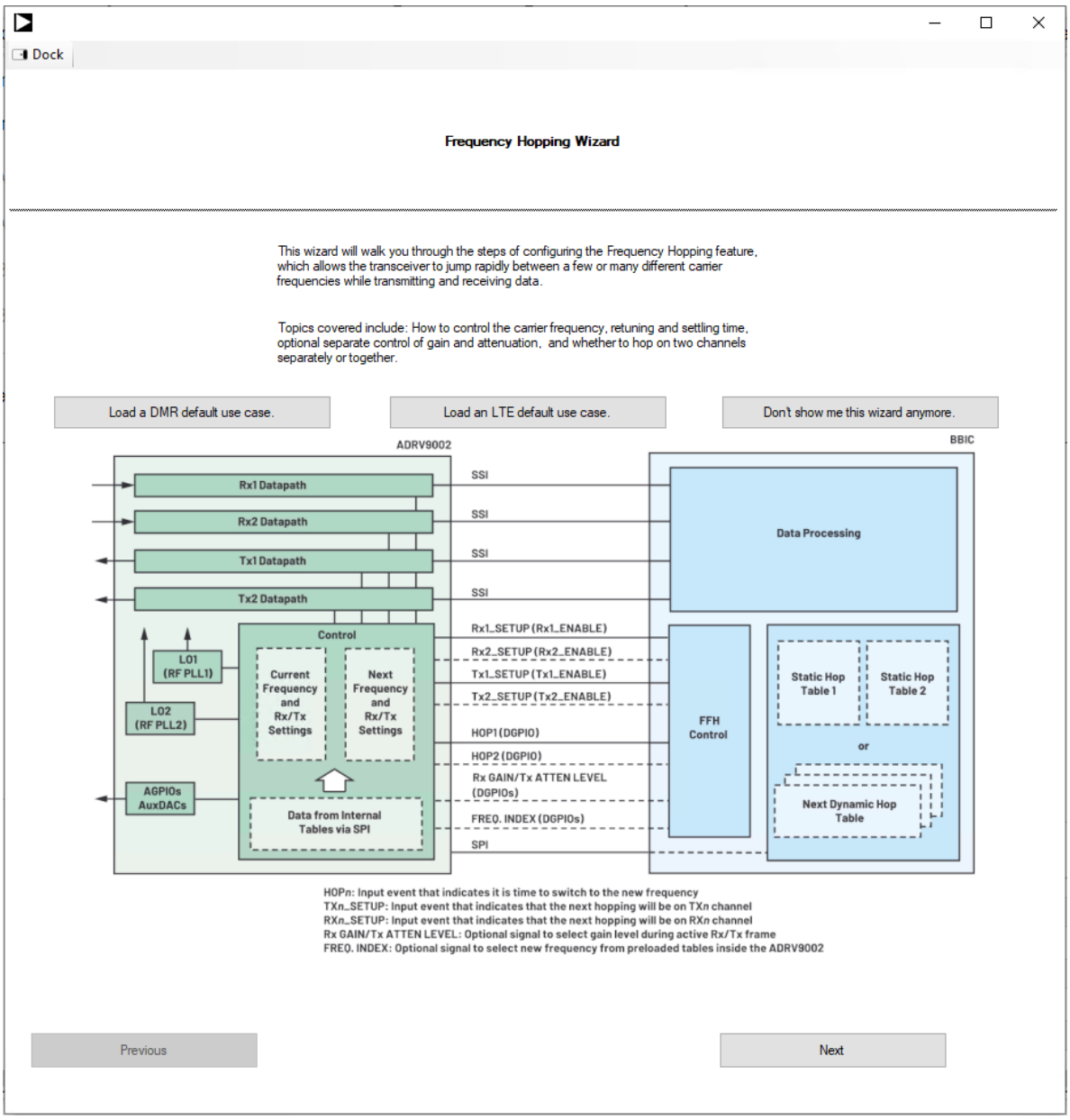


Figure 356. Frequency Hopping Wizard

Use this wizard to set up all the frequency hopping tables, starting with **Use Case** and **Timing Constraints**. Select the type of **Carrier Frequency Strategy** from ping pong to dynamic table loading. This dictates how many tables to initialize and how the frequency information is loaded from the table. Use the **Gain and Attenuation by Pin** section to optionally add attenuation and gain codes that are GPIO operated instead of the hopping table gain and attenuation settings. Select the **NCO-Only** option to use an Rx offset. Note that Rx1 and Rx2 are mapped to the same LO and therefore must have the same offset. Upload the hop tables in the **Hop Tables** page. There are sample tables available in the TES folders **"~\Examples\Hop Tables"**. Use the dropdown to select "Upload Existing Table" to find them. The **Executed Hop Sequence** then displays a summary table from all the inputs. This then populates the hopping and gain tables in the frequency hopping section of the TES.

Interface Gain Seed / Save

Use the **Interface Gain Seed / Save** window to seed or save the Rx digital interface gain. This is only enabled in the automatic gain control mode with an assigned seed/save GPIO pin. Seed the interface gain from the window and on the next rising edge of the selected GPIO pin.

ADRV9001 EVALUATION SYSTEM

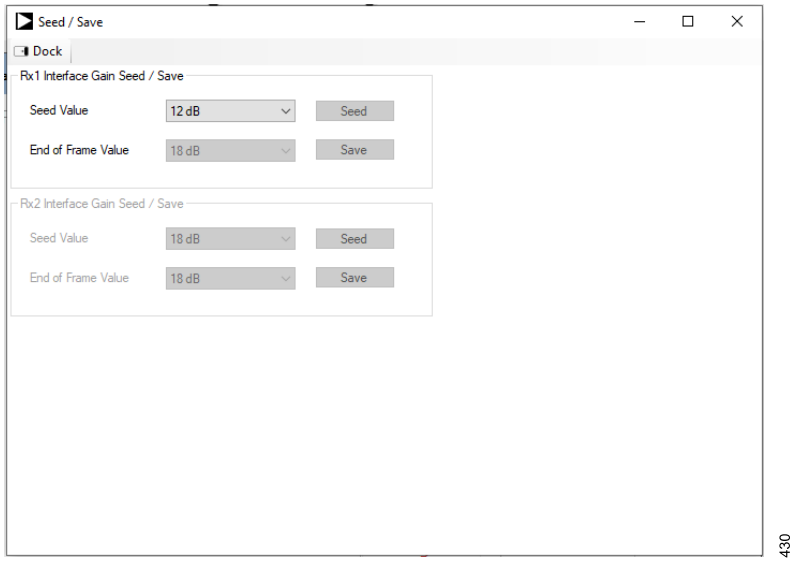


Figure 357. Interface Gain Seed / Save

Log File

The **Log File** button at the top of the GUI shows the logging information of the system. If the PC is connected to the evaluation platform, the log file shows the version numbers for the different components of the system, including the firmware, FPGA, API, and others. If errors occur, for example, programming the chip fails, the log file provides certain debugging information on what is failing.

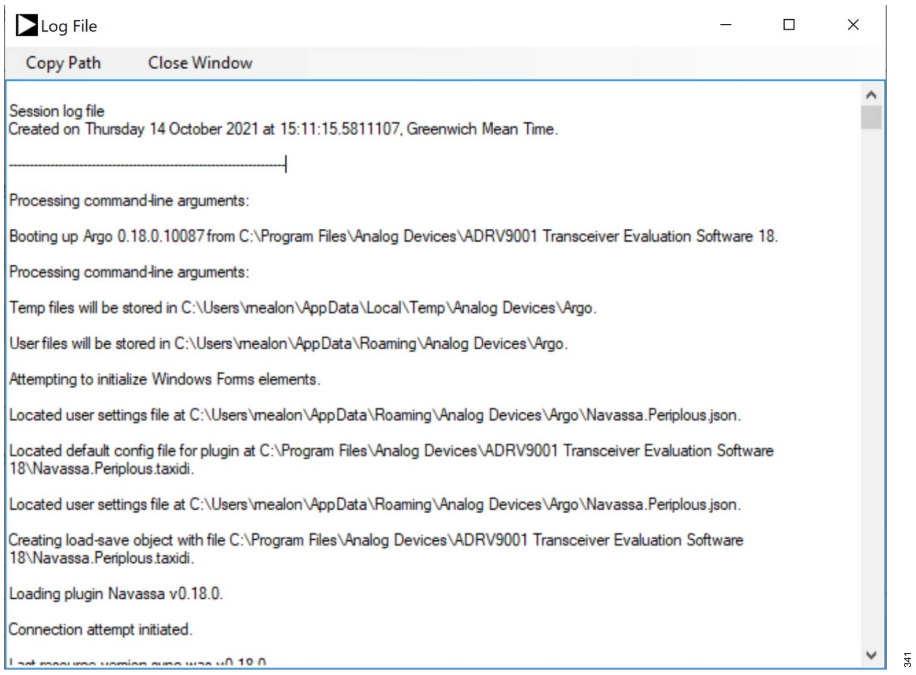


Figure 358. Log File Window

Sample Code

Use Matlab and Python to initialize the system. First, connect the PC to the part by clicking **Connect**. Then, configure the system to the desired state and program. After the programming is successful, click **Sample Code-> Matlab / Python/ C99/ C#** in the GUI to generate Matlab, Python, C, and C# initialization code. Then execute the generated Matlab Python or C code to bring the part to the same desired state as the GUI.

ADRV9001 EVALUATION SYSTEM

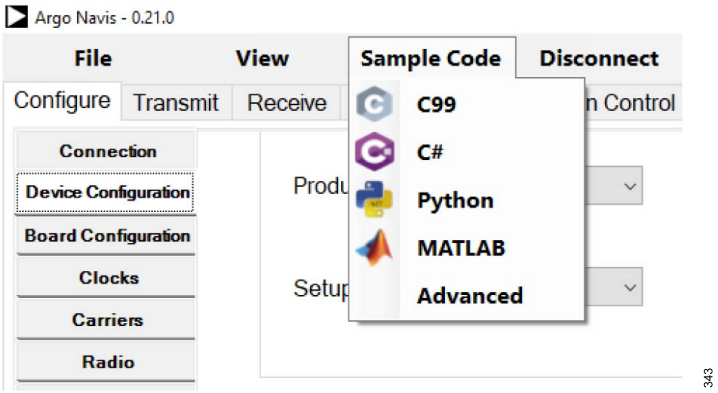


Figure 359. Auto Generated Code Options

Frequency Hopping TES Examples

This section shows three examples to use the TES to achieve frequency hopping for the ADRV9001. For details on the frequency hopping operation, see the Frequency Hopping section.

Example 1: Manual Frequency Hopping

In this mode, there is no need to specify the timing. In most cases, this is a mode to understand how frequency hopping operates in the ADRV9001. This example uses the DMR profile. Other profiles are largely the same.

This example shows the frequency hopping for the transmitter only, receiver only, and transceiver.

1. Connect.
2. Go to **Carriers** tab on the left pane.
3. Select "Frequency Hopping" under "Carrier Configuration Mode".

   The first three steps are the same for "Automated TDD Frequency Hopping" as well.

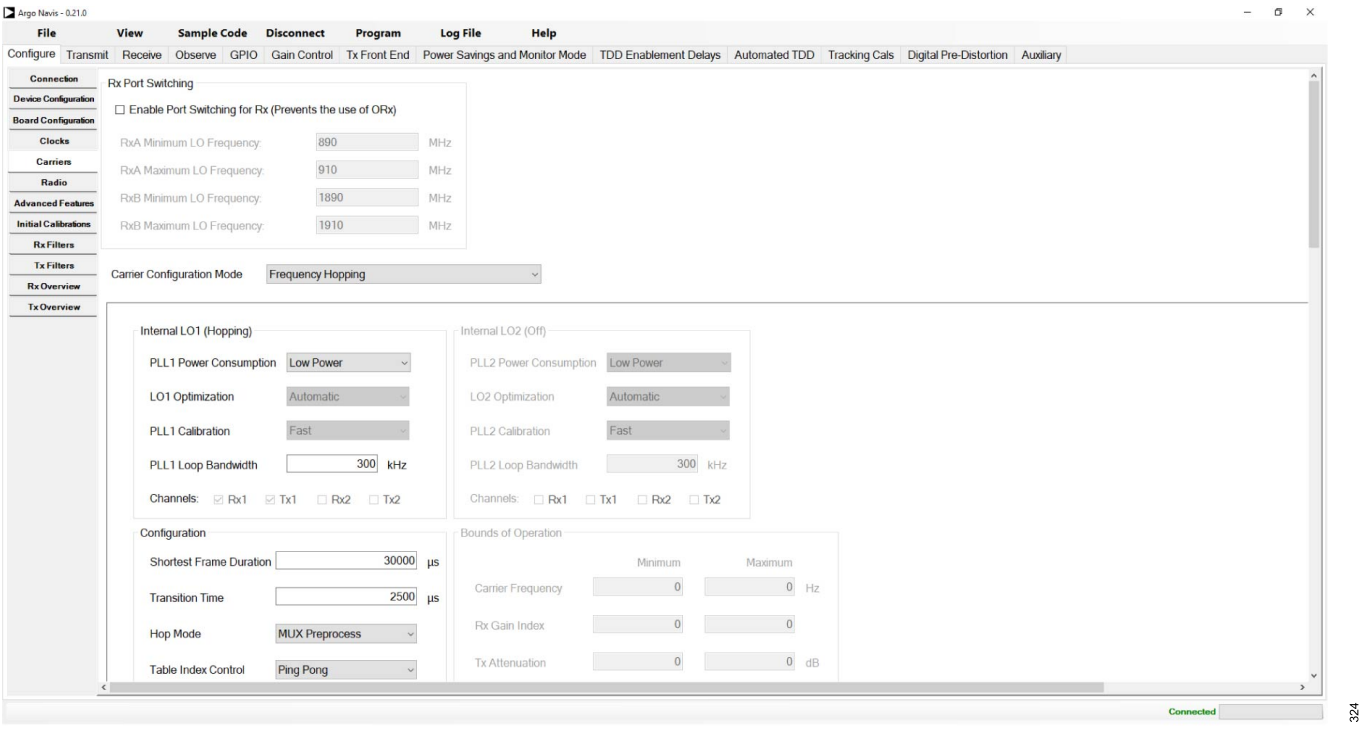


Figure 360. Manual Frequency Hopping Example

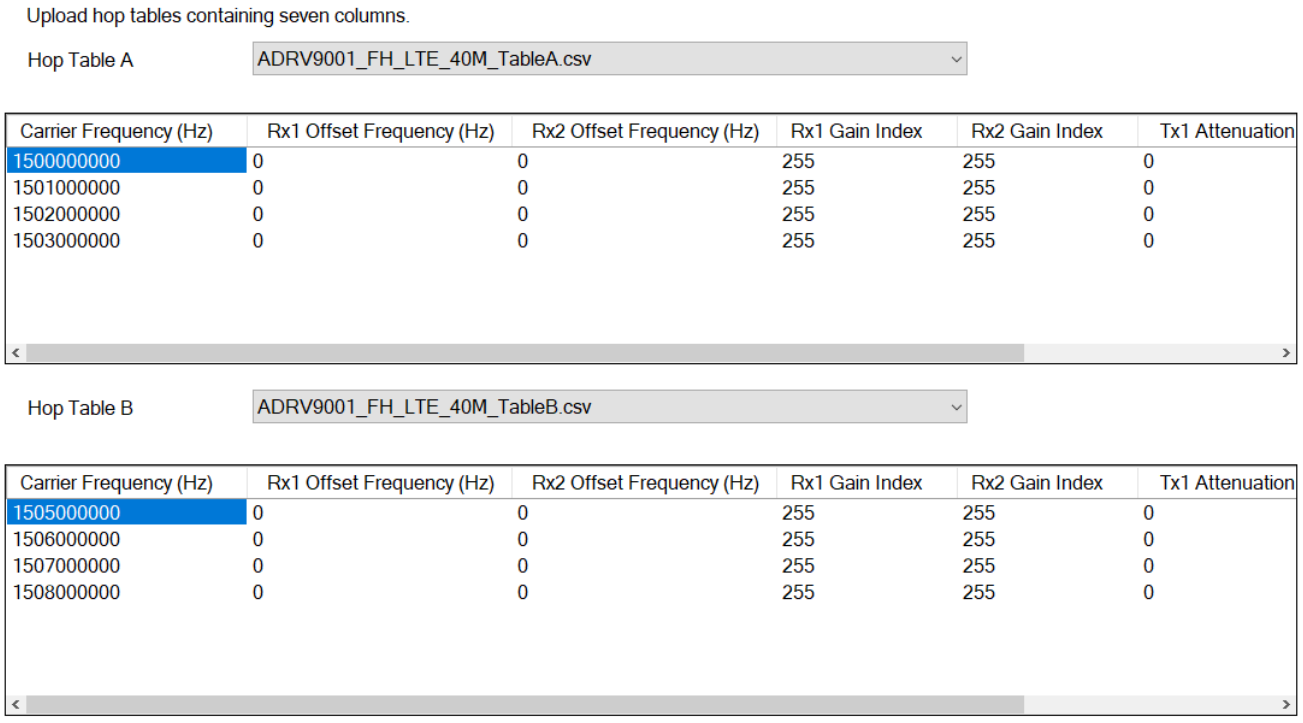
ADRV9001 EVALUATION SYSTEM

4. Specify the "Hop Pin" . By default, set it to "Pin 01".
5. Specify the "Hop Mode". Different profiles have certain modes enabled/disabled.
   a. "Mux Preprocess" indicates two LOs in use for frequency hopping, and the tables are preprocessed before hopping.
   b. "Mux Real-Time Process" indicates two LOs are in use for frequency hopping, and tables are processed at the hopping stage.
   c. "LO Retuning (Real-Time)" indicates that only one LO is used for frequency hopping and the tables are processed at the hopping stage.

   This example uses the default "Mux Preprocess".
6. Specify "Operation".
   a. Automatic Loop: Automatically increment through a frequency hopping table and wrap around once end is reached.
   b. Automatic Ping Pong: Ping pong operation between ADI_ADRV9001_FHHOPTABLE_A and ADI_ADRV9001_FHHOPTABLE_B. Automatically increment through one frequency hopping table and switch to the other once the end is reached.
   c. GPIO: Use DGPIO pins to index the entries within a frequency hopping table.

   This example uses the default "Ping Pong" mode.
7. Load the frequency hopping tables.
   a. Load two tables (Tables A and B) or load only one table (Table A or B). The GUI provides the frequency hopping table examples.
   b. At this point, edit the frequency tables in the loaded files. Do not edit the values directly on the GUI. Once the values in the table are changed, load the table again.

   Two tables are loaded in this example.



434

Figure 361. Frequency Hopping Tables

8. "Executed Sequence" displays in the table at the bottom of the tab. Any changes made to the frequency hopping setting automatically populates this table to show the programmed sequence.

ADRV9001 EVALUATION SYSTEM

Executed Sequence

Hop Table	Hop Index	Carrier (Hz)	Gain / Atten Index 1	Gain 1	Attenuation 1 (dB)	Gain / Atten Index 2	Gain 2	Attenuation 2 (dB)
A	1	1501000000	N/A	255	0	N/A	255	0
A	2	1502000000	N/A	255	0	N/A	255	0
A	3	1503000000	N/A	255	0	N/A	255	0
B	0	1505000000	N/A	255	0	N/A	255	0
B	1	1506000000	N/A	255	0	N/A	255	0
B	2	1507000000	N/A	255	0	N/A	255	0
B	3	1508000000	N/A	255	0	N/A	255	0
A	0	1500000000	N/A	255	0	N/A	255	0
A	1	1501000000	N/A	255	0	N/A	255	0

435

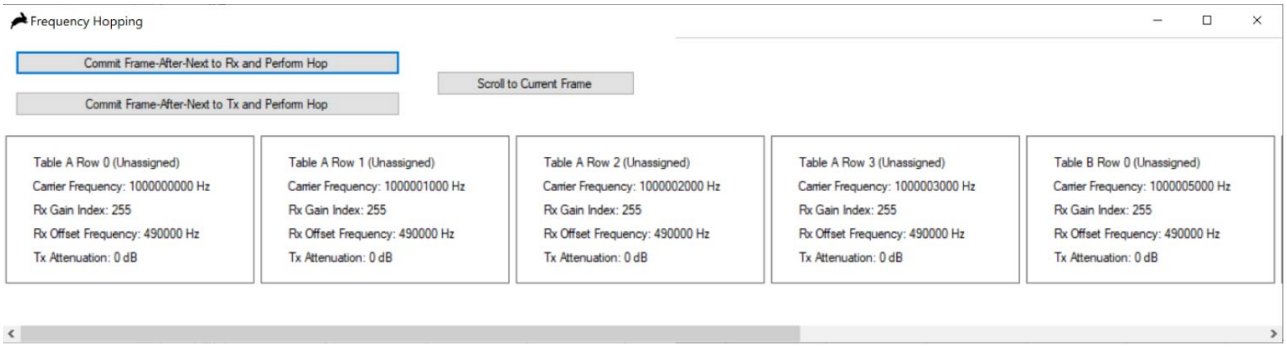
Figure 362. Frequency Hopping Executed Sequence

9. Click **Program**.
10. Upon successful programming, go to the **Transmit** tab and click **Play**. A window pops up and indicates the frequency hopping is working in the manual mode.
    a. "Commit Frame-After-Next to Rx/Tx and **Perform Hop**" button commits the "Frame-After-Next" to the receiver/transmitter. Note that this is not the same as the next frame, but the one after.
    b. Scrolling to "Current Frame" locates the current frame to show. This is very useful if there is a large set of frequencies in the frequency table.
    c. Each rectangle box under it shows a frequency entry in a frequency table. This includes the:
        1. Table being used (Table A or B)
        2. Carrier frequency
        3. Receiver gain index
        4. Receiver offset frequency
        5. Transmitter attenuation

    Frequency, gain, and attenuation values must match those in the loaded tables.

Transmitter Only

1. Change the transmitter 'Data Source Single Tone' in the main TES window, and enter 0 Hz under "Tone 1 Frequency".
    a. See the figure



327

Figure 363. Manual Frequency Hopping Using TES

    b. This indicates:
        1. All entries of the frequencies are shown, both in Table A and B.
        2. All entries should display 'Unassigned'.
        3. The upcoming frame is not assigned.
        4. There should not be any signal coming out of the transmitter.
2. Click **Commit Frame-After-Next to Tx and Perform Hop**.

ADRV9001 EVALUATION SYSTEM

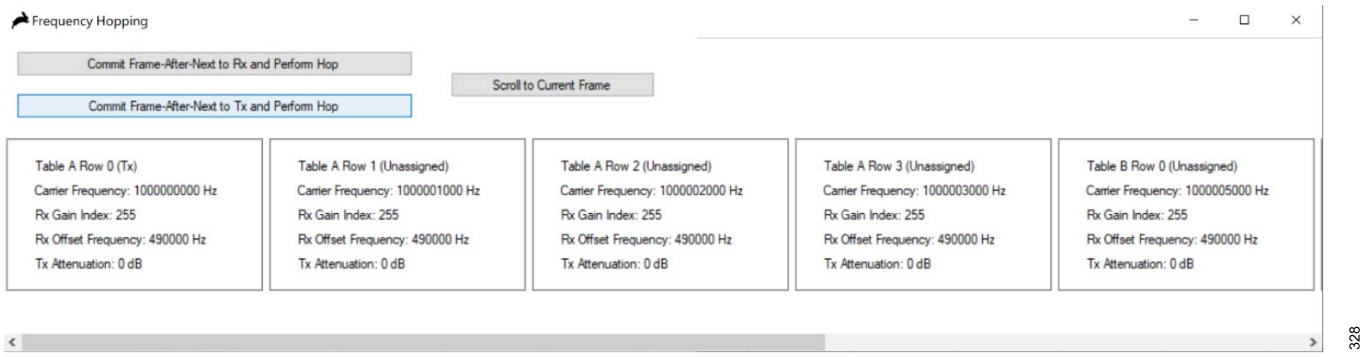


Figure 364. Manual Frequency Hopping Using TES

a. The upcoming frame is assigned to the transmitter (first box).
b. There should not be any signal coming out of the transmitter.

3. Click **Commit Frame-After-Next to Tx and Perform Hop** again.

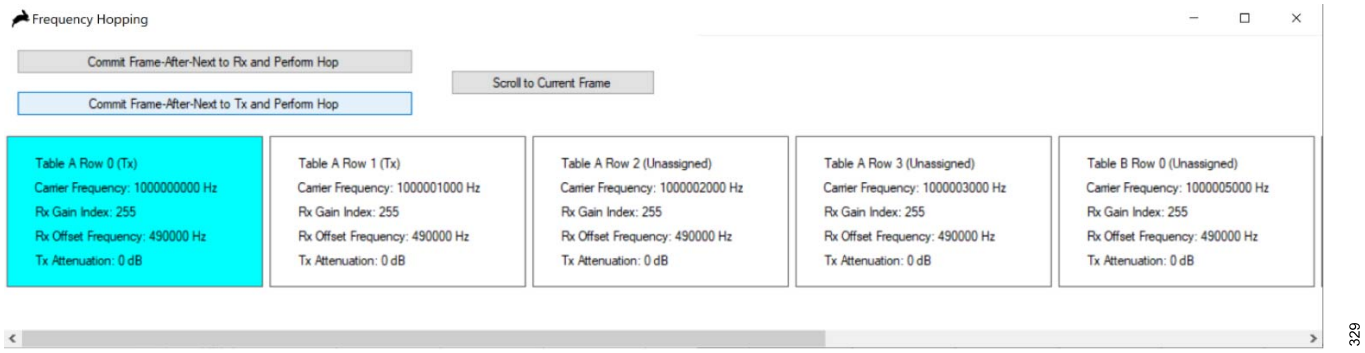


Figure 365. Manual Frequency Hopping Using TES

a. The current frame is played for a frequency of 1000 MHz (highlighted in blue).
b. Assign the next frame as the transmitter as well.
c. The signal should appear at the transmitter output as 1000.001 MHz single tone.

4. Repeat the last step.

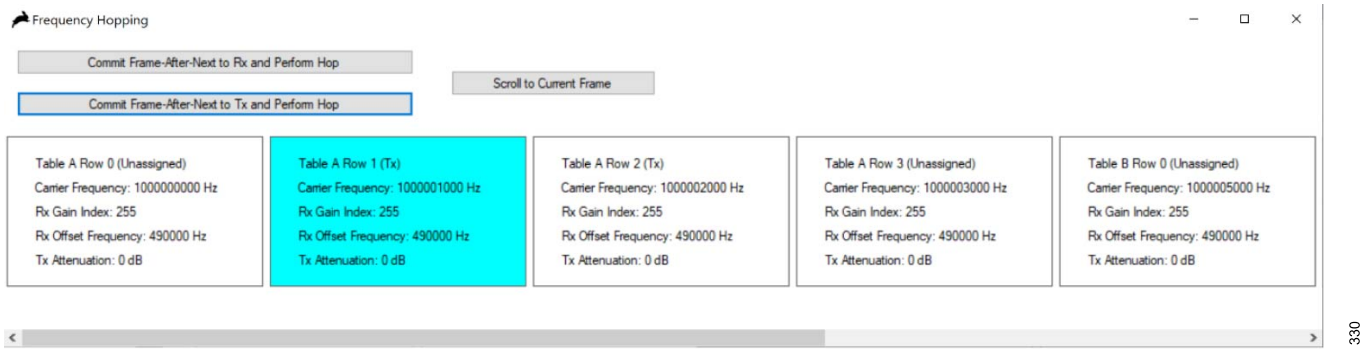


Figure 366. Manual Frequency Hopping Using TES

a. Advance the current frame to the next 1000.001 MHz.
b. The previous entry returns to the 'Unassigned' state.
c. Set the next frame to 1000.002 MHz.
d. The signal must appear at the transmitter output as 1000.001 MHz single tone.

ADRV9001 EVALUATION SYSTEM

Receiver Only

The **Receiver Only** steps are the same as Transmitter Only . The only difference is that instead of clicking **Play** on the **Transmit** tab, click **Play** on the **Receive** tab.

Transceiver

The transceiver steps are also very similar, except that click **Play** on both the **Transmit** and **Receive** tabs before operating frequency hopping.

Note: Upon reset, all frequencies entries must assign back to "Unassigned".

Example 2: GPIO-Controlled Frequency Hopping

Following the setup of **Example 1** through to step 8, implement GPIO controls with the following instructions:

1. Enable separate gain and attenuation tables that correspond with the GPIO codes to call for individual hops. Do this by loading tables into the "Gain Table" and "Attenuation Table" in the **Frequency Hopping** tab.

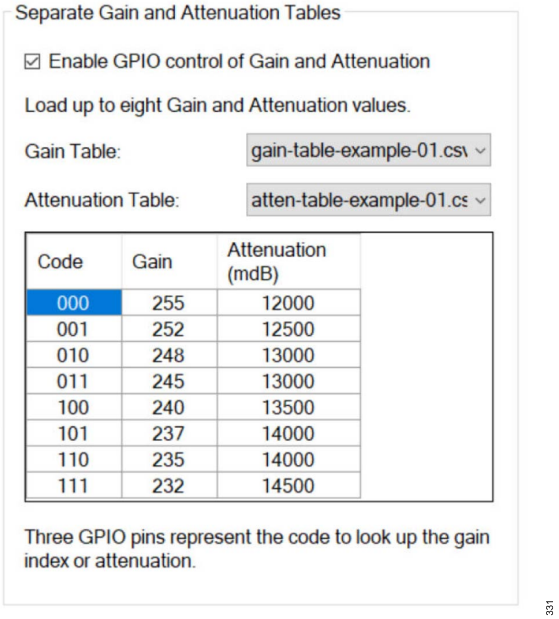


Figure 367. Separate Gain and Attenuation Tables

2. When the tables are loaded, enable an upload sequence in the "Control Hopping by GPIO" section, shown in Figure 368. This CSV file contains a row of integers from 0 to 7 in the required sequence. The integer 0 corresponds to the gain and attenuation value used in the table shown in Figure 367.

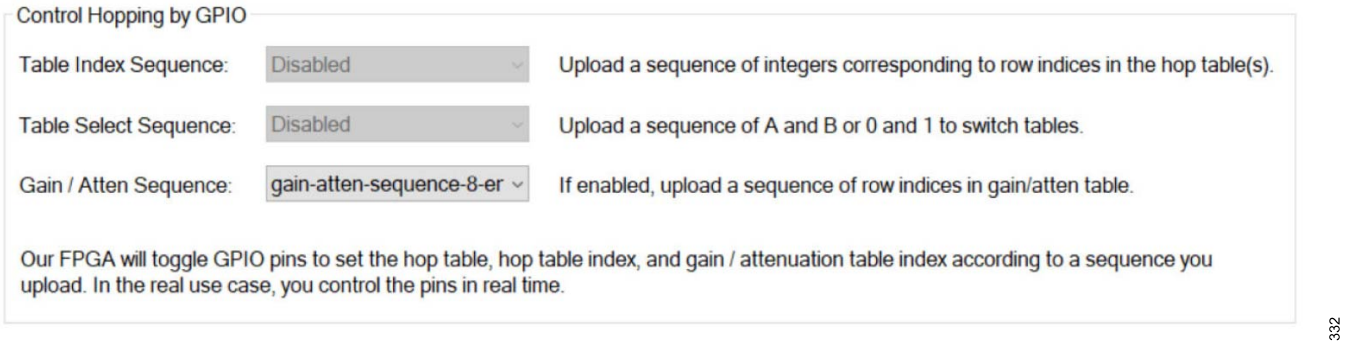


Figure 368. Control Hopping by GPIO

ADRV9001 EVALUATION SYSTEM

3. The FPGA board toggles the GPIO pins to set the hop table index according to the loaded sequence. Select the GPIO pins used in the "GPIO Pins for Control of Hop Table" section.

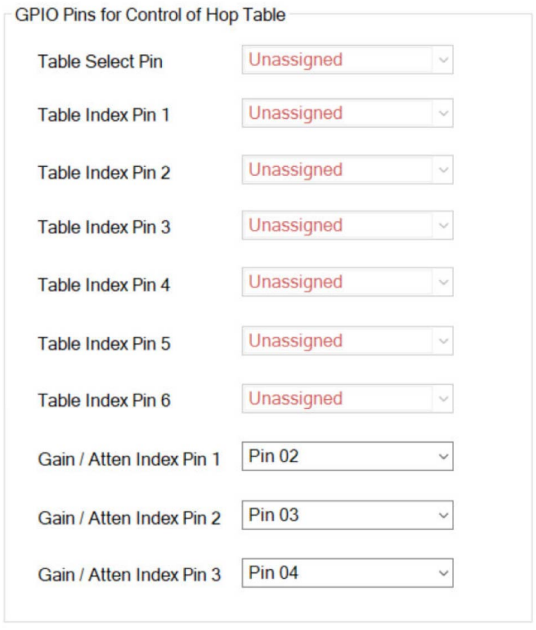


Figure 369. GPIO Pins for Control of Hop Table

4. The "Executed Sequence" table automatically updates with the gain and attenuation settings as indicated by the sequence table loaded in step 2.

Executed Sequence

Hop Table	Hop Index	Gain / Atten Index	Carrier (Hz)	Gain	Attenuation (mdB)
A	0	000	1000000000	255	12000
A	1	001	1000001000	252	12500
A	2	010	1000002000	248	13000
A	3	011	1000003000	245	13000
B	0	100	1000005000	240	13500
B	1	101	1000006000	237	14000
B	2	110	1000007000	235	14000
B	3	111	1000008000	232	14500
A	0	000	1000000000	255	12000

334

Figure 370. Executed Sequence Table (GPIO Settings)

5. Per **Example 1**, when the users program the part and click **Play** in the **Receive tab**, they are shown the "Frequency Hopping" window and can commit frames to the receiver or transmitter. In this example, notice that the signal is attenuated at the rate described in the gain and attenuation tables.

Example 3: Automated TDD with Frequency Hopping

Unlike the previous mode, in this mode, specify the TDD timing. Also, there is no frequency hopping window to advance frames manually. Instead the frames are played automatically.

To achieve this:

1. Follow the steps in **Example 2** before programming.
2. In the **Automated TDD** tab, click **Enable Automated TDD State Machine for FPGA**.
3. The ADRV9001 TES includes several predefined examples in the **/Examples** folder.

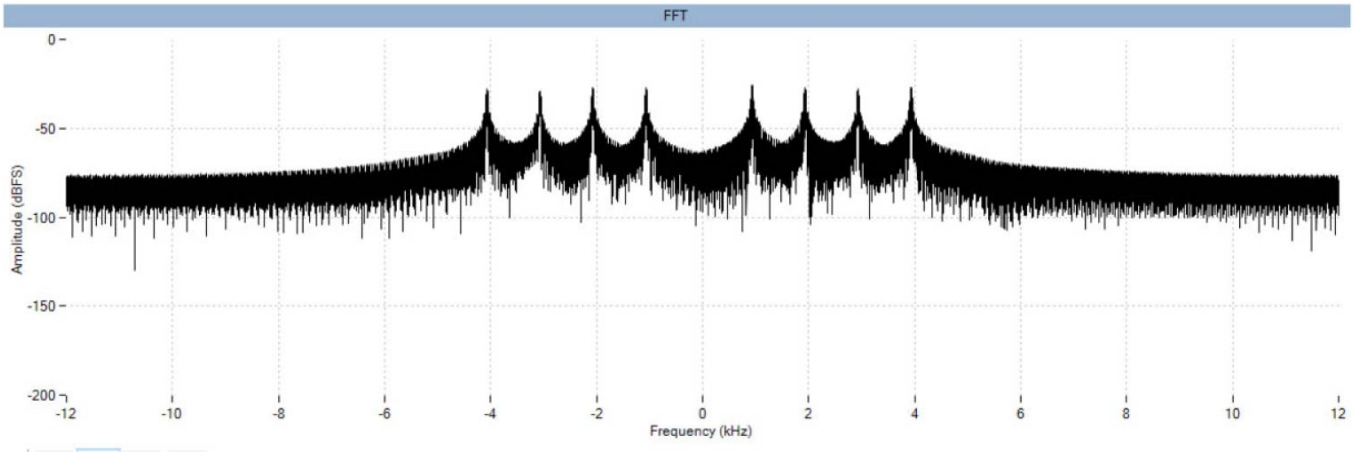
ADRV9001 EVALUATION SYSTEM

Receiver Only

1. Select the predefined JSON file DMR_24K_RX_ONLY_FH.json.
2. Hop Pin is set automatically.
3. Go to the **Receive** tab, set the capture length to the longer value. Here, it is set to 65536.
4. Click **Play**.

See the signal being played for multiple frequencies.

Note that in the time domain, the TES only shows actual data frames, i.e., without gap between data frames.



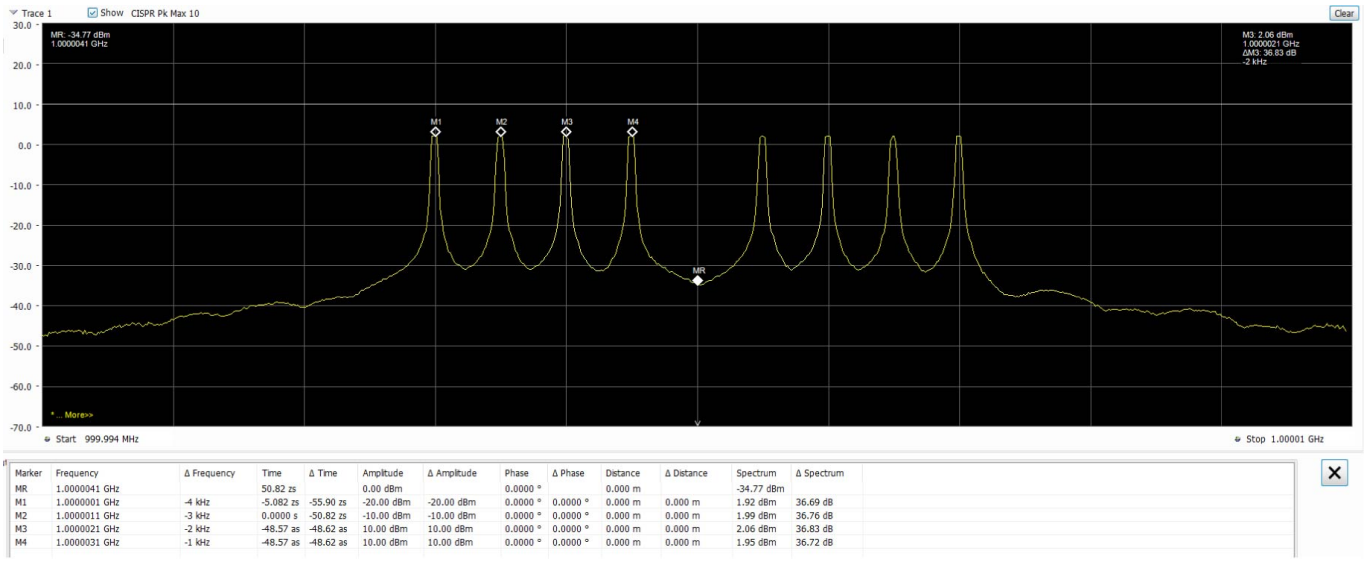
335

Figure 371. Receiver TDD DMR with Frequency Hopping

Transmitter Only

The steps are the same as **Receiver Only**, except that the predefined JSON file must be DMR_24K_TX_ONLY_FH.json.

Use 0 Hz tone as an example. There are six frequencies here, and the plot is done with maximum hold.



336

Figure 372. Transmitter TDD DMR with Frequency Hopping

Transceiver

The steps are the same as **Transmitter Only** or **Receiver Only**, except that the predefined JSON file must be DMR_24K_TX_RX_FH.json.

ADRV9001 EVALUATION SYSTEM

In this mode, the frames are interleaved between the transmitter and receiver. Therefore, from the time domain, the transmitter spectrum analyzer must show gaps between the frames.

EVALUATION SYSTEM TROUBLESHOOTING

The following is a quick help guide describing what to do if the system is not operational. This guide assumes the user followed instructions and assembled the setup according to the hardware configuration described in this document.

TES Connection Issues

1. Ethernet connection (firewall on IP address, port blocked, and others).
   a. Make sure the firewall settings allow communication to the Xilinx platform.
   b. Check that the IP address for the Ethernet is set correctly. With a direct connection to the PC, the IP address in the TES must be 192.168.1.10 port 55557. Set the LAN connection to 192.168.1.2. With an indirect connection to the PC, set the IP address to the IP address that the router dynamically assigned to the FPGA.
2. FPGA platform incorrectly configured (refer to the user guide for jumper settings).
   a. Check that the switch on the Xilinx platform is set properly to boot from the SD card. Find these settings in Hardware Operation .
3. SD card not compatible with FPGA platform.
   a. There is a different SD card image required for the different supported Xilinx platforms. Analog Devices provides the SD card imaging software. To find the settings, revisit SD Card Imaging
4. SD card not compatible with TES version.
   a. An early image is required for all SDK versions previous to 0.13.0. For SDK releases 0.13.0 or later, a new SD card image is required. This is all taken care of with the Analog Devices SD card imaging software. But check if older versions have been used before. The .img files are also now available on the ADRV9001 product page.

No LED Activity (ZYNQ ZC706)

1. Check if the board is properly powered. There must be 12 V at the J22 input, and after powering the Xilinx platform on (SW1 turned on), the following must be true:
   a. Activate the fan on the ZYNQ platform. Ensure the fan cable is reconnected to the ADRV9001 evaluation platform fan header P702.
   b. A number of green LEDs on the ZYNQ platform near SW1 are ON with no red LEDs active on the ZYNQ platform.
   c. ZYNQ GPIO LEDs follow the sequence described in the section.
   d. Two green LEDs (D801 and D901) on the ADRV9001 evaluation card must be ON.
2. If the LED sequence does not follow the described one, check the jumper settings and SW11 positions on the ZYNQ platform. If these are correct, check if the SD card is correct and properly inserted in the J30 socket. Use the SD card supplied with the evaluation kit.
3. If there is still a problem and the ZYNQ platform is certainly operational, contact an Analog Devices representative for help.

LED Active But TES Reports Hardware is Not Connected

1. Check if the Ethernet cable is properly connected between the PC used to run the TES and the Xilinx platform. The LEDs on the Xilinx platform next to the Ethernet socket must flash when the connection is active.
2. If the cable is properly connected, then check if MS Windows OS is able to communicate over the Ethernet port with the Xilinx platform. Check if the IP number and open ports for the Ethernet connection used to communicate with the Xilinx platform follow the advice described in the section.
   a. Run cmd.exe and then type: ping 192.168.1.10. There must be a reply from the Xilinx platform. If there is no reply, re-examine the connection with the Xilinx platform.
   b. If the connection with the Xilinx platform is established but the TES still reports that hardware is not available, ensure the firewall on the Ethernet connection used to communicate with the Xilinx platform does not block ports 22 (SSH) and 55557 (Evaluation Software). Both ports must be open for normal operation.
3. If issues still persist, find some common connection issues on the Engineer Zone forum here.
4. Check for physical damage to the EVB.

a. Look for FB E803 located on the TOP side of the PCB, next to the mounting hole. There is a possibility that during transit or when in use, the nut used to keep the PCB in place damaged E803. Ensure that E803 is in place with good connection. If E803 gets broken, replace it with BLM41PG600SN1L from Murata or similar.

Red LED Constantly On

1. The Xilinx platform generates the power domain for IOs that control the ADRV9001 over the FPGA mezzanine card (FMC) interface. This power domain is called VADJ. For proper operation, voltage on that power domain must not exceed 1.89 V. The SD card provided with the evaluation card ensures that VADJ is properly set. On an evaluation card, a red LED is installed close to the FMC connector. The role of this LED is to indicate if the VADJ voltage exceeds 2.0 V. If that is the case, this LED is ON.
   a. When the FPGA is powered on, the LED is "on" for a few seconds as the SD card image readjusts the FPGA VADJ voltage to 1.8 V.
   b. If this LED does not turn off, then there is an issue, and while the part might still operate, this is exceeding the recommended level for VADJ. This decreases the lifetime of the part and can lead to permanent damage of the IC in the worst case. Possible causes for this LED not turning off are connected to the SD card image.
2. The chip might still operate correctly after this issue but the VADJ may have exceeded the recommended level. The only way to remove uncertainty here is to change the ADRV9001 on an evaluation board to the new one.

Init Calibration Fail

There may be program failure due to init calibration. This is usually caused if the receiver input connected to the signal generator and RF output is ON. This causes the ADRV9001 to interfere with its own internal receiver calibration. Turn off the RF signal during programming.

DLL Bug Fix

If more than one installation of the SDK exists on the computer, the TES software can call an incorrect set of DLLs, causing unexpected bugs.

Consider errors and bugs pertaining to implicit declarations, undeclared pointers, and missing brackets as indications of DLL errors in the SDK installation.

The most effective solution for this is to remove all the SDK installations from the drive and reinstall only one valid release of the software.

EngineerZone Support Forum

The ADRV9001 family of products is actively supported on EngineerZone (ez.analog.com). EngineerZone is a public forum where users can ask questions and search through previous answers to find solutions to their issues.

For support on the hardware design, visit: https://ez.analog.com/rf/wide-band-rf-transceivers/design-support-adrv9001-adrv9007.

For support on the software design (product line SDK), visit: https://ez.analog.com/rf/wide-band-rf-transceivers/tes-gui-software-support-adrv9001-adrv9007.

The Design Support ADRV9001 to ADRV9007 and TES GUI & Software Support ADRV9001 to ADRV9007 forums are supported directly by the product line.

For support on the software design using other tools (e.g., IIO, No-OS, etc.), visit the following:

- https://ez.analog.com/fpga/
- https://ez.analog.com/linux-software-drivers/
- https://ez.analog.com/microcontroller-no-os-drivers/
- https://ez.analog.com/sw-interface-tools/

To help determine which approach is better suited to specific projects, see the *ADRV9001 Software and Hardware Selection Guide*.

ADDITIONAL RESOURCES

Along with this user guide, there are several other resources to aid with evaluation and development with ADRV9001:

- Product pages for access to documentation, SDK, design resources and ordering information: ADRV9002, ADRV9003, ADRV9004, ADRV9005, ADRV9006
- Technical articles:
  - *The Complete Guide to Troubleshoot and Fine Tune Digital Predistortion*
  - *How a High Dynamic Range RF Transceiver Solves the Blocking Challenge for Mission Critical Communications*
  - *Transceiver with Scalable Power and Performance: A Solution to Mission Critical Communications*
  - *The Next-Gen, Software-Defined Radio (SDR) Transceiver Delivers Big Advances in Frequency Hopping (FH)*
- EngineerZone forums:
  - TES GUI & Software Support
  - Design Support
  - FAQs and tutorials

Q FORMATTING STANDARD DESCRIPTION

The following sections describe the two types of Q Formatting that is mentioned in this document.

- Ua.b Format(Unsigned Fixed-Point Format)
- Sa.b Format(Signed Fixed-Point Format)

Ua.b Format (Unsigned Fixed-Point Format)

Description: In the Ua.b format, "U" stands for unsigned, meaning all numbers are nonnegative. The "a" represents the number of integer bits, and "b" represents the number of fractional bits.

Range: The range of values that can be represented in Ua.b format is from 0 to $2^a - 2^{-b}$.

Example: For U2.3 format:

- Binary Representation: 10.011
- Binary to Decimal Conversion:
  - Integer Part: $1 \times 2^1 + 0 \times 2^0 = 2$
  - Decimal Part: $0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 0.375$
  - Total: $2 + 0.375 = 2.375$
- Decimal to Binary Conversion:
  - Multiply the decimal number by 2^b (2.375×2^b)
  - That equals to $2.375 \times 2^3 = 19$
  - 19 in Binary format is 10011

Sa.b Format (Signed Fixed-Point Format)

Description: In Sa.b format, "S" indicates the format is signed, allowing for both positive and negative numbers. The "a" includes the sign bit along with the integer bits, and "b" is the number of fractional bits.

Range: The range of values in Sa.b format is from -2^{a-1} to $+2^{a-1} - +2^{-b}$.

Example: For S2.2 format:

- Binary Representation: 10.11 (Most Significant Bit is a signed bit)
- Binary to Decimal Conversion:
  - Integer Part: $-1 \times +2^{+1} + 0 \times +2^0 = -2$
  - Decimal Part: $+1 \times +2^{-1} + +1 \times +2^{-2} = +0.75$
  - Total: $-2 + +0.75 = -1.25$
- Decimal to binary Conversion:

ADDITIONAL RESOURCES

- Multiply the positive decimal number by 2^b ($1.25 \times 2^2 = 5$)
- 5 in Binary format is 0101
- two's Complement of 0101 is 1011

FREQUENTLY ASKED QUESTIONS

Q1: Will there be an evaluation board for the other family members (ADRV9003/ADRV9004/ADRV9005/ADRV9006)?

A1: Not necessary as ADRV9002 is the full featured part that customers can use to emulate and use for all the family members with TES software.

Q2: Why do we not have one evaluation board for the entire operating frequency band 30 MHz to 6 GHz?

A2: We have two boards as each board provides optimal matching, minimizes loss and provides the best RF performance for the given band of operation. Customer can choose to make their own broad band match.

Q3: Why is the User Guide and TES software called ADRV9001 (ADRV9001 System Development User Guide)?

A3: One user guide and TES software covers the planned generics and future generics, so a non-product generic was created in order to generically refer to any product within the ADRV9001 family.

Q4: Why do we have two SD cards supplied with the evaluation boards?

A4: See the '*ADRV9001 SOFTWARE AND HARDWARE SELECTION GUIDE*'

Q5: Why do you call it TES (Transceiver Evaluation Software)?

A5: Historically, this was the name given, today users can do more than evaluation including develop code for their target system. The TES software allows for evaluation, prototyping, production target system development.

Q6: What is the power consumption of the ADRV9001?

A6: Power consumption is dependent upon may factors, most important is the configuration and set up, then the performance and IP blocks enabled. Users can see what the active power consumption for each of the ADRV9001 power supply rails in the TES software, special tab is provided in the GUI. For sleep modes, please review the Power Saving and Monitor Mode section.

Q7: What is the performance and resolution of the receiver ADCs?

A7: As ADRV9001 provides the entire signal path the resolution and that the ADCs are oversampled sigma delta not comparable with pipeline converters. The resolution is truncated to the output interface 16 bits. See the ADRV9001 data sheet for linearity and dynamic range performance.

Q8: What are the main differences between the ADRV9002 and the ADRV9006?

A8: The main differences are a reduced feature set to make the part more affordable for specific applications. The features that have been limited on the ADRV9006 include Fast Profile Switching, Digital Pre-Distortion, monitor mode, External LO support, fast PLL-retune and the high-performance ADC. See the ADRV9001 Product Family Comparison section for further details.

Q9: What data interface mode to use and consider?

A9: For narrow band operation signal bandwidths 1 MHz to 2 MHz CMOS serial interface is best option (can be considered up to 10 MHz) as CMOS is limited to max 80 MHz clock rate. For arbitrary bandwidths from kHz to 40 MHz the users should use the LVDS serial interface. Details on the interface are in the Data Interface section.

Q10: What is the clock rate and sample rate and how that equates to data rate and sample rate needs?

A10: ADRV9001 supports arbitrary sample and data rates. There is a dedicated baseband PLL on chip that is used to generate the sample rates and baseband interface rates independent of the RF PLL synthesizers. See the Clock Generation section.

NOTES