ADRV9026 System Development User Guide

UG-1727

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

Preliminary Technical Data

ADRV9026 Integrated Quad RF Transceiver with Observation Path

SCOPE

This user guide is the main source of information for system engineers and software developers using the Analog Devices, Inc., ADRV9026 software defined radio transceiver. Updates to this user guide can be expected after additional ADRV902x products are added to this family of radio transceivers. This user guide must be used in conjunction with the ADRV9026 data sheet.

TABLE OF CONTENTS

Scope ... 1
General Overview ... 4
System Overview ... 5
System Architecture Description ... 6
Software Architecture ... 6
API Folder Structure ... 6
Private vs. Public API functions ... 7
Hardware Abstraction Layer ... 8
Software Integration ... 10
Software Integration Process Overview ... 10
Software Package Folder Structure Overview ... 10
API Software Architecture ... 11
Implementing Hardware Abstraction Interface ... 11
Developing the Application ... 11
Serial Peripheral Interface (SPI) ... 20
SPI Bus Signals ... 20
SPI Data Transfer Protocol ... 20
SPI Configuration Using API Function ... 21
Timing Diagrams ... 22
System Initialization ... 24
Initialization Sequence ... 24
Serializer/Deserializer (SERDES) Interface ... 25
JESD204 Standard ... 25
Differences between JESD204B and JESD204C ... 26
Clock Distribution ... 26
Receiver (ADC) Datapath ... 27
Transmitter (DAC) Datapath ... 38
Supported Deframer Link Parameters ... 39
API Software Integration ... 49
Implementation Recommendations ... 49
Link Initialization and Debugging ... 50
First time system bring up—Checking Link Integrity ... 50
Sample Iron Python Code for PRBS Testing ... 51
PRBS Errors ... 51
SPO (Static Phase Offset) TEST to Verify Eye Width ... 53
Checking JESD204C Link Status ... 60
Selecting the Optimal LMFC/LEMC Offset for a Deframer 60
Synthesizer Configuration ... 72
Overview ... 72
Connections for external Reference clock(DEVCLK) ... 72
External Reference Clock (DEVCLK) Requirements ... 73
Clock Synthesizer ... 74
RF Synthesizer ... 74
Auxiliary Synthesizer ... 76
Setting the LO Frequencies ... 76
External LO ... 79
Lock Status ... 79
RF PLL Phase Synchronization ... 80
Arm Processor and Device Calibrations ... 82
Arm State Machine Overview ... 82
System Initialization ... 82
Pre MCS Init ... 83
Post MCS Init ... 83
Device Calibrations ... 83
Initial Calibrations ... 84
System Considerations for Initial Calibrations ... 87
Tracking Calibrations ... 91
Calibration Guidelines after PLL Frequency Changes ... 95
Initialization Calibrations to be Run after Device Initialization ... 104
Tracking Calibration Timing ... 105
Stream Processor and System Control ... 106
Slice Stream Processors ... 106
System Control ... 106
Use Cases ... 113
Transmitter Overview and Path Control ... 119
API Commands ... 119
DAC Full-Scale Function (DAC Boost) ... 124
adi_adrv9025_TxChannelCfg API Structure ... 126
Transmitter Power Amplifier Protection ... 127
PA Protection Description ... 127
Receiver Gain Control and Gain Compensation ... 134
Overview ... 134
Receiver Data Path ... 135
Gain Control Modes ... 136
Manual Gain Control (MGC) ... 137
Automatic Gain Control ... 139
AGC Clock and Gain Block Timing ... 147
Analog Peak Detector (APD) ... 148
Half-Band 2 Peak Detector ... 149

Power Detector 151
API Programming 152
AGC Holdover Function 153
Rx Gain Mode Switching using GPIO 153
Gain Control Data Structures 155
Sample Python Script—Peak Detect Mode with Fast Attack 160
Gain Compensation, Floating Point Formatter and Slicer 162
Receiver Data Format Data Structure 169
Digital Filter Configuration 173
Overview 173
Receiver Signal Path 173
Receiver Signal Path Example 176
Receiver Filter API Structure 177
Transmitter Signal Path 180
Tx Signal Path Example 181
Transmitter Filter API Structure 182
Observation Receivers Signal path 183
Observation Receiver Signal Path Example 184
Observation Receiver Filter API Structure 186
General-Purpose Input/Output Configuration 187
Digital GPIO Operation 187
GPIO_ANA Operation 195
General-Purpose Interrupt 196
PLL GPINT Sources 198
JESD204B/JESD204C GPINT Sources 198
PA Protection GPINT Sources 199
Arm GPINT Sources 199
Stream Processor Sources 200
Memory ECC Error 200
Software Procedures for GPINT 200
API Commands for GPINT 200
Auxiliary Converters and Temperature Sensor 203
Auxiliary DAC (AuxDAC) 203
Auxiliary ADC (AuxADC) 205
Temperature Sensor 207
SPI2 Description 208
SPI2 Configuration 208
Transmitter Control with SPI2 208
Receiver and Observation Receiver Control with SPI2 211
RF Port Interface Overview 212
RF Port Impedance Data 212
ADS Setup Using DAC (Data Access Component) and SEDZ File 215
Transmitter Bias and Port Interface 216
General Receiver Path Interface 218
Impedance Matching Network Examples 218
Matching Component Recommendations 220
Power Management Considerations 222
Important 222
Power Supply Sequence 222
Power Supply Domain Connections 222
Power Supply Architecture 225
Current Consumption 226
PCB Layout Considerations 228
Overview 228
PCB Material and Stack Up Selection 228
Fanout and Trace Spacing Guidelines 231
Component Placement and Routing Guidelines 232
RF and JESD Transmission Line Layout 234
Isolation Techniques 238
Power Management Layout Design 240
Analog Signal Routing Considerations 248
Digital Signal Routing Considerations 248
Unused Pin Instructions 249
Transceiver Evaluation Software (TES) Operation 250
Initial Setup 250
Hardware Kit 250
Requirements 250
Hardware Setup 251
Hardware Operation 253
TES Installation 253
Starting the Transceiver Evaluation Software 255
Normal Operation 256
Transmitter Operation 261
Receiver Operation 263
Scripting 264
C Code Generation 266

GENERAL OVERVIEW

There are several sections to this user guide, organized to make it easier for users to find the information pertinent to their area of interest. A synopsis of each section follows:

System Overview section: explains the capability of the ADRV9026 and provides an introduction to all the subsystems and functions, including block diagrams and interfaces.

System Architecture Description section: explains the software design approach using APIs and all details required to develop code to operate the device.

Software Integration section: explains the structure of the API developed by Analog Devices and how to integrate the API into the code of the customer.

Serial Peripheral Interface (SPI) section: explains the main control interface between the baseband processor (also referred to as BBIC) and the device.

System Initialization section: explains the sequence of steps required at startup.

Serializer/Deserializer (SERDES) Interface section: describes the high speed digital interface that transfers data to/from a baseband processor.

Synthesizer Configuration section: describes the design, control, and versatility of the synthesizer subsystem.

Arm Processor and Device Calibrations section: explains the calibrations scheduled and controlled by the internal Arm® processor.

Stream Processor and System Control section: explains the stream processor functions and how these functions are implemented.

Transmitter Overview and Path Control section: describes operation of the transmitter attenuation settings and available software API used for control.

Transmitter Power Amplifier Protection: describes the protection circuitry and how it works in conjunction with the general-purpose interrupt feature to enable transmitter attenuation and notify the baseband processor that such an event has taken place.

Receiver Gain Control and Gain Compensation section: describes automatic and manual gain control options that the API uses for making adjustments.

Digital Filter Configuration section: describes the digital processing portion of each receiver and transmitter and provides details on configuration options.

General-Purpose Input/Output Configuration section: describes the different GPIO capabilities provided by the ADRV9026 device and how to configure those capabilities for various functions.

General-Purpose Interrupt section: describes the various interrupt options that can be routed to the GPINT pins for monitoring purposes.

Auxiliary Converters and Temperature Sensor section: describes the implementation and functionality of the AuxDAC, AuxADC, and internal temperature sensor.

SPI2 Description section: explains the implementation and functionality of the SPI2 bus using designated GPIO pins.

RF Port Interface Overviewsection: describes the RF port impedance matching process and explains different topologies that can be used to achieve proper impedance matching.

Power Management Considerations section: explains how to connect power supplies to the device, the inputs that supply the various blocks, and precautions to take when completing a schematic and layout for power routing implementation.

PCB Layout Considerations section: provides guidelines for proper printed circuit board (PCB) layout and techniques for maximizing performance and minimizing channel-to-channel interference.

Transceiver Evaluation Software (TES) Operation section: explains setup and control of the device using the graphical user interface (GUI) software.

SYSTEM OVERVIEW

The ADRV9026 is part of a family of highly integrated RF agile transceivers designed for use in small cell, massive MIMO, and macro base station equipment used in advanced communications systems. The device contains four independently controlled transmitters, dedicated observation receiver inputs for monitoring transmitter channel outputs, four independently controlled receivers, integrated synthesizers, and digital signal processing functions to provide a complete transceiver solution. The device provides the high radio performance and low power consumption demanded by cellular infrastructure applications such as macro 2G/3G/4G/5G and massive MIMO base stations. This document is designed to encompass description of all functions available in the ADRV9026. Note that some variants may be developed for specific design targets that do not encompass all available functions, so refer to the data sheet for the specific device to determine which features are included. To avoid confusion, the term "device" is used throughout this user guide to refer to any variant that employs a specific function. When a function that applies to a specific part is described, the device part number is used to delineate which device is being described.

The ADRV9026 is designed to operate over the wide frequency ranges of 650 MHz to 6 GHz. The receiver channels support bandwidth up to 200 MHz with data transfer across (up to) four JESD204B/JESD204C lanes at rates up to 16.22 Gbps. The transmitter channels operate over the same frequency range as the receivers. Each transmitter channel supports up to 450 MHz synthesis bandwidth with data input across (up to) four JESD204B/JESD204C lanes. In addition, local oscillator (LO) routing allows the transmitters to operate at different frequencies than the receivers for additional flexibility. Two observation receiver channels are included to provide the capability to monitor feedback from the transmitter outputs. The feedback loops can be used to implement error correction, calibration, and signal enhancing algorithms. These receivers operate in the same frequency range as the transmitter channels, and they support up to 450 MHz channel bandwidth to match the output synthesis bandwidth of the transmitter channels. These channels provide digital data paths to the internal Arm processor for use in calibration and signal enhancement algorithms.

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

Power supply for each block is distributed across four different voltage supplies: 3 analog and 1 digital. The analog supplies are 1.8 V, 1.3 V, and 1.0 V. These supplies are fed directly to the power inputs for some blocks and buffered by internal low dropout (LDO) regulators for other functions for maximum performance. The digital processing blocks are supplied by a 1.0 V source. In addition, a 1.8 V supply supplies all GPIO and interface ports that connect with the baseband processor.

See the functional block diagram in the ADRV9026 data sheet for a high level view of the functions in the ADRV9026. Descriptions of each block with setup and control details are provided in subsequent sections.

SYSTEM ARCHITECTURE DESCRIPTION

Analog Devices developed a proprietary application programming interface (API) software for the ADRV9026 transceiver device. Whereas this section outlines the overall architecture, folder structure, and methods for using API software on the customer platform, this section does not explain the API library functions. Detailed information regarding the API functions is in the doxygen document included with the API software (adrv9025.chm) located at /c_src/doc. This file can also be viewed in the **Help** tab on the ADRVTRX transceiver evaluation software (TES) used for controlling the evaluation platform. Note that the ADRV9025 is the baseline device for the ADRV902x family; all API and evaluation systems use the ADRV9025 product number to delineate the product. With respect to this user guide, the ADRV9025 and ADRV9026 product numbers are interchangeable.

SOFTWARE ARCHITECTURE

Figure 2 illustrates the software architecture for the system evaluation platform.

This architecture can be broadly divided into three main layers:

- Hardware abstration layer: consists of device drivers and device specific code.
- Middleware layer: consists of device APIs and other auxiliary layer functions, and resides in the platform layer.
- Application layer: consists of radio application software running on a baseband processor. The baseband processor can be an embedded processor or a PC running a digital signal processing application, such as MATLAB® that processes baseband data.

API FOLDER STRUCTURE

Source files are provided by Analog Devices in the folder structure shown in Figure 1. Note that the baseline device, ADRV9025, is used in the source file folder structure. Each subfolder is explained in the following sections. Analog Devices understands that the developer may desire to use a different folder structure. Whereas Analog Devices provides API source code releases in the folder structure shown in Figure 1, the developer may organize the API into a custom folder organization. Creating a new folder structure, however, does not permit the developer the right to modify the content of the API source code. Modifying the content of any API source file is not allowed because such modification causes issues with supporting the API and complicates updates to future API code releases.

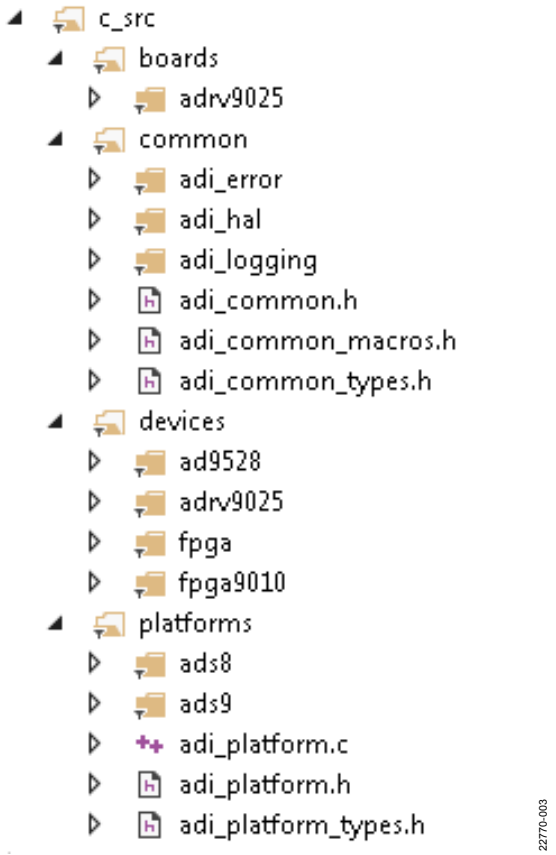


Figure 1. API Folder Structure

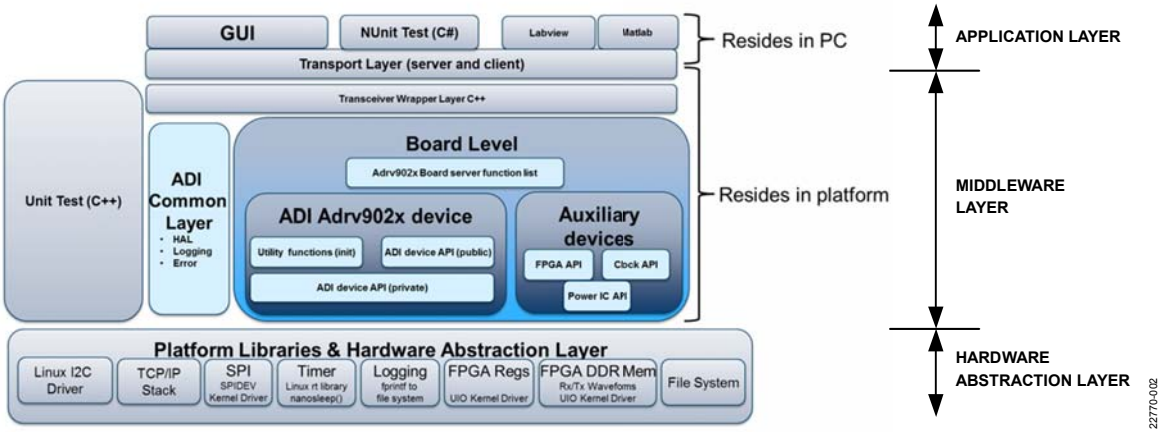


Figure 2. ADRV9025 API Software Architecture (Analog Devices Evaluation Platform)

Devices Folder (/c_src/devices)

The devices folder (**/c_src/devices**) includes the main API code for the transceiver as well as the Analog Devices clock chip AD9528 (**/ad9528** folder). The **/adrv9025** folder contains the high level function prototypes, data types, macros, and source code to build the final user software system. The user is strictly forbidden to modify the files contained in the **/adrv9025** and **/ad9528** folders. Note that software support cannot be provided if these files have been modified. Analog Devices maintains this code. The only exception is that the developer may modify user-selectable #define macros such as ADI_ADRV9025_VERBOSE mode to enable or disable API logging, and user configurable macros defined in **/adrv9025/public/include/adi_adrv9025_user.h**.

Platforms Folder (/c_src/platforms)

The platforms folder, named **/c_src/platforms**, provides the means for a developer to insert custom platform hardware driver code for system integration with the API. The **adi_platform.c/.h** files contain function pointers and the required prototypes necessary for the API to work correctly. It is important that the function prototypes in **adi_platform.c** do not change. The developer is responsible for implementing the code for each **adi_platform.c** function to insure the correct hardware drivers are called for the platform hardware of the user. In the example code provided by Analog Devices in **adi_platform.c**, the function pointers are assigned to call the Analog Devices ADS9 platform functions. To allow for easy platform swapping, Analog Devices maintains a generic implementation of **adi_platform.c**. To support another platform, assign the function pointers in **adi_platform.c** to call the platform functions specific for the platform hardware of the user.

API doxygen (adrv9025.chm) File (/c_src/doc)

This folder contains the device API doxygen (**adrv9025.chm**) file for user reference. It is in compressed HTML format. For security reasons, .chm files can only be opened from a local drive. If you attempt to open from a network drive, the file may look empty.

PRIVATE vs. PUBLIC API FUNCTIONS

The API is made up of multiple .c and .h files. The API is written in C, so there are no language modifiers to identify a function as private or public as commonly used in object-oriented languages. Per the Analog Devices coding standard, public API functions are denoted by the function name prepended with adi_adrv9025_FunctionName(). The application layer is free to use any API function prepended with the adi_adrv9025_ naming. Private helper functions lack the adi_ prefix, and are not intended to be called by the customer application.

Most functions in the API are prefixed with adi_adrv9025_ and are for public use. However, many of these functions are never called directly from the application layer of the developer. Utility functions that abstract some common operations, specifically initialization of the device, are provided in adi_adrv9025_utility.c. For this reason, the majority of the initialization and other helper functions have been separated from the top level **adi_adrv9025.c/adi_adrv9025.h** files to help the developer focus on the most commonly and widely used functions by the application layer program.

HARDWARE ABSTRACTION LAYER

The hardware abstraction layer (HAL) interface is a library of functions that the transceiver API uses when it needs to access the target platform hardware. The implementation of this interface is platform dependent and needs to be implemented by the end user in **adi_platform.c**. The current **adi_platform.c** provides example code that calls the HAL functions for the ADS9 evaluation platform specific functions.

The adi_platform.c HAL functions are function pointers that must be initialized by creating a customer supplied, platform specific function and pointing the associated HAL function pointer to the customer supplied function.

A snippet is given here from the **adi_platform.c** provided for the ADS9 mother board demonstrating assignment of adi_hal_ function pointers to ADS9 specific functions:

```
adi_hal_HwOpen = ads9_HwOpen;
adi_hal_HwClose = ads9_HwClose;
adi_hal_HwReset = ads9_HwReset;
adi_hal_DevHalCfgCreate = ads9_DevHalCfgCreate;
adi_hal_DevHalCfgFree = ads9_DevHalCfgFree;

adi_hal_SpiInit = ads9_SpiInit;
adi_hal_SpiWrite = ads9_SpiWrite_v2;
adi_hal_SpiRead = ads9_SpiRead_v2;

adi_hal_LogFileOpen = ads9_LogFileOpen;
adi_hal_LogLevelSet = ads9_LogLevelSet;
adi_hal_LogLevelGet = ads9_LogLevelGet;
adi_hal_LogWrite = ads9_LogWrite;
adi_hal_LogFileClose = ads9_LogFileClose;



adi_hal_Wait_us = ads9_TimerWait_us;
adi_hal_Wait_ms = ads9_TimerWait_ms;

/* only required to support the ADI FPGA*/
adi_hal_BbicRegisterRead   = ads9_BbicRegisterRead;
adi_hal_BbicRegisterWrite  = ads9_BbicRegisterWrite;
adi_hal_BbicRegistersRead  = ads9_BbicRegistersRead;
adi_hal_BbicRegistersWrite = ads9_BbicRegistersWrite;
```

Hardware Functions

Access to the SPI controller that communicates with the Analog Devices transceiver is required. The SPI details are illustrated in the Serial Peripheral Interface (SPI) section of this document. In addition, control of the hardware reset signal that controls the $\overline{\text{RESET}}$ pin is required. This is usually implemented using a platform processor GPIO. Refer to the target platform schematic and transceiver data sheet for more details of the $\overline{\text{RESET}}$ pin.

Logging Functions

The API provides a simple logging feature function that may be enabled for debugging purposes. This feature requires an implementation for the adi_hal_LogWrite function. The APIs optionally call to send debug information to the system via the HAL. The function adi_hal_LogLevelSet, may be used to configure HAL flags to configure how the HAL processes the various message types from the API layer. Analog Devices transceiver open-hardware function, adi_hal_HwOpen calls this function to set the desired logging operation. Available logging levels are given by adi_common_LogLevel_e, as shown in Table 1.

Table 1. Logging Levels

Function Name	Purpose
ADI_COMMON_LOG_NONE	All types of log messages not selected
ADI_COMMON_LOG_MSG	Log message type
ADI_COMMON_LOG_WARN	Warning message type
ADI_COMMON_LOG_ERR	Error message type
ADI_COMMON_LOG_API	API function entry for logging purposes
ADI_COMMON_LOG_API_PRIV	Private API function entry for logging purposes
ADI_COMMON_LOG_BF	BF function entry for logging purposes
ADI_COMMON_LOG_HAL	Analog Devices HAL function entry for logging purposes
ADI_COMMON_LOG_SPI	SPI transaction type
ADI_COMMON_LOG_ALL	All types of log messages selected

Multiple Device Support

For applications with multiple transceivers, the HAL layer requires a reference to the targeted device and its hardware particulars, for example SPI chip select, reset signal. The HAL function prototypes first parameter, void* devHalCfg, provides the platform layer functions with device specific settings such as SPI chip select, log file names, and so forth. The devHalCfg pointer is void to the device API layer because the device API layer has no knowledge of the platform. This allows each platform to use a different devHalCfg structure that properly represents the specific hardware on the platform.

Note for the Analog Devices transceiver API: there is a requirement that only one thread may control and configure a specific device instance at any given time.

devHalInfo

To pass a target device information from the application to the adi_platform.c HAL functions, the API layer passes a void pointer parameter, called devHalInfo. This void pointer shall act as a state container for the relevant hardware information for a particular device. Note that within the platform layer (adi_platform.h), this is the same as devHalCfg.

The API user must define this state container as per system HAL implementation requirements. User may implement any structure to pass any hardware configuration information that the hardware requires between application layer and platform layer. This state container may be used to transfer device reference information in multi-threaded and multi-transceiver systems.

The application passes the device state container, devHalInfo, via the API transceiver device structure, for example the adi_adrv9025_Device_t. The API function does not read or write the (void *) devHalInfo but passes it as a parameter to all HAL function calls.

Table 2. HAL Interface Functions for User Integration

Function Name	Purpose
adi_hal_HwOpen	Open and initialize all platform drivers/resources and peripherals required to control the transceiver device (for example, SPI, timer, and logging)
adi_hal_HwClose	Close any resources opened by adi_hal_HwOpen
adi_hal_HwReset	Toggle the hardware reset signal for the transceiver device
adi_hal_SpiWrite	Write an array of data bytes on a targeted SPI device (address bytes are packed into the byte array before calling this function)
adi_hal_SpiRead	Read an array of data bytes from a targeted SPI device (address bytes are provided by a TxData array, which are packed into the byte array before calling this function)
adi_hal_Wait_us	Perform a wait/thread sleep in units of microseconds
adi_hal_Wait_ms	Perform a wait/thread sleep in units of milliseconds
adi_hal_LogFileOpen	Open a file for logging
adi_hal_LogLevelSet	Mask to set the severity of information to write to the log (Error/Warning/Message)
adi_hal_LogLevelGet	Get the current log level setting
adi_hal_LogWrite	Log a debug message (message, warning, error) from the API to the platform log
adi_hal_LogFileClose	Function to close the log file
adi_hal_DevHalCfgCreate	This function allows the platform to allocate and configure the devHalCfg structure
adi_hal_DevHalCfgFree	This function allows the platform to deallocate the devHalCfg structure
adi_hal_BbicRegisterRead	This function is used to communicate with the baseband processor (FPGA)
adi_hal_BbicRegisterWrite	This function is used to communicate with the baseband processor (FPGA)
adi_hal_BbicRegistersRead	This function is used to communicate with the baseband processor (FPGA)
adi_hal_BbicRegistersWrite	This function is used to communicate with the baseband processor (FPGA)

SOFTWARE INTEGRATION

The ADRV9025 API package was developed on the Analog Devices ADS9 reference platform utilizing a Xilinx MicroZed running a Linux variant. This section describe how to use the provided API in a custom hardware/software environment. This is readily accomplished because the API was developed abiding by ANSI C constructs while maintaining Linux system call transparency. The ANSI C standard was followed to ensure agnostic processor and operating system integration with the API code.

SOFTWARE INTEGRATION PROCESS OVERVIEW

The following steps can be followed to integrated Analog Devices API into functioning system software.

- Transceiver Device API Integration: The API source code can be integrated into the radio system software deployed on the baseband processor to control the Analog Devices transceiver operations.
- Integration of Transceiver Specific Files: Platform files which are necessary for the Analog Devices transceiver to function are added to the system software.
- Integration of Drivers in Hardware Abstraction Layer: The API software provided by Analog Devices communicates with the transceiver through a SPI interface, accessed via Hardware Abstraction Layer (HAL). The references to the SPI driver need to be updated by the user in the HAL.
- Compilation and Programming: Once the files required for software integration are available, the device API can be compiled, and the transceiver specific platform files programmed into the transceiver.

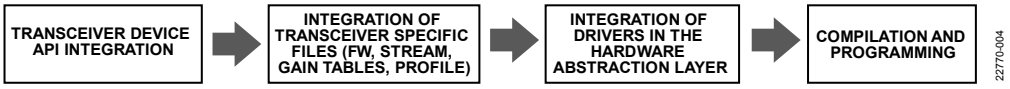


Figure 3. Software Integration Process Steps

SOFTWARE PACKAGE FOLDER STRUCTURE OVERVIEW

The software package delivered follows the structure shown in Figure 4. The software package consists of 4 main folders:

- API—contains the API C source code for the ADRV902x family of transceiver devices.
- Firmware—contains the firmware binaries generated for the embedded Arm processor core in the ADRV902x family devices.
- Gain Tables—contains the receiver gain table, receiver gain compensated gain table, and the transmit path attenuation table used by the ADRV902x family devices.
- GUI—contains an installation package for the Transceiver Evaluation Software, which can be used to evaluate the transceiver, and generate important platform files such as the stream and the use case profile used to initialize the device.

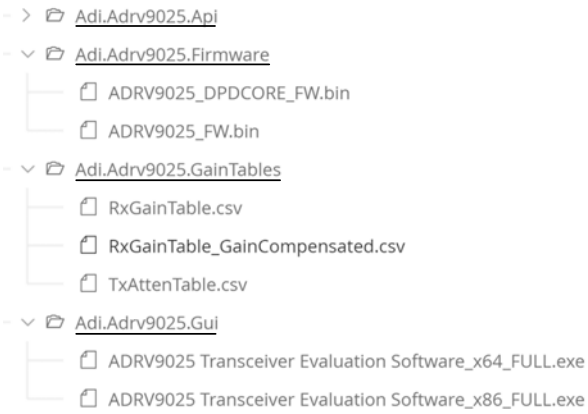


Figure 4. Software Package Folder Structure

API SOFTWARE ARCHITECTURE

The API architecture is implemented as 3 main layers as shown in Figure 5. This section describes how to use the API in a custom hardware/software environment. This is readily accomplished because the API was developed abiding by ANSI C constructs while maintaining Linux system call transparency. The ANSI C standard was followed to ensure agnostic processor and operating system integration with the ADRV902x transceiver family-based API code.

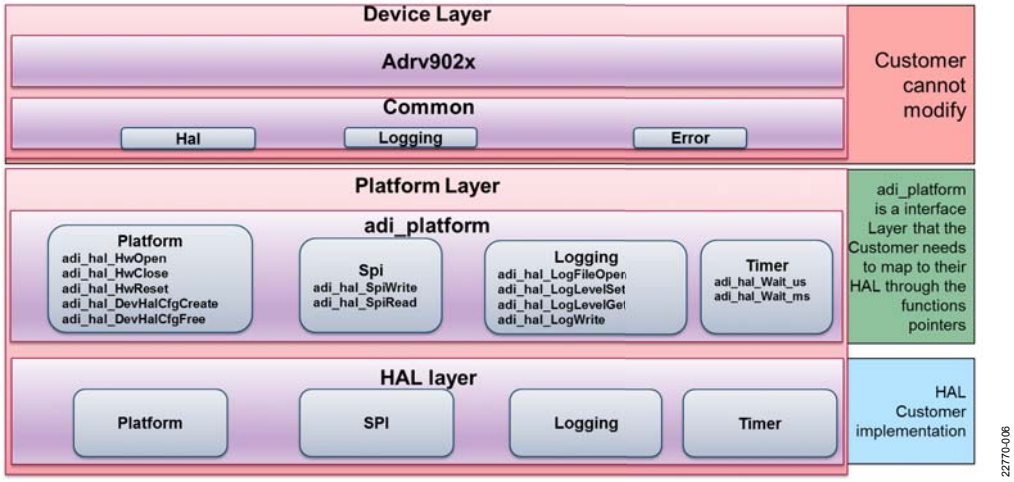


Figure 5. Software Integration

IMPLEMENTING HARDWARE ABSTRACTION INTERFACE

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 Analog Devices evaluation platform. The Analog Devices HAL interface is a library of functions that the API uses when it needs to access the target platform hardware. The Analog Devices HAL is defined by adi_platform.h. The implementation of this interface is platform dependent and shall be implemented by the developer in a platform specific subfolder. The prototypes of the required functions defined in adi_platform.h may not be modified, as this breaks the API.

Refer to Table 2 for the functions required by the HAL interface for integration.

DEVELOPING THE APPLICATION

The /c_src/app/main.c file provides a user example demonstrating top-level initialization. The example application was written to demonstrate initialization of one device, initialize the transmitter, and provide examples of calling the HAL functions and key initialization functions such as adi_adrv9025_PreMcsInit_v2. Initialization of the transmitter and loading of the adi_adrv9025_Init_t structure are omitted from the example code contained here for brevity. The example project also demonstrates how to load the adi_adrv9025_Init_t structure from a JSON file or using initdata.c files.

The user application needs to allocate and clear the device and init structures. The adi_adrv9025_Device_t data structure is used to describe or point to a particular device. The adi_adrv9025_Init_t structure is used to contain the init profile of the user.

An adi_adrv9025_Device_t pointer to the specific device instance is as follows:

```
typedef struct adi_adrv9025_Device
{
    adi_common_Device_t          common;
    adi_adrv9025_Info_t          devStateInfo;
    adi_adrv9025_SpiSettings_t   spiSettings;
} adi_adrv9025_Device_t;

typedef struct adi_adrv9025_Init
{
    adi_adrv9025_ClockSettings_t        clocks;
    adi_adrv9025_GpInterruptSettings_t        gpInterrupts;
    adi_adrv9025_RxSettings_t           rx;
```

```
    adi_adrv9025_TxSettings_t           tx;
    adi_adrv9025_DataInterfaceCfg_t     dataInterface;
} adi_adrv9025_Init_t;
```

To support multiple ADRV9026 devices, the application layer code needs to instantiate multiple adi_adrv9025_Device_t structures to describe each physical device. Multiple devices can have their own adi_adrv9025_Init_t structure instance, or share a common init structure if they are configured the same.

The devHalInfo is defined as a void pointer and allows the user to define and pass any platform hardware settings to the platform HAL layer functions. For example, devHalInfo might contain information such as the SPI chip select to be used for the physical ADRV9026 device. The API does not use the devHalInfo member, and therefore does not define the information it contains. Note that the API functions are shared across all instances of physical ADRV9026 devices. The devHalInfo structure defined by the developer identifies which physical ADRV9026 device is targeted (SPI chip select) when a particular API function is called. The developer may need to store other hardware information unique to a particular ADRV9026 device in this structure such as timer instances, log file information to allow for multithreading. It is expected that only one thread uses the API to a particular ADRV9026 device.

The devStateInfo member is of type adi_adrv9025_Info_t and is a runtime state container for the API. The application layer must allocate memory for this structure, but only the 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 (for example, has it been initialized and Arm loaded), as well as a debug store for any run-time data, such as error codes and error sources. It is not intended for the application layer to access the devStateInfo member directly, as API functions are provided to access the last error code and source information.

The adi_adrv9025_Init_t structure is used to contain the customer profile initialization settings to configure an ADRV9026 device. This init structure is passed to the API init functions during the initialization phase. This structure contains the receiver/transmitter/observation receiver profile settings, system clock settings, JESD204B/JESD204C settings, and transceiver specific SPI slave controller settings. The application layer passes a pointer to an instance of the adi_adrv9025_Init_t structure for a particular ADRV9026 device to handle the majority of the device core initialization. After initialization is complete, the adi_adrv9025_Init_t structure may be disposed of or deallocated if desired.

```
#include <stdio.h>

#include "adi_platform.h"
#include "adi_adrv9025_utilities.h"
#include "adi_adrv9025.h"
#include "adi_adrv9025_radioctrl.h"

static void adi_LoadADRV9025InitStructUseCase24(adi_adrv9025_Init_t *init);
static int32_t adi_ADRV9025InitExample(adi_adrv9025_Device_t *adrv9025Device);
static int32_t adi_ADRV9025EnableTxExample(adi_adrv9025_Device_t *adrv9025Device);
int main()
{
     int32_t recoveryAction = 0;
     adi_adrv9025_Device_t adrv9025Device = {0} ;
     adi_ADRV9025InitExample(&adrv9025Device);
     adi_ADRV9025EnableTxExample(&adrv9025Device_) ;
     recoveryAction = adi_adrv9025_HwClose(&adrv9025Device);
     if (recoveryAction != ADI_ADRV9025_ACT_NO_ACTION)
     {
           printf("Failed closing platform hardware drivers\n");
           return -1;
     }
    adi_hal_DevHalCfgFree(adrv9025Device.devHalInfo);
   return 0;
```

```
}

static int32_t adi_ADRV9025InitExample(adi_adrv9025_Device_t *adrv9025Device)
{
                int32_t recoveryAction = 0;

                printf("Example Init sequence for ADRV9025\n");

                if (adrv9025Device == NULL)
                {
                                printf("NULL ADRV9025 device pointer\n");
                                return -1
                }

                adi_adrv9025_Init_t adrv9025Init = {0};

                /* Platform layer function adi_hal_DevHalCfgCreate allocates platform specific
settings structure for SPI
                     driver, logging, etc (per device)*/
               void *adrv9025hal = adi_hal_DevHalCfgCreate((ADI_HAL_INTERFACE_SPI |
ADI_HAL_INTERFACE_LOG |
                                                            ADI_HAL_INTERFACE_HWRESET |
ADI_HAL_INTERFACE_TIMER), 0, "adrv9025Log.txt");
                if (adrv9025hal == NULL)
                {
                                printf("Failed allocating platform hardware settings
structure\n");
                                return -1;
                }
                adrv9025Device->devHalInfo = adrv9025hal;

                /* Load ADRV9025 init structure */
                adi_LoadADRV9025InitStructUseCase24(&adrv9025Init);

                recoveryAction = adi_adrv9025_HwOpen(adrv9025Device);
                if (recoveryAction != ADI_ADRV9025_ACT_NO_ACTION)
                {
                                printf("Failed opening platform hardware drivers\n");
                                return -1;
                }

                /* Initialize ADRV9025 */
                recoveryAction = adi_adrv9025_PreMcsInit_v2(adrv9025Device,
                                                &adrv9025Init,

"/home/analog/adrv9025_server/resources/Tokelau_M4.bin",
                                                "/home/analog/adrv9025_server/resources/stream_imag
e.bin",
```

```
"/home/analog/adrv9025_server/resources/RxGainTable.csv",

"/home/analog/adrv9025_server/resources/TxAttenTable.csv");

                recoveryAction = adi_adrv9025_PllFrequencySet(adrv9025Device,
ADI_ADRV9025_LO1_PLL, 3500000000);

                return 0;
}
```

Include Files

For each major function block, there are generally three files: adi_feature.c, adi_feature.h, and adi_feature_types.h. For core API functionality, Table 3 shows the mandatory .h header files that must be included in the application layer program. Optional add-on API functions can be included if the application of the developer requires those features. Note: the API places typedef definitions in files with _types postfix such as ADRV9025_types.h. These _types.h files are included within their corresponding .h files and do no need to be manually included in the application layer code.

Note that the ADRV9025_user.h contain the #defines for API timeouts and SPI read intervals which may be set as needed by the customer platform. The user files are the only API files that the developer may change.

Table 3. API Mandatory .h Header Files

Mandatory Include Files	Description
adi_adrv9025.h	Core init functions
adi_adrv9025_error.h	Error extension from common error
adi_adrv9025_arm.h	Arm related functions
adi_adrv9025_cals.h	Calibration related functions
adi_adrv9025_gpio.h	General-purpose input/output (GPIO) related functions
adi_adrv9025_data_interface.h	Data interface related functions, JESD204B/JESD204C
adi_adrv9025_hal.h	Contains prototypes and macro definitions for transceiver specific HAL wrapper functions
adi_adrv9025_radioctrl.h	Functions for controlling the Radio
adi_adrv9025_rx.h	Receiver related functions
adi_adrv9025_tx.h	Transmitter related functions
adi_adrv9025_user.h	API timeout and retry definitions
adi_adrv9025_utilities.h	Higher level utility functions for init, loading Arm and Stream binaries, loading Rx Gain Table, Tx attenuation table (Most require file system support)
adi_adrv9025_version.h	Version structure

Table 4. API Optional .h Files

Optional (Add On) Include Files	Description
adi_adrv9025_agc.h	Add-on receiver AGC functionality

API Error Handling and Debug

Each API function returns an int32_t value representing a recovery action. Recovery actions are divided into:

- Warning actions are those that don't have an impact at the time of executing the device API, but can cause performance issues or logging problems. The value of this actions are positive.
- Error actions are those that cause API not to be able to run and an action is required for API to go back to a good state. The value of this actions are negative.

The API error structure that is accessed via device.error contains various members to narrow the action to be taken.

- errSource: current source of error detected, indicating the source file where the error.
- errCode: current error code,
- errLine: line of the source code where the error was returned
- errFunc: function name where the error occurred
- errFile: file name where the error occurred
- varName: variable name which has the error
- errorMessage: error message to describe the error
- lastAction: last action detected
- newAction: new action detected

API functions respond by telling the application layer what action needs to be taken due to a possible error in the API function call. The error structure contains further information in order to take the adequate action. The possible recovery action return values are listed in Table 5.

Table 5. API Recovery Actions

Recovery Action Name	Value	Description
ADI_COMMON_ACT_WARN_CHECK_PARAM	+3	API OK: parameter exceeds the range of values allowed
ADI_COMMON_ACT_WARN_RERUN_FEATURE	+2	API OK: rerun device feature (Arm init cals)
ADI_COMMON_ACT_WARN_CHECK_INTERFACE	+1	API OK: log not working, this is a warning device programing can continue, upper layer must decide action to be taken
ADI_COMMON_ACT_NO_ACTION	0	API function completed: no error handling action is required.
ADI_COMMON_ACT_ERR_CHECK_TIMER	−1	API OK: timer not working
ADI_COMMON_ACT_ERR_CHECK_PARAM	−2	API OK: invalid parameter detected in API
ADI_COMMON_ACT_ERR_RESET_INTERFACE	−3	API NG: interface Not Working, device cannot be program or access, timer/I²C/SPI/data interface
ADI_COMMON_ACT_ERR_RESET_FEATURE	−4	API NG: reset device feature (for example, arm init cals)
ADI_COMMON_ACT_ERR_RESET_MODULE	−5	API NG: module of device not working (arm not accessible)
ADI_COMMON_ACT_ERR_RESET_FULL	−6	API NG: full system reset required

The actions can be divided into different blocks: parameter, feature, module, interface, and device.

Parameter:

- Parameter either passed to function or member of structure
- This action can be assigned when we want to set a feature/Module/Interface and it is not configure correctly

Feature: (parts of a module or device)

- GPIO control for transmitter attenuation
- GP interrupt
- Arm initial calibrations
- Arm tracking calibrations
- Arm control
- AGC control
- PA protection

Module: (individual blocks that are contained in the device that are to contain features)

- Arm
- Caching/merging/streaming…

Interface:

- Devices interface
- SPI/I²C/data interface
- Log

Device:

- Target device

API Recovery Action: ADI_COMMON_ACT_NO_ACTION

The ADI_COMMON_ACT_NO_ACTION API recovery action is returned when an API function completes. There is no recovery action to be performed.

API Recovery Action: ADI_COMMON_ACT_WARN_RERUN_FEATURE

The ADI_COMMON_ACT_WARN_RERUN_FEATURE recovery action is returned when the API detects a failure in any of the device features.

If a tracking calibration error is detected, it usually is not a catastrophic error, usually resulting in degraded radio performance. The application layer attempts to recover by resetting the tracking calibration.

If the API detects an error with the transceiver init calibrations, at this point the error severity is high enough that re-running all init calibrations is required. A full transceiver device reset is not required. It is also not required to reload the Arm firmware of the device.

Suggested application layer action:

1. Set PA and other RF front-end components in powered down / init state.
2. Call adi_adrv9025_ErrorCodeGet() to determine the specific ADIHAL error code and verify ADIHAL is the error source. Log error code and source.
3. Read Arm calibration status to log debug information on calibration failure - call adi_adrv9025_InitCalDetailedStatusGet()
4. Call adi_adrv9025_InitCalsRun() to re-run the init calibrations.
5. Call adi_adrv9025_InitCalsWait () and adi_adrv9025_InitCalDetailedStatusGet () to confirm that there is no error in init calibrations.

API Recovery Action: ADI_COMMON_ACT_WARN_CHECK_INTERFACE

The ADI_COMMON_ACT_WARN_CHECK_INTERFACE API recovery action is returned if the adi_platform has return an error in any interface. Further information can be extracted by reading the error structure which contains extended information about the error.

The following are possible scenarios for a check interface action.

Issue: Logging Interface When the Log File Cannot Be Opened Or Written to

The API layer does not return this as an error because it does not directly affect transceiver performance. In addition, this recovery action does not prevent the API function from completing. Analog Devices suggests that the application layer attempt to close the log file and reopen to resolve the log file access issue.

Issue: Baseband Processor GPIO Failed to Operate Correctly, but the API Circumvented the Error by Using the SPI port or Other Control Mechanism

Because the API was able to complete the API function, the issue is not critical, but the application layer attempts to debug and fix the issue reported by the adi_common layer with respect to the baseband processor GPIO control. The device.common.error contains the information for decoding the error, the application layer can use it to debug the root cause of the error further.

Issue: adi_common Returns an Error Reporting that the Timer Is Not Working as Expected

The API uses the timer adi_common functions to perform thread blocking waits to insure that the API does not poll the SPI bus with 100% utilization.

If the timer is reporting an error from the adi_common, it is possible that the API function works correctly, but there may be an impact on the system due to incorrect usage of system resources.

Issue: adi_common layer reports a HAL error while attempting to control the baseband processor GPIO pins

If the API function cannot circumvent the error, this action is returned. If the API can circumvent the error, only a warning is returned. Currently, the only baseband processor GPIO pin used in the adi_common is to reset the transceiver device ($\overline{\text{RESET}}$ pin).

If this error is reported, the application layer attempts to reset the baseband processor GPIO pins that are used within the adi_common layer of code. If the application layer can resolve the GPIO hardware driver issue, normal operation of the API can resume by retrying the failed API function.

Suggested application layer actions:

- Attempt to reset interface.
- Continue use of API monitoring for future check interface recovery action reports.
- If continued reports of ADI_COMMON_ACT_WARN_ CHECK_INTERFACE, a system diagnostic may be required for the particular hardware.

API Recovery Action: ADI_COMMON_ACT_ERR_CHECK_PARAM

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

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

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

API Recovery Action: ADI_COMMON_ACT_ERR_CHECK_DEVICE

The ADI_COMMON_ACT_ERR_CHECK_DEVICE recovery action is returned when the device detected is not compatible with the API being executed.

API Recovery Action: ADI_COMMON_ACT_ERR_RESET_INTERFACE

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

Suggested application layer action:

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

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

Suggested application layer action:

1. Record the error.
2. Perform any baseband processor determined recovery actions.

API Recovery Action: ADI_COMMON_ACT_ERR_RESET_FEATURE

The ADI_COMMON_ACT_ERR_RESET_FEATURE API recovery action is returned by the API when an error has been detected that required the reset of a feature of the device. To reset the feature a reconfiguration of same has to be performed.

API Recovery Action: ADI_COMMON_ACT_ERR_RESET_MODULE

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

- Arm processor module that requires a complete reset and reload of the Arm firmware. This type of action might be required if the communication interface to the ADRV9026 Arm processor fails or the Arm watchdog timer reports an error. These events are not expected in production code, but are failsafe mechanisms in the event of a catastrophic error.
  - Issue adi_adrv9025_RxTxEnableSet() to disable transmitter to keep hardware in a safe state. If this fails, a full transceiver reset is required.
  - Set PA and other RF front-end components in powered down / init state.
  - Call adi_adrv9025_ErrorCodeGet() to determine the specific error code and verify the error source. Log error code and source.
  - Dump Arm memory if necessary for debug.
    - Dump SPI registers if necessary for debug.
    - Reload the stream processor and Arm binary firmware files.
    - Continue with normal init sequence to run init calibrations and enable tracking calibrations.

API Recovery Action: ADI_COMMON_ACT_ERR_RESET_DEVICE

The ADI_COMMON_ACT_ERR_RESET_DEVICE recovery action is returned if an API function cannot complete due to a detected error. If the API cannot correct or circumvent the error, and the severity of the error requires a complete reset of the transceiver device, this action is returned.

Suggested application layer action:

1. Put system hardware in safe state.
   a. Set PA and other RF front end components in powered down / init state
   b. Hard reset ADRV9026 device (adi_adrv9025_HwReset())
2. Read API error code information for debug.
   a. Dump Arm memory if necessary
   b. Dump SPI registers if necessary
3. Reinitialize transceiver using normal full initialization sequence.

Restrictions

Developers may not modify any code located in /c_src/devices/* folder other than changing the adi_platform.c, adi_platform.h code bodies for hardware driver insertion. Analog Devices maintains the code in /c_src/devices/adrv9025 and /c_src/devices/ad9528. Analog Devices provides new releases to fix any code bugs in these folders.

No direct SPI read/write operation is permitted when configuring the transceiver or Analog Devices clock chip device. Only use the high-level API functions defined in /c_src/devices/ad9528/ad9528.h or other public .h files. Do not directly use any SPI read/write functions in the application layer code for transceiver configuration or control. Analog Devices does not support any customer code containing SPI writes reverse-engineered from the original API.

Multiple Thread and Multiple Transceiver Application Considerations

For applications with multiple transceivers, the API requires a reference to the targeted device and its hard and soft particulars – for example, SPI chip-select, reset and configuration status. The adi_adrv9025_Device_t structure is used to identify each instance of a physical transceiver device.

For multi-threaded applications, there is a requirement that a particular device may only be controlled and configured by a single thread. Concurrent thread configuration of the same instance of a physical transceiver device is **not supported** by the API.

Delays, Waits and Sleeps

A small number of APIs require some time to allow the hardware to complete internal configurations, for example, adi_adrv9025_PllFrequencySet(). These APIs request the system to perform a wait or sleep by calling the HAL interface function adi_hal_Wait_us/adi_hal_Wait_ms. If the HAL interface implementation of the target platform chooses to implement a thread sleep, it is not permitted for the application to call another API targeting the same transceiver device. The application is required to wait/sleep and the API to complete before continuing with the configuration of the device.

Table 6 lists the wait/sleep period used by the API. They are defined in adi_adrv9025_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 time-out period, the status of the transceiver is polled. The frequency of the polling the status during this timeout period may be modified by the user by adjusting the value of the polling interval.

Note that these recommendations may change once evaluation of the device is fully complete.

Table 6. API internal Wait/Sleep Intervals

Wait/Sleep Reference	Purpose	Recommended Timeout Period Per µs	Recommended Poll Interval Per µs
VERIFY_ARM_CHKSUM_XXX	Calculation of arm checksum	200000	5000
CLKPLL_CPCAL_XXX	Internal clock and PLL configuration	1000000	100000
CLKPLL_LOCK_XXX	Internal clock and PLL locking period	1000000	100000
SETARMGPIO_XXX	Update arm information on GPIOs for TDD pin control	1000000	100000
SETRFPLL_XXX	Configure RF PLL frequency	1000000	100000
GETRFPLL_XXX	Retrieve RF PLL frequency	1000000	100000
ABORTINITCALS_ XXX	Abort initial calibrations	1000000	100000
GETINITCALSTATUS_XXX	Retrieving initial calibrations status	1000000	100000
RADIOON_XXXS	Enabling radio transmit and receive	1000000	100000
READARMCFG_XXX	Reading arm configurations	1000000	100000
WRITEARMCFG_XXX	Updating arm configurations	1000000	100000
RADIOOFF_XXX	Disabling radio transmit and receive	1000000	100000
ENTRACKINGCALS_XXX	Enabling tracking calibrations	1000000	100000
RESCHEDULETRACKINGCALS_XXX	Schedule a tracking calibration to run	1000000	100000
SETTXTOORXMAP_	Set Tx to ORx external signal routing	1000000	100000
GETTXLOLSTATUS_	Status of TxLOL external tracking cal	1000000	100000
GETTXQECSTATUS_	Status of Tx QEC tracking cal	1000000	100000
GETRXQECSTATUS_	Status of Rx QEC tracking cal	1000000	100000
GETORXQECSTATUS_	Status of ORx QEC tracking cal	1000000	100000
GETRXHD2STATUS_	Status of Rx HD2 tracking cal	1000000	100000
SENDARMCMD_XXX	Sending requests to arm firmware	2000000	100000
GETTEMPERATURE_	Read current temperature	1000000	100000
GETARMBOOTUP_	Waiting for arm to boot up	1000000	100000

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 functions of the device. This section is mainly an information-only section meant to give the user an understanding of 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 BUS SIGNALS

The SPI bus consists of the following signals:

$\overline{CS}$

$\overline{CS}$ is the active-low chip select that functions as the bus enable signal driven from the baseband processor to the device. This signal is an input to the $\overline{SPI\_EN}$ pin. $\overline{CS}$ 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 $\overline{CS}$ is high. $\overline{CS}$ also frames communication to and from the device and returns the SPI interface to the ready state when it is driven high.

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

SCLK

SCLK is the serial interface reference clock driven by the baseband processor. This signal is an input to the SPI_CLK pin. It is only active while $\overline{CS}$ is low. The minimum SCLK frequency is 10 MHz and the maximum SCLK frequency is 25 MHz. These limits are determined based on the practical timing requirements of the transceiver system and the physical limitations of the device.

SDIO and SDO

When configured as a 4-wire bus, the SPI utilizes two data signals: SDIO and SDO. SDIO is the data input line driven from the baseband processor. The signal input to the device is the SPI_DIO pin. SDO is the data output from the device to the baseband processor in this configuration. The output signal is driven by 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 SPI_SDO and SPI_SDIO pins transition to a high impedance state when the $\overline{CS}$ input is high. The device does not provide any weak pull-ups or pull-downs on these pins. When SPI_SDO is inactive, it is floated in a high impedance state. If a valid logic state on SPI_SDO is required at all times, add an external weak pull-up/down (10 kΩ value) on the PCB.

SPI DATA TRANSFER PROTOCOL

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

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 following information in Table 8.

Table 7. 16-Bit Control Field

MSB	D14:D0
R/$\overline{W}$	A[14:0]

R/$\overline{W}$—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 IO operation.

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

SPI CONFIGURATION USING API FUNCTION

SPI operation is configured via calling adi_adrv9025_SpiCfgSet(). This function is called by the adi_adrv9025_Initialize(), which is called by adi_adrv9025_PreMcsInit_v2().

The input parameters for adi_adrv9025_PreMcsInit_v2() include the init structure which is of type adi_adrv9025_Init_t. The adi_ADRV9025InitExample() function shows an example of configuring a hard-coded init function which includes the SPI related parameters.

Users can configure SPI settings for the device with different SPI controller configurations by configuring member values of the adi_adrv9025_SpiSettings_t data structure. The adi_adrv9025_ SpiSettings_t data structure contains the following:

The parameters for this structure are listed in Table 8. Any value that is not listed in the table is invalid.

```
typedef struct adi_adrv9025_SpiSettings
{
    uint8_t msbFirst;
    uint8_t enSpiStreaming;
    uint8_t autoIncAddrUp;
    uint8_t fourWireMode;
    adi_adrv9025_CmosPadDrvStr_e cmosPadDrvStrength;
} adi_adrv9025_SpiSettings_t;
```

Table 8. SPI Bus Setup Parameters

Structure Member	Value	Function	Default
MSBFirst	0x00	Least significant bit first	0x01
	0x01	Most significant bit first	
enSpiStreaming	0x00	Enable single-byte data transfer mode. All communication between the baseband processor and the device uses this mode. Note: not implemented in the Analog Devices platform layer.	0x00
	0x01	Enable streaming to improve SPI throughput for indirect data transfer using an internal DMA controller. Note: not implemented in the Analog Devices platform layer.	
autoIncAddrUp	0x00	Auto-increment. Functionality intended to be used with SPI streaming. Sets address auto-increment -> next addr = addr – 4. Note: not implemented in the Analog Devices platform layer.	0x01
	0x01	Auto-decrement. Functionality intended to be used with SPI streaming. Sets address auto-decrement -> next addr = addr + 4. Note: not implemented in the Analog Devices platform layer.	
fourWireMode	0x00	SPI hardware implementation. Use 3-wire SPI (SDIO pin is bidirectional). Figure 8 shows example of SPI 3-wire mode of operation. Note: Analog Devices FPGA platform always uses 4-wire mode.	0x01
	0x01	SPI hardware implementation. Use 4-wire SPI. Figure 6 and Figure 7 show examples of SPI 4-wire mode of operation. The default mode for Analog Devices FPGA platform is 4-wire mode.	
cmosPadDrvStrength	0x00	5 pF load at 75 MHz	0x01
	0x01	100 pF load at 100 MHz	

Single-Byte Data Transfer

When enSpiStreaming = 0, a single-byte data transfer is chosen. In this mode, $\overline{CS}$ goes active-low, the SCLK signal activates, and the address is transferred from the baseband processor to the device. This mode is always used in direct communication between the baseband processor and 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 next LSB to MSB. The next bit signifies if 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.

In MSB mode, the first bit transmitted is the R/$\overline{W}$ bit that determines if the operation is a read (set) or write (clear). The MSB of the address is the next bit transmitted from the baseband processor, followed by the remaining 14 bits in order from 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.

Single-byte data transfer can continue in either mode for multiple byte transfers using the transfer format of address followed by data (A D A A D …) until the $\overline{CS}$ signal is driven high. The address must be written for each data byte transfer when using this mode.

Multiple Byte Data Transfer (SPI Streaming)

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

TIMING DIAGRAMS

The diagrams in Figure 6 and Figure 7 illustrate the SPI bus waveforms for a single-register write operation and a single-register read operation, respectively. In the first figure, the value 0x55 is written to register 0x00A. In the second value, register 0x00A is read and the value returned by the device is 0x55. If the same operations are performed with a 3-wire bus, the SDO line in Figure 6 is eliminated, and the SDIO and SDO lines in Figure 7 are combined on the SDIO line. Note that both operations use MSB-first mode and all data is latched on the rising edge of the SCLK signal.

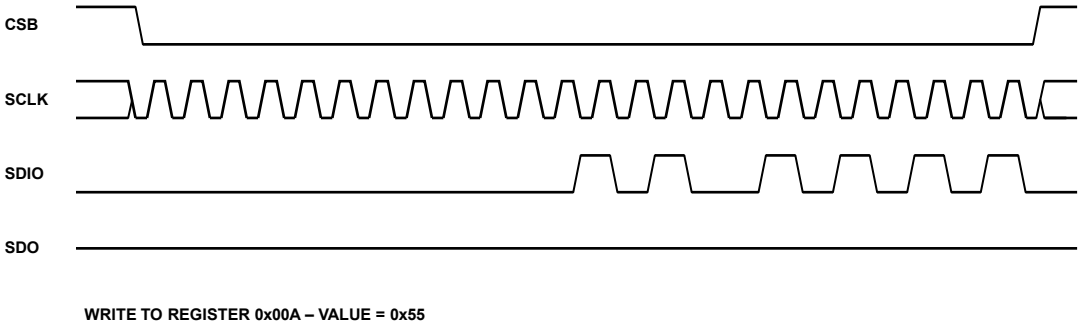


Figure 6. Nominal Timing Diagram, SPI Write Operation

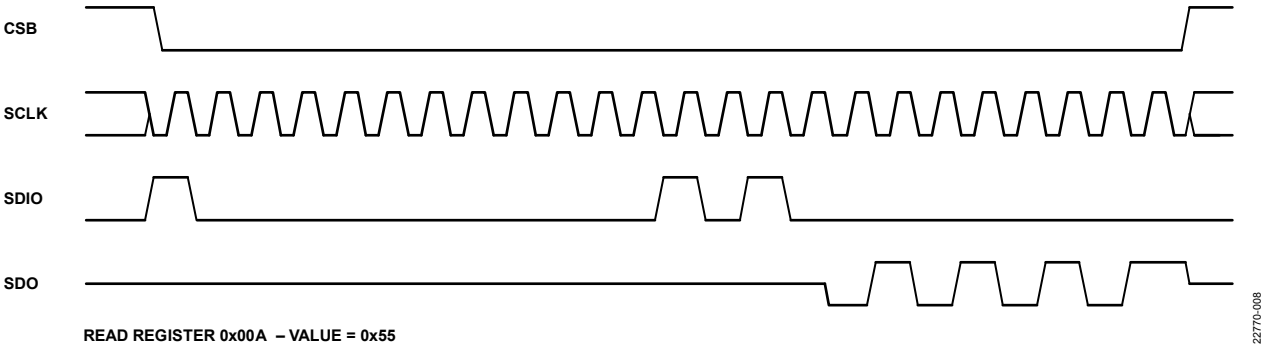


Figure 7. Nominal Timing Diagram, SPI Read Operation

Table 9 lists the timing specifications for the SPI bus. The relationship between these parameters is shown in Figure 8. This diagram shows a 3-wire SPI bus timing diagram with the device returning a value of 0xD4 from register 0x00A and timing parameters marked. Note that this is a single read operation, so the bus-ready parameter after the data is driven from the device (t_{HZS}) is not shown in the diagram.

Table 9. SPI Bus Timing Constraint Values

Parameter	Min	Typ	Max	Description
t_{CP}	40 ns		100 ns	SCLK cycle time (clock period)
t_{MP}	10 ns			SCLK pulse width
t_{SC}	4 ns			$\overline{CS}$ setup time to first SCLK rising edge
t_{HC}	0 ns			Last SCLK falling edge to $\overline{CS}$ hold
t_S	4 ns			SDIO data input setup time to SCLK
t_H	0 ns			SDIO data input hold time to SCLK
t_{CO}	10 ns		16 ns	SCLK falling edge to output data delay (3-wire or 4-wire mode)
t_{HZM}	t_H		$t_{CO\ (max)}$	Bus turnaround time after baseband processor drives the last address bit
t_{HZS}	3 ns		$t_{CO\ (max)}$	Bus turnaround time after device drives the last data bit
t_{INT}			400 ns	Byte to byte delay time during any single read or write operation

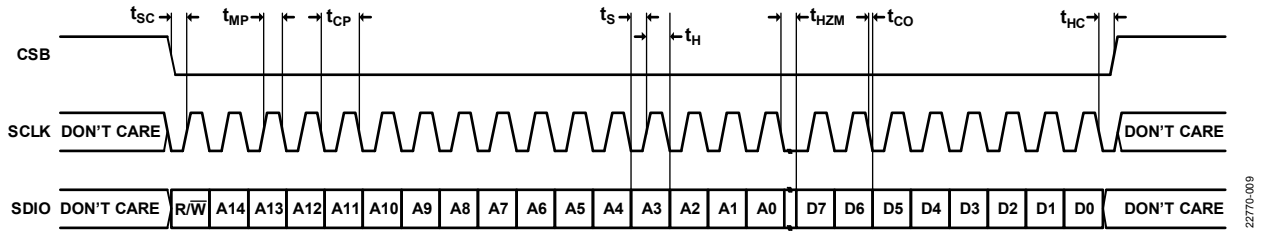


Figure 8. 3-Wire SPI Timing with Parameter Labels

SYSTEM INITIALIZATION

This section provides information about the initialization process for the device utilizing the API developed by Analog Devices. The following sections describe the developer preparation requirements and the initialization sequence. This section does not explain the API library functions. Detailed information regarding the API functions can be found in the API doxygen document (adrv9025.chm) located at /src/doc. This section does not describe API integration and the hardware abstraction Interface. Details of such can be found in the Software Integration section and the Hardware Abstraction Layer section.

INITIALIZATION SEQUENCE

The initialization sequence is comprised of API calls intermixed with user defined function calls specific to the hardware platform. The API functions perform all of the necessary tasks for device configuration, calibration, and control. The user is required to insert their code into the initialization sequence specific to the hardware platform requirements. These platform requirements include but are not limited to: user clock device, user FPGA\ASIC\baseband processor JESD204B interface, data path control, and various system checks governed by the application.

The initialization process consist of the following steps. Some of the steps are done by the Arm. All functions before loading the stream must be write only (use SPI write or bit field write, no SPI read).

Pre MCS initialization:

1. adi_adrv9025_Initialize
    a. Set SPI controller settings
    b. Set master bias
    c. Enable pin pads
    d. Set device clock hsdig divider
    e. Load PFIRs per channel
    f. Load gain tables
    g. Load Tx attenuation tables
    h. Load stream binary
    i. Load arm binary
    j. Write init struct/Rx/Tx profile info into Arm mem
    k. Arm run = 1
    l. Wait for Arm boot to complete
    m. Verify Arm checksum

Arm configuration:

1. Rx/Tx channel configuration (all half-band filter enables, clock dividers)
2. CLKPLL/SERDES PLL configuration
3. JESD204 configuration
4. Arm switches to CLKPLL output once PLL locked

Post MCS initialization:

1. MCS:
    a. Set Arm run = 0
    b. Enable MCS state machine to listen for new SYSREF pulses
    c. Customer sends SYSREF pulses
    d. When MCS state machine complete, Arm run = 1
2. Pass License /cap info to Arm
3. Request Capabilities from Arm
4. Create function vector table (for optional /licensed features)
5. Load factory cal data
6. Run Arm init cals
7. Enable tracking cals
    a. Enable radio control pin mode or not

    Setup any GPIO for Arm/streams

The system is now ready.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

The ADRV9026 employs a SERDES high speed serial interface based on the JESD204B/JESD204C standards to transfer ADC and DAC samples between the device and a baseband processor. The device can support high-speed serial lane rates up to 16.22 Gbps. An external clock distribution solution provides a device clock and SYSREF to both the device and the baseband processor. The SYSREF signal ensures deterministic latency between the device and the baseband processor.

Note that the initialization sequence of the part is critical to guarantee deterministic latency. Specifically, the Arm init calibrations must be run before the JESD links are established, as described in the initialization sequence section of this document. It is also imperative to check the FIFO depth after the link has been established, as described in this section.

Major blocks in the interface include clock distribution, SERDES framer, and SERDES deframer.

JESD204 STANDARD

The JESD204 specification defines four key layers that implement the protocol data stream, as shown in Figure 9. The transport layer maps the conversion between samples and framed, unscrambled octets. The optional scrambling layer scrambles/descrambles the octets, spreading the spectral peaks to reduce EMI. The data link layer handles link synchronization, setup, and maintenance, and encodes/decodes the optionally scrambled octets to/from 10-bit characters in the case of JESD204B (8-bit/10-bit encoding) and 66-bit characters in the case of JESD204C (64-bit/66-bit encoding). The physical layer is responsible for transmission and reception of characters at the bit rate.

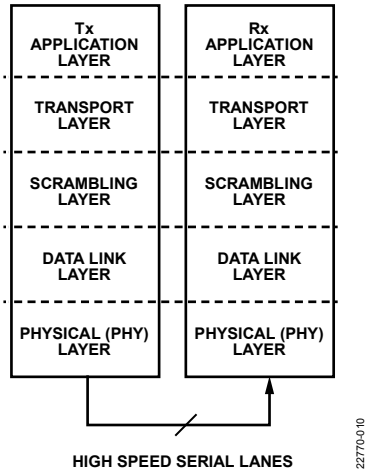


Figure 9. Key Layers of the JESD 204B/C standard

Figure 10 and Figure 11 illustrate how the JESD204 transmit and receive protocols are implemented.

The data interface blocks in the ADRV9026 can operate in either 204B or 204C modes. Fewer number of lanes may be needed when operating in JESD204C, which results in simpler PCB layout and less power consumption.

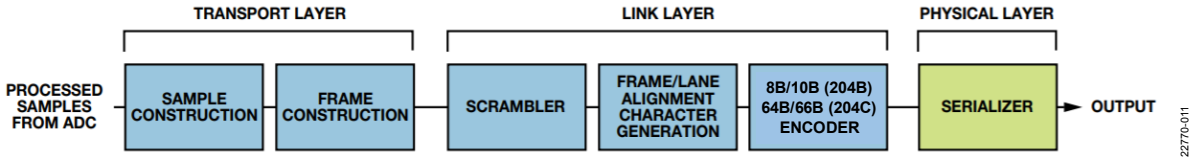


Figure 10 JESD 204B/C Framer (JTX)

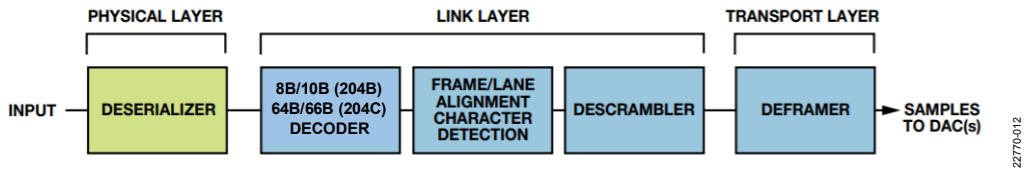


Figure 11 JESD 204 B/C Deframer (JRX)

DIFFERENCES BETWEEN JESD204B AND JESD204C

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

In 8-bit/10-bit link layer, the data is organized into multiframes where in 64-bit/66-bit link layer data is organized into multiblocks of 32 blocks where each block contains 8 octets. In 8-bit/10-bit link layer, phase synchronization is done by Local Multiframe clock (LMFC) where 64-bit/66-bit uses the Local Extended Multiblock Clock (LEMC). In 8-bit/10-bit link layer, LMFC marks multiframe boundaries where in 64-bit/66-bit link layer LEMC is used to mark extended multiblock boundaries. Deterministic latency can be achieved by both local multiframe clock (LMFC) or local extended multiblock clock (LEMC) as per the link layer used.

The 8-bit/10-bit link layer does the alignment between multiple converters by the alignment of their LMFCs to an external signal SYSREF. In 64-bit/66-bit link layer, the alignment between multiple converter devices is done by the alignment of their LEMC to an external signal SYSREF/MULTIREF in Subclass 1. Each converter device can then adjust its LEMC phase to match with the common LEMC of the logic device. The 64-bit/66-bit link layer only supports subclass 1 based LEMC alignment. In this case, the RBD adjustment resolution must not be greater than 255 steps, and if more than one multiframe or multiblock per lane fits in the buffer, the RBD adjustment resolution must be at least 16 steps per multiframe or multiblock. The 64-bit/66-bit link layer also defines a sync header stream, which transmits the information parallel to the user data. This information is encoded using the sync header portion of the 66-bit word block. One sync header per block is decoded to a single bit, and 32 of these bits across a multiblock makes a 32-bit sync word. The sync word can contain the following information:

- Pilot signal (used to mark the borders of the multiblocks and extended multiblocks)
- CRC-3 signal (used for error detection)
- CRC-12 signal (used for error detection)
- FEC signal (used for error detection and correction)
- Command channel (used for transmitting commands and status information)

With the 8-bit/10-bit link layer, JESD204 uses the SYNC interface for synchronization and error reporting where as in 64-bit/66-bit encoding sync headers within the encoded data are used for the synchronization process and the reporting of errors is left to the application layer.

CLOCK DISTRIBUTION

The clock distribution in the ADRV9026 allows the SERDES to be driven by the either by the SERDES PLL or the Clock PLL depending on the use case. Analog Devices provides tested predefined profiles with the appropriate settings so that each use case has a known working setup configurations. For other profile configurations, a Profile Generator application is planned for future release allowing customers to change bandwidths and sampling rates for custom configuration support.

RECEIVER (ADC) DATAPATH

Figure 12 is a block diagram of the transceiver receive side (SERDES framer).

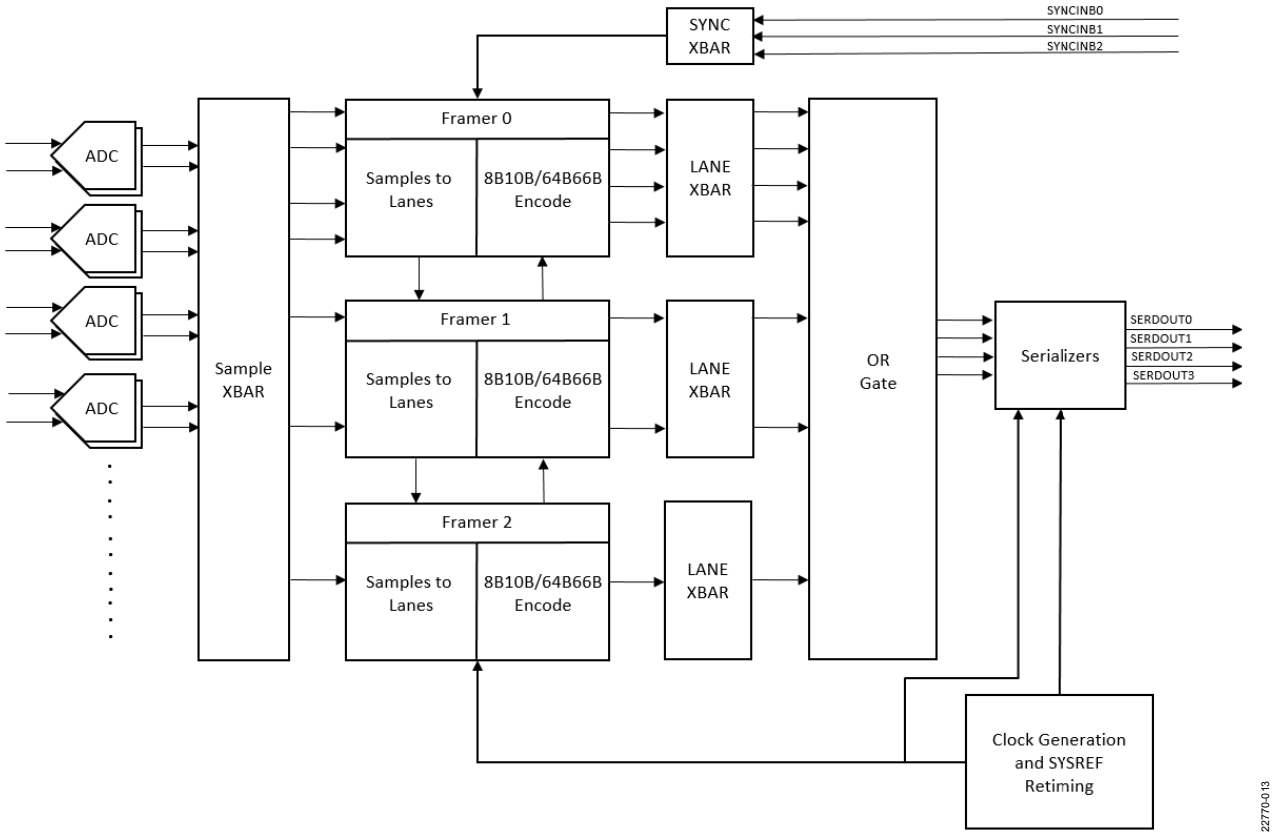
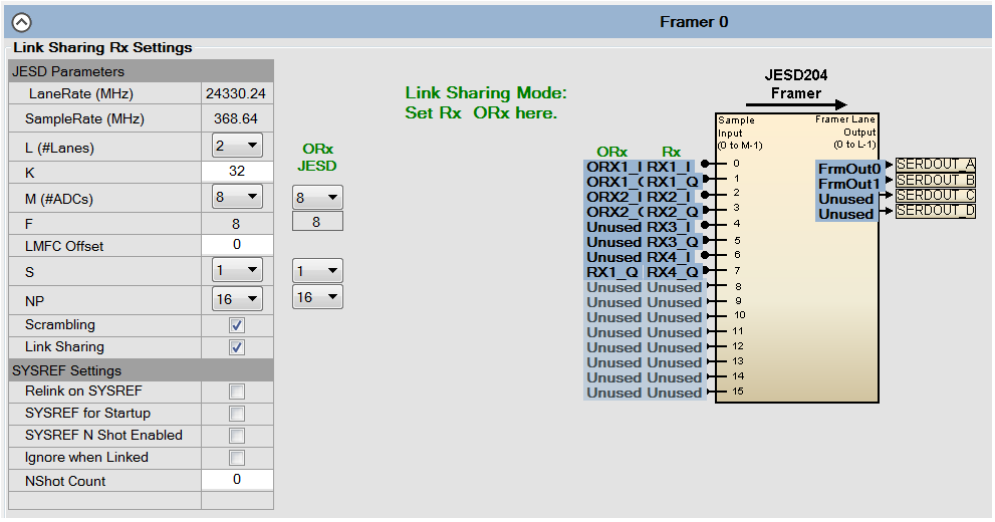


Figure 12. High Level JESD204B/JESD204C Interface Block Diagram (Rx Only)

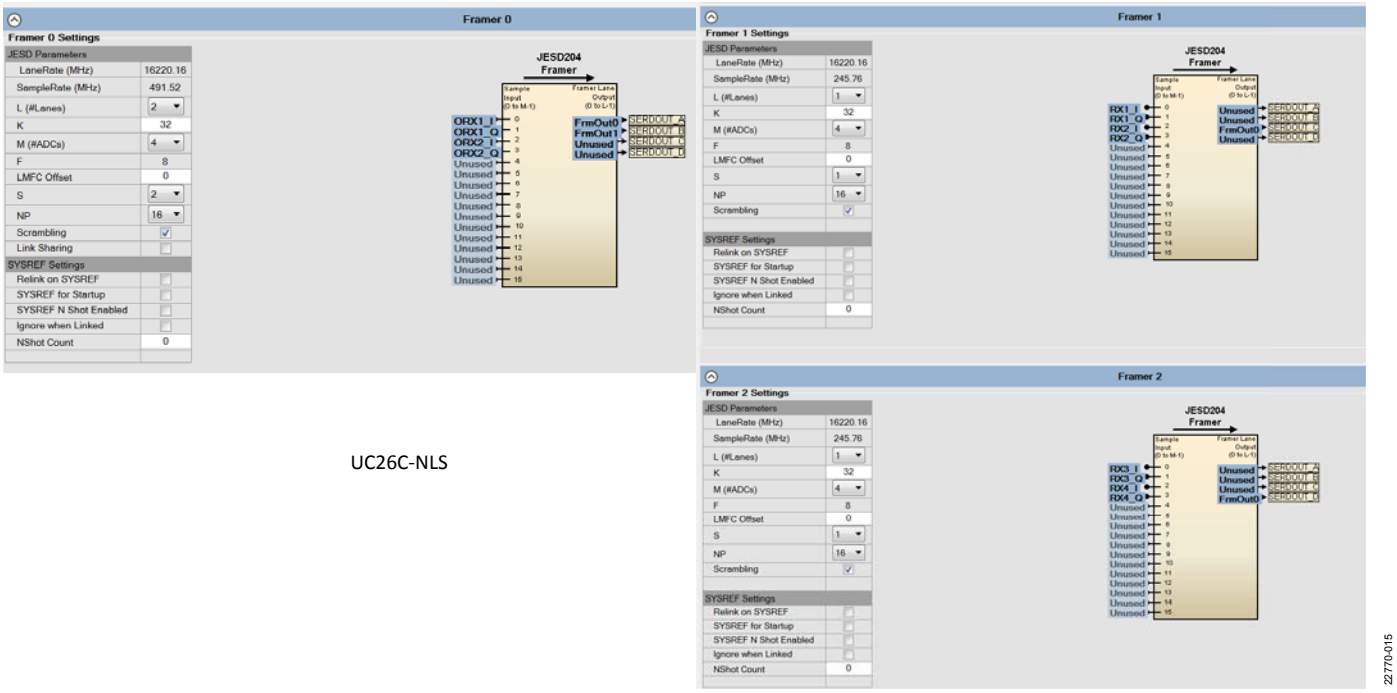
The framers take care of all the encoding functions of the interface and is highly configurable with regard to interface rates and combinations of RF receiver/observation receiver data streams, either separately or utilizing link sharing (Rx/ORx data time multiplexed on the same lane according to the receiver-transmitter frame timing) for up to four lanes. To assist in debugging it contains an internal data generator allowing a number of test patterns and PBRS patterns to be sent across the link.

There are three framers in the ADRV9026 to allow flexibility in configuring the output data streams. Data samples from the receivers and observation receivers can be routed through a cross bar to put specific data on a particular lane. The framer supports separate lanes for receiver and observation receiver, as well supporting link sharing in TDD mode that reduces the number of physical lanes needed by putting receiver data on the lanes during the receiver slot and observation receiver data on the same lanes during the transmitter slot. Figure 13 shows the configuration for use case 83C with link sharing (UC83C-LS) where all the signals are routed into Framer0. Framer1 and Framer2 are not needed and are unused. This profile is a 25G 204C profile.



UC83C-LS

Figure 13. Example Framer Configuration for UC83C-LS



UC26C-NLS

Figure 14. Example Framer Configuration for UC26C-NLS

Figure 14 shows a configuration for a non-link sharing use case UC26C-NLS. This profile has a unique configuration where the datalink on the ORx must have the data in a specific format (IIQQ). Framer0 has more flexibility that the other two framers. For this case Framer0 is used to format the ORx data as needed, and the other two framers are used to route the RX data on the lanes. This is a 16G 204C profile.

The transport and link layers for JESD204B/JESD204C are performed in the framers. This device has three JESD204B/JESD204C framers that get OR'ed together into four serial lanes. There are 20 logical converters to choose from, and samples from any of the logical converters can be connected to any framer using the sample crossbar. Each framer has its own $\overline{\text{SYNC}}$ signal. This allows links to be brought up/down for reconfiguration without interrupting the other links.

The three framers are capable of operating at different sample rates. The highest sample rate must be a power of two multiple of the lower sample rates (2×, 4×, 8×, and so forth). There are two options to make this work: oversample at the framer input or bit repeat at the framer output.

Oversample mode samples the same converter samples of the lower sample rate multiple time, essentially oversampling the converter output. This allows for all serializers to run at the same bit rate. In oversample mode, the baseband processor must decimate the data after the transport layer to remove the extra samples.

Bit repeat mode repeats each bit at the framer output on the lane or lanes that carry the slower data, before it enters the serializer. Because this is after the 8B10B/64B66B encoding, it appears as if the lane is running at a slower data rate than the other lanes. This essentially expands the eye of the signal in the horizontal direction. In bit repeat mode, the baseband processor must be able to configure the lane rates on the individual lanes independently as the lanes with the slower link must be sampled at a slower lane rate than the lanes with the faster link.

All framers must share the four serializer lanes. Each framer must be configured for 0, 1, 2 or 4 lanes such that at a time all framers combine for no more than 4 lanes.

Each framer is capable of generating a pseudo-random bit sequence (PRBS) on the enabled lanes. Once the PRBS is enabled, errors can be injected. Enabling the PRBS generator disables the normal JESD204B/JESD204C framing, and causes the link to drop.

The serializers can be configured to adjust the amplitude and pre-emphasis of the physical signal to help combat bit errors due to various PCB trace lengths.

Supported Framer Link Parameters

This device supports a subset of possible JESD204B/JESD204C link configurations. The number of virtual converters and the number of serial lanes implemented in the silicon limit these configurations.

Table 10. JESD204B/JESD204C Framer Parameters

JESD204B/JESD204C Parameter	Description
M	Number of converters. Framer 0 supports M maximum of 8, Framers 1 and 2 support M maximum of 4.
L	Number of lanes (L can be 1, 2, or 4).
S	Number of samples per converter per frame cycle (S can be 1, 2, or 4).
N	Converter resolution (N can be 12, 16, or 24).
N′	Total number of bits per sample (N′ can be 12, 16, or 24).
CF	Number of control words/frame clock. Cycle/converter device.
CS	Number of control bits/conversion sample.
K	JESD204B only: Number of frames in 1 multiframe, (20 ≤ F*K ≤ 256), F*K must be a multiple of 4.
E	JESD204C only: Number of multiblocks in an extended multiblock.

For the JESD204B/JESD204C configuration to be valid, the lane rate must be within the range 3686.4 Mbps to 16220.16 Mbps. The lane rate is the serial bitrate for one lane of the JESD204B/JESD204C link. The lane rates can be calculated using Equation 1 for JESD204B configurations and using Equation 2 for JESD204C configurations.

$$JESD204B\ Lane\ Rate = IQ\ Sample\ Rate \times M \times N' \times \frac{10}{8} \div L \quad (1)$$

$$JESD204C\ Lane\ Rate = IQ\ Sample\ Rate \times M \times N' \times \frac{66}{64} \div L \quad (2)$$

Serializer Configuration

The amplitude of the serializer is represented by a 3-bit number that is not linearly weighted. The JESD204B/JESD204C transmitter mask requires a differential amplitude greater than 360 mV and less than 770 mV.

Table 11. Serializer Amplitude Settings

Serializer Amplitude (Decimal)	Average Differential Amplitude (V_{TT} = 1 V)
0	$1 \times V_{TT}$
1	$0.85 \times V_{TT}$
2	$0.75 \times V_{TT}$
3	$0.5 \times V_{TT}$

It is always recommended to verify the eye diagram in the system after building a PCB to verify any layout related performance differences. If possible verify the eye using an internal eye monitor after the equalizer circuit of the receiver as this shows the true eye that the receiver circuit receives.

A three-tap FIR equalizer is implemented in the serializer as shown in Figure 15. Here, the cursor, or largest tap weight multiplying a_k is in the center. There is a precursor tap, b_{-1}, multiplying a_{k+1} and a postcursor tap, b_1, multiplying a_{k-1} to realize the following difference

equation for y_k. Transmit pre-emphasis is used because it is simpler to realize bit delays with flip flops than trying to implement analog delays at the receiver.

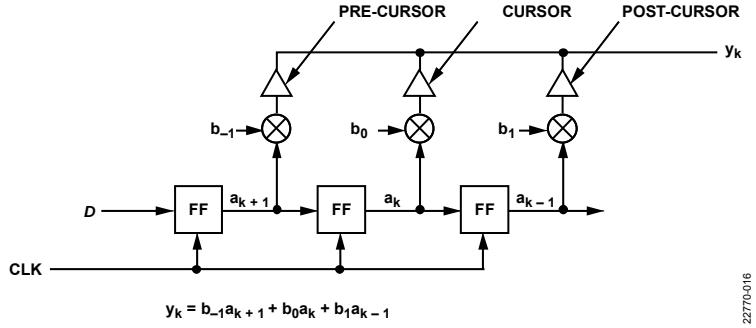


Figure 15. Serializer Emphasis implementation

This serializer pre-emphasis circuit allows boosting the amplitude anytime the serial bit changes state. If no bit transition occurs, the amplitude is left unchanged. Pre-emphasis helps open the eye for longer PCB traces or when the parasitic loading of connectors has a noticeable effect. In most cases, to find the best setting, a simulation or measurement of the eye diagram with a high-speed scope at the receiver is recommended, or as mentioned above an internal eye monitor after the equalizer is the optimum solution. The serializer pre-emphasis is controlled by setting a precursor and a postcursor setting, which are listed in Table 12 and Table 13, respectively.

Table 12. Precursor Amplitude Settings

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

Table 13. Postcursor Amplitude Settings

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

The adi_adrv9025_SerCfg_t data structure contains the information required to properly configure the serializer. Details of each member can be found in API documentation (/c_src/doc). The Transceiver Evaluation Software has the option to output example data structures with values chosen from the configuration tab of the software.

```
typedef struct adi_adrv9025_SerCfg
{
    uint8_t serAmplitude;
    uint8_t serPreEmphasis;
    uint8_t serPostEmphasis;
    uint8_t serInvertLanePolarity;
} adi_adrv9025_SerCfg_t;
```

Framer

Each framer receives logical converter samples and maps them to high speed serial lanes. The mapping changes depending on the JESD204B/JESD204C configuration chosen, specifically the number of lanes, the number of converters, and the number of samples per converter. Figure 16 provides one valid framer configuration for this device.

The converter samples are passed into the framer through a sample crossbar. The sample crossbar allows any of the 20 logical converters to be mapped to any input of any framer. For example, this can be used to swap I and Q samples or to mix and match different receivers' data across different logical lanes. The framer lane data outputs also pass through a lane crossbar. This allows mapping any framer output

lane (internal to the silicon) to any physical JESD204B/JESD204C lane at the package pin. The framer packs the converter samples into lane data following the JESD204B/JESD204C specification. Figure 16 shows the data packing for M = 2, L = 1, S = 1 as an example.

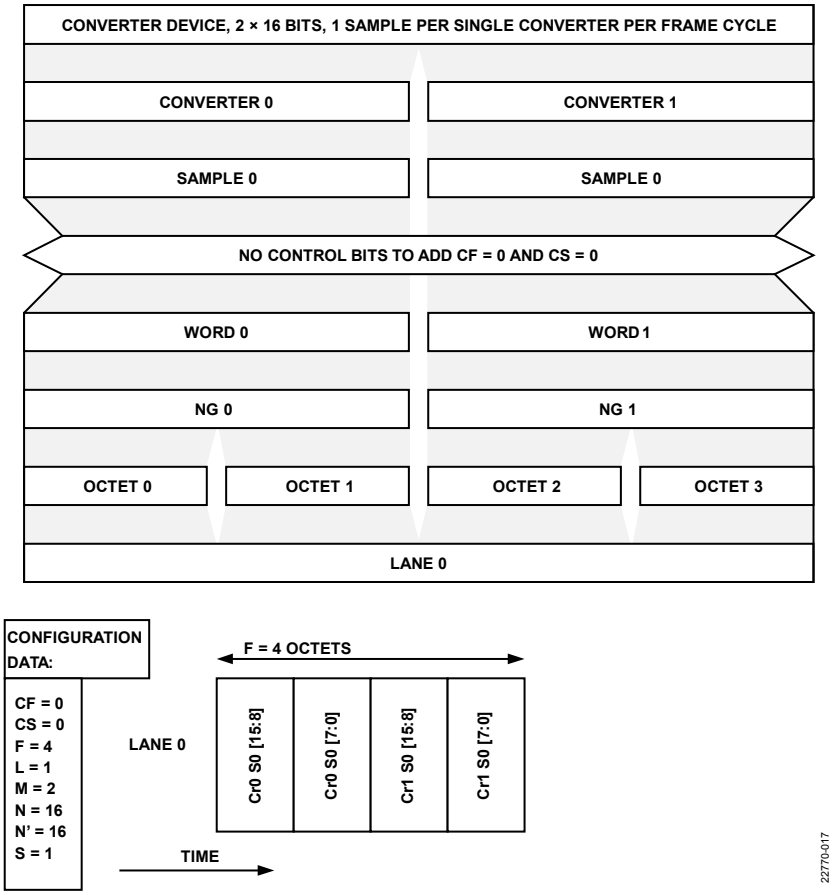


Figure 16. Framer Data Packing for M = 2, L = 1, S = 1

Other Useful Framer IP Features

PRBS Generator

Each framer has a built in PRBS test pattern generator to aid in debugging the JESD204B/JESD204C serial link. The pattern generator is capable of generating PRBS7, PRBS9, PRBS15, PRBS23, or PRBS31 patterns. If errors caused by signal integrity exist, it may be difficult to get the JESD204B/JESD204C framer-to-deframer link to work properly. The PRBS generator built into the framer allows the device to output serial data even when the link cannot be established. With this mode enabled, the serializer amplitude and pre-emphasis can be adjusted to find the best setting to minimize bit errors seen by the PRBS checker at the receiver side of the link. For this mode to be utilized, the baseband processor must have a PRBS checker to check the PRBS sequence for errors.

Typical usage sequence:

1. Initialize the device as outlined in the link establishment section
2. Run the adi_adrv9025_FramerTestDataSet(…) with the required framer, test data source set to desired PRBS order, and injection point to serializer input
3. Enable PRBS checker on the baseband processor and reset its error count
4. Wait a specific amount of time to allow a good number of samples to be transmitted, and then check the PRBS error count of the baseband processor.
5. Adjust framer amplitude and pre-emphasis settings and/or deframer equalization settings and repeat steps 3 and 4 to find the optimum settings.

Pattern Generator

The framer also has the capability to generate some other patterns that can be used during debug like RAMP, CHECKERBOARD. There is also a way the user can load a custom pattern into the framer which can be verified on the baseband processor. The pattern can sent once, or be repeated continuously.

API Software Integration

Configuration of the serializer and framers are all handled by the adi_adrv9025_Initialize(…) API function. Set all JESD204B/JESD204C link options for the framer in the adi_adrv9025_FrmCfg_t data structure before calling adi_adrv9025_Initialize(…). After initialization, there are some other API functions to aid in debug and monitoring the status of the JESD204B/JESD204C link.

JESD204B/JESD204C Framer API Data Structures

adi_adrv9025_FrmCfg_t

The adi_adrv9025_FrmCfg_t data structure contains the information required to properly configure each framer. Details of each member can be found in API documentation. The transceiver evaluation software has the option to output example data structures with values chosen from the configuration tab of the software.

```
typedef struct adi_adrv9025_FrmCfg
{
    uint8_t enableJesd204C;
    uint8_t bankId;
    uint8_t deviceId;
    uint8_t lane0Id;
    uint8_t jesd204M;
    uint16_t jesd204K;
    uint8_t jesd204F;
    uint8_t jesd204Np;
    uint8_t jesd204E;
    uint8_t scramble;
    uint8_t externalSysref;
    uint8_t serializerLanesEnabled;
    uint16_t lmfcOffset;
    uint8_t reserved;
    uint8_t syncbInSelect;
    uint8_t overSample;
    uint8_t syncbInLvdsMode;
    uint8_t syncbInLvdsPnInvert;
    uint8_t enableManualLaneXbar;
    adi_adrv9025_SerLaneXbar_t serializerLaneCrossbar;
    adi_adrv9025_AdcSampleXbarCfg_t adcCrossbar;
    uint8_t newSysrefOnRelink;
    uint8_t sysrefForStartup;
    uint8_t sysrefNShotEnable;
    uint8_t sysrefNShotCount;
    uint8_t sysrefIgnoreWhenLinked;
} adi_adrv9025_FrmCfg_t;
```

Table 14. JESD204B/JESD204C Framer Configuration Structure Member Description

Structure Member	Valid Values	Description
enableJesd204C	0, 1	0 = enable JESD204B framer; 1 = enable JESD204C framer
bankId	0..15	JESD204B/JESD204C configuration Bank ID—extension to device ID
deviceId	0..255	JESD204B/JESD204C configuration Device ID—link identification number
lane0Id	0..31	JESD204B/JESD204C configuration Lane ID—if more than one lane is used, each subsequent lane increments from this number
jesd204M	0, 1, 2, 4, 8	Number of converters—typically two converters per receive chain
jesd204K	1 to 32	Number of frames in a multiframe—default is 32; F × K must be a multiple of 4
jesd204F	1, 2, 3, 4, 6, 8, 12, 16	Number of octets per frame
jesd204Np	12, 16, 24	Number of bits per sample
Scramble (JESD204B Only)	0, 1	Scrambling enabled If scramble = 0, then scrambling is disabled If scramble = 1, then scrambling is enabled
externalSysref	0, 1	External SYSREF enabled If externalSysref = 0, then use internal SYSREF If externalSysref = 1, then use external SYSREF
serializerLanesEnabled	0x0 to 0x0F	Serializer lane enabled—one bit per lane
serializerLaneCrossbar	0x0 to 0xFF	Serializer lane crossbar—two bits per lane
lmfcOffset	0 to 31	LMFC Offset—set the local multiframe counter offset value for deterministic latency setting, such that 0 ≤ lmfcOffset ≤ (K-1)
reserved		
syncinbSelect	0, 1, 2	$\overline{\text{SYNC}}$ selection – selects which $\overline{\text{SYNC}}$ input is connected to the framer If syncinbSelect = 0, then $\overline{\text{SYNCIN0}}$ is connected to the framer If syncinbSelect = 1, then $\overline{\text{SYNCIN1}}$ is connected to the framer If syncinbSelect = 2, then $\overline{\text{SYNCIN2}}$ is connected to the framer
overSample	0, 1	Oversample mode—selects which method is chosen when oversample or bit repeat is needed If oversample = 0, then bit repeat mode is selected If oversample = 1, then oversample is selected
enableManualLaneXbar	0, 1	0 = automatic lane crossbar mapping; 1 = manual lane crossbar mapping (using serializerLaneCrossbar value)
syncbInLvdsMode	0, 1	1 = Enables LVDS input pad; 0 = enables CMOS input pad
syncbInLvdsPnInvert	0, 1	0 = $\overline{\text{SYNC}}$ LVDS PN not inverted; 1 = $\overline{\text{SYNC}}$ LVDS PN inverted
newSysrefOnRelink	0, 1	Set the flag for determining if SYSREF on relink. Where, if > 0 = set, 0 = not set
sysrefForStartup	0, 1	1 = framer: require a SYSREF before CGS is output from serializer, 0: Allow CGS to output before SYSREF occurs (recommended on framer to allow deframer CDR to lock and EQ to train)
sysrefNShotEnable	0, 1	1 = enable SYSREF NShot (ability to ignore first rising edge of SYSREF to ignore possible runt pulses)
sysrefNShotCount	0..15	Count value of which SYSREF edge to use to reset LMFC phase
sysrefIgnoreWhenLinked	0, 1	When JESD204 link is up and valid, 1 = ignore any SYSREF pulses

JESD204B/JESD204C Framer Enumerated Types

adi_adrv9025_FramerDataSource

The adi_adrv9025_FramerDataSource_e is an enumerated data type to select the Framer test data source. The allowable values are listed in Table 15.

Table 15. Framer Data Source Enumeration Description

Enumeration Value	Description
FTD_ADC_DATA	Framer test data ADC data source – this is used for normal operation
FTD_CHECKERBOARD	Framer test data checkerboard data source
FTD_TOGGLE0_1	Framer test data toggle 0 to 1 data source
FTD_PRBS31	Framer test data PRBS31 data source
FTD_PRBS23	Framer test data PSRB23 data source
FTD_PRBS15	Framer test data PRBS15 data source
FTD_PRBS9	Framer test data PRBS9 data source
FTD_PRBS7	Framer test data PRBS7 data source
FTD_RAMP	Framer test data ramp data source
FTD_PATTERN_REPEAT	Framer test data 16-bit programmed pattern repeat source
FTD_PATTERN_ONCE	Framer test data 16-bit programmed pattern executed once source

adi_adrv9025_FramerDataInjectPoint

The adi_adrv9025_FramerDataInjectPoint is an enumerated data type to select the Framer test data injection point. The allowable values are listed in Table 16.

Table 16. Framer Injection Point Enumeration Description

Enumeration Value	Description
FTD_FRAMERINPUT	Framer test data injection point at framer input
FTD_SERIALIZER	Framer test data injection point at serializer input
FTD_POST_LANEMAP	Framer test data injection point after lane mapping

adi_adrv9025_FramerSel

The adi_adrv9025_FramerSel is an enumerated data type to select the desired Framer. The allowable values are listed in Table 17.

Table 17. Framer Selection Enumeration Description

Enumeration Value	Description
ADI_ADRV9025_FRAMER_0	Framer 0 selection
ADI_ADRV9025_FRAMER_1	Framer 1 selection
ADI_ADRV9025_FRAMER_2	Framer 2 selection
ADI_ADRV9025_ ALL_FRAMERS	All Framers selected

API Functions

adi_adrv9025_FramerSysrefCtrlSet(…)

```
adi_adrv9025_FramerSysrefCtrlSet(adi_adrv9025_Device_t *device, uint8_t framerSelMask, uint8_t
  enable);
```

This function enables or disables the external SYSREF JESD204B/JESD204C signal connection to the framers.

For the framer to retime its LMFC/LEMF (local multi frame clock/local extended multiblock clock), a SYSREF rising edge is required. The external SYSREF signal at the pin can be gated off internally so the framer does not see a potentially invalid SYSREF pulse before it is configured correctly.

By default the device has the SYSREF signal ungated. However, the multichip sync state machine still does not allow the external SYSREF to reach the framer until the other stages of multichip sync have completed. As long as the external SYSREF is correctly configured before performing MCS, this function may not be needed by the BBP, because the MCS state machine gates the SYSREF to the framer.

Precondition

This function is called after the device has been initialized and the JESD204B/JESD204C framer is enabled.

Dependencies

device->devHalInfo

Parameters

Table 18.

Parameter	Description
*device	is a pointer to the device settings structure
framerSelMask	Select framer to enable/disable SYSREF input for (Valid Any OR'ed combination of enums ADI_ADRV9025_FRAMER_0, ADI_ADRV9025_FRAMER_1, ADI_ADRV9025_FRAMER_2 or ADI_ADRV9025_ALL_FRAMERS)
enable	= 1 enables SYSREF to framer, '0' disables SYSREF to framer

Return Values

Table 19.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_FramerStatusGet(…)

```
adi_adrv9025_FramerStatusGet(adi_adrv9025_Device_t *device, adi_adrv9025_FramerSel_e framerSel,
  adi_adrv9025_FramerStatus_t *framerStatus);
```

This function reads back the status of the selected framer to determine the state of the JESD204B/JESD204C link. The framer status return value is an 8-bit status word as shown in Table 20. It also returns the qbfStateStatus and sync signal used by the selected framer.

Table 20. Framer Status Return Value

framerStatus	Description
[7]	Reserved
[6]	Reserved
[5]	Reserved
[4]	Reserved
[3]	Current $\overline{\text{SYNCIN}}$ level(1 = high, 0 = low)
[2]	SYSREF phase error. Is set when a new SYSREF had different timing than the first that set the LMFC timing.
[1]	SYSREF phase established by framer
[0]	Flag indicating that configuration parameters are not supported when set(1)

Precondition

The Rx JESD204B/JESD204C link(s) needs to be configured and running to use this function

Dependencies

device->devHalInfo

Parameters

Table 21.

Parameter	Description
*device	is a pointer to the device settings structure
framerSel	Read back the framer status of the selected framer (Framer0, Framer1 or Framer2)
framerStatus	is the framer status structure read

Return Values

Table 22.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_FramerTestDataSet*(…)*

```
adi_adrv9025_FramerTestDataSet(adi_adrv9025_Device_t *device, adi_adrv9025_FrmTestDataCfg_t
  *frmTestDataCfg);
```

This function selects the PRBS type and enables or disables Rx Framer PRBS generation. This is a debug function to be used for debug of the Rx JESD204B/JESD204C lanes. Rx data transmission on the JESD204B/JESD204C link(s) is not possible when the framer test data is activated.

Precondition

This function may be called any time after device initialization.

Dependencies

device->devHalInfo

Parameters

Table 23.

Parameter	Description
*device	is a pointer to the device settings structure
frmTestDataCfg	is a pointer to a structure which contains the framer(s) of interest, testDataSource and injectPoint

Return Values

Table 24.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_FramerTestDataInjectError (…)

```
adi_adrv9025_FramerTestDataInjectError(adi_adrv9025_Device_t *device, adi_adrv9025_FramerSel_e
  framerSelect, uint8_t laneMask);
```

This function injects an error into the Framer test data by inverting the data. This is a debug function to be used for debug of the receiver JESD204B/JESD204C lanes. Receiver data transmission on the JESD204B/JESD204C link(s) is not possible when the framer test data is activated.

Precondition

This function is called after the framer test data is enabled.

Dependencies

device->devHalInfo

Parameters

Table 25.

Parameter	Description
*device	is a pointer to the device settings structure
framerSelect	Select the desired framer ADI_ADRV9025_FRAMER_0, ADI_ADRV9025_FRAMER_1, ADI_ADRV9025_FRAMER_2
laneMask	is an four bit mask allowing selection of lanes 0-3 for the selected framer

Return Values

Table 26.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_FramerLinkStateSet (…)

```
adi_adrv9025_FramerLinkStateSet(adi_adrv9025_Device_t *device, uint8_t framerSelMask, uint8_t
  enable);
```

This function enables and disables the JESD204B/JESD204C Framer. This function is normally not necessary. In the event that the link needs to be reset, this function allows a framer to be disabled and reenabled. .

Precondition

This function may be called any time after device initialization.

Dependencies

device->devHalInfo

Parameters

Table 27.

Parameter	Description
*device	is a pointer to the device settings structure
framerSelMask	Desired framer(s) to set/reset.
enable	0 = Disable the selected framers, 1 = enable the selected framer link

Return Values

Table 28.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

TRANSMITTER (DAC) DATAPATH

Figure 17 shows a block diagram of the transceiver transmit side (SERDES deframer).

The SERDES deframer receives the transmitter data from the baseband processor, decodes it and distributes the data streams to the transmitters. The ADRV9026 includes two deframers that share up to four lanes that can operate at up to 25G. Figure 18 shows the configuration for UC26C-NLS that uses Deframer0 and utilizes four lanes at 16G to support 4 Tx at maximum bandwidth.

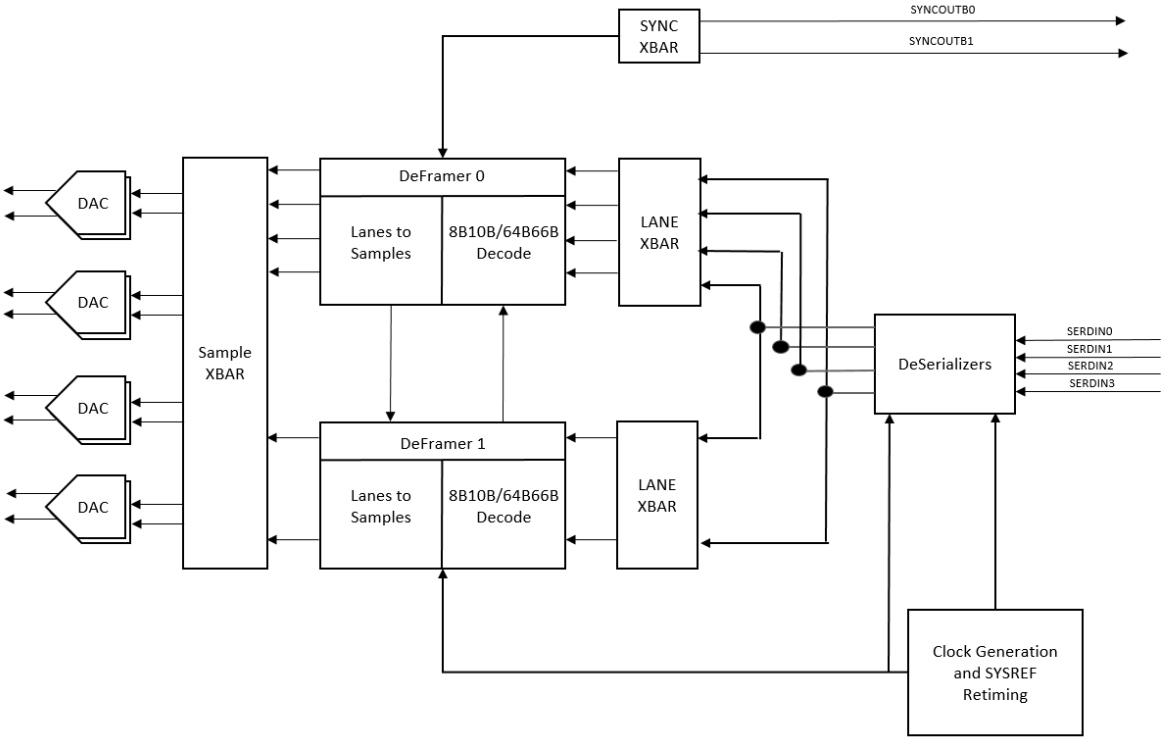
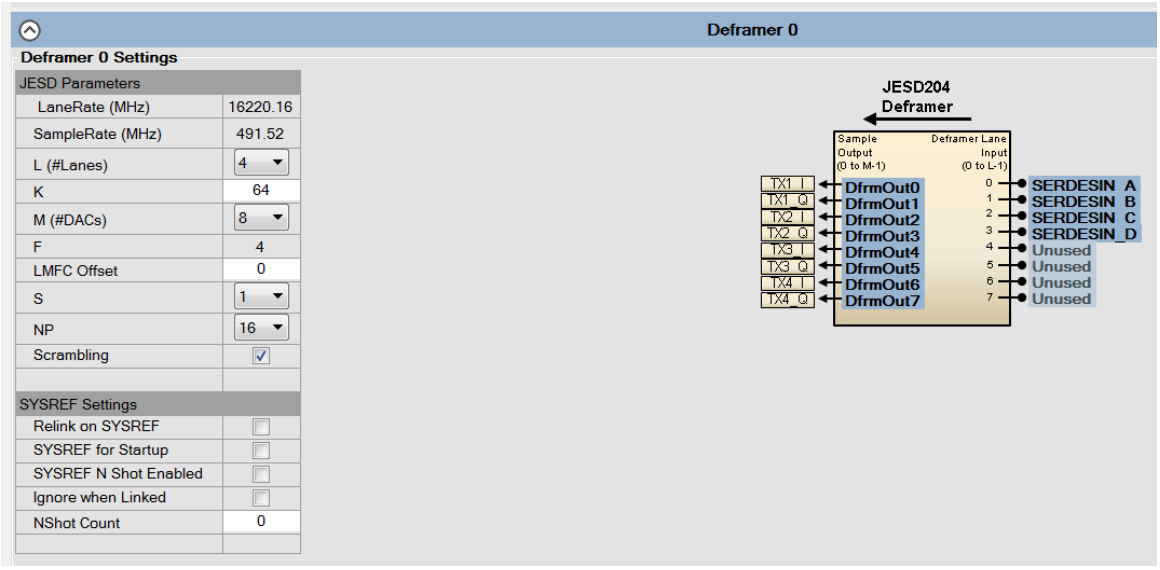


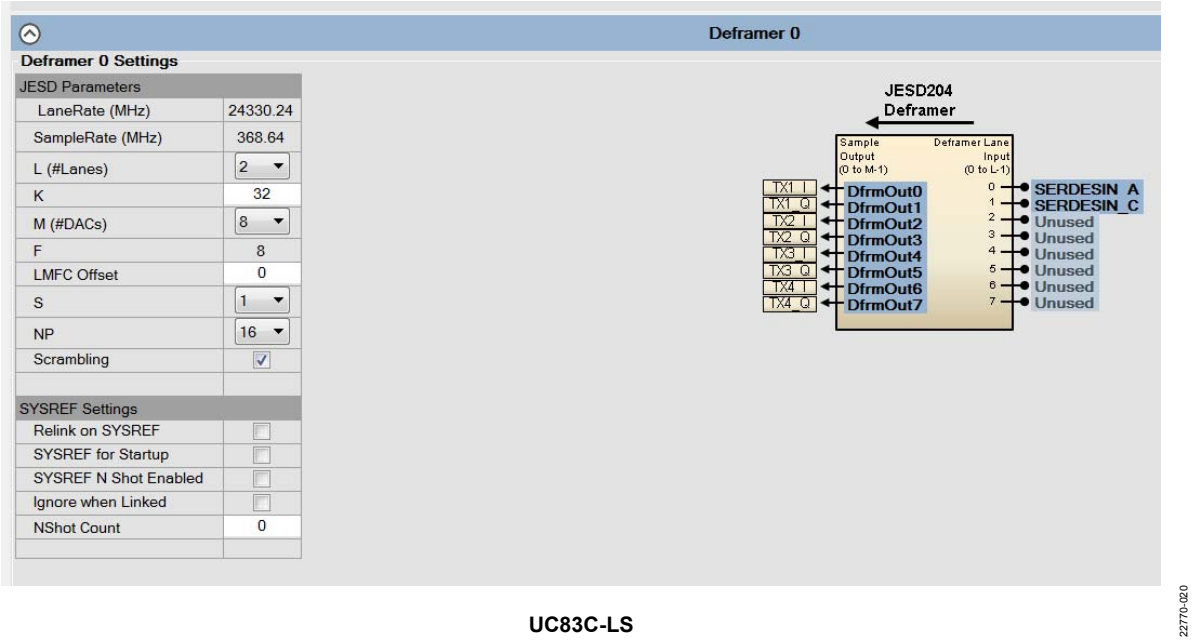
Figure 17. High Level JESD204B/JESD204C Interface Block Diagram (Tx Only)



UC26C-NLS

Figure 18. Example Deframer Configuration for UC26C-NLS

Figure 19 shows the configuration for UC83C-LS that uses Deframer0. Only two lanes are needed to realize the maximum chip RF bandwidth (450 MHz) across all four Tx. This device has two JESD204B/JESD204C deframers that share four physical lanes. The two deframers feed a sample crossbar that connects to eight DACs. All converters must run at the same sample rate. Likewise, all lanes must run at the same data rate. Each deframer is capable of receiving a PRBS sequence and accumulating error counts. The deserializers have adjustable equalization circuits to counteract the insertion loss due to various PCB trace lengths and material.



UC83C-LS

Figure 19. Example Deframer Configuration for UC83C-LS

SUPPORTED DEFRAMER LINK PARAMETERS

The product supports a subset of possible JESD204B/JESD204C link configurations. The modes are limited by the number of DACs and the number of serial lanes implemented in the silicon.

Table 29. JESD204B/JESD204C Deframer Parameters

JESD204B/JESD204C Parameter	Description
M	Number of converters (M can be 1, 2, 4 or 8)
L	Number of lanes (L can be 1, 2, or 4)
S	Number of samples per converter per frame cycle
N	Converter resolution (N can be 12 or 16)
N'	Total number of bits per sample (N' can be 12 or 16)
CF	Number of control words/frame clock cycle/converter device
CS	Number of control bits/conversion sample
HD	High density mode.
K	JESD204B only: Number of frames in 1 multiframe, ($20 \le F \times K \le 256$), $F \times K$ must be a multiple of 4, $K \le 32$
E	JESD204C only: Number of multiblocks in an extended multiblock.

For a particular converter sample rate, not all combinations listed in the above table are valid. Calculate the JESD204B or JESD204C lane rate using the equations described in the Supported Framer Link Parameters section.

The deserializer link is allowed to run at a different lane rate than the serializer link, under the condition that both lane rates are possible with respect to the clock divider settings. Both the deserializer and serializer link rates are derived from the same PLL, but there are separate dividers to generate the deserializer clock and the serializer clock.

Deserializer Configuration

The deserializer includes a non-adaptive, programmable equalizer. This helps in compensating for signal integrity distortions for each channel due to PCB trace length and impedance. The table below summarizes the amount of insertion loss each EQ setting can overcome. Equalizer boost settings can range from 0 (maximum boost) to 3 (default).

Table 30. Deserializer EQ Boost Correction

EQ BoostSettings	Boost (dB)
0	0
1	–3
2	–6
3	–12

If the insertion loss is greater than this, one of the other settings may be appropriate. Note that any setting can be used in conjunction with transmitter pre-emphasis to ensure functionality and/or to optimize for power. The equalizer setting can be changed in the API using desEqGainSetting parameter in the data structure adi_adrv9025_DesCfg_t.

The adi_adrv9025_DesCfg_t data structure contains the information required to properly configure the deserializer. Details of each member can be found in API documentation. The Transceiver Evaluation Software has the option to output example data structures with values chosen from the configuration tab of the software.

```
typedef struct adi_adrv9025_DesCfg
{
    uint8_t desInvertLanePolarity;
    uint8_t desEqBoostSetting;
    uint8_t desEqGainSetting;
    uint8_t desEqFeedbackSetting;
} adi_adrv9025_ DesCfg_t;
```

In JESD204B mode, the ADRV9026 uses passive equalizer architecture that de-emphasizes low frequencies in relation to the high frequencies and then amplifies the signal. This provides the required equalization or 'boost' to properly capture the signal. A brief description of the data members in adi_adrv9025_DesCfg_t is given in Table 31.

Table 31. Deserializer EQ Data Members

Structure Member	Description
desInvertLanePolarity	Deserializer lane PN inversion select. Bit 0 = invert PN of Lane 0, Bit 1 = invert PN of Lane 1, and so on.
desEqBoostSetting	It sets how much high frequency attenuation you are trying to compensate.
desEqGainSetting	Gain is setting the number of stages of limiting amplifier. This compensates for the amount of EqBoost added above.
desEqFeedbackSetting	This is the amount of feedback set for each gain stage. It works as a basic op amp, where the feedback network can be tuned depending on the feedback setting in the equalizer. This feedback setting is applied to each of the limiting amplifiers (depending on number of stages). It causes peaking in the total channel response. It is not recommended to tune this data member while compensating for insertion losses.

When operating in JESD 204C mode, the equalization is done with a CTLE (continuous time linear equalizer) that is configured during device initialization with a SERDES INIT calibration.

Deframer

The active deframers receive 8B10B/64B66B encoded data from the deserializer and decode the data into converter samples. The deserializer-to-converter sample mapping changes depending on the JESD204B/JESD204C link configuration setting. Responsibilities of the deframer are:

1. Monitor the health of the JESD204B/JESD204C link
2. Control the JESD204B/JESD204C Interrupt signal (can output on General Purpose Interrupt pin) to signal baseband processor when certain JESD204B/JESD204C error conditions arise.
3. Remove character replacement (valid for only JESD204B).
4. Perform 8B10B/64B66B decoding.
5. Map JESD204B/JESD204C lane data to converter samples.

A lane crossbar provides the ability to reorder the lanes into each deframer input. A sample crossbar provides the ability to reorder the converter samples at the output of the deframers. The lane and sample crossbars enable flexiblity on which physical lanes are used and which data is on each link.

The deframer unpacks the converter samples from lane data following the JESD204B/JESD204C specification. Figure 20 shows the data unpacking for M = 4, L = 2, S = 1 as an example.

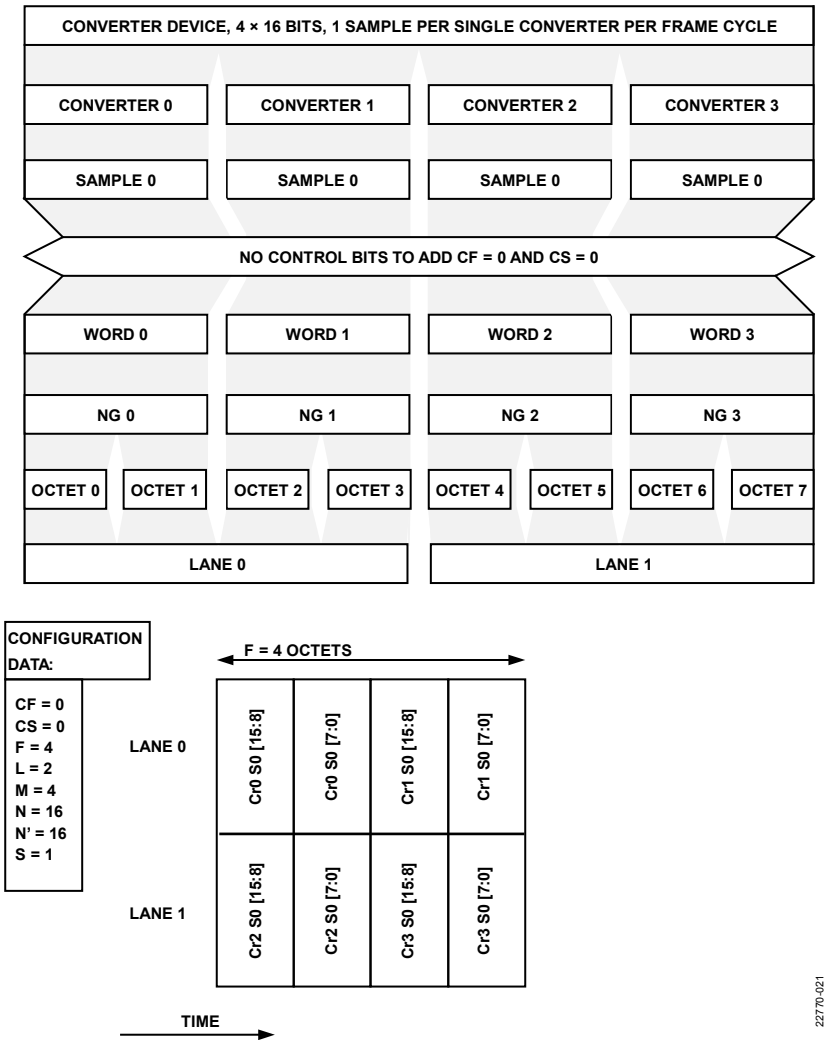


Figure 20. JESD204B Deframer Configuration (M = 4, L = 2)

Other Useful Deframer IP Features

PRBS Checker

The deframer has a built in pseudo random bit sequence (PRBS) checker. The PRBS checker can self synchronize and check for PRBS errors on a PRBS7, PRBS15, or PRBS31 sequence. Since this mode works even in the midst of potential bit errors on each lane, the physical link can be debugged even when the JESD204B/JESD204C link cannot be established. This mode can be used to check the robustness of the physical link during initial testing and/or factory test. For this mode to be fully utilized, the BPP must have a PRBS generator capable of creating PRBS7, PRBS15, or PRBS31 data.

A typical usage sequence is as follows:

1. Initialize the device as outlined in the link establishment section.
2. Enable the PRBS generator on the baseband processor with the desired PRBS sequence.
3. Call the API adi_adrv9025_DfrmPrbsCheckerStateSet(…) passing the actual device being evaluated, the PRBS sequence to check, and the location at which to check the PRBS sequence.
4. After some amount of time, call the API function to check the PRBS errors. This can be done by calling the API function adi_adrv9025_DfrmPrbsErrCountGet(…) passing the actual device being evaluated, the counter selection lane to be read and the error count is returned in the third parameter passed.

To prove an error count of 0 is valid, the baseband processor may have a PRBS error inject feature. Alternatively, the baseband processor amplitude and emphasis settings can be set to a setting where errors occur. To reset the error count call the API function that clears the counters: adi_adrv9025_DfrmPrbsCountReset(…).

API Software Configuration

Configuration of the deserializer and deframers are handled by the adi_adrv9025_Initialize(…) API function. Set all JESD204B/JESD204C link options for the framer in the adi_adrv9025_DfrmCfg_t data structure before calling adi_adrv9025_Initialize(…). After initialization, there are some other API functions to aid in debug and monitoring the status of the JESD204B/JESD204C link.

JESD204B/JESD204C Deframer API Data Structures

adi_adrv9025_DfrmCfg_t

The adi_adrv9025_DfrmCfg_t data structure contains the information required to properly configure each deframer. Details of each member can be found in API documentation. The transceiver evaluation software has the option to output example data structures with values chosen from the configuration tab of the software.

```
typedef struct adi_adrv9025_DfrmCfg
{
    uint8_t enableJesd204C;
    uint8_t bankId;
    uint8_t deviceId;
    uint8_t lane0Id;
    uint8_t jesd204M;
    uint16_t jesd204K;
    uint8_t jesd204Np;
    uint8_t jesd204E;
    uint8_t scramble;
    uint8_t externalSysref;
    uint8_t deserializerLanesEnabled;
    uint16_t lmfcOffset;
    uint8_t reserved;
    uint8_t syncbOutSelect;
    uint8_t syncbOutLvdsMode;
    uint8_t syncbOutLvdsPnInvert;
    uint8_t syncbOutCmosSlewRate;
    uint8_t syncbOutCmosDriveLevel;
```

```
    uint8_t enableManualLaneXbar;
    adi_adrv9025_DeserLaneXbar_t deserializerLaneCrossbar;
    adi_adrv9025_DacSampleXbarCfg_t dacCrossbar;
    uint8_t newSysrefOnRelink;
    uint8_t sysrefForStartup;
    uint8_t sysrefNShotEnable;
    uint8_t sysrefNShotCount;
    uint8_t sysrefIgnoreWhenLinked;

} adi_adrv9025_DfrmCfg_t;
```

Table 32. JESD204B/JESD204C Deframer Configuration Structure Member Description

Structure Member	Valid Values	Description
enableJesd204C	0, 1	0 = Enable JESD204B framer; 1= Enable JESD204C framer
bankId	0..15	JESD204B/JESD204C Configuration Bank ID (extension to device ID)
deviceId	0..255	JESD204B/JESD204C Configuration Device ID (link identification number)
lane0Id	0..31	JESD204B/JESD204C Configuration Lane ID (if more than one lane is used, each subsequent lane increments from this number)
jesd204M	0, 2, 4, 8	Number of converters: 2 converters per transmit chain
jesd204K (JESD204B Only)	1..32	Number of frames in a multiframe (default is 32); F × K must be a multiple of 4
jesd204Np	12, 16	Number of bits per sample
jesd204E	0..255	JESD204C E parameter
Scramble (JESD204B Only)	0, 1	Scrambling enabled If scramble = 0, then scrambling is disabled If scramble = 1, then scrambling is enabled
externalSysref	0, 1	External SYSREF enabled If externalSysref = 0, then use internal SYSREF If externalSysref = 1, then use external SYSREF
deserializerLanesEnabled	0x0 to 0xF	Deserializer lane enabled: one bit per lane
deserializerLaneCrossbar	0x0 to 0xFF	Deserializer lane crossbar: three bits per lane
lmfcOffset	0 to 31	LMFC offset: Set the Local Multi Frame Counter Offset value for deterministic latency setting, such that 0 ≤ lmfcOffset ≤ (K-1)
syncbOutSelect	0,1	New SYSREF on Relink: flag to indicate that a SYSREF is required to re-establish the link if newSysrefOnRelink = 0, then no SYSREF is required if newSysrefOnRelink = 1, then SYSREF is required
enableManualLaneXbar	0, 1	$\overline{\text{SYNC}}$ Selection: selects which $\overline{\text{SYNCOUT}}$ output is driven by the deframer If syncbOutSelect = 0, then the deframer drives $\overline{\text{SYNCOUT0}}$ If syncbOutSelect = 1, then the deframer drives $\overline{\text{SYNCOUT1}}$
syncbInLvdsMode	0, 1	0 = automatic lane crossbar mapping; 1 = Manual lane crossbar mapping (using deserializerLaneCrossbar value)
syncbInLvdsPnInvert	0, 1	1 = enables LVDS input pad; 0 = enables CMOS input pad
syncbOutCmosSlewRate	0 to 3	0 = $\overline{\text{SYNC}}$ LVDS PN not inverted; 1 = $\overline{\text{SYNC}}$ LVDS PN inverted
syncbOutCmosDriveLevel	0, 1	0 = fastest rise/fall times, 3 = slowest rise/fall times
newSysrefOnRelink	0, 1	Set the flag for determining if SYSREF on relink. 1 = set, 0 = not set
sysrefForStartup	0, 1	1 = framer: requires a SYSREF before CGS outputs from serializer, 0: allow CGS to output before SYSREF occurs (recommended on framer to allow deframer CDR to lock and EQ to train)
sysrefNShotEnable	0, 1	1 = enable SYSREF NShot (ability to ignore first rising edge of SYSREF to ignore possible runt pulses)
sysrefNShotCount	0 to 15	Count value of which SYSREF edge to use to reset LMFC phase
sysrefIgnoreWhenLinked	0, 1	When JESD204 link is up and valid, 1= ignore any sysref pulses

adi_adrv9025_ DataInterfaceCfg _t

The adi_adrv9025_ DataInterfaceCfg_t data structure contains the information required to properly configure each framer, each deframer, the serializers, and deserializers. Details of each member can be found in API documentation (/c_src/doc).

```
typedef struct adi_adrv9025_DataInterfaceCfg
{
    adi_adrv9025_FrmCfg_t framer[3];
    adi_adrv9025_DfrmCfg_t deframer[2];
    adi_adrv9025_SerCfg_t serCfg[8];
    adi_adrv9025_DesCfg_t desCfg[8];
    uint8_t sysrefLvdsMode;
    uint8_t sysrefLvdsPnInvert;
    adi_adrv9025_LinkSharingCfg_t linkSharingCfg;
} adi_adrv9025_DataInterfaceCfg_t;
```

Table 33. JESD204B/JESD204C Settings Structure Member Description

Structure Member	Valid Values	Description
framer0	data structure	Framer 0 configuration data structure
framer1	data structure	Framer 1 configuration data structure
framer2	data structure	Framer 2 configuration data structure
deframer0	data structure	Deframer 0 configuration data structure
deframer1	data structure	Deframer 1 configuration data structure
serAmplitude	0 to 3	Serializer amplitude setting. Default = 1.
serPreEmphasis	0 to 2	Serializer pre-emphasis setting. Default = 0.
serInvertLanePolarity	0x0 to 0x0F	Serializer Lane Polarity Inversion Select – one bit per lane
desInvertLanePolarity	0x0 to 0x0F	Deserializer Lane Polarity Inversion Select – one bit per lane
desEqSetting	0 to 3	Deserializer Equalizer setting. Applied to all deserializer lanes.

JESD204B/JESD204C Deframer Enumerated Types

adi_adrv9025_ DeframerSel

The adi_adrv9025_DeframerSel is an enumerated data type to select the desired Deframer. The allowable values are listed in Table 34.

Table 34. Deframer Selection Enumeration Description

Enumeration Value	Description
ADI_ADRV9025_DEFRAMER_0	Deframer 0 selection
ADI_ADRV9025_DEFRAMER_1	Deframer 1 selection
ADI_ADRV9025_DEFRAMER_0_AND_1	Deframer 0 and 1 selection

adi_adrv9025_ DeframerPrbsOrder

The adi_adrv9025_DeframerPrbsOrder is an enumerated data type to select the desired Deframer PRBS pattern. The allowable values are listed in Table 35.

Table 35. Deframer PRBS Polynomial Order Enumeration Description

Enumeration Value	Description
ADI_ADRV9025_PRBS_DISABLE	Deframer PRBS pattern disable
ADI_ADRV9025_PRBS7	Deframer PRBS7 pattern select
ADI_ADRV9025_PRBS15	Deframer PRBS15 pattern select
ADI_ADRV9025_PRBS31	Deframer PRBS31 pattern select

adi_adrv9025_DeframerPrbsCheckLoc

The adi_adrv9025_DeframerPrbsCheckLoc is an enumerated data type to select the desired location within the Deframer to check the PRBS pattern. The allowable values are listed in Table 36.

Table 36. Deframer PRBS Check Location Enumeration Description

Enumeration Value	Description
ADI_ADRV9025_PRBSCHECK_LANEDATA	Check PRBS at deserializer lane output (does not allow JESD204B/JESD204C link to be established)
ADI_ADRV9025_PRBSCHECK_SAMPLEDATA	Check PRBS at output of deframer (JESD204B/JESD204C deframed sample)

API Functions

adi_adrv9025_DeframerSysrefCtrlSet(…)

```
adi_adrv9025_DeframerSysrefCtrlSet(adi_adrv9025_Device_t *device, adi_adrv9025_DeframerSel_e
  deframerSel, uint8_t enable)
```

This function enables or disables the external SYSREF to the deframers of the transceiver.

For the deframer to retime its LMFC /LEMC (local multi frame clock/local extended multiblock clock), a SYSREF rising edge is required. The external SYSREF signal at the pin can be gated off internally so the deframer does not see a potential invalid SYSREF pulse before it is configured correctly.

By default the device has the SYSREF signal ungated, however, the Multichip Sync state machine still does not allow the external SYSREF to reach the deframer until the other stages of multichip sync have completed. As long as the external SYSREF is correctly configured before performing MCS, this function may not be needed by the baseband processor, since the MCS state machine gates the SYSREF to the deframer.

Precondition

This function is called after the device has been initialized and the JESD204B/JESD204C deframer is enabled.

Dependencies

device->devHalInfo

Parameters

Table 37.

Parameter	Description
*device	Pointer to the device settings structure
deframerSel	Select deframer to enable/disable SYSREF input for (Valid ADI_ADRV9025_DEFRAMER_0, ADI_ADRV9025_DEFRAMER_1 or ADI_ADRV9025_DEFRAMER_0_AND_1)
enable	1 = enable SYSREF to deframer, 0 = disable SYSREF to deframer

Return Values

Table 38.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_DfrmLinkStateSet (…)

```
adi_adrv9025_DfrmLinkStateSet(adi_adrv9025_Device_t *device, uint8_t deframerSelMask, uint8_t
  enable)
```

This function is normally not necessary. In the event that the link needs to be reset, this function allows a deframer to be disabled and re-enabled.

During disable, the lane FIFOs for the selected deframer are also disabled. When the deframer link is enabled, the lane FIFOs for the selected deframer are reenabled (reset). The baseband processor sends valid serializer data before enabling the link so the device CDR (recovered clock) is locked.

Precondition

This function may be called any time after device initialization.

Dependencies

device->devHalInfo

Parameters

Table 39.

Parameter	Description
*device	Pointer to the device settings data structure
deframerSelMask	Desired deframer to reset. Valid ADI_ADRV9025_DEFRAMER_0, ADI_ADRV9025_DEFRAMER_1 or ADI_ADRV9025_DEFRAMER_0_AND_1
enable	0 = disable the selected deframer, 1 = enable the selected deframer link

Return Values

Table 40.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_DeframerStatusGet(…)

```
adi_adrv9025_DeframerStatusGet(adi_adrv9025_Device_t *device, adi_adrv9025_DeframerSel_e
  deframerSel, adi_adrv9025_DeframerStatus_t *deframerStatus)
```

After bringing up the deframer JESD204B/JESD204C link, the baseband processor can check the status of the deframer for the parameters shown in Table 41.

Table 41. Deframer Status Parameters

deframerStatus	Bit Name	Description
7	Valid checksum	1 if the checksum calculated by the device matched the one sent in the ILAS data.
6	EOF Event	This bit captures the internal status of the End of Frame event of the deframer. Value =1 if framing error during ILAS
5	EOMF Event	This bit captures the internal status of the End of Multiframe event of the deframer. Value = 1 if framing error during ILAS
4	FS Lost	This bit captures the internal status of the Frame Symbol event of the deframer. Value = 1 if framing error during ILAS or user data (invalid replacement characters)
3	Reserved	
2	User Data Valid	= 1 when in user data (deframer link is up and sending valid DAC data)
1	SYSREF Received	Deframer has received the external SYSREF signal
0	Syncb level	Current level of $\overline{\text{SYNC}}$ signal internal to deframer (= 1 means link is up)

Precondition

The Tx JESD204B/JESD204C link(s) needs to be configured and running to use this function.

Dependencies

device->devHalInfo

Parameters

Table 42.

Parameter	Description
*device	is a pointer to the device settings structure
deframerSel	Select the Deframer to read back the status of ADI_ADRV9025_DEFRAMER_0, ADI_ADRV9025_DEFRAMER_1 or ADI_ADRV9025_DEFRAMER_0_AND_1
deframerStatus	8 bit deframer status word return value

Return Values

Table 43.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_DfrmPrbsCheckerStateSet(…)

```
adi_adrv9025_DfrmPrbsCheckerStateSet(adi_adrv9025_Device_t *device, adi_adrv9025_DfrmPrbsCfg_t
*dfrmPrbsCfg)
```

This function configures and enables or disables the transceiver lane or sample PRBS checker. This is a debug function to be used for debug of the Tx JESD204B/JESD204C lanes.

If the checkerLocation is ADI_ADRV9025_PRBSCHECK_LANEDATA, the PRBS is checked at the output of the deserializer. If the checkLocation is ADI_ADRV9025_PRBSCHECK_SAMPLEDATA the PRBS data is expected to be framed JESD204B/JESD204C data and the PRBS is checked after the JESD204B/JESD204C data is deframed. For the sample data, there is only a PRBS checker on DAC 0 input. The lane PRBS has a checker on each deserializer lane.

Precondition

This function may be called any time after device initialization.

Dependencies

device->devHalInfo

Parameters

Table 44.

Parameter	Description
*device	is a pointer to the device settings structure
polyOrder	selects the PRBS type based on enum values (ADI_ADRV9025_PRBS_DISABLE, ADI_ADRV9025_PRBS7, ADI_ADRV9025_PRBS15, ADI_ADRV9025_PRBS31)
checkerLocation	Check at deserializer (deframer input) or sample (deframer output).

Return Values

Table 45.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_DfrmPrbsCountReset(…)

```
adi_adrv9025_DfrmPrbsCheckerStateSet(adi_adrv9025_Device_t *device, adi_adrv9025_DfrmPrbsCfg_t
  *dfrmPrbsCfg)
```

This function allows the baseband processor to clear the Deframer PRBS counters. It resets the PRBS error counters for all lanes. It is recommended to clear the error counters after enabling the deframer PRBS checker.

Precondition

The Tx JESD204B/JESD204C link(s) needs to be configured to use this function.

Dependencies

device->devHalInfo

Parameters

Table 46.

Parameter	Description
*device	is a pointer to the device settings structure

Return Values

Table 47.

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

adi_adrv9025_DfrmPrbsErrCountGet(…)

```
adi_adrv9025_DfrmPrbsErrCountGet(adi_adrv9025_Device_t *device,
adi_adrv9025_DfrmPrbsErrCounters_t *counters)
```

After enabling the deframer PRBS checker and clearing the PRBS error counters, use this function to read back the PRBS error counters. The lane parameter allows the baseband processor to select which lane error counter to read. Only one lane error counter can be read at a time. To read error counters for all four lanes, the baseband processor calls this function four times.

In the case that the PRBS checker is set to check at the deframer output sample, there is only a checker on the DAC 0 input. In this case the lane function parameter is ignored and the sample 0 PRBS counter is returned. The sample crossbar can be used to switch all deframer outputs to DAC0 in turn.

Precondition

The Tx JESD204B/JESD204C link(s) needs to be configured to use this function.

Dependencies

device->devHalInfo

Parameters

Table 48.

Parameter	Description
*device	Pointer to the device settings structure
counters	Pointer to PRBS Error counter structure to be returned

Return Values

Return Value	Description
ADI_ADRV9025_ACT_WARN_RESET_LOG	Recovery action for log reset
ADI_ADRV9025_ACT_ERR_CHECK_PARAM	Recovery action for bad parameter check
ADI_ADRV9025_ACT_ERR_RESET_SPI	Recovery action for SPI reset required
ADI_ADRV9025_ACT_NO_ACTION	Function completed, no action required

API SOFTWARE INTEGRATION

Configuration of the JESD204B/JESD204C circuitry is handled by the adi_adrv9025_Initialize(…) API function. Set all JESD204B/JESD204C link options in the adi_adrv9025_Init_t data structure before calling adi_adrv9025_Initialize(…).

JESD204B/JESD204C API Data Structures

adi_adrv9025_ DataInterfaceCfg _t

The adi_adrv9025_ DataInterfaceCfg_t data structure contains the information required to properly configure each framer, each deframer, the serializers, and deserializers. Details of each member can be found in API documentation (/c_src/doc).

```
typedef struct adi_adrv9025_DataInterfaceCfg
{
    adi_adrv9025_FrmCfg_t framer[3];
    adi_adrv9025_DfrmCfg_t deframer[2];
    adi_adrv9025_SerCfg_t serCfg[8];
    adi_adrv9025_DesCfg_t desCfg[8];
    uint8_t sysrefLvdsMode;
    uint8_t sysrefLvdsPnInvert;
    adi_adrv9025_LinkSharingCfg_t linkSharingCfg;
} adi_adrv9025_DataInterfaceCfg_t;
```

Table 49. JESD204B/JESD204C Settings Structure Member Description

Structure Member	Valid Values	Description
framer0	data structure	Framer 0 configuration data structure
framer1	data structure	Framer 1 configuration data structure
framer2	data structure	Framer 2 configuration data structure
deframer0	data structure	Deframer 0 configuration data structure
deframer1	data structure	Deframer 1 configuration data structure
serAmplitude	0..3	Serializer amplitude setting. Default = 1.
serPreEmphasis	0..2	Serializer pre-emphasis setting. Default = 0.
serInvertLanePolarity	0x0 to 0x0F	Serializer Lane Polarity Inversion Select – one bit per lane
desInvertLanePolarity	0x0 to 0x0F	Deserializer Lane Polarity Inversion Select – one bit per lane
desEqSetting	0 to 3	Deserializer Equalizer setting. Applied to all deserializer lanes.

IMPLEMENTATION RECOMMENDATIONS

- SYSREF must be dc-coupled. If SYSREF is generated by GPIO pins for example, both pins being in the low state at startup is not valid. Ensure that the signals are active and/or in a known valid state prior to enabling the MCS gate.
- For 25G operation, it is recommended to use deframer Lane A and Lane C to minimize crosstalk possibilities.
- Deframer input amplitude is on the order of 500 mV p-p to 700 mV p-p if insertion loss is on the order of 5 dB at room temp.
- Minimizing data link uncertainty:
  - Ensure setup and hold times are met for each SYSREF/DCLK pair
  - Separate the SYSREF/DCLK pairs for each device in the system
  - Match the trace length within each pair so that the propagation time is the same

LINK INITIALIZATION AND DEBUGGING

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

JESD204B

For the deframer side, follow these steps:

1. Initialize and bring up the baseband processor framer side.
2. Deframer is held in reset state until INIT command, then deframer issues a synchronization request by asserting the SYNC signal.
3. Framer starts sending K28.5 characters, then deframer is brought out of reset.
4. Deframer identifies four consecutive K28.5 characters then deasserts SYNC and goes into ILAS phase.
5. If SYNC stays asserted, this indicates it is stuck in CGS phase. Check that the link parameters match. If they do, check the signal integrity (refer to the Sample Iron Python Code for PRBS Testing section).

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

JESD204C

For the deframer side, follow these steps:

1. Initialize and bring up the baseband processor framer side.
2. Send the JESD204C initialization calibration command. This brings the link up since it is now protocol based.
3. Enable the JESD204C tracking calibrations. This maintains the link parameters on a 60 second schedule.

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

The API function adi_board_adrv9025_JesdBringup is used to configure and establish the datalinks. The overall detailed sequence including the MCS is in the file adi_adrv9025_daughter_board.c.

FIRST TIME SYSTEM BRING UP—CHECKING LINK INTEGRITY

1. For ease of debug during bring up, it is recommended to start with single lane on both sides and with minimum possible link speed.
2. Check that the parameters are configured the same at both ends transceiver and FPGA. The adi_adrv9025_DfrmCfg_t data structure contains the information required to properly configure each deframer.
3. There is a PRBS checker available that can be used to check signal integrity related issues. Initialize the device as outlined in the link establishment section. Enable the PRBS generator on the baseband processor with the desired PRBS sequence.
4. Confirm that the lanes baseband processor is transmitting PRBS on are the actually configured in the ADRV9026. Start with the PRBS errors. Ensure baseband processor and the ADRV9026 are both using the same PRBS signal and ADRV9026 expects the same PRBS 7 from baseband processor.
5. Call the API adi_adrv9025_DfrmPrbsCheckerStateSet(…) passing the actual device being evaluated, the PRBS sequence to check, and the location at which to check the PRBS sequence.
6. After some amount of time, call the API function to check the PRBS errors. This can be done by calling the API function adi_adrv9025_DfrmPrbsErrCountGet(…) passing the actual device being evaluated, the counter selection lane to be read and the error count is returned in the third parameter passed.
7. The user can use adi_adrv9025_DeframerSysrefCtrlSet(…) API so that The external SYSREF signal at the pin can be gated off internally so the deframer does not see a potential invalid SYSREF pulse before it is configured correctly.
8. After bringing up of JESD204B link or for debugging the deframer, the baseband processor can check the status of the deframer using adi_adrv9025_DeframerStatusGet(…).

SAMPLE IRON PYTHON CODE FOR PRBS TESTING

The following Iron Python script can be loaded into the Iron Python tab in the GUI to run the PRBS test. To use this code, select File →New and place this code just after the ##### YOUR CODE GOES HERE ##### note.

```
#Create an Instance of the Class
link = AdiEvaluationSystem.Instance
connect = False
adrv9025 = link.Adrv9025Get(1)

FrmTestDataCfg=Types.adi_adrv9025_FrmTestDataCfg_t()
FrmTestDataCfg.framerSelMask=int(Types.adi_adrv9025_FramerSel_e.ADI_ADRV9025_FRAMER_0)
print FrmTestDataCfg.framerSelMask
FrmTestDataCfg.testDataSource=Types.adi_adrv9025_FramerDataSource_e.ADI_ADRV9025_FTD_PRBS7
FrmTestDataCfg.injectPoint=Types.adi_adrv9025_FramerDataInjectPoint_e.ADI_ADRV9025_FTD_SERIALIZE
R
adrv9025.DataInterface.FramerTestDataSet(FrmTestDataCfg)

#Enable Deserializer
link.platform.board.Fpga.Prbs.PrbsDeserializerEnable(0xF,0x1) #1:PRBS7;2:PRBS9;3:PRBS15;5:PRBS31
link.platform.board.Fpga.Prbs.PrbsDeserializerEnable(0xF,0x1) #1:PRBS7;2:PRBS9;3:PRBS15;5:PRBS31
#clear PRBS error
link.platform.board.Fpga.Prbs.PrbsErrorClear(0xF)
#Read PRBS error
#adrv9025.DataInterface.FramerTestDataInjectError(Types.adi_adrv9025_FramerSel_e.ADI_ADRV9025_FR
AMER_0,0x0)
time.sleep(1)

errCounts=Array[System.UInt32]([0,0,0,0,0,0,0,0])
errCounts=link.platform.board.Fpga.Prbs.PrbsErrorCountsRead(errCounts)[1]
errCounts=[int(data) for data in errCounts]
print errCounts   #[0,0,0,0,0,0,0,0]
```

When this script is run, it results in the number of errors per enabled lane. Note only the first 4 positions are valid and the last four positions is always be 0. To create errors as a test, change the 0x1 in the line immediately below "Enable Deserializer" comment to one of the other values indicated. The enabled lanes show errors by enabled lane position.

PRBS ERRORS

When the baseband processor is transmitting PRBS, confirm that the active lanes are also configured properly in the ADRV9026. Start with the PRBS errors. Ensure the baseband processor and the ADRV9026 are both using the same PRBS signal and the ADRV9026 expects the same PRBS 7 from baseband processor. The following are some scenarios that might occur and how to resolve issues.

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

1. Check the board, unpowered for the following:
   a. SYSREF and SYNC signaling is dc-coupled.
   b. Check that the pull-down or pull-up resistors are not dominating the signaling, for example if values are too small or shorted and therefore cannot be driven correctly.
   c. Verify that the differential-pairs traces are length matched.
   d. Verify differential impedance of the traces is 100 Ω.

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

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

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

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

1. Invalid setup and hold time of periodic or gapped periodic SYSREF or SYNC signal.
2. Link parameter conflicts
3. Character replacement conflicts
4. Scrambling problem, if enabled
5. Lane data corruption, noisy or jitter can force the eye diagram to close
6. Spurious clocking or excessive jitter on device clock

SPO (STATIC PHASE OFFSET) TEST TO VERIFY EYE WIDTH

High speed data rates present a tougher challenge because signal integrity is required for reliable error free data transfer. See the PCB Layout Considerations section for differential line layout recommendations.

When debugging lane errors, it can be useful to understand how large the 'eye' of the waveform is to determine how reliable the link is. In the case of the Deframer, in 204C mode the channel is estimated during an initialization cal that configures the CTLE (continuous time linear equalizer) and automatically adjusts the sampling position on the waveform. To gain confidence in the link stability, the opening of the eye over the operating conditions is one measure of robustness. A method of determining the opening size is to sweep the sampling position, searching for 'dead space' where no transitions are occurring therefore the sampling point is in the eye. This is called a SPO (Static Phase Offset) test that offsets the clocks to move the sampling edge left or right on the waveform and the resulting 'dead' steps total at least 4 steps left and right from center, over all operating conditions the link is considered 'good'. The SPO test requires PRBS transmission in the FPGA and setup of the PRBS pattern checker in the TRX.

A typical test output report is shown below. In this case, two lanes are in use. The phase is swept in 128 steps The resolution is of course dependent on the lane rate, but in general this result shown is considered good with approximately 16 phase steps open in the center of the eye as shown in the resulting output files.

SPO Test Example Python Script

The SPO test code can be run in the GUI and works for both JESD204B and for JESD204C. The user needs to set the first line appropriately and also configure the output file path to a folder on the PC. Insert these functions in the def section of the New Script, as follows:

```
def FpgaWrite(address, data):
    link.platform.board.Fpga.Hal.RegisterWrite(address, data)
    #print "FPGA Write Address " + hex(address) + ": " + hex(data)


def FpgaRead(address):
    data = link.platform.board.Fpga.Hal.RegisterRead(address, 0)
    print "FPGA Read Address " + hex(address) + ": " + hex(data[1])



def FPGAPRBSSetup(mode_is_204c=0):
    enablePRBS_ch1 = link.platform.board.Fpga.Hal.RegisterRead(0x43400220,0)
                    #Read the value in PRBS control register (FPGA ch1 testmodes register)
    disablePRBS = enablePRBS_ch1[1] & 0xF0ffFFFF
                                            #Zero bits 27-24 without affecting the other bits
in the register.
    enablePRBS7 = disablePRBS | 0x01000000
                                            #Set the enablePRBS variable bits 27-24 to 0001
to enable PRBS7
    enablePRBS23 = disablePRBS | 0x05000000
                                        #Set the enablePRBS variable bits 27-24 to 0101 to
enable PRBS23


    for fpgaregister in range (0x43400100, 0x43400900, 0x100):
                                #Update all FPGA ch0-7
        if (mode_is_204c == 1):
            FpgaWrite(fpgaregister, 0x00000004)
                                        #Puts the lane transmit side in reset
            FpgaWrite(fpgaregister + 0x48, 0x20800080)
                                        #Sets the data and data mask for the DRP write to enable
the buffer and disable the gearbox
            FpgaWrite(fpgaregister + 0x40, 0x0003007C)
                                        #Initiates the write to the DRP
```

```
            FpgaWrite(fpgaregister + 0x10, 0x02015233)
                                        #Sets the transmit clock source to the PMA clock
        FpgaWrite(fpgaregister + 0x20, enablePRBS7)
                                        #Write the new value back to the FPGA to enable PRBS7 -
ch1(fpga) to ch7 =serdinA to H
        if (mode_is_204c):
            FpgaWrite(fpgaregister, 0x00000000)
                                            #Remove reset


    print "PRBS7 is enabled", hex(disablePRBS), hex(enablePRBS7), hex(enablePRBS23)



    ErrorCount = Types.adi_adrv9025_DfrmPrbsErrCounters_t()
    dfrmPrbsCfg = Types.adi_adrv9025_DfrmPrbsCfg_t()
    dfrmPrbsCfg.deframerSel = dfrm_sel
    dfrmPrbsCfg.polyOrder = Types.adi_adrv9025_DeframerPrbsOrder_e.ADI_ADRV9025_PRBS7
    dfrmPrbsCfg.checkerLocation =
Types.adi_adrv9025_DeframerPrbsCheckLoc_e.ADI_ADRV9025_PRBSCHECK_LANEDATA
    adrv9025.DataInterface.DfrmPrbsCheckerStateSet(dfrmPrbsCfg)
    #check config matches what you've written
    dfrmPrbsCfgRead = Types.adi_adrv9025_DfrmPrbsCfg_t()
    adrv9025.DataInterface.DfrmPrbsCheckerStateGet(dfrmPrbsCfgRead)
    print "PRBS config setup, Poly, location,drmrSel", dfrmPrbsCfgRead.polyOrder,
dfrmPrbsCfgRead.checkerLocation, dfrmPrbsCfgRead.deframerSel


    adrv9025.DataInterface.DfrmPrbsCountReset()
    adrv9025.DataInterface.DfrmPrbsErrCountGet(ErrorCount)          #api method to read error
counters + flags


    for lanes in range(len(ErrorCount.laneErrors)):
        print "Initial laneError count for lane", lanes, "is :", ErrorCount.laneErrors[lanes]
        print "Initial ErrorStatus for lane", lanes, "is :", ErrorCount.errorStatus[lanes]  #Bit
0 = Lane inverted, bit 1 = invalid data flag, bit 2 = sample/lane error flag


    if ErrorCount.laneErrors[0] == 0:
        print "No Errors detected as expected in PRBS7 mode. Will switch to PRBS23 now"
    else:
        print "Errors detected!! Link not good, please check link"


    for fpgaregister in range (0x43400100, 0x43400900, 0x100):
                                #Update all FPGA ch0-7
        link.platform.board.Fpga.Hal.RegisterWrite(fpgaregister + 0x20, enablePRBS23)
                        #Write to the FPGA to enable PRBS23 on all Ch
    print "Changing to PRBS23"

    adrv9025.DataInterface.DfrmPrbsCountReset()
    adrv9025.DataInterface.DfrmPrbsErrCountGet(ErrorCount)


    for lanes in range(len(ErrorCount.laneErrors)):
        print "PRBS23 laneError count for lane", lanes, "is :", ErrorCount.laneErrors[lanes]
```

```
        print "PRBS23 ErrorStatus for lane", lanes, "is :", ErrorCount.errorStatus[lanes]

    if ErrorCount.laneErrors[0] != 0:
        print "Errors detected as expected with PRBS mismatch. Will switch back to PRBS7 now"
    else:
        print "Errors not detected with PRBS mismatch !! Please verify PRBS generator in FPGA"

    for fpgaregister in range (0x43400100, 0x43400900, 0x100):
                                #Update all FPGA ch0-7
        link.platform.board.Fpga.Hal.RegisterWrite(fpgaregister + 0x20, enablePRBS7)
    print "PRBS7 is enabled again on all channels"
    adrv9025.DataInterface.DfrmPrbsCountReset()
    adrv9025.DataInterface.DfrmPrbsErrCountGet(ErrorCount)
    if ErrorCount.laneErrors[0] == 0:
       print "No Errors detected after switching back to PRBS7 mode. Will move onto phase/amp
eye sweep"
    else:
       print "Errors detected!! please check setup - may need to reboot"
```

Insert the following in the **Iron Python** tab after the line: ***#####* YOUR CODE GOES HERE *#####*** (approximately Line 40). See Figure 21 for the SPO test measurement result.

```
mode_is_204c = 0                    # need to setup FPGA differently for 204c vs. 204b mode, so
set this bit appropriately.
foldername = "C:\\tmp"

errorTimeDuration = 0.001           #time duration to allow PRBS errors to accumulate
LaneErrorFlag = []                  #containers to store ErrorFlag Data for each lane to print
to file
LaneErrorCntr= []

dfrmPrbsCfg = Types.adi_adrv9025_DfrmPrbsCfg_t()
ErrorCount = Types.adi_adrv9025_DfrmPrbsErrCounters_t()
dfrm_sel = Types.adi_adrv9025_DeframerSel_e.ADI_ADRV9025_DEFRAMER_0

FPGAPRBSSetup(mode_is_204c)     #Setup PRBS TestMode on FPGA side

dfrmPrbsCfg.deframerSel = dfrm_sel
dfrmPrbsCfg.polyOrder = Types.adi_adrv9025_DeframerPrbsOrder_e.ADI_ADRV9025_PRBS7       #can
configure PRBS mode on Madura
dfrmPrbsCfg.checkerLocation =
Types.adi_adrv9025_DeframerPrbsCheckLoc_e.ADI_ADRV9025_PRBSCHECK_LANEDATA
adrv9025.DataInterface.DfrmPrbsCheckerStateSet(dfrmPrbsCfg)

adrv9025.DataInterface.DfrmPrbsCountReset()
adrv9025.DataInterface.DfrmPrbsErrCountGet(ErrorCount)          #Run initial PRBS error check -
should have zero errors initially

for lanes in range(len(ErrorCount.laneErrors)):
```

```
    print "Initial laneError count for lane", lanes, "is :", ErrorCount.laneErrors[lanes]
    print "Initial ErrorStatus for lane", lanes, "is :", ErrorCount.errorStatus[lanes]  #Bit 0 =
Lane inverted, bit 1 = invalid data flag, bit 2 = sample/lane error flag



for phase in range (64,192,1):
    phase = phase % 128                             #Offset the phase to centre the eye
    spiWrite(0x6805, 0xD)           # Write the serdes submap addr
    spiWrite(0x6808, phase | 0x80)  # Write the phase data
    spiWrite(0x6806, 0x0F)          # Latch in phase data for all lanes
    spiWrite(0x6806, 0x00)          #clear latch
    adrv9025.DataInterface.DfrmPrbsCountReset()

    time.sleep(errorTimeDuration)                                   #Set a wait time to allow errors
to accumulate
    adrv9025.DataInterface.DfrmPrbsErrCountGet(ErrorCount)


    for lanes in range(len(ErrorCount.laneErrors)):             #readback errors from each lanes
and store in an array
        LaneErrorFlag.append(int(ErrorCount.errorStatus[lanes] >> 2) & 0x1)
        LaneErrorCntr.append(ErrorCount.laneErrors[lanes])


# Print ErrorFlag & ErrorCounter data to files
filename = "{0}\\eyedata_lane.txt".format(foldername)
filename2 = "{0}\\cntrdata_lane.txt".format(foldername)
with open(filename, 'w') as f1, open(filename2, 'w') as f2:
    f1.write("LaneErrorFlag[0]\tLaneErrorFlag[1]\tLaneErrorFlag[2]\tLaneErrorFlag[3]\n")
    f2.write("LaneErrorCntr[0]\tLaneErrorCntr[1]\tLaneErrorCntr[2]\tLaneErrorCntr[3]\n")
    for i in range(0, len(LaneErrorFlag),4):                #print out the eye diagram ascii
symbols to file
        f1.write("{0}\t{1}\t{2}\t{3}\n".format(LaneErrorFlag[i],
LaneErrorFlag[i+1],LaneErrorFlag[i+2],LaneErrorFlag[i+3]))
        f2.write("{0}\t{1}\t{2}\t{3}\n".format(LaneErrorCntr[i],
LaneErrorCntr[i+1],LaneErrorCntr[i+2],LaneErrorCntr[i+3]))
```

```
Connected
PRBS7 is enabled 0x0L 0x1000000L 0x5000000L
PRBS config setup, Poly, location,drmrSel ADI_ADRV9010_PRBS7 ADI_ADRV9010_PRBSCHECK_LANEDATA 0
Initial laneError count for lane 0 is : 0
Initial ErrorStatus for lane 0 is : 0
Initial laneError count for lane 1 is : 0
Initial ErrorStatus for lane 1 is : 0
Initial laneError count for lane 2 is : 0
Initial ErrorStatus for lane 2 is : 0
Initial laneError count for lane 3 is : 0
Initial ErrorStatus for lane 3 is : 0
No Errors detected as expected in PRBS7 mode. Will switch to PRBS23 now
Changing to PRBS23
PRBS23 laneError count for lane 0 is : 333222
PRBS23 ErrorStatus for lane 0 is : 4
PRBS23 laneError count for lane 1 is : 0
PRBS23 ErrorStatus for lane 1 is : 0
PRBS23 laneError count for lane 2 is : 517779
PRBS23 ErrorStatus for lane 2 is : 4
PRBS23 laneError count for lane 3 is : 0
PRBS23 ErrorStatus for lane 3 is : 0
Errors detected as expected with PRBS mismatch. Will switch back to PRBS7 now
PRBS7 is enabled again on all channels
No Errors detected after switching back to PRBS7 mode. Will move onto phase/amp eye sweep
Initial laneError count for lane 0 is : 0
Initial ErrorStatus for lane 0 is : 0
Initial laneError count for lane 1 is : 0
Initial ErrorStatus for lane 1 is : 0
Initial laneError count for lane 2 is : 0
Initial ErrorStatus for lane 2 is : 0
Initial laneError count for lane 3 is : 0
Initial ErrorStatus for lane 3 is : 0
```

22770-022

Figure 21. SPO Test Measurement Result

The test reported in Figure 21 was run on UC14C-LS on the evaluation board platform with the result indicating that initially there are no PRBS errors. Then errors are injected with the resulting error counts, and the eye sweep is run with no errors being reported. In this use case only two deframer lanes are in use: Lane A and Lane C. Data for the unused lanes are 0.

Two files are also generated by the script: cntrdata_lane.txt and eyedata_lane.txt.

The cntrdata_lane.txt indicates the number of errors counted as the phase is adjusted, and the count goes to 0 in the center of the eye.

In the eyedata_lane.txt file, errors are represented by 1 and the eye indicated by 0. Similarly, the 0s occur toward the center of the waveform indicating and acceptable eye width. Following, in Figure 22 and Figure 23, are excerpts from the center of the files.

683046	0	657487	0
602268	0	614251	0
397518	0	469438	0
817921	0	719822	0
685900	0	527792	0
689797	0	611034	0
648607	0	765663	0
437121	0	522234	0
253707	0	442740	0
85281	0	179763	0
18818	0	60046	0
815	0	9462	0
25	0	1529	0
0	0	159	0
0	0	2	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	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	1	0
113	0	2	0
5567	0	170	0
39999	0	2632	0
122796	0	33358	0
230292	0	277960	0
734022	0	748024	0
701161	0	783019	0
532885	0	628545	0
782405	0	896331	0
445538	0	562745	0

22770-023

Figure 22. cntrdata_lane.txt Showing PRBS Error Counts About the Eye Center

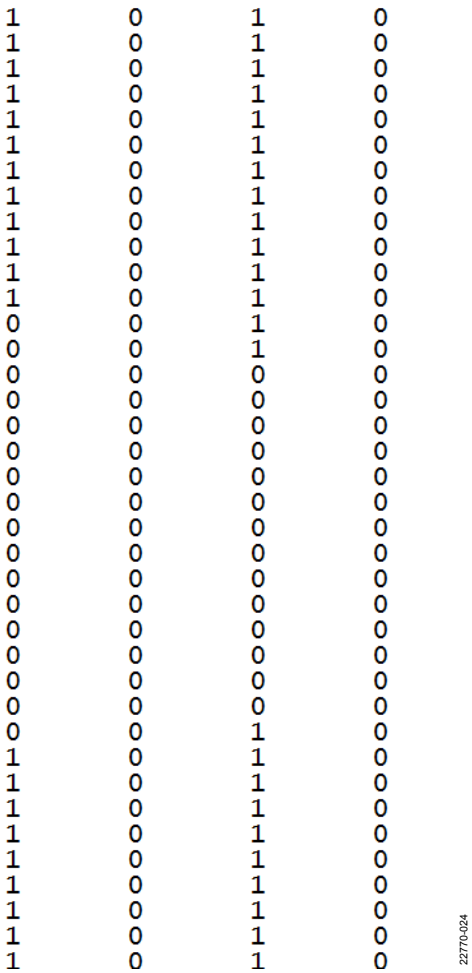


Figure 23. eyedata_lane.txt Showing Center of the Eye

CHECKING JESD204C LINK STATUS

The API for checking the link status is currently not available. Until it is, the registers can be read directly with SPI commands.

Address: 0x6B2B to Address 0x6B2E are for Deframer 0 for Lane A, Lane B, Lane C, and Lane D, respectively.

Address: 0x6D2B to Address 0x6D2E are for Deframer 1 for Lane A, Lane B, Lane C, and Lane D, respectively.

It is only necessary to check as many lanes as the deframer is using. For example, if both deframers are in use and each one uses two lanes, then it is only necessary to check the first two registers in each deframer, not all four.

Table 50.

Bits	Name	Description
7:3	Reserved	Reserved
2:0	Jrx_dl_204c_state	Current Lock State

Table 51.

Bits[2:0]	Description
0	Reset
1	Unlocked
2	Block (blocks aligned)
3	M_Block (lanes aligned)
4	E_M_Block (multiblock aligned)
5	FEC_BUF
6	FEC_READY (good state)
7	Reserved

SELECTING THE OPTIMAL LMFC/LEMC OFFSET FOR A DEFRAMER

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

Deterministic latency in JESD 204B mode

In JESD204B mode, the ADRV9026 digital data interface follows the JESD204B Subclass 1 standard, which has provisions to ensure repeatable latencies across the link from power-up to power-up or over link re-establishment by using the SYSREF signal.

To achieve this deterministic latency, the ADRV9026 deframers include elastic buffers for each of their lanes. The elastic buffers are also used to de-skew each lane before aligning them with the LMFC signal. The depth of the elastic buffers can therefore be different for each lane of a given deframer.

A deframer starts outputting data out of its elastic buffers on the next LMFC (that is, multiframe) boundary following the reception of the first characters in the ILA sequence by all the active lanes. It is therefore possible to adjust when the data is output from the elastic buffers, and therefore how much data is stored in those buffers (called buffer depth), by adjusting the phase relationship between the external SYSREF signal and the internally generated LMFC signal. This phase relationship is adjustable by using the LMFC offset parameter, which is programmable for each of the deframers. This is illustrated on Figure 24 and Figure 25.

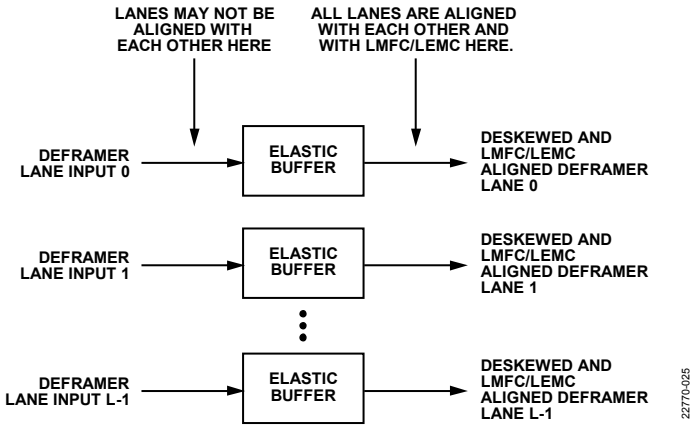


Figure 24. Elastic Buffers in the Deframers

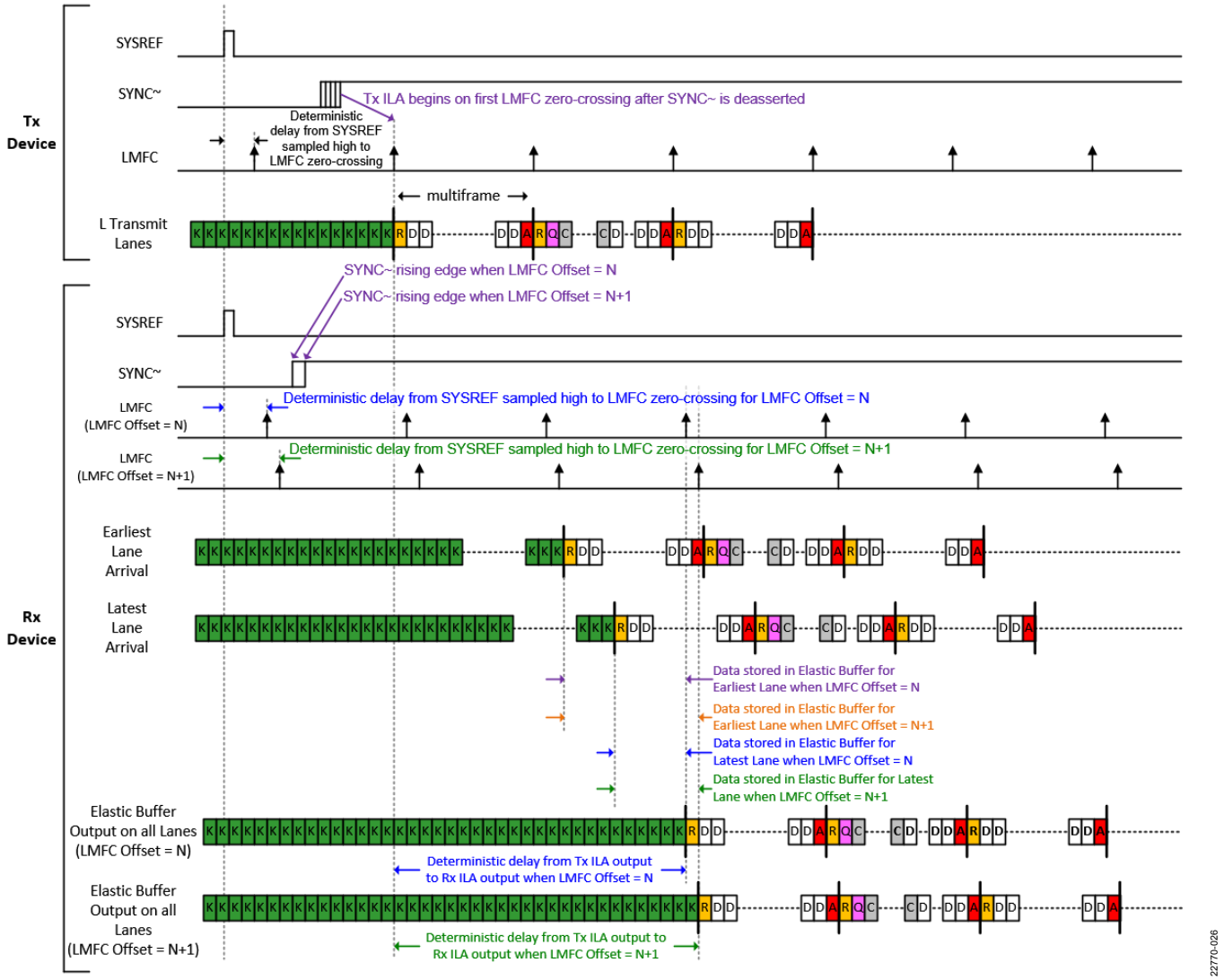


Figure 25. Impact of LMFC Offset on Elastic Buffer Depth in JESD204B Mode

Deterministic Latency in JESD204C Mode

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

A deframer starts outputting data from its elastic buffers on the next LEMC (extended multiblock) boundary following the reception of the first multiblock in an extended multiblock by all the active lanes. As a result, it is possible to adjust when the data is output from the elastic buffers and, therefore, how much data is stored in those buffers (the buffer depth) by adjusting the phase relationship between the external SYSREF signal and the internally generated LEMC signal. This phase relationship is adjustable by using the LEMC offset parameter, which is programmable for each of the deframers. This is illustrated on Figure 24 and Figure 26.

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

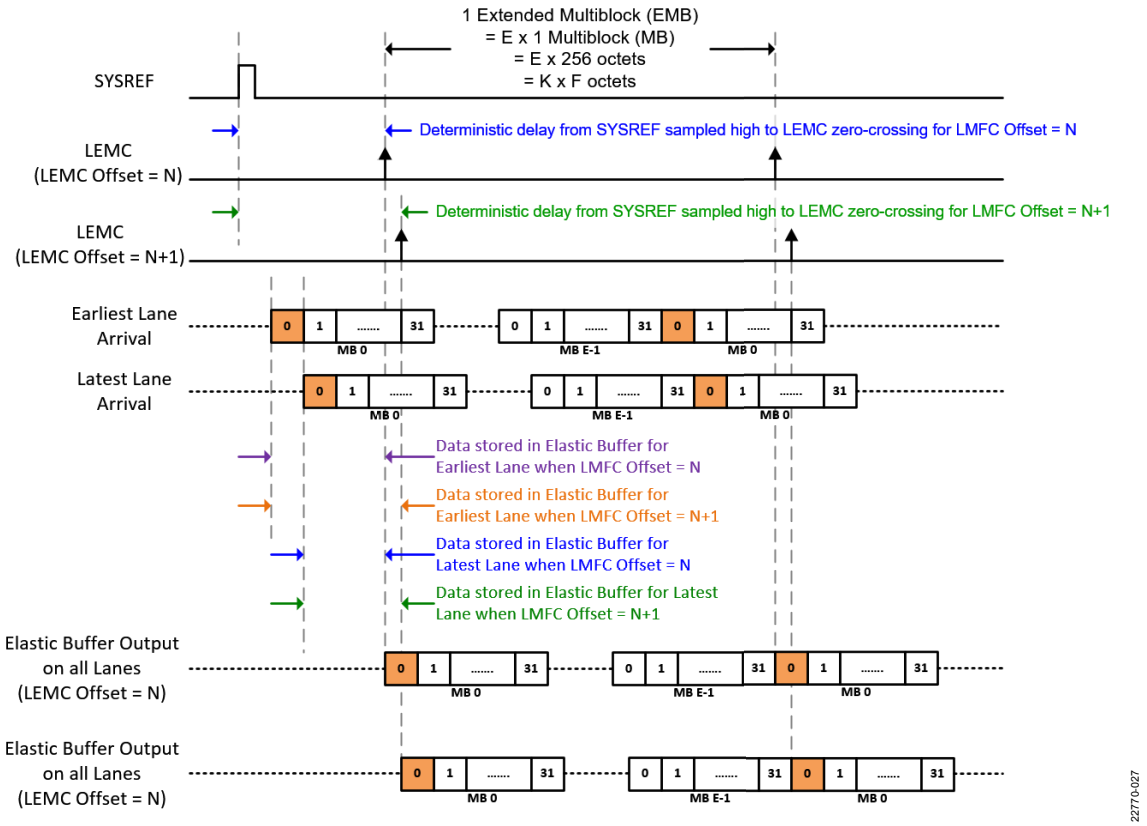


Figure 26. Impact of LEMC Offset on Elastic Buffer Depth in JESD204C Mode

Programming the LMFC Offset for a Deframer

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

1. By modifying the profile file being used
2. By using the adi_adrv9025_DfrmCfg data structure
3. By writing directly to the relevant SPI registers

Each method is addressed in the following sections.

Setting the LMFC/LEMC Offset in the Profile File

There is an lmfcOffset field for each of the two deframers in the profile file. This field corresponds to the LMFC offset in JESD 204B mode, and to the LEMC offset in JESD 204C mode. It can be set to a decimal value between 0 and (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle). For example, for the ADRV9025Init_StdUseCase26C_nonLinkSharing.profile file, the **"lmfcOffset"** field is located around Line 189 for Deframer 0 and around Line 229 for Deframer 1 (see Figure 27).

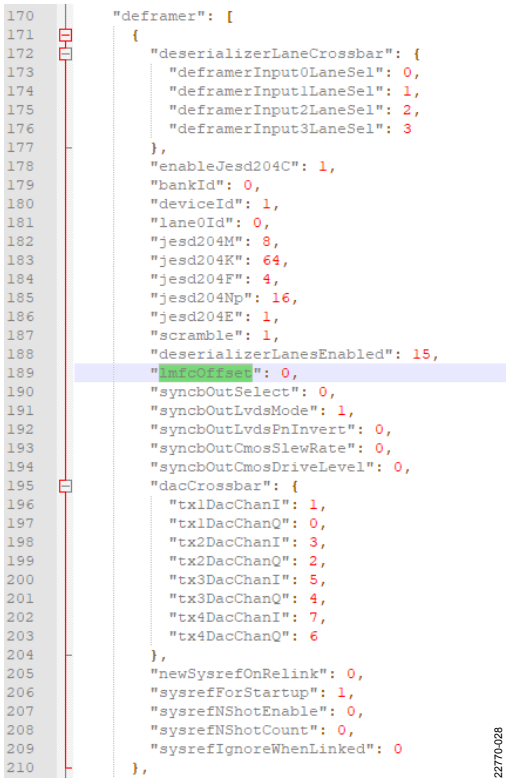


Figure 27. Deframer 0 lmfcOffset Field for the ADRV9025Init_StdUseCase26C_nonLinkSharing.profile File

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

Setting the LMFC/LEMC Offset in the adi_adrv9025_DfrmCfg Data Structure

An alternative way of programming the LMFC/LEMC offset consists in using the lmfcOffset field of the adi_adrv9025_DfrmCfg data structure for the relevant deframer (see Figure 28). Note that the device must be reprogrammed after changing the LMFC/LEMC offset for a given deframer in the adi_adrv9025_DfrmCfg data structure for the change to take effect. Also note that if the goal is to sweep the LMFC/LEMC offset values for test purposes without any need for RF performance (for example, to determine the optimal LMFC/LEMC value), it is not necessary to run the init cals when programming the device. Not running the init cals make the programming process quicker.

```
typedef struct adi_adrv9025_DfrmCfg
{
    uint8_t enableJesd204C;
    uint8_t bankId;
    uint8_t deviceId;
    uint8_t lane0Id;
    uint8_t jesd204M;
    uint16_t jesd204K;
    uint8_t jesd204F;
    uint8_t jesd204Np;
    uint8_t jesd204E;
    uint8_t scramble;
    uint8_t deserializerLanesEnabled;
    uint16_t lmfcOffset;
    uint8_t syncbOutSelect;
    uint8_t syncbOutLvdsMode;
    uint8_t syncbOutLvdsPnInvert;
    uint8_t syncbOutCmosSlewRate;
    uint8_t syncbOutCmosDriveLevel;
    adi_adrv9025_DeserLaneXbar_t deserializerLaneCrossbar;
    adi_adrv9025_DacSampleXbarCfg_t dacCrossbar;
    uint8_t newSysrefOnRelink;
    uint8_t sysrefForStartup;
    uint8_t sysrefNShotEnable;
    uint8_t sysrefNShotCount;
    uint8_t sysrefIgnoreWhenLinked;
} adi_adrv9025_DfrmCfg_t;
```

22770-029

Figure 28. LMFC Offset Field in adi_adrv9025_DfrmCfg Data Structure

It is possible to set the LMFC/LEMC offset value by writing to the following SPI registers: Deframer 0 and Deframer 1.

Deframer 0:

- Register 0x6A8E, Bits[7:0]: jrx_tpl_phase_adjust[7:0]. Bits[7:0] of the LMFC/LEMC phase adjustment 16-bit word for Deframer 0. The valid range of phase adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).
- Register 0x6A8F, Bits[7:0]: jrx_tpl_phase_adjust[15:8]. Bits[15:8] of the LMFC/LEMC phase adjustment 16-bit word for Deframer 0. The valid range of phase adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).

Deframer 1:

- Register 0x6C8E, Bits[7:0]: jrx_tpl_phase_adjust[7:0]. Bits[7:0] of the global LMFC/LEMC phase adjustment 16-bit word for Deframer 1. The valid range of phase adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).
- Register 0x6C8F, Bits[7:0]: jrx_tpl_phase_adjust[15:8]. Bits[15:8] of the global LMFC/LEMC phase adjustment 16-bit word for Deframer 1. The valid range of phase adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).

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

Setting the LMFC/LEMC Offset Through SPI Registers Controls

It is possible to set the LMFC/LEMC offset value by writing to the following SPI registers:

Deframer 0:

- Register 0x6A50, Bits[7:0]: jrx_tpl_phase_adjust[7:0]. Bits[7:0] of the LMFC/LEMC phase adjustment 16-bit word for deframer 0. The valid range of phase adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).
- Register 0x6A51, Bits[7:0]: jrx_tpl_phase_adjust[15:8]. Bits[15:8] of the LMFC/LEMC phase adjustment 16-bit word for deframer 0. The valid range of phase adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).

Deframer 1:

- Register 0x6C50, Bits[7:0]: jrx_tpl_phase_adjust[7:0]. Bits[7:0] of the global LMFC/LEMC phase adjustment 16-bit word for deframer 1. The valid range of phase adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).
- Register 0x6C51, Bits[7:0]: jrx_tpl_phase_adjust[15:8]. Bits[15:8] of the global LMFC/LEMC phase adjustment 16-bit word for deframer 1. The valid range of phase adjustment values is 0 to (K × S) − 1 (where K is the number of frames per multiframe/extended multiblock, and S is the number of samples per converter per frame cycle).

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

Reading Back the Buffer Depths for Each Deframer Lanes

It is possible to read back the depths of the elastic buffers for each deframer lanes in the following SPI registers: Deframer 0 and Deframer 1.

Deframer 0:

- Register 0x6A8A, Bits[7:0]: buffer depth for Lane 0 of Deframer 0
- Register 0x6A8B, Bits[7:0]: buffer depth for Lane 1 of Deframer 0
- Register 0x6A8C, Bits[7:0]: buffer depth for Lane 2 of Deframer 0
- Register 0x6A8D, Bits[7:0]: buffer depth for Lane 3 of Deframer 0

Deframer 1:

- Register 0x6C8A, Bits[7:0]: buffer depth for Lane 0 of Deframer 1
- Register 0x6C8B, Bits[7:0]: buffer depth for Lane 1 of Deframer 1
- Register 0x6C8C, Bits[7:0]: buffer depth for Lane 2 of Deframer 1
- Register 0x6C8D[7:0]: buffer depth for Lane 3 of Deframer 1

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

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

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

Buffer Protection

By default, an automatic buffer protection is enabled for the elastic buffers. This automatic buffer protection prevents the read and write pointers from being too close, which can lead to corrupted data being read out of the elastic buffers, as data can be read at the same time it is being written. When the automatic buffer protection detects that the read and write pointers are too close to each other for any of the elastic buffers, a pre-determined buffer depth is used, the data out of the elastic buffer no longer aligns to the LMFC/LEMC output signal, and deterministic latency is lost.

Checking if the Buffer Protection Is Active

It is possible to read back if the buffer protection is active in the following SPI register bits: Deframer 0 and Deframer 1.

Table 52. Deframer 0—Register 0x6A89, Bit 7: jrx_tpl_buf_protection

Bit Setting	Description
0	Buffer protection not active for Deframer 0
1	Buffer protection active for Deframer 0. Buffer read and write pointers were too close with the chosen LMFC/LEMC offset setting. A predetermined buffer depth is used. Deterministic latency is lost.

Table 53. Deframer 1—Register 0x6C89, Bit 7: jrx_tpl_buf_protection

Bit Setting	Description
0	Buffer protection not active for Deframer 1.
1	Buffer protection active for Deframer 1. Buffer read and write pointers were too close with the chosen LMFC/LEMC offset setting. A predetermined buffer depth is used. Deterministic latency is lost.

Disabling the Automatic Buffer Protection

It is possible to disable the automatic buffer protection by using the following SPI register bits: Deframer 0 and Deframer 1.

Table 54. Deframer 0—Register 0x6A89, Bit 6: jrx_tpl_buf_protection_en

Bit Setting	Description
0	Automatic buffer protection disabled for Deframer 0
1	Automatic buffer protection enabled for Deframer 0

Table 55. Deframer 1— Register 0x6C89, Bit 6: jrx_tpl_buf_protection_en

Bit Setting	Description
0	Automatic buffer protection disabled for Deframer 1
1	Automatic buffer protection enabled for Deframer 1

Following is an example of buffer depth values vs. LMFC offset values in JESD 204B mode with buffer protection enabled:

Figure 29 corresponds to the elastic FIFO buffer depths for Lane 0 and Lane 1 vs. the LMFC offset setting measured for Deframer 0 on the ADRV9026 customer evaluation (CE) board with the ADRV9025Init_StdUseCase50_nonLinkSharing profile. In this example, the buffer protection activated for LMFC offset values between 23 and 26, and the buffer depths were fixed to values between 7 and 9 independently of the LMFC offset. For other LMFC offset values, the buffer depths read back changed with the LMFC offset.

During the measurement, the link between the JESD framer and JESD deframer of the transceiver was reestablished 10 times (with application of a new SYSREF pulse each time) for each LMFC offset value and each time the buffer depth was read. That is why several buffer depth values can be seen for a given LMFC offset. This variation in buffer depth is due to the variance in, for example, synchronization delays and physical lane skews, during the JESD link establishments that the elastic buffers are correcting for.

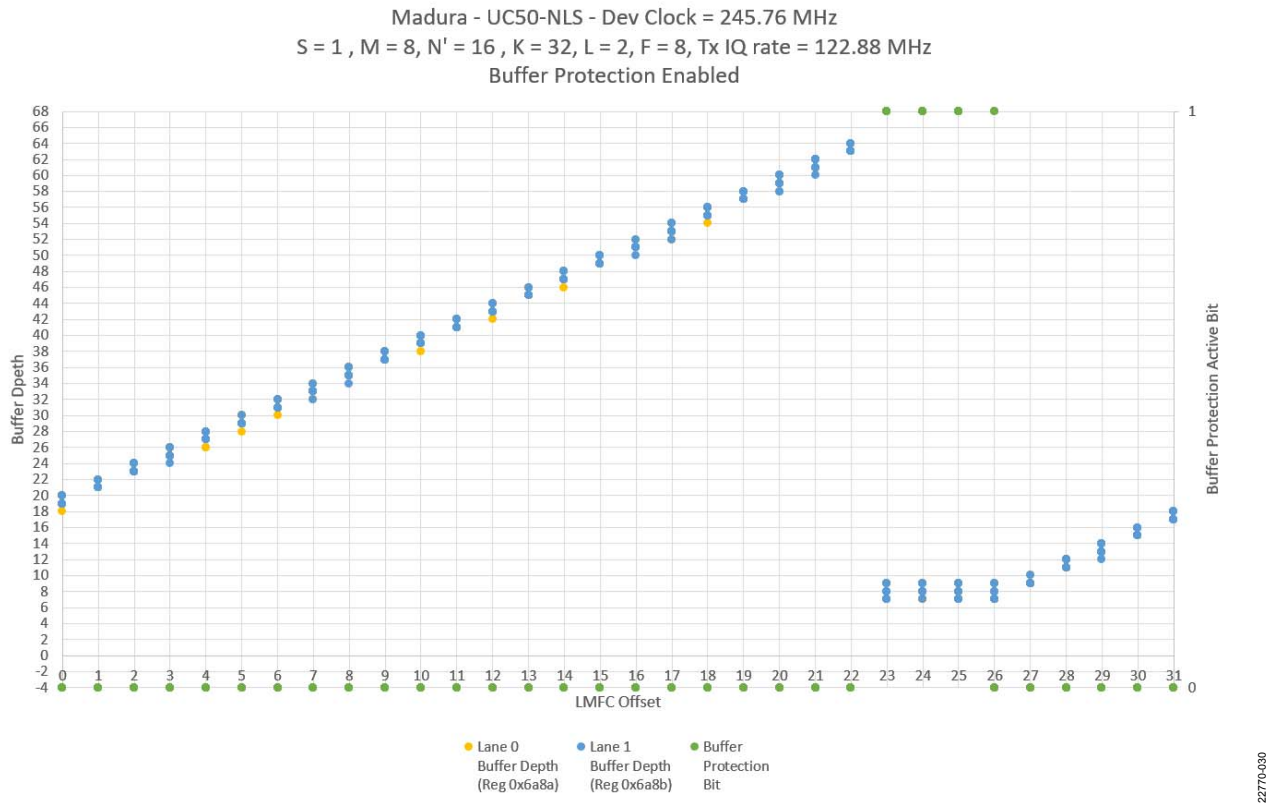


Figure 29. Buffer Depths for Lane 0 and Lane 1 vs. LMFC Offset on the ADRV9026 CE Board with the ADRV9025Init_StdUseCase50_nonLinkSharing Profile and Buffer Protection Enabled

Selecting the optimal LMFC/LEMC offset for a system

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

It is therefore recommended to select an LMFC/LEMC offset value resulting in optimal buffer depths to account for those variations and maintain deterministic latency on all boards for a given system. The LMFC/LEMC offset to be selected depends on whether buffer protection is enabled or not.

Selecting the Optimal LMFC Offset for a System in JESD204B Mode with Buffer Protection Enabled

To ensure deterministic latency when buffer protection is enabled, it is recommended to select an LMFC offset value that gives buffer depths values as close as possible to the center of the linear part of the buffer depth vs. LMFC Offset plot for all the lanes used. To find the LMFC offset corresponding to those optimal buffer depths, read back the buffer depth values for all the used lanes for all LMFC offset values with buffer protection enabled on a sample board for a given system. Measuring the buffer depths per LMFC offset for 10 power cycles or JESD link establishments (with application of a new SYSREF pulse each time) provides a good indication of the buffer depths spread for each LMFC offset values. Select an LMFC offset value giving buffer depths as close as possible to the center of the linear part of the buffer depth vs. LMFC Offset plot for all the used lanes.

Figure 29 illustrates this process using the ADRV9026 CE board programmed with the ADRV9025Init_StdUseCase50_nonLinkSharing profile, with automatic buffer protection enabled. In that example, an LMFC offset value of 9 is a good choice because it results in a buffer depth around 37 or 38 for each lane, which is in the middle of the linear part of the plot and therefore guarantees deterministic latency.

If the goal for the system is to achieve deterministic latency with a latency as short as possible, it may be desirable to select an LMFC offset value giving buffer depths as small as possible. In that case, an LMFC offset value above the highest LMFC offset resulting in the automatic buffer protection being active with some additional headroom to account for PVT variations can be selected. In that situation, carry out thorough system testing over all possible temperature, supply and board variations to ensure that the automatic buffer protection never gets activated and that deterministic latency is maintained in all possible operating conditions for the system.

Avoid LMFC offset values giving large buffer depths (that is, near a value of (K × F)/4) because, for some combinations of JESD parameters, it can lead to the write and read pointers being too close and therefore can result in data corruption.

Selecting the Optimal LMFC Offset for a System in a JESD 204B Mode with Buffer Protection Disabled

When buffer protection is disabled, it is recommended to select an LMFC offset value that gives buffer depths as close as possible to (K × F)/8 to account for variations and maintain deterministic latency on all boards for a given system.

To find the LMFC offset corresponding to that optimal buffer depth, read back the buffer depth values for all LMFC offset values for all the used lanes with buffer protection disabled on a sample board for a given system. Measuring the buffer depths per LMFC offset for 10 power cycles or JESD link establishments (with application of a new SYSREF pulse each time) provides a good indication of the buffer depth spread for each LMFC offset values.

Select an LMFC offset value giving buffer depths as close as possible to (K × F)/8 for all lanes.

Figure 30 illustrates this process using the same ADRV9026 CE board with the ADRV9025Init_StdUseCase50_nonLinkSharing profile example with automatic buffer protection disabled.

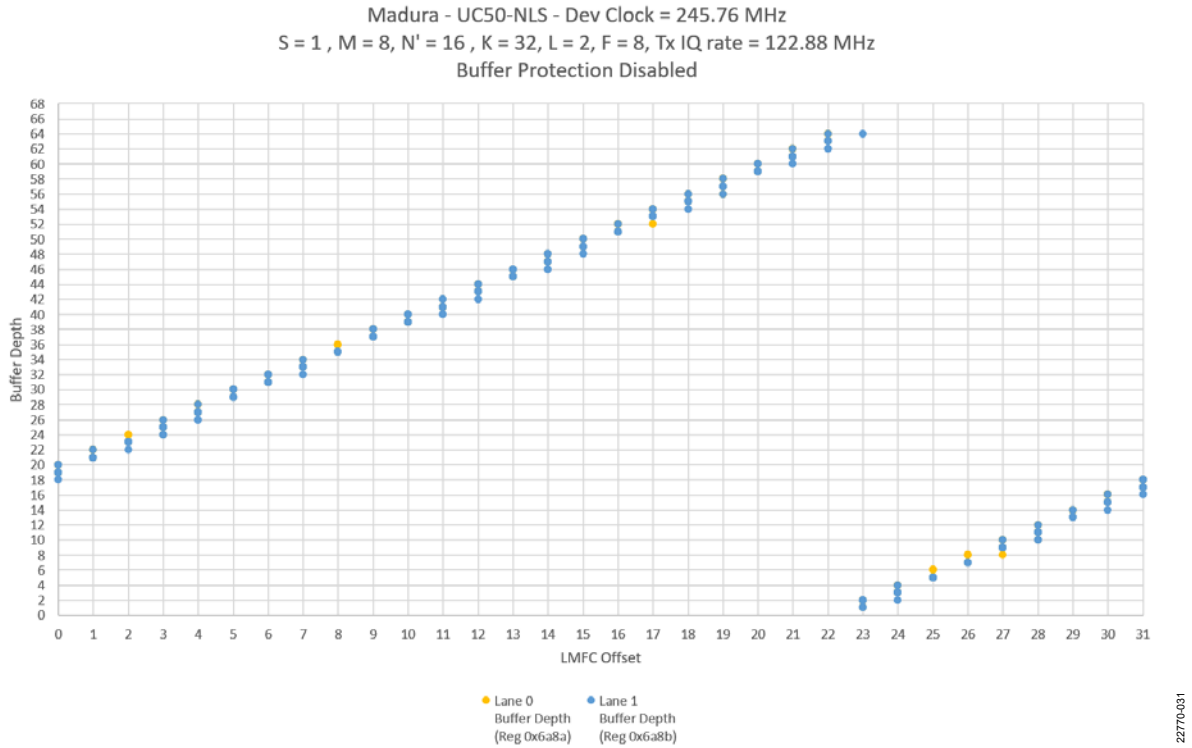


Figure 30. Buffer Depths Read Back for Lane 0 and Lane 1 vs. LMFC Offset on the ADRV9026 CE Board with ADRV9025Init_StdUseCase50_nonLinkSharing Profile and Buffer Protection Disabled

In this example, an LMFC offset value of 6 or 7 is a good choice because the result is buffer depths around 31 and 34 for all the used lanes, guaranteeing deterministic latency with no chance of data corruption due to the read and write pointers being too close.

If the goal for the system is to achieve deterministic latency with a latency as short as possible, it may be desirable to select an LMFC offset value giving buffer depths as small as possible but still well above a small number (for example, 10 or 12) to avoid data corruption due to the read and write pointers being too close. Note that in that situation, carry out thorough system testing over all possible temperature, supply and board variations to ensure that data corruption never occurs in all possible operating conditions for the system.

Avoid LMFC offset values giving a large buffer depth (that is, near a value of (K × F)/4) because, for some combinations of JESD parameters, it can lead to the write and read pointers being too close and therefore can result in data corruption.

Selecting the Optimal LEMC Offset for a System in JESD 204C Mode When E ≤ 2 with Buffer Protection Enabled

In JESD 204C mode when E ≤ 2, it is also recommended to select an LEMC offset that gives buffer depths values as close as possible to the center of the linear part of the buffer depth vs. LEMC Offset plot for all the lanes used. To find that LEMC offset, read back the buffer depth values for all the used lanes for all LEMC offset values with buffer protection enabled on a sample board for a given system. Measuring the buffer depths per LEMC offset for 10 power cycles or JESD link establishments (with application of a new SYSREF pulse each time) provides a good indication of the buffer depths spread for each LEMC offset. Select an LEMC offset value giving buffer depths as close as possible to the center of the linear part of the buffer depth vs. LEMC Offset plot for all the used lanes.

Figure 31 illustrates this process using the ADRV9026 CE board programmed with the ADRV9025Init_StdUseCase26C_nonLinkSharing profile, with automatic buffer protection enabled.

In this example, LEMC offset values between 36 and 40 are good choices because the result is a buffer depth around 24 for each lane, which is in the middle of the linear part of the plot and, therefore, guarantees deterministic latency.

If the goal for the system is to achieve deterministic latency with a latency as short as possible, it may be desirable to select an LEMC offset value giving buffer depths as small as possible. In that case, an LEMC offset value above the highest LEMC offset resulting in the automatic buffer protection being active with some additional headroom to account for PVT variations can be selected. In that situation, carry out thorough system testing over all possible temperature, supply and board variations to ensure that the automatic buffer protection never gets activated and that deterministic latency is maintained in all possible operating conditions for the system.

Avoid LEMC offset values giving large buffer depths (near a value of E × 32) because, for some combinations of JESD parameters, it can lead to the write and read pointers being too close and therefore can result in data corruption.

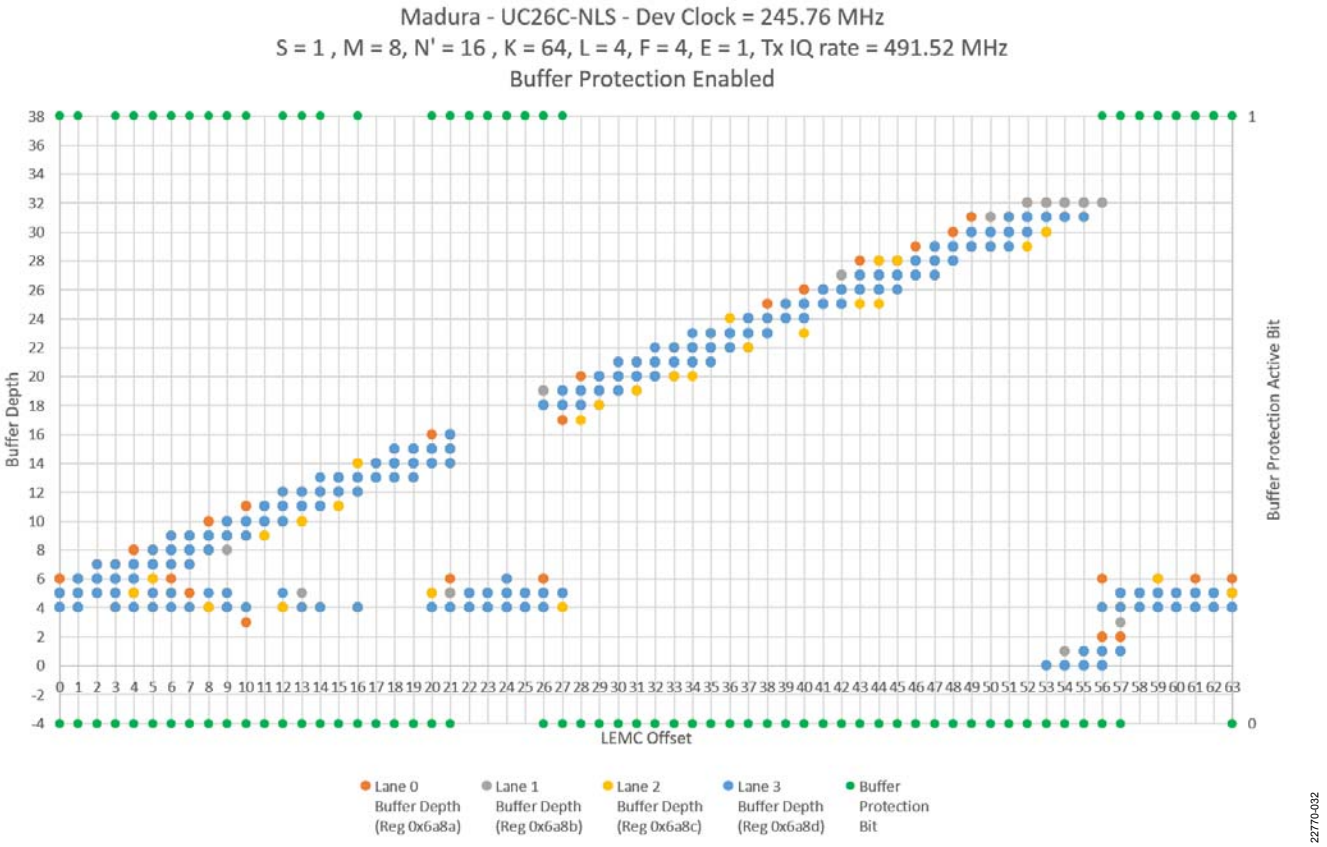


Figure 31. Buffer Depths for Lane 0, Lane 1, Lane 2, and Lane 3 vs. LEMC Offset on the ADRV9026 CE Board with the ADRV9025Init_StdUseCase26C_nonLinkSharing Profile and Buffer Protection Enabled

Selecting the Optimal LEMC Offset for a System in JESD 204C Mode When E ≤ 2 with Buffer Protection Disabled

When buffer protection is disabled, it is recommended to select an LEMC offset value that gives buffer depths as close as possible to (E × 32)/2 to account for variations and maintain deterministic latency on all boards for a given system. To find the LEMC offset corresponding to that optimal buffer depth, read back the buffer depth values for all LEMC offset values for all the used lanes with buffer protection disabled on a sample board for a given system. Measuring the buffer depths per LEMC offset for 10 power cycles or JESD link establishments (with application of a new SYSREF pulse each time) provided a good indication of the buffer depth spread for each LEMC offset values. Select an LEMC offset value giving buffer depths as close as possible to (E × 32)/2 for all lanes.

Figure 32 illustrates this process using the same ADRV9026 CE board with the ADRV9025Init_StdUseCase26C_nonLinkSharing profile and automatic buffer protection disabled.

In this example, an LEMC offset value between 21 and 24 is a good choice because it results in buffer depths around 16 for all the used lanes, guaranteeing deterministic latency with no chance of data corruption due to the read and write pointers being too close.

If the goal for the system is to achieve deterministic latency with a latency as short as possible, it may be desirable to select an LEMC offset value giving buffer depths as small as possible but still well above a small number (for example, 10 or 12) to avoid data corruption due to the read and write pointers being too close. Note that in that situation, carry out thorough system testing over all possible temperature, supply and board variations to ensure that data corruption never occurs in all possible operating conditions for the system.

Avoid LEMC offset values giving a large buffer depth (near a value of E × 32) because, for some combinations of JESD parameters, it can lead to the write and read pointers being too close and therefore can result in data corruption.

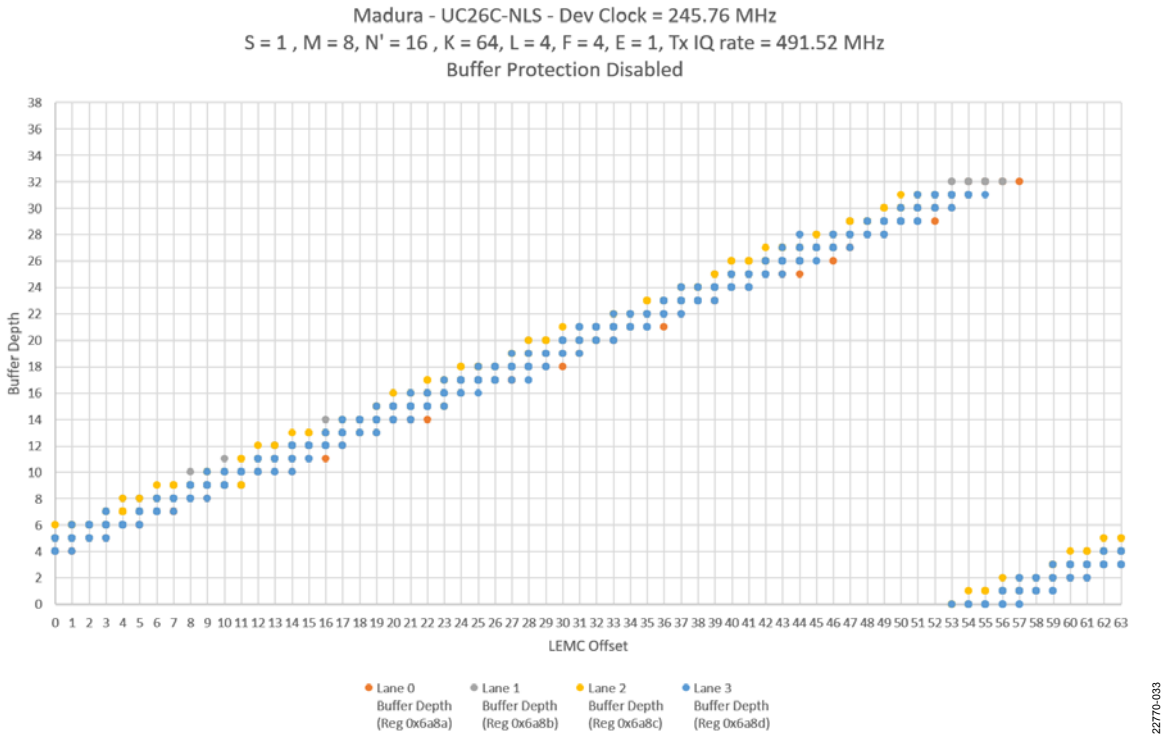


Figure 32. Buffer Depths for Lane 0, Lane 1, Lane 2, and Lane 3 vs. LEMC Offset on the ADRV9026 CE Board with the ADRV9025Init_StdUseCase26C_nonLinkSharing Profile and Buffer Protection Disabled

Selecting the Optimal LEMC Offset for a System in JESD204C Mode When E > 2

As mentioned previously, the size of each elastic buffer is 512 octets. When E is bigger than 2, there is some LEMC offset values for which more than 512 octets are needed to be stored in the elastic buffer to be able to release the data on the next LEMC edge. As this is not possible due to the elastic buffer size, buffer protection gets activated for such LEMC offset values when it is enabled. It is therefore recommended to have buffer protection enabled when E > 2.

In JESD204C mode when E >2, it is recommended to select an LEMC offset that gives buffer depths values as close as possible to the center of the linear part of the buffer depths vs. LEMC Offset plot for all the lanes used.

To find that LEMC offset, read back the buffer depths values for all the used lanes for all LEMC offset values with buffer protection enabled on a sample board for a given system. Measuring the buffer depths per LEMC offset for 10 power cycles or JESD link establishments (with application of a new SYSREF pulse each time) provides a good indication of the buffer depths spread for each LEMC offset.

Select an LEMC offset value giving buffer depths as close as possible to the center of the linear part of the buffer depth vs. LEMC Offset plot for all the used lanes.

Figure 33 illustrates this process using an ADRV9026 CE board programmed with the ADRV9025Init_StdUseCase14C_LinkSharing profile, with automatic buffer protection enabled.

In this example, LEMC offset values between 87 and 89 are good choices because it results in a buffer depth around 41 for each lane, which is in the middle of the linear part of the plot and therefore guarantees deterministic latency.

If the goal for the system is to achieve deterministic latency with a latency as short as possible, it may be desirable to select an LEMC offset value giving buffer depths as small as possible. In that case, an LEMC offset value above the highest LEMC offset resulting in the automatic buffer protection being active with some additional headroom to account for PVT variations can be selected. In that situation,

carry out thorough system testing over all possible temperature, supply and board variations to ensure that the automatic buffer protection never gets activated and that deterministic latency is maintained in all possible operating conditions for the system.

Avoid LEMC offset values giving large buffer depths (near a value of 64) because, for some combinations of JESD parameters, it can lead to the write and read pointers being too close and therefore can result in data corruption.

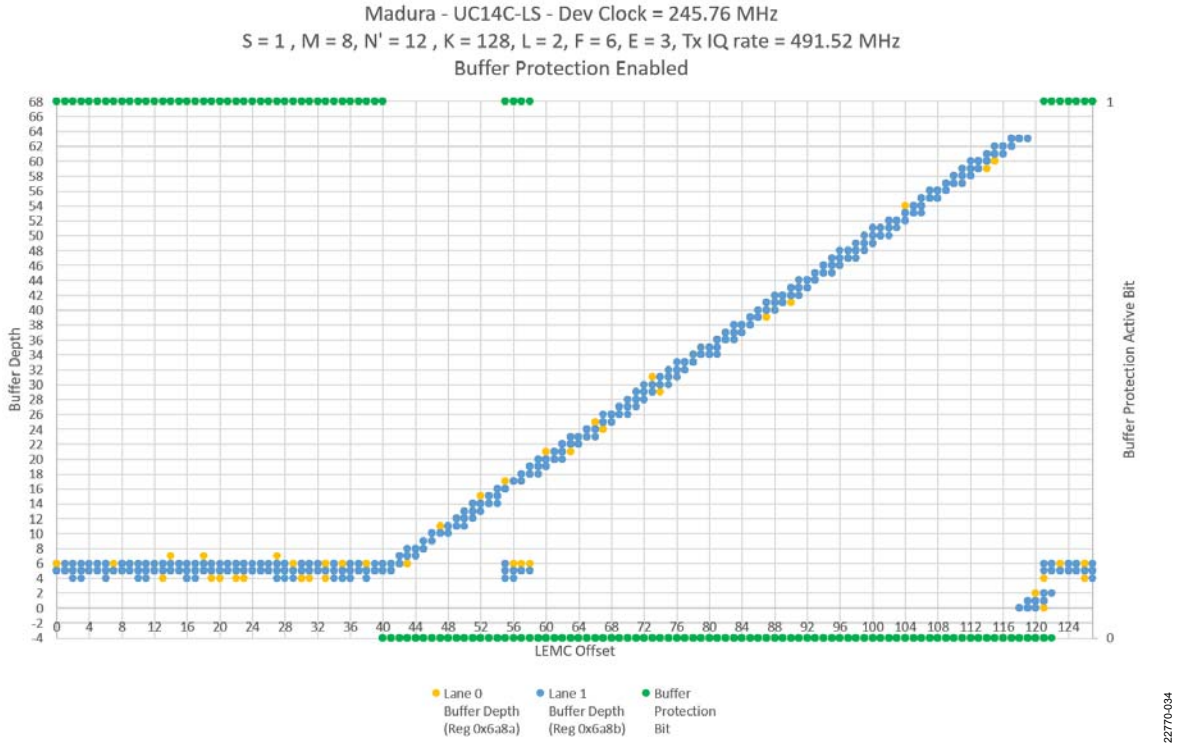


Figure 33. Buffer Depths for Lane 0 and Lane 1 vs. LEMC Offset on the ADRV9026 CE Board with the ADRV9025Init_StdUseCase14C_LinkSharing Profile and Buffer Protection Enabled

SYNTHESIZER CONFIGURATION

OVERVIEW

The ADRV9026 employs four phase-locked loop (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, and digital control block and either a 1 or 4 core voltage-controlled oscillator (VCO). The AuxPLL and CLKPLL VCO have a tuning range of 6.5 GHz to 13 GHz. The RFPLL1 and RFPLL2 VCO have a tuning range of 6.4 GHz to 12.8 GHz. Each PLL drives its own local oscillator (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. This configuration allows the use of any convenient reference frequency for operation on any channel with any sample rate. The reference frequency for the PLL is scaled from the reference clock applied to the device. Figure 35 below illustrates the common PLL block used in the ADRV9026.

CONNECTIONS FOR EXTERNAL REFERENCE CLOCK (DEVCLK)

The external clock is used as a reference clock for the clock synthesizer, two RF synthesizers, and auxiliary synthesizer in the device and thus needs to be a very clean clock source with respect to noise. Connect the external clock inputs to the DEVCLK+ (C8) and DEVCLK– (C9) pins via ac coupling capacitors and terminate them with 100 Ω close to the device as shown in Figure 34. The device clock receiver is a noise sensitive differential RF receiver. The frequency range of the DEVCLK signal must be between 10 MHz and 1 GHz. Ensure that the external clock peak to peak amplitude does is not less than 50 mV or greater than 1 V.

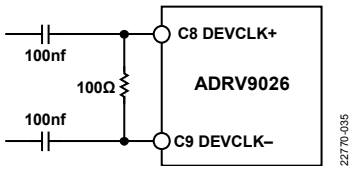


Figure 34. Reference Clock Input Connections

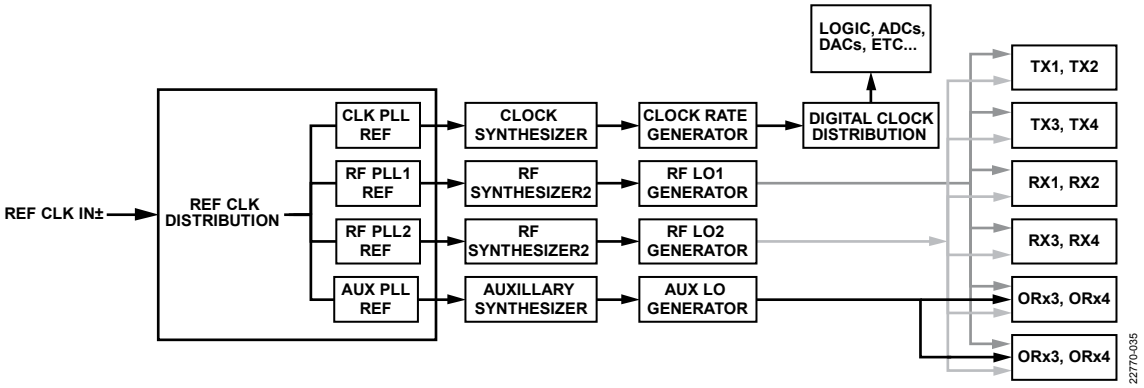


Figure 35. Synthesizer Interconnection and Clock/LO Distribution Block Diagram

EXTERNAL REFERENCE CLOCK (DEVCLK) REQUIREMENTS

Each RF synthesizer takes a lower frequency reference and multiplies it up to a higher frequency. The phase noise performance at the final frequency subsequently has some dependency on the phase noise of the input reference clock. This section discusses the impact of the reference (DEVCLK input) on the phase noise performance of the RF synthesizers. In general, the reference clock requirements are derived from the desired LO frequencies, PLL loop bandwidths, and somewhat on the phase margin.

The phase noise plots provided in the ADRV9026 data sheet are taken with a nearly ideal reference clock. An example is shown in Figure 36. Any noise on the reference is an additional noise source and can be RSS (root square sum) added to the phase noise specified in the data sheet.

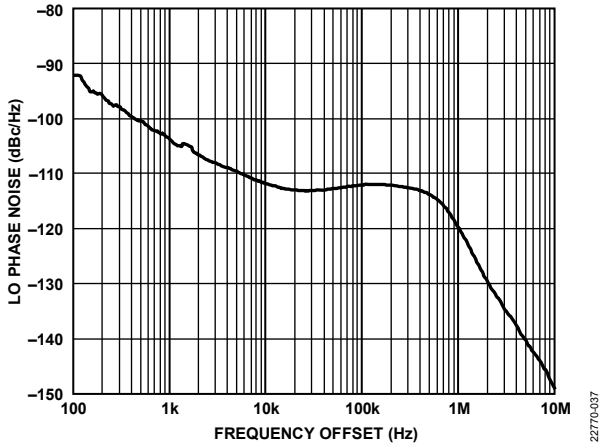


Figure 36. LO Phase Noise vs. Frequency Offset, F_{LO} =2600 MHz, Loop Bandwidth = 500 kHz, Phase Margin = 60°, DEVCLK Supplied by a Wenzel VCXO

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

$$f_{LO} = N \times f_{REF}$$

$$Loop\ Gain = 20 \times \log_{10}(N)$$

where N is the multiplier applied to the reference clock frequency (f_{REF}) to generate the desired LO frequency.

Noise power from the reference sees a multiplication factor equal to the loop gain. What is missing from this equation is the transfer function of the PLL that depends on the PLL loop bandwidth and somewhat on the loop phase margin. The loop bandwidth and phase margin are provided in the caption of the phase noise figures provided in the data sheet (as shown in Figure 36).

Figure 37 illustrates several closed loop responses with different loop bandwidths and phase margins listed. Each response is normalized to 0 dB using the loop gain calculation value for each to factor in the amount of gain that each response shifts. For example, for a f_{LO} of 2600 MHz and a f_{REF} of 245.76 MHz the gain is 20.5 dB. With the reference clock noise, the data sheet RF LO phase noise, and this transfer function the total noise can be calculated.

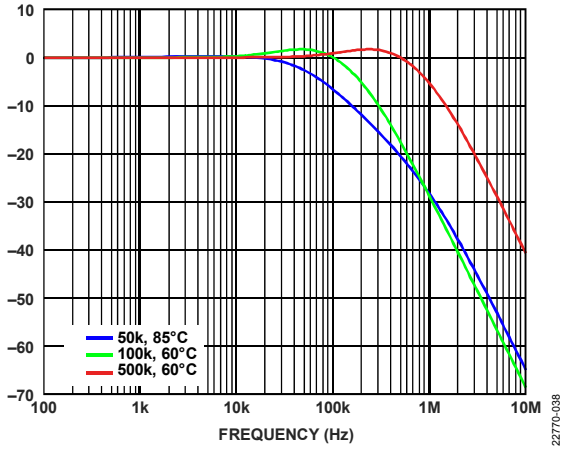


Figure 37. Normalized PLL Closed Loop Response

A 245.76 MHz reference with relatively high noise content is shown in Figure 38. This can be used to calculate the reference clock noise impact to the RF PLL using the following process:

- Multiply the phase noise of the reference clock by the PLL closed loop transfer function.
- RSS add this product to the corresponding RF PLL phase noise response for the given LO frequency provided in the data sheet.

The result of this process using the example data in Figure 36 through Figure 38 is illustrated by the plot shown in Figure 39. Note that the data sheet reference has much better phase noise at low frequency because it was measured during device characterization testing using an extremely low phase noise VCXO as the reference clock.

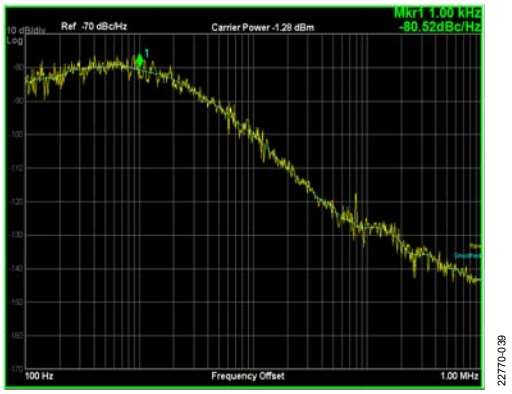


Figure 38. Phase Noise Plot for a Noisy 245.76 MHz Reference Clock

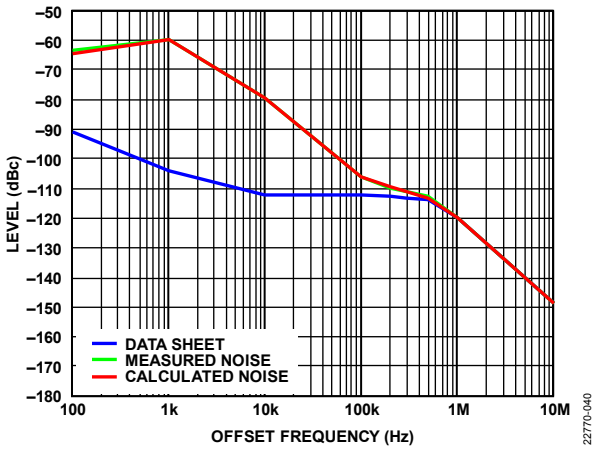


Figure 39. Example of Measured Phase Noise vs. Calculated Phase Noise—High Phase Noise Reference Clock

CLOCK SYNTHESIZER

The clock synthesizer is used to generate 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. Although the clock PLL is a fractional-N architecture, the signal sampling relationships to the JESD interface rates typically require that the synthesizer operates in integer mode. Profiles that are included in the ADRVTRX TES configure the clock synthesizer appropriately. Reconfiguration of the clock synthesizer is typically not necessary after initialization. The most direct approach to configuration is to follow the recommended programming sequence and utilize 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, Rx ADC sample and interface clocks, ORx ADC sample and interface clocks, and Tx DAC sample and interface clocks.

RF SYNTHESIZER

The device contains two RF PLLs. Each RF PLL uses the PLL block common to all synthesizers in the device and employ a 4 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 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 8386560 is programmed to provide an exact frequency on at least a 2 kHz raster using reference clocks that are integer multiples of 122.88 MHz. More details of the divider options are given in Table 56.

The RF LO frequency is derived by dividing down the VCO output in the LOGEN block. The tunable range of the RF LO is 400-6400 MHz. The LO divider boundaries are given in Table 57. Note that it is recommend to rerun the init cals when crossing a divide by 2 boundary or when changing the LO frequency by ±100 MHz or more from the frequency at which the init cals were performed.

Table 56. RF Synthesizer Divider Ranges

	LO Frequency Limits (MHz)									
	Lower Limit	**Upper Limit**	**Lower Limit**	**Upper Limit**	**Lower Limit**	**Upper Limit**	**Lower Limit**	**Upper Limit**	**Lower Limit**	**Upper Limit**
AuxPLL	203.125	406.25	406.25	812.5	812.5	1625	1625	3250	3250	6500
RFPLL1/2	200	400	400	800	800	1600	1600	3200	3200	6400
Div by	32		16		8		4		2	

Table 57. RF Synthesizer LO Boundaries

	DEV_CLK_IN (MHz)	**PLL PFD (MHz)**	**Desired LO Frequency Ranges (MHz)**									
			Lower Limit	**Upper Limit**	**Lower Limit**	**Upper Limit**	**Lower Limit**	**Upper Limit**	**Lower Limit**	**Upper Limit**	**Lower Limit**	**Upper Limit**
			200	400	400	800	800	1600	1600	3200	3200	6400
LO Step Size (Hz)	491.52, 245.76	245.76	0.92		1.83		3.66		7.33		14.65	
	307.2	307.2	1.14		2.29		4.58		9.16		18.32	
	122.88	122.88	0.46		0.92		1.83		3.66		7.33	
	153.6	153.6	0.57		1.14		2.29		4.58		9.16	
Exact Decimal Frequency Raster (Hz)	491.52, 245.76	245.76	250		500		1000		2000		4000	
	307.2	307.2	312.5		625		1250		2500		5000	
	122.88	122.88	125		250		500		1000		2000	
	153.6	153.6	156.25		312.5		625		1250		2500	

A switching network is implemented in the device to provide flexibility in LO assignment for the two RF LO sources. The switching network is diagrammed in Figure 40. Note that it is not recommended to set RFLO1 = RFLO2; this cab cause unwanted coupling between the two PLLs. To set RFLO1 = RFLO2, then set either RFPLL1 or RFPLL2 to the desired frequency and mux that PLL to both TxLO and RxLO. That is, set either TXLO = RXLO = RFLO1 or TXLO = RXLO = RFLO2 and power down the unused RFLO.

AUXILIARY SYNTHESIZER

An auxiliary synthesizer is integrated to generate the signals necessary to calibrate the device. This synthesizer utilizes a single core VCO. The reference frequency for the AUX synthesizer is scaled from the device clock via 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 Rx signal chain as an offset LO. Calibrations are executed during the initialization sequence at startup. There is no signal present at the Rx/ORx input during tone calibration time. Calibrations are fully autonomous. During the calibration, the auxiliary synthesizer is controlled solely by the internal Arm processor and does not require any user interactions. The AUX LO signal is also available as an LO source for the observation receiver mixers.

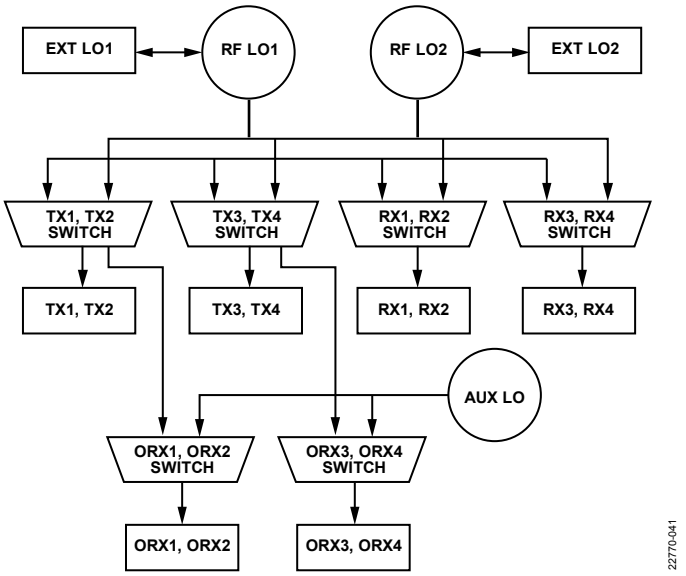


Figure 40. LO Switching Network

SETTING THE LO FREQUENCIES

There are two commands that the user can execute to select the LO frequency in the device. One is used when the user does not have special phase requirements between the Tx LO and the AUX LO; the other is used when the user has special phase requirements. When no phase requirements exist, the user can run the following API command:

```
int32_t adi_adrv9025_PllFrequencySet(adi_adrv9025_Device_t* device, adi_adrv9025_PllName_e
  pllName, uint64_t pllLoFrequency_Hz)
```

If the user has special phase requirements, relies on their own LOL/QEC tracking calibrations, or requires a faster lock time, the user can use the following function which provides more control over these settings.

```
int32_t adi_adrv9025_PllFrequencySet_v2(adi_adrv9025_Device_t* device, adi_adrv9025_PllConfig_t
  *pllConfig)
```

An example of this situation involves placing the AUX LO at a user defined offset from the TX LO that is normally defaulted to +(bandwidth/2 + 5) MHz. If the user has no specific requirements on the phase or frequency of the auxiliary LO, use the adi_adrv9025_PllFrequencySet(…) command. More details about these commands are in the API Functions section of this chapter.

Both commands can be run any time after device initialization, and neither has any prerequisite commands or requirements. The structures and enumerators for these API commands are detailed in Table 58 through Table 62.

Table 58. adi_adrv9025_PllConfig_t Structure

Data Type	Parameter	Range	Description
PllName_e	pllName	Table 59	Name of the PLL the user wants to control.
PllModeSel_e	pllModeSel	Table 60	The user can select between Slow locking or fast locking mode.
PllAuxLoResyncSel_e	pllAuxLoResyncSel	Table 61	The user can select between resyncing and not resyncing the AUX LO to the TX LO after a frequency change.
PllAuxLoOffsetProgSel	pllAuxLoOffsetProgSel	Table 62	The user can select whether the auxiliary LO frequency is changed to be +(bandwidth/2 + 5) MHz or to not be changed after a frequency change.
Uint64_t	pllLoFrequency_Hz	400 × 106 to 6000 × 106	The LO frequency which the customer wants to set in Hz.

Table 59. adi_adrv9025_PllName_e Enumerator

Enum	Enum Values	Description
PllName_e	ADI_ADRV9025_LO1_PLL	Selects LO1 PLL for Tx/Rx/ORx
	ADI_ADRV9025_LO2_PLL	Selects LO2 PLL for Tx/Rx/ORx.
	ADI_ADRV9025_AUX_PLL	Selects AUX PLL for ORx.

Table 60. adi_adrv9025_Pll_ModeSel_e Enumerator

Enum	Enum Values	Description
pllModeSel_e	ADI_ADRV9025_PLL_SLOW_MODE	Slow lock mode. This mode skips some calibrations in order to lock the PLL faster.

Table 61. adi_adrv9025_pllAuxLoResyncSel_e Enumerator

Enum	Enum Values	Description
pllAuxLoResyncSel_e	ADI_ADRV9025_PLL_AUX_LO_RESYNC_ENABLE	Resyncs the AUX LO to the Tx LO after a frequency change.
	ADI_ADRV9025_PLL_AUX_LO_RESYNC_DISABLE	Does not resync the AUX LO to the Tx LO after a frequency change.

Table 62. adi_adrv9025_pllAuxLoOffsetProgSel_e Enumerator

Enum	Enum Values	Description
pllAuxLoOffsetProgSel_e	ADI_ADRV9025_PLL_AUX_LO_OFFSET_PROG_ENABLE	Programs the AUX LO to be +(bandwidth/2 + 5) MHz from the Tx LO after every frequency change.
	ADI_ADRV9025_PLL_AUX_LO_OFFSET_PROG_DISABLE	Does not set the AUX LO after a frequency change.

API Functions

adi_adrv9025_PllFrequencySet(…)

```
int32_t adi_adrv9025_PllFrequencySet(adi_adrv9025_Device_t* device, adi_adrv9025_PllName_e
  pllName, uint64_t pllLoFrequency_Hz)
```

Description

This function sets the LO frequency of the chosen PLL.

Precondition

After device initialization.

Parameters

Table 63.

Parameter	Description
*device	Pointer to device structure.
pllName	The PLL selected for setting the frequency.
pllLoFrequency_Hz	Frequency of the LO the user wants to set in Hz.

adi_adrv9025_PllFrequencyGet(…)

```
int32_t adi_adrv9025_PllFrequencyGet(adi_adrv9025_Device_t* device, adi_adrv9025_PllName_e
  pllName, uint64_t *pllLoFrequency_Hz)
```

Description

This function gets the LO frequency of the chosen PLL.

Precondition

After device initialization.

Parameters

Table 64.

Parameter	Description
*device	Pointer to device structure.
pllName	The PLL selected for getting the frequency.
*pllLoFrequency_Hz	Pointer to the frequency of the LO the user wants to set in Hz.

adi_adrv9025_PllFrequencySet_v2(…)

```
int32_t adi_adrv9025_PllFrequencySet_v2(adi_adrv9025_Device_t* device, adi_adrv9025_PllConfig_t
  *pllConfig);
```

Description

Use this function when the user has special phase constraints that they need to put on certain PLLs to meet system requirements. adi_adrv9025_PllFrequencySet_v2(…) is equivalent to adi_adrv9025_PllFrequencySet(…) with the following parameters set in the adi_adrv9025_PllConfig_t structure:

Table 65.

Parameter	Description
pllModeSel	ADI_ADRV9025_PLL_SLOW_MODE
pllAuxLoResyncSel	ADI_ADRV9025_PLL_AUX_LO_RESYNC_ENABLE
pllAuxLoOffsetProgSel	ADI_ADRV9025_PLL_AUX_LO_OFFSET_PROG_ENABLE

Precondition

After device initialization.

Parameters

Parameter	Description
*device	Pointer to device structure.
*pllConfig	Pointer to PLL configuration structure.

adi_adrv9025_PllLoopFilterSet(…)

```
int32_t adi_adrv9025_PllLoopFilterSet(adi_adrv9025_Device_t* device, adi_adrv9025_PllName_e
  pllName, adi_adrv9025_PllLoopFilterCfg_t *pllLoopFilterConfig);
```

Description

This function allows the user to set the PLL loop filter bandwidth, phase margin, and power scale of the device.

Precondition

After device initialization.

Parameters

Table 66.

Parameter	Description
*device	Pointer to device structure.
pllName	PLL selected for changing settings.
*pllLoopFilterConfig	Pointer to loop filter configuration structure passed to the device.

adi_adrv9025_PllLoopFilterGet(…)

```
int32_t adi_adrv9025_PllLoopFilterGet(adi_adrv9025_Device_t* device, adi_adrv9025_PllName_e
  pllName, adi_adrv9025_PllLoopFilterCfg_t *pllLoopFilterConfig);
```

Description

This function allows the user to get the PLL loop filter bandwidth, phase margin, and power scale of the device.

Precondition

After device initialization.

Parameters

Table 67.

Parameter	Description
*device	Pointer to device structure.
pllName	PLL selected for getting settings.
*pllLoopFilterConfig	Pointer to loop filter configuration structure passed to the device, returns the current configuration.

EXTERNAL LO

The device is provisioned with 2 external LO ports. These ports share a pair of balls and can be configured to be input or output for LO signals. In an input configuration, they provide the option to drive the LO with another device in order to synchronize multiple devices in the same system or to provide an LO with better phase noise performance in order to meet stringent requirements as in the case of multicarrier GSM receivers. In an output configuration, the LO of one device can drive the LO input of another device. Refer to Figure 40 for illustration of EXT LO connection in reference to the RF LO sources.

EXT LO IN

External LO in can receive a signal between 200 MHz and 13 GHz through a matched differential impedance of 100 Ω and delivers a programmable signal between 25 MHz and 6.5 GHz as the LO for transmitters and receivers in the device. Amplitude must be maintained between ±6 dBm.

EXT LO OUT

In the output mode, the port delivers the internal LO divided by 1 to 128 providing a frequency range of 25 MHz to 6.5 GHz.

LOCK STATUS

Lock status of the clock, RF and auxiliary PLLs is provided through the following API command:

```
adi_adrv9025_PllStatusGet(adi_adrv9025_Device_t* device, uint32_t *pllLockStatus);
```

Table 68. pllLockStatus Return Values

Variable	Bit Position	Return Values
*pllLockStatus	D0	1 = CLK PLL Locked; 0 = Unlocked
	D1	1 = LO1 PLL Locked; 0 = Unlocked
	D2	1 = LO2 PLL Locked; 0 = Unlocked
	D3	1 = AUX (LO3) PLL Locked; 0 = Unlocked

Additionally, PLL lock status can be set to assert via a general purpose interrupt pin. For monitoring the lock status over the GPINT signal, see the General-Purpose Interrupt section.

API Functions

Adi_adrv9025_PllStatusGet(…)

```
int32_t adi_adrv9025_PllStatusGet(adi_adrv9025_Device_t* device, uint32_t *pllLockStatus)
```

Description

This function gets the PLL lock status of Clock, LO1, LO2 and AUX PLLs and returns them in the parameter *pllLockStatus.

Precondition

After device initialization.

Parameters

Table 69.

Parameter	Description
*device	Pointer to device structure.
*pllLockStatus	Pointer to structure element that contains the PLL Lock Status.

RF PLL PHASE SYNCHRONIZATION

The RFPLL Phase Sync description is included at this time for prototyping and evaluation purposes only. Consult Analog Devices for function availability.

This function has been added to allow the internally generated LO to be phase synchronized and aligned to the applied reference clock. In multiple transceiver systems, this function allows all devices to align the RFPLL to the same point, and therefore the phase between each device is aligned at startup so that phasing between devices is repeatable and fixed. At startup, the standard JESD204B multi-chip synchronization mechanism (MCS) implemented with the device clock (DEVCLK) and system reference signal (SYSREF) are used to reset the data converter clocks and all other clocks at the baseband rate. These same signals are also used to initialize an on-chip counter which is later used during PLL programming to synchronize the LO phase. No additional signals are required to take advantage of the LO phase synchronization mechanism. From the on-chip counter and a PLL fractional word programming, a digital representation of the desired LO phase can be computed at each PLL reference clock edge and is remembered in the digital phase accumulator (DPA).

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

After the phase difference has been measured, the adjustment can be applied into the first stage Σ-Δ modulator (SDM) of the PLL by adding it to the first stage modulator input. The total adjustment amount is added over many reference clock cycles in order to stay within the PLL loop bandwidth and not cause the PLL to come unlocked. To counteract temperature effects after calibration, a PLL phase tracking mode can be activated. Figure 41 is a block diagram of the phase synchronization system.

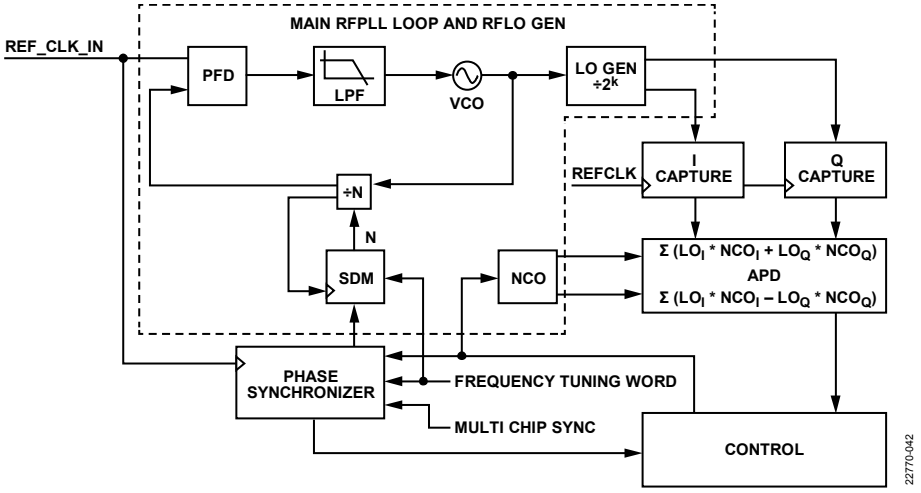


Figure 41. LO Phase Sync Functional Diagram

System Level Considerations

Overall phase synchronization is determined by a number of factors, including the board level clock routing (t_{CLK}), the on-chip reference path routing ($t_{REFPATH}$), the PLL and LO divider path (t_{PLL}), and the RF and antenna paths (t_{RF}). These time delays are illustrated in Figure 42. In a beam forming/MIMO system, there is a system level antenna calibration that is performed to equalize the sum of these paths between all channels. The goals of this transceiver mechanism are:

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

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

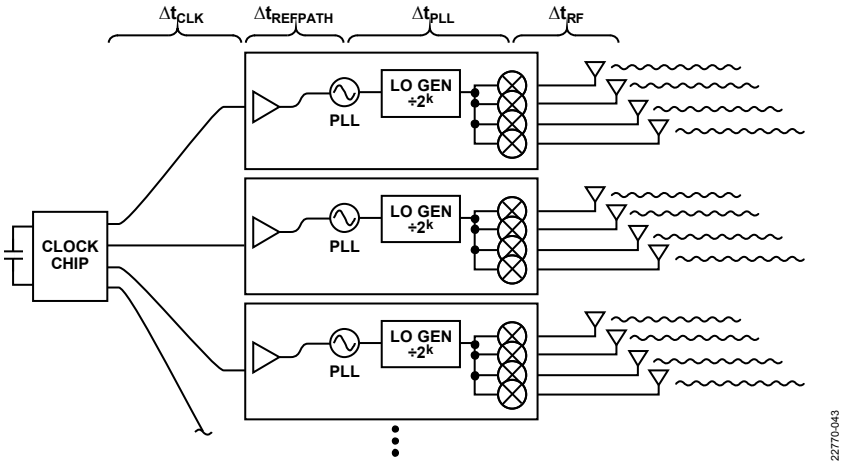


Figure 42. High Level Contributions to System Phase Per Antenna

ARM PROCESSOR AND DEVICE CALIBRATIONS

The ADRV9026 is equipped with a built in Arm M4 processor. The firmware for this Arm processor is loaded during the initialization process. The firmware memory size is 224 kB, and the Arm has access to a further 160 kB of data memory to utilize. The Arm is tasked with configuring the device for the selected use case, performing initial calibrations of the signal paths and maintaining device performance over time through tracking calibrations.

ARM STATE MACHINE OVERVIEW

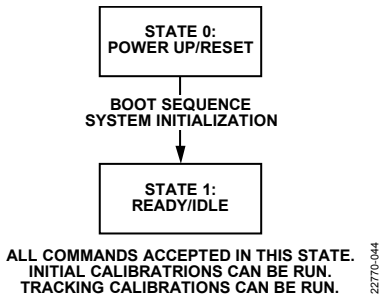


Figure 43. Arm State Machine

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

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

SYSTEM INITIALIZATION

The System Initialization section of this document provides a detailed description of the initialization procedure. There are three main sections to the initialization procedure.

Pre MCS Init initializes the device up to the multichip sync procedure. The Pre-MCS init sequence is split into two commands that the application layer function calls. These are adi_adrv9025_PreMcsInit_v2(…) and adi_adrv9025_PreMcsInit_NonBroadCast(…). adi_adrv9025_PreMcsInit_v2(…) is a broadcastable command that can simultaneously issue commands to multiple transceivers to save time during system initialization for systems with multiple transceivers. Arm and Stream binaries are programmed to the chip during this step. The broadcast functionality is realized by issuing SPI write commands only. The adi_adrv9025_PreMcsInit_NonBroadCast(…) verifies that the Arm is programmed properly by verifying the Arm checksum and that the Arm is in the Ready/Idle state.

The multichip sync (MCS) step uses SYSREF pulses to synchronize internal clocks within the transceiver. Required for deterministic latency.

Post-MCS Init continues initialization following MCS. The application layer command that performs the post-MCS initialization is adi_adrv9025_PostMcsInit(…). This command programs the PLLs, configures the radio control initialization structure and instructs the Arm to perform initialization calibrations.

PRE MCS INITIALIZATION

This section explains the Arm related function calls in adi_adrv9025_PreMcsInit_v2( ). Run adi_adrv9025_PreMcsInit_v2(…) as part of the initialization sequence.

```
adi_adrv9025_PreMcsInit_v2(adi_adrv9025_Device_t *device,
                           adi_adrv9025_Init_t *init,
                           const char *armImagePath,
                           const char *streamImagePath,
                           adi_adrv9025_RxGainTableFile_t rxGainTableFileArr[],
                           uint8_t rxGainTableFileArrSize,
                           adi_adrv9025_TxAttenTableFile_t txAttenTableFileArr[],
                           uint8_t txAttenTableFileArrSize);
```

Of importance from the perspective of the Arm is the armImagePath, a file system location where the Arm binary is stored, which is required for the Arm to be loaded.

The adi_adrv9025_PreMcsInit_v2(…) function is in the adi_adrv9025_utilities.c/h file. It performs a sizeable part of the full chip initialization. From the point of view of the Arm, it performs a number of tasks. The first step is to load the Arm image: adi_adrv9025_ArmImageLoad(device, armImagePath), where device is the transceiver device structure.

armImagePath is the path to the Arm image binary passed as a parameter to adi_adrv9025_PreMcsInit_v2( )

The Arm image is provided in the **Resources/ArmFiles** folder of the GUI installation folder.

Following the Arm firmware image being loaded, the next step is to load the device configuration into data memory using adi_adrv9025_ArmProfileWrite(adi_adrv9025_Device_t *device, const adi_adrv9025_Init_t *init).

*init is the initialization settings data structure.

The Arm is then started and begins its boot sequence. This process is initiated by:

adi_adrv9025_ArmStart(adi_adrv9025_Device_t *device, const adi_adrv9025_Init_t *init)

As part of the boot sequence, the Arm configures the device for the required profile (Tx/Rx/ORx path configuration as determined by the use case), configures and enables the clock PLL (the device starts initialization on the device clock), and configures the JESD framers and deframers. The Arm also computes a checksum for the Arm firmware image loaded, for each of the streams loaded and the profiles loaded (determining if they are valid profiles). The following API function waits for the Arm boot to complete, compares the computed checksums during booth to precomputed checksums, for example comparing the Arm firmware checksum vs. the Arm checksum which is calculated upon compilation of the Arm firmware and stored within the Arm firmware image: adi_adrv9025_ArmStartStatusCheck(adi_adrv9025_Device_t *device, uint32_t timeout_us), where timeout_us is a timing parameter that dictates the longest time that the function waits for arm booting to complete

If a checksum is found not to be valid, this function returns an error.

POST MCS INITIALIZATION

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

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

DEVICE CALIBRATIONS

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

A number of Tx calibrations use an observation path to observe the signal at the output of the Tx. For the most part, they use an internal loopback path from Tx to ORx. The exception is the external LOL initialization and tracking algorithms that require the use of an external path connection between the Rx output and an ORx input (typically the DPD feedback path).

A requirement for this device is that the ORx channel used to calibrate a Tx channel must be on the same side of the chip as that Tx channel. Table 70 provides the possible feedback combinations. For example, it is not possible for LO Leakage tracking to calibrate Tx4 by providing its output to ORx1 or ORx2.

Table 70. External Feedback Path Possibilities

Channel	Available Feedback Channels
Tx1	ORx1 or ORx2
Tx2	ORx1 or ORx2
Tx3	ORx3 or ORx4
Tx4	ORx3 or ORx4

Figure 44 shows an example of four feedback paths, each Tx going back to an ORx, obeying the principle of each Tx being fed back to an ORx on the same side of the device. It is also possible to have both Tx1 and Tx2 going back to a single ORx input (either ORx1 or ORx2) through a switch, likewise Tx3 and Tx4 can go back to a single ORx input (either ORx3 or ORx4).

Note: for the diagrams outlining the operation of individual calibrations, the Tx and ORx inputs are not numbered. Instead it is assumed that the principle of a Tx being fed back to an ORx on the same side of the device is being obeyed.

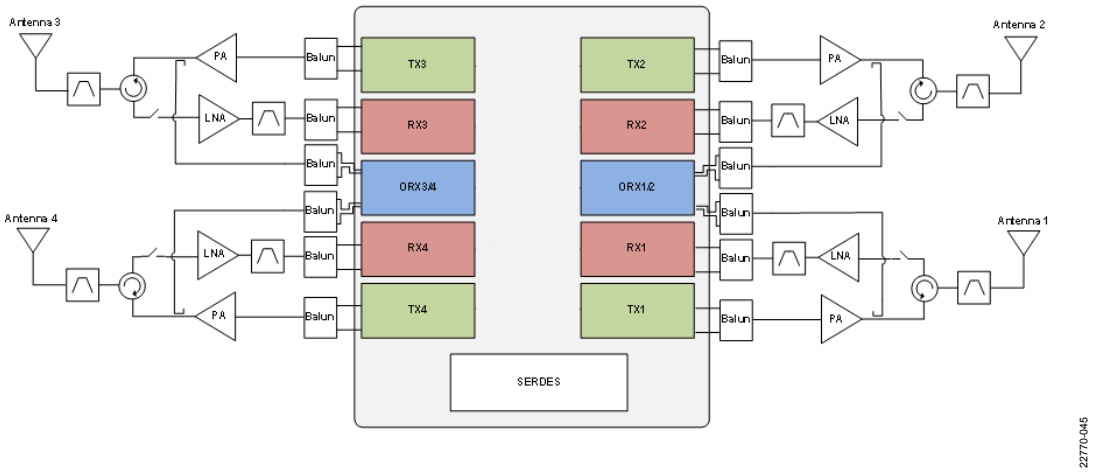


Figure 44. External Feedback for Tx Tracking Calibrations

INITIAL CALIBRATIONS

The Arm processor in the device is tasked with scheduling/performing initial calibrations to optimize the performance of the signal paths prior to device operation. These calibrations are run as part of the utility API function adi_adrv9025_PostMcsInit( ). To correctly perform the initial calibrations, this utility function needs to be called. This section also provides details of the procedure invoked in adi_adrv9025_PostMcsInit( ) to perform the initial calibrations, principally for further information, but also in case there is a need to run initial calibrations outside of the post MCS initialization procedure. The function definition for the post MCS initialization is:

```
adi_adrv9025_PostMcsInit(adi_adrv9025_Device_t *device, adi_adrv9025_PostMcsInit_t *utilityInit)
```

*utilityInit is a structure containing a structure determining the initial calibrations to be run as part of the post MCS initialization routine

In some cases, it is required to run an initial calibration outside of adi_adrv9025_PostMcsInit(…). This following command instructs the Arm to perform the requested calibrations:

```
adi_adrv9025_InitCalsRun(adi_adrv9025_Device_t *device, adi_adrv9025_InitCals_t *initCals)
```

*initCals is the initial calibration structure, passed to adi_adrv9025_PostMcsInit as part of utilityInit, that informs the Arm processor which calibrations to run on which enabled path. initCals is composed of a uint32_t calMask and a uint8_t channelMask. calMask indicates which calibrations are to run in this call of adrv9025_InitCalsRun( ).

Table 71 shows the bit assignments of the calibration mask. Note that Table 71 provides a full list of initialization calibrations for the device. Some initial calibrations are not available for certain variants of the ADRV9026 device.

The channelMask parameter, a member of the adi_adrv9025_InitCals_t structure, advises which channels the selected calibrations run. Each bit of the bitmask refers to an individual channel as shown in Table 72. The mask is universally applied to all calibrations selected in the current call of adi_adrv9025_initCalsRun( ), regardless of the paths that the calibrations are being run for. For example, if 0xF is

chosen as a mask and both Rx and Tx calibrations are selected in the calMask, then when the Arm runs an Rx calibration it does so on all four Rx channels. Likewise, when it runs a Tx calibration it does so on all four Tx channels.

Table 71. calMask Bit Assignments

Bits	Corresponding Enum	Calibration	Description
D0	ADI_ADRV9025_TX_BB_FILTER	Tx baseband filter calibration	This is used to tune the corner frequency of the Tx Baseband filter.
D1	ADI_ADRV9025_ADC_TUNER	ADC tuner calibration	This is used to configure the ADC for the required profile bandwidth.
D2	ADI_ADRV9025_RX_TIA	Rx TIA filter calibration	This is used to tune the corner frequency of the Rx TIA Filter.
D3	ADI_ADRV9025_ORX_TIA	ORx TIA filter calibration	This is used to tune the corner frequency of the ORx TIA Filter.
D4	ADI_ADRV9025_LBRX_TIA	Loopback Rx TIA Filter calibration	This is used to tune the corner frequency of the Loopback Rx TIA Filter.
D5	ADI_ADRV9025_RX_DC_OFFSET	Rx dc offset calibration	This is used to correct for dc Offset within the Rx chain.
D6	ADI_ADRV9025_ORX_DC_OFFSET	ORx dc offset calibration	This is used to correct for dc Offset within the ORx chain.
D7	ADI_ADRV9025_LBRX_DC_OFFSET	Loopback Rx dc offset calibration	This is used to correct for dc Offset within the loopback Rx chain.
D8	ADI_ADRV9025_FLASH_CAL	ADC flash calibration	This is used to optimally configure the ADC Flash converters.
D9	ADI_ADRV9025_INTERNAL_PATH_DELAY	Internal path delay calibration	This computes the Tx to internal loopback path delay, which is required for the TxQEC initial calibration and tracking.
D10	ADI_ADRV9025_TX_LO_LEAKAGE_ INTERNAL	Tx LO leakage initial calibration	This performs an initial LO leakage calibration for the Tx path. It utilizes the Tx path and the internal loopback path (see Figure 47).
D11	ADI_ADRV9025_TX_LO_LEAKAGE_EXTERNAL	Tx LO leakage external initial calibration	This performs an initial external LO leakage calibration for the Tx path. It utilizes the Tx path, a required external loopback path and the ORx path (see Figure 48). The external loop must be enabled such that the Tx output is observable by the ORx.
D12	ADI_ADRV9025_TX_QEC_INIT	Tx QEC initial calibration	This performs an initial QEC calibration for the Tx path. It utilizes the Tx path and an internal loopback path (see Figure 47).
D13	ADI_ADRV9025_LOOPBACK_RX_LO_DELAY	Loopback Rx LO delay calibration	This is used to perform an LO delay calibration for the loopback path.
D14	ADI_ADRV9025_LOOPBACK_RX_RX_QEC_INIT	Loopback RxQEC initial calibration	This performs an initial QEC calibration for the Rx path.
D15	ADI_ADRV9025_RX_LO_DELAY	Rx LO delay calibration	This is used to perform an LO delay calibration for the receiver path.
D16	ADI_ADRV9025_RX_QEC_INIT	Rx QEC initial calibration	This performs an initial QEC calibration for the Rx path.
D17	ADI_ADRV9025_ORX_LO_DELAY	ORx LO delay calibration	This is used to perform an LO delay calibration for the observation receiver path.
D18	ADI_ADRV9025_ORX_QEC_INIT	ORx QEC initial calibration	This performs an initial QEC calibration for the observation receiver path.
D19	ADI_ADRV9025_TX_DAC	Tx DAC initial calibration	This performs a calibration of the Tx DAC.
D20	Reserved		
D21	ADI_ADRV9025_EXTERNAL_PATH_DELAY	External Tx to ORx path delay initial calibration	This acquires an estimation of the Tx to ORx path delay (not required if CLGC tracking is not used)
D22	Reserved		
D23	ADI_ADRV9025_HD2	HD2 initial calibration	This performs an initial calibration of the HD2 product in the Rx path (typically required only in GSM applications).

Bits	Corresponding Enum	Calibration	Description
D24	ADI_ADRV9025_TX_ATTENUATION_DELAY	Tx attenuation delay calibration	This is used to calculate the path delay between the Tx analog and digital attenuation blocks. This delay is then used to delay the onset of Tx analog attenuation when the Tx attenuation changes. This synchronizes the attenuation change at the Tx output.
D25	ADI_ADRV9025_TX_ATTEN_TABLE	Tx attenuation table linearization calibration	This is used to correct for phase changes between different attenuation indices in the Tx attenuation table.
D26	ADI_ADRV9025_RX_GAIN_DELAY	Rx gain delay calibration	This is used to calculate the path delay between the Rx analog and digital attenuation blocks. This delay is then used to delay the onset of Rx analog attenuation when the Rx gain index is changed. This synchronizes the gain change in the baseband data.
D27	ADI_ADRV9025_RX_GAIN_PHASE	Rx gain phase calibration	This is used to correct for phase changes between different gain indices in the Rx gain table.
D28	Reserved		
D29	ADI_ADRV9025_CFR_INIT	Crest factor reduction initialization calibration	This performs an initialization calibration for the ADRV9026 CFR hardware.
D30	ADI_ADRV9025_SERDES_INIT	SERDES initialization cal	This performs an initialization calibration for the ADRV9026 JESD 204C data interface.
D31	Reserved		

Table 72. channelMask Bit Assignments

Bits	Channel
D0	Channel 1 (either Rx1/Tx1/ORx1 depending on calibration being performed)
D1	Channel 2 (either Rx2/Tx2/ORx2 depending on calibration being performed)
D2	Channel 3 (either Rx3/Tx3/ORx3 depending on calibration being performed)
D3	Channel 4 (either Rx4/Tx4/ORx4 depending on calibration being performed)

The Arm sequences the initial calibrations as required, not necessarily in the bit order presented above. It is mandatory that the user wait for calibrations to complete before continuing with the initialization of the device. The following API command is used to verify that the initial calibrations are complete:

```
adi_adrv9025_InitCalsWait(adi_adrv9025_Device_t *device, uint32_t timeoutMs, uint8_t *errorFlag)
```

timeoutMs is the time in milliseconds (ms) that the function must wait for the calibrations to complete before returning an error

errorFlag indicates if there was an Arm error when the running the initialization calibrations

This function implements a blocking wait until the initial calibrations have been completed. An alternative function can be used instead which determines if the initial calibrations are still running using the following API:

```
adi_adrv9025_InitCalsCheckCompleteGet(adi_adrv 9025_Device_t *device, uint8_t *areCalsRunning,
   uint8_t *errorFlag);
```

*areCalsRunning is a value to indicate if calibrations are still running (0 = initial calibrations have completed, 1 = initial calibrations are still running)

errorFlag indicates if there was an Arm error when the running the initialization calibrations.

In the case when an initial calibration error occurs, information about the error can be obtained with the following command:

```
adi_adrv9025_InitCalsDetailedStatusGet(adi_adr v9025_Device_t *device,
   adi_adrv9025_InitCalStatus_t *initStatus);
```

*initStatus is a pointer to a data structure that contains initial calibration status information. The adi_adrv9025_InitCalStatus_t data structure details are described in Table 73.

Table 73. Definition of adi_adrv9025_InitCalStatus_t

Parameter	Interpretation
initErrCode	Returns the Object ID and error code reported for the initialization calibration failure. The object ID is contained within Bits[D15:D8] and error bits are contained within Bits[D7:D0].
initErrCal	Returns the object ID of the calibration reporting an error.
calsDurationUsec	Time duration in microseconds of the most recent InitCalsRun invocation.
calsSincePowerUp[4]	A 4-element array indicating the bitmask of initial calibrations run since power up. Each element of the array corresponds to calibrations performed on each channel.
calsLastRun[4]	A 4-element array indication the bitmask of initial calibrations run in the most recent invocation of InitCalsRun. Each element of the array corresponds to calibrations performed on each channel.

SYSTEM CONSIDERATIONS FOR INITIAL CALIBRATIONS

The following diagrams are used to show how the device is configured for notable calibrations with external system requirements, such as the QEC and LOL calibrations. In all diagrams, gray lines and blocks are not active in the calibration. Lines showing the path of the LOs are shown in color to distinguish them from the signal paths. A brief explanation of the calibration is provided. Note that as the Arm performs each of the calibrations, it is tasked with configuring the device as per Figure 45, with respect to enabling/disabling paths, for example. No user input is required in this regard.

It is important that the user ensures that external conditions are met, such as having the PA off for all calibrations other than the external LOL initialization calibration, or having the Rx input properly terminated for an Rx QEC initialization calibration.

Rx QEC Initial Calibration

The Rx QEC initialization calibration algorithm is utilized to improve the Rx path QEC performance. The Rx QEC calibration routine sweeps a number of internally generated test tones across the desired frequency band, measuring quadrature performance and calculating correction coefficients. Tone generation is performed by the CAL PLL in the figure above, which is the AUX PLL. When the Rx QEC initialization calibration runs, the Arm configures the Rx to the maximum gain index (255).

System requirement: The input port must be isolated from incoming signals or the calibration may fail to complete. The calibration tones appear on the Rx pins and, therefore, must be prevented from reaching the antenna through the Rx port being properly terminated into a 50 Ω load. If an LNA is present at the Rx input, it is recommended to disable the LNA during the calibration.

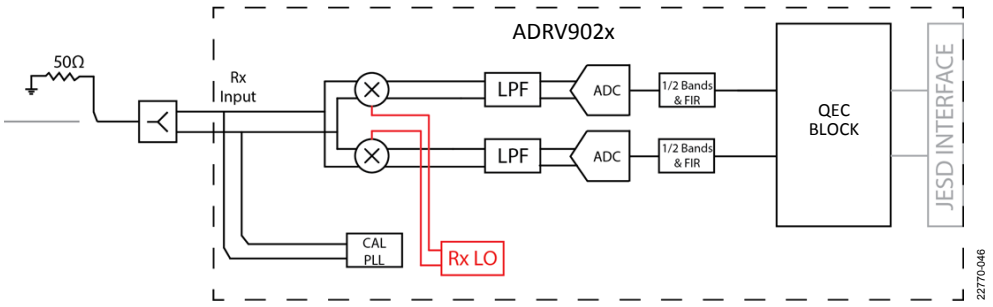


Figure 45. Rx QEC Initial Calibration System Configuration

ORx QEC Initial Calibration

The ORx QEC calibration functions by sweeping a number of internally generated test tones across the band measuring quadrature performance and calculating correction coefficients. The Arm determines which PLL is free for use as a calibration source given the LO selections. In the figure above, the Tx LO is the LO source for the ORx channel and the AUX PLL acts as the CAL PLL.

System Requirement: For optimum performance, the ORx QEC initialization must run at the same attenuation setting as described in the External Tx LOL Init Cal system requirement, that is, it is recommended to set the internal ORx attenuation to 10 dB for TXLO ≤ 2.8 GHz or 14 dB to 16 dB for TXLO > 2.8 GHz.

Isolate the ORx input from incoming signals and be properly terminated into a 50 Ω load while the calibration is running. The calibration tones appear on the ORx pins and, therefore, must be prevented from reaching the antenna.

Rx/ORx TIA Initial Calibration

The Rx/ORx TIA calibration is used to calibrate the corner frequency of the analog baseband TIA filter in the Rx/ORx signal path. The signal path used for this calibration is the same as the Rx QEC initialization calibration shown in Figure 45. The calibration applies two tones sequentially, one in-band and another at the TIA corner frequency, and compares the amplitude of both of these signals to ensure that the corner frequency produces the appropriate roll-off.

System requirement: Isolate the input port from incoming signals or the calibration may fail to complete.

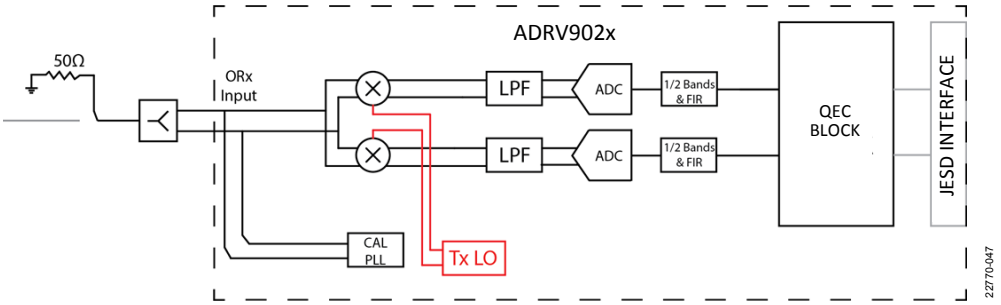


Figure 46. ORx QEC Initial Calibration System Configuration

Internal Tx LO Leakage and Tx QEC Initial Calibrations

The Tx LO Leakage and Tx QEC Initial calibrations utilize the internal loopback path and the baseband section of the ORx path to calculate its initial correction factors. During these calibrations, test signals (tones and wideband signals) are output. These appear at the Tx output, so it is important that the PA connected to the device output be switched off. Both calibrations sweep through a series of attenuation values, creating a table of initial calibration values over attenuation. Then during operation and upon application of a new Tx attenuation setting, the corresponding QEC and LOL correction values are applied to the Tx channel by the Arm. The device configuration for this calibration is shown in Figure 47.

System requirement: The PA in the Tx path must be powered off during these calibrations to prevent potential damage to the PA. When the PA is disabled, ensure the load seen at the Tx output is 50 Ω.

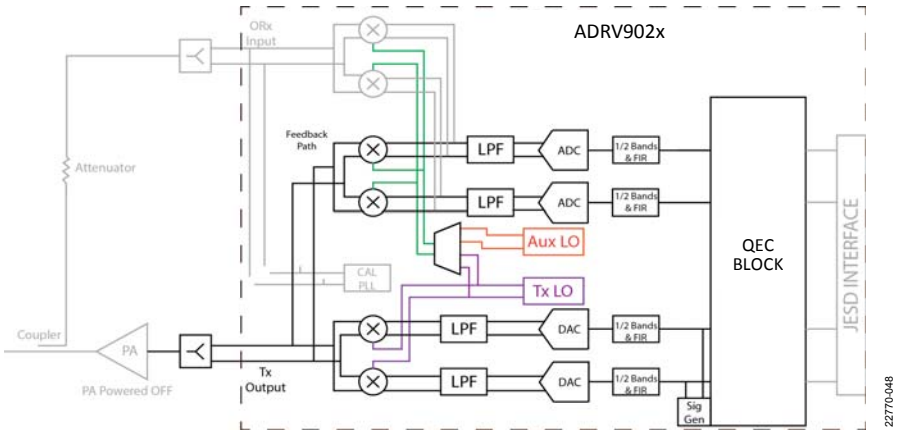


Figure 47. Device Path Configuration for Tx LOL and QEC Initial Calibrations

External Tx LO Leakage Initial Calibration

The external LOL initialization calibration requires that the PA be enabled such that a full external loop is made between the Tx outputs and the ORx inputs. The purpose of this calibration is to obtain a good estimate (gain/phase) of the external loop channel conditions prior to operation. The device configuration is shown in Figure 48. The calibration utilizes a pseudo-random noise signal to estimate the channel conditions. This is a broadband signal with a nominal level of -78 dBFS out of the DAC.

It is important that a suitable attenuator be chosen between the PA output and the ORx input. This is to prevent Tx data from saturating the ORx input. This is also necessary from the perspective of DPD operation.

Note: If the ORx receives an input signal larger than the ADC full scale, the channel overloads and calibration results are poor. The arm does not issue a warning or error condition in this case. Similarly the arm does not issue a warning if the physical Tx to ORx mapping does not match the programmed Tx to ORx mapping.

System requirement: The output of the Tx channel to be calibrated must be routed to the utilized ORx path to properly observe the calibration signal. If there is a PA in the path, it must be enabled during this calibration. The Tx to ORx mapping must be configured (via API or GPIO) prior to the calibration to indicate which Tx is routed back to which ORx (see the Tx to ORx Feedback section).

Choose the combined external coupler plus attenuation to provide a peak input power close to PHIGH at the ORx input pin such that the peak power is close to −2 dBFS at the digital output with the programmed internal attenuation. For optimal external LOL initial calibration and LOL tracking calibration, it is recommended to set the internal ORx attenuation to 10 dB for TXLO ≤ 2.8 GHz or 14 dB to 16 dB for TXLO > 2.8 GHz.

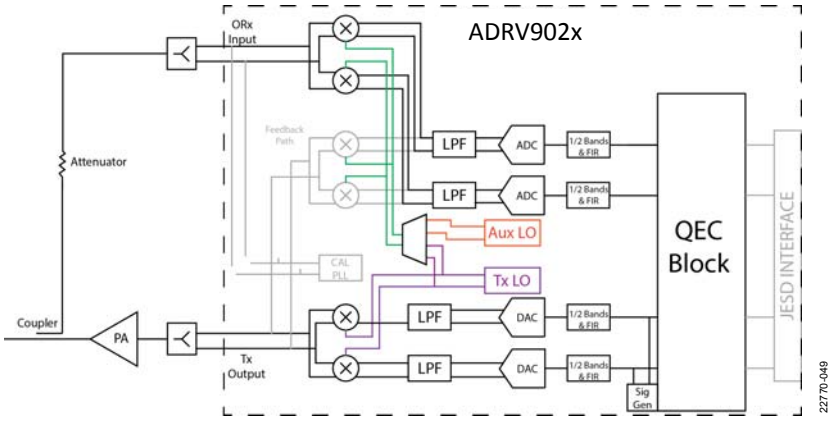


Figure 48. External LOL System Configuration (grayed-out circuitry not in use)

Rx Gain Delay Initial Calibration

The Rx datapath features an analog and a digital gain/attenuation element. If the analog and digital gain changed simultaneously, then the received baseband data shows a two-step change in the gain index. The first gain change seen in the baseband is from the digital gain change and the second gain change is from the analog gain change. This is due to the non-zero data path latency between the analog and digital gain/attenuation elements.

The Rx gain delay calibration measures the latency between the analog and digital gain/attenuation elements in order to delay the onset of digital gain. This ensures that when the analog and digital gain change that the baseband data shows a single coordinated gain change between these two elements. Because the analog gain change is not delayed there are no consequences to AGC timing due to this calibration.

Rx Gain Phase

The Rx gain phase calibration is used to minimize the phase differences between different gain indices. This calibration scans the gain table for unique analog attenuation settings and applies a phase shift for each setting to minimize the phase difference between gain index settings. The AUX PLL is used to transmit a tone at the Rx input and measure the phase difference. The phase shift is introduced by a digital phase shifting element.

Tx Attenuation Phase Initial Calibration

This calibration is called ADI_ADRV9025_TX_ATTEN_TABLE in the API enumerations. This calibration corrects for phase differences between different attenuation settings in the Tx attenuation table. A tone is transmitted during this calibration at -12 dBFS and it is advised to disable the PA during this calibration. No external loopback is necessary during the operation of this calibration.

Run this calibration run prior to any LO Leakage initial calibrations. When combined in the initial calibration mask with LO Leakage calibrations, the Arm sequences this cal before LO Leakage initial calibrations.

The attenuation phase calibration supports up to 20 dB of attenuation.

This calibration has known performance issues below 1 GHz LO frequency operation.

Tx Attenuation Delay

Similar to the Rx, the Tx datapath features an analog and digital gain/attenuation element. The Tx attenuation delay calibration helps to ensure that when a change in attenuation occurs in both analog and digital that the Tx output only sees a single change in output power rather than a two-step effect. This is done by delaying the onset of the analog attenuator change, unlike the Rx Gain Delay calibration, which delays the onset of digital gain/attenuator changes.

Tx to ORx Feedback

For the external Tx LO leakage initial calibration to complete, the Arm must be advised of the current Tx to ORx feedback paths through the external circuitry. Specify this at initialization, through the adi_adrv9025_PostMcsInit_t structure that is passed to adi_adrv9025_PostMcsInit( ). In this structure, there are four variables which indicate which Tx is being fed back to each ORx. These are shown in Table 74.

Table 74. Definition of adi_adrv9025_TxToOrxMappingConfig_t

ORx	Permissible Values
orx1Map	ADI_ADRV9025_MAP_NONE_ORX1 ADI_ADRV9025_MAP_TX1_ORX1 ADI_ADRV9025_MAP_TX2_ORX1
orx2Map	ADI_ADRV9025_MAP_NONE_ORX2 ADI_ADRV9025_MAP_TX1_ORX2 ADI_ADRV9025_MAP_TX2_ORX2
orx3Map	ADI_ADRV9025_MAP_NONE_ORX3 ADI_ADRV9025_MAP_TX3_ORX3 ADI_ADRV9025_MAP_TX4_ORX3
orx4Map	ADI_ADRV9025_MAP_NONE_ORX4 ADI_ADRV9025_MAP_TX3_ORX4 ADI_ADRV9025_MAP_TX4_ORX4

Note: In the case of multiple Tx channels being fed back to a single ORx, a multiple pass is required for the External Tx LO Leakage initial calibration. During the first pass when adi_adrv9025_PostMcsInit( ) is called, the current feedback paths must be advised to the device. When the external LOL initial calibration is run, the Arm performs the calibration on Tx paths that have a feedback path to an ORx. In a second pass, the feedback paths are modified and advised to the device, and the external LOL initial calibration must be called again.

Note Regarding AUX LO Settings During Initialization Calibrations

For users who intend to use an AUX LO frequency other than the default AUX LO frequency for their given use case, note that initial calibrations must run with the default AUX PLL frequency. Therefore, the user must use the following procedure if a non default AUX PLL frequency is used in their application. This procedure is as follows:

1. Set the Tx PLL frequency to the desired frequency.
   a. If the user uses adi_adrv9025_PllFrequencySet(…), the AUX PLL is configured to the default offset frequency when the Tx PLL is programmed.
   b. If the user uses adi_adrv9025_PllFrequencySet_v2(…), the AUX PLL is configured to the default offset frequency if the adi_adrv9025_PllConfig_t->pllAuxLoOffsetProgSel parameter is set to ADI_ADRV9025_PLL_AUX_LO_OFFSET_PROG_ENABLE.
2. Run initialization calibrations.
3. After all initialization calibrations are complete, the user can set the AUX PLL frequency to the desired application frequency.

If the user sets the AUX PLL to a different frequency and requires initial calibrations to be rerun, follow this procedure.

Summary of Initial Calibration Requirements

Table 75 summarizes initial calibration requirements and other related details as mentioned previously.

Table 75. Recommended Initial Calibrations

Initial Calibration	Recommendations
Rx QEC	ADI_ADRV9025_MAP_NONE_ORX1 ADI_ADRV9025_MAP_TX1_ORX1 ADI_ADRV9025_MAP_TX2_ORX1
Rx TIA	ADI_ADRV9025_MAP_NONE_ORX2 ADI_ADRV9025_MAP_TX1_ORX2 ADI_ADRV9025_MAP_TX2_ORX2
ORx TIA	ADI_ADRV9025_MAP_NONE_ORX3 ADI_ADRV9025_MAP_TX3_ORX3 ADI_ADRV9025_MAP_TX4_ORX3
orx4Map	ADI_ADRV9025_MAP_NONE_ORX4 ADI_ADRV9025_MAP_TX3_ORX4 ADI_ADRV9025_MAP_TX4_ORX4

TRACKING CALIBRATIONS

The Arm processor is tasked with ensuring that QEC and LOL (and HD2 for GSM applications) corrections are optimal throughout device operation over time, attenuation, and temperature. It achieves this by performing calibrations at regular intervals. These calibrations are termed "tracking calibrations", and utilize normal traffic data to update the path correction coefficients.

The following API function enables the tracking calibrations in the Arm:

```
adi_adrv9025_TrackingCalsEnableSet(adi_adrv9025_Device_t *device, uint32_t enableMask,
  adi_adrv9025_TrackingCalEnableDisable_e enableDiasbleFlag)
```

enableMask is a mask that informs the Arm processor which tracking calibrations to run (Table 76 shows the bit assignments of the enable mask (Presently only Rx/ORx QEC calibrations are available))

enableDiasbleFlag is an enable or disable parameter (valid enums are shown in Table 77)

Based on the enum chosen for enableDiasbleFlag, the selected tracking calibrations in enableMask is enabled or disabled.

Table 76. Tracking Calibrations Enable Mask Bit Assignments

Cal Mask Bits	Function
D0	Rx1 QEC Tracking
D1	Rx2 QEC Tracking
D2	Rx3 QEC Tracking
D3	Rx4 QEC Tracking
D4	ORx1 QEC Tracking
D5	ORx2 QEC Tracking
D6	ORx3 QEC Tracking
D7	ORx4 QEC Tracking
D8	Tx1 LOL Tracking
D9	Tx2 LOL Tracking
D10	Tx3 LOL Tracking
D11	Tx4 LOL Tracking
D12	Tx1 QEC Tracking
D13	Tx2 QEC Tracking
D14	Tx3 QEC Tracking
D15	Tx4 QEC Tracking

Table 77. adi_adrv9025_TrackingCalEnableDisable_e Definition

ENUM	Description
ADI_ADRV9025_TRACKING_CAL_DISABLE	When used, the selected tracking calibrations in enableMask is disabled upon the call to adi_adrv9025_TrackingCalsEnableSet.
ADI_ADRV9025_TRACKING_CAL_ENABLE	When used, the selected tracking calibrations in enableMask is enabled upon the call to adi_adrv9025_TrackingCalsEnableSet.

The arm is tasked with the scheduling of the tracking calibrations. No user input is required to initiate a tracking calibration.

System Considerations for Tracking Calibrations

This section describes the operation of the tracking calibrations. Diagrams are used to show how the device is configured for each calibration, and a brief explanation of the calibration is provided. In all diagrams, grayed-out lines and blocks are not active in the calibration. Lines showing the path of the LOs are shown in color to distinguish them from the signal paths. As the Arm performs each of the calibrations, it is tasked with configuring the feedback path or ORx input as per the following diagrams. No user input is required in this regard. However, for external LOL tracking the user must ensure that the feedback path is available to use.

The following sections show the requirements for GPIO and enable pins during each of the tracking calibrations. These calibrations may need many milliseconds of observation to calculate an update. The Arm reduces the total time needed by splitting up this time into batches such that observations do not need to be continuous. The Arm algorithms are optimized to process batches of 100 µs, but smaller batches are acceptable.

The Rx/ORx tracking algorithms run while the channels are in normal use, using the data in the channel to calculate updates to the correction coefficients. The Tx correction algorithms utilize the ORx path when they are run, feeding back transmission data for observation to calculate updates to the correction coefficients. Thus ORx paths must be time shared with other uses of the ORx path.

Given the device has two observation paths, the expectation is that the calibrations always have access to a single ORx path, an equal amount of time for ORx paths on either side of the device (that is, an equal amount of time on ORx1/ORx2 and ORx3/ORx4). When an ORx on one side of the device is being assigned to calibrations, the other ORx(s) on the other side of the device are available to the user for observation.

Rx QEC Tracking Calibration

The Rx QEC tracking algorithm improves the Rx path QEC performance during operation. It utilizes normal traffic data to calculate updated corrected coefficients. It runs continuously while the receivers are active.

System requirement: Rx channels must be enabled. For example, in TDD mode Rx QEC tracking only runs during Rx periods. If only one channel is enabled, the Rx QEC only runs on this channel.

Note: In FDD modes, Rx Enable is high at all times. Rx Enable refers to the enable of any of Rx1-4.

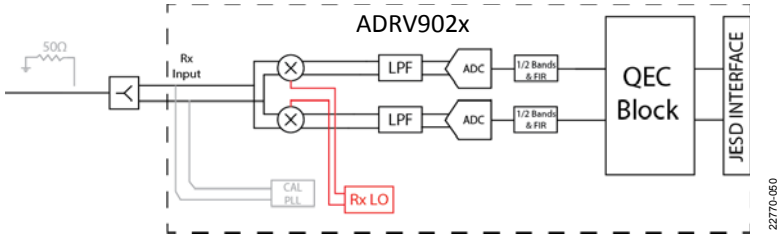


Figure 49. Rx QEC Tracking

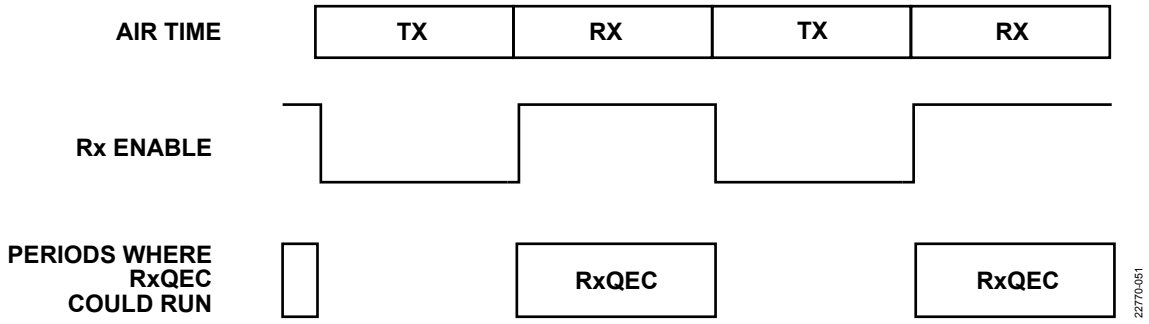


Figure 50. Timing Diagram Showing When RxQEC Can Run in TDD Mode

ORx QEC Tracking Calibration

The ORx QEC tracking algorithm improves the ORx path QEC performance during operation. It utilizes normal traffic data to calculate updated corrected coefficients. It runs continuously in the background while the observation receiver is active.

System Requirement: ORx channels must be enabled. For example, in TDD mode ORx QEC tracking only runs during ORx periods. If only one channel is enabled, the ORx QEC only runs on this channel.

Do not change the ORx gain index while the tracking cal runs. If the ORx gain index changes, re-run the ORx QEC initial calibration.

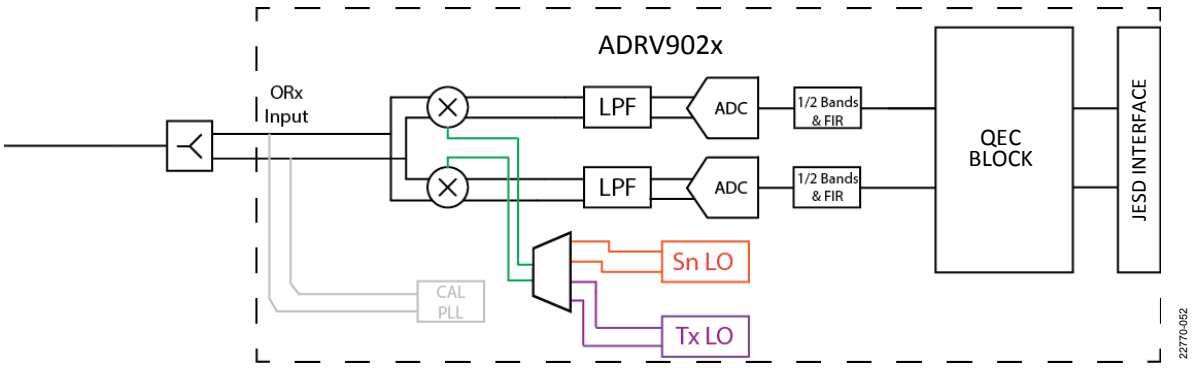


Figure 51. ORx QEC Tracking

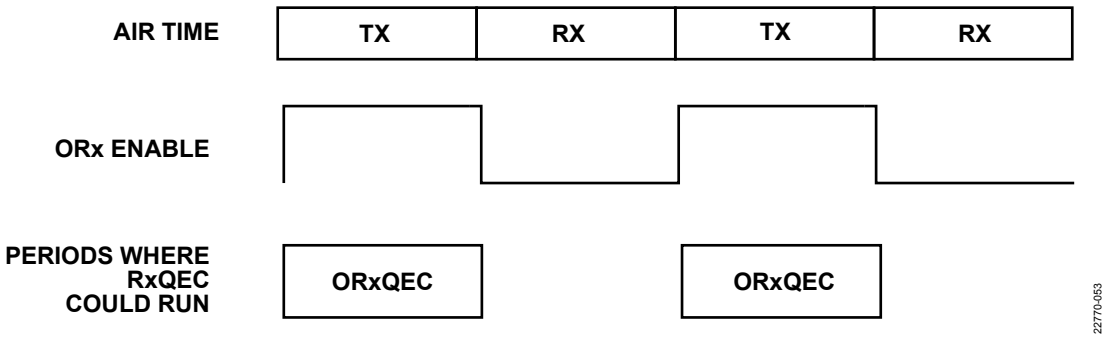


Figure 52. Timing Diagram Showing When ORx QEC Can Run in TDD Mode (ORx Enable Refers to the Internal Enable Control of ORx1 to ORx4)

Tx QEC Tracking Calibration

The Tx QEC tracking is an online calibration that is run to improve the QEC performance using transmit data. It utilizes the loopback (feedback) path for operation. Therefore, the Tx QEC tracking must be interleaved with normal other captures that utilize the ORx path. This tracking determines optimal coefficients for the current gain setting, updating the table stored during the Tx QEC initialization to make sure this table has the best values for the current operating conditions. Figure 53 shows the device configuration for Tx QEC tracking calibration.

System Requirement: Tx channel(s) must be enabled. To run, the ORx path must be available for the Arm to use (ORx enable low). That means the required ORx path cannot be required by the user for other (or VSWR and so forth) captures.

Note: In FDD modes, Tx Enable is high at all times. Tx Enable refers to the enable of any of Tx1-Tx4. ORx Enable refers to the internal enable signal for the selected ORx channel.

QEC tracking uses an offset LO on the feedback path during tracking. This ensures that the quadrature errors of the Tx path are not aligned with those of the ORx path. This frequency is set to

$$f_{OFFSET} = ((Primary\ Tx\ Bandwidth/4) + 5\ MHz$$

Continuous wave tones placed at $\pm f_{OFFSET}$, or 2× ($\pm f_{OFFSET}$), show reduced QEC performance. However, modulated signals centered at these frequencies do not have reduced performance.

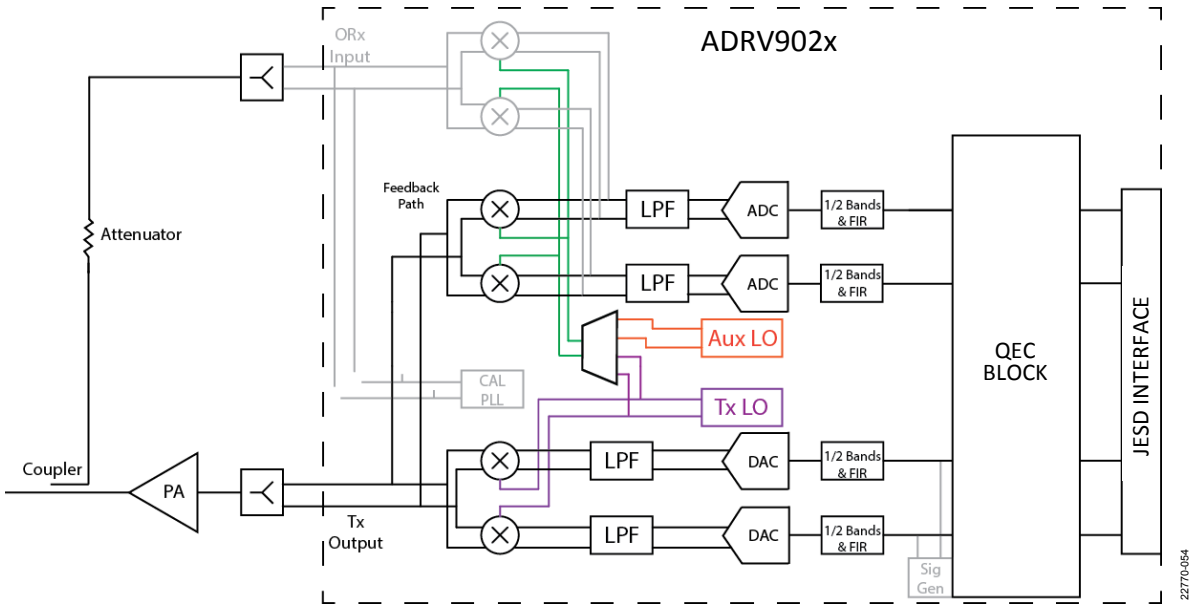


Figure 53. Tx QEC Tracking Calibration Configuration

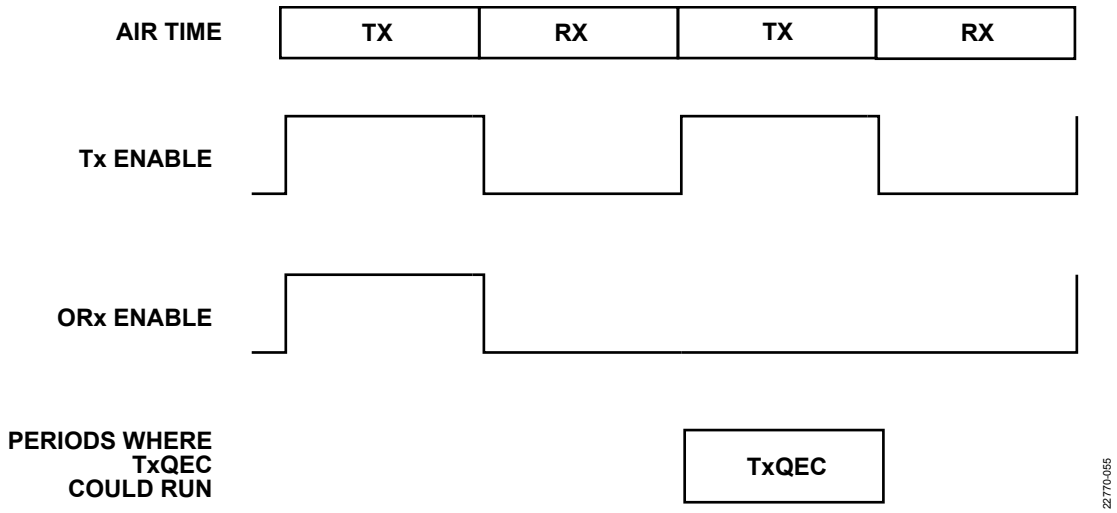


Figure 54. Timing Diagram Showing When TxQEC Can Run in TDD Mode

Tx LOL Tracking Calibration

The Tx LO leakage tracking calibration uses an external path between the Tx output and ORx input to measure LO leakage and calculate correction factors. This calibration is run while user data is being transmitted (with the PA operational). For this calibration, the AuxLO is used in the ORx path to offset the Tx LO leakage from the ORx LO. Figure 55 shows the device configuration for the TX LO leakage tracking calibration with the Tx output looped back to the ORx input (an ORx on the same side of the chip as the Tx being calibrated).

Note: If the observation receiver receives an input signal larger than the ADC full scale, the channel is overloads and calibration results are poor. The arm does not issue a warning or error condition in this case.

System Requirement: Tx channel(s) must be enabled. The ORx path must be available for the Arm to use (that is, not required by the user for DPD (or VSWR) captures). The ORx path must be connected to the appropriate Tx to be calibrated, and the Arm must be advised which Tx output has a connection to which ORx.

A proper channel estimate is required for optimal LOL tracking performance. A new initial channel estimate must be acquired when the LO frequency changes or ORx gain index changes. There are two methods to achieve this as follows; however, it is highly recommended to follow the first procedure.

1. Run External Tx LO Leakage Initial Calibration. Ensure that mapping is setup properly, PA is enabled, and all tracking calibrations are disabled.
2. If not running External Tx LO Leakage Initial Calibration, follow the procedure below.
    a. Recommended Sequence to run Tx LOL Tracking calibration if either skip Ext Tx LOL Init Cal, change LO Frequency or ORx attenuation: Disable Tx LOL tracking (if it is running)
    b. If Tx traffic has content at dc, disable data transmission. If data is offset from dc it can be left on.
    c. Reset the desired channels using the ExtTxLolchannelReset() command
    d. Call TrackingCalTxLolStatusGet() and note the value of iterCount
    e. Enable Tx LOL tracking
    f. Call TrackingCalTxLolStatusGet() again and note the value of iterCount
    g. If the iterCount value has increased by at least 1, enable Tx data transmission

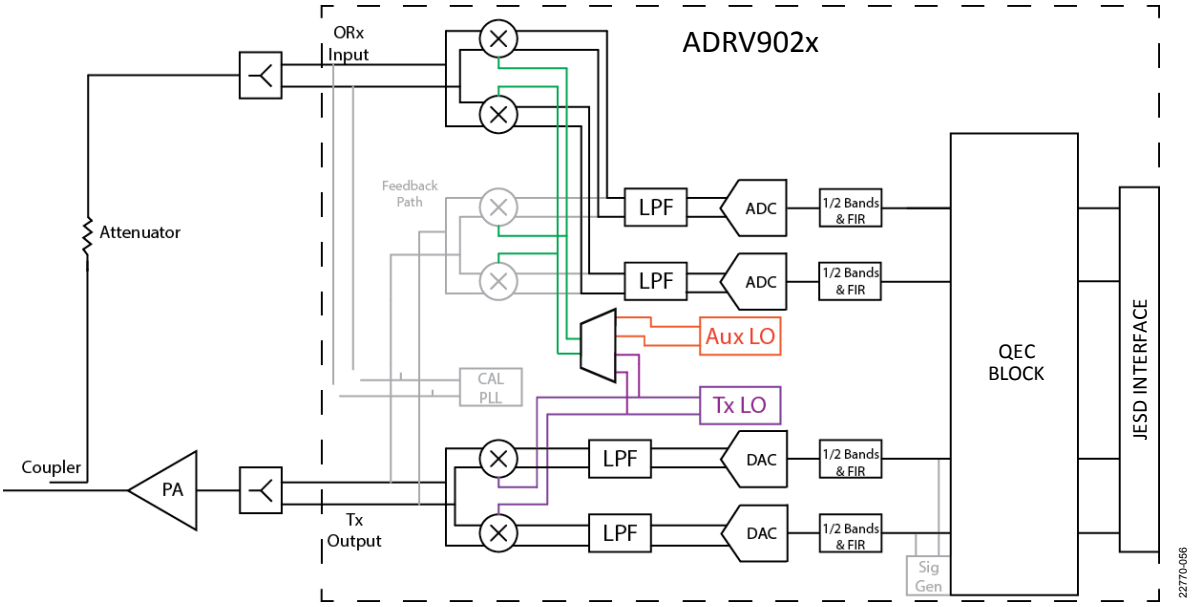


Figure 55. Tx LOL Tracking Configuration

CALIBRATION GUIDELINES AFTER PLL FREQUENCY CHANGES

Some applications require changing the PLL frequency for Tx, Rx, or ORx signal paths after the transceiver has started normal operation and tracking calibrations have improved performance. Some tracking calibrations require re-running initial calibrations after the PLL frequency change in order to relearn the new channel conditions. It is important that certain procedures are followed in order to maintain proper operation of the tracking cals.

The LO frequency changes fall into one of two types:

Type 1: LO frequency change that is described by both of the following criteria:

- The LO frequency change is less than 100 MHz.
- The LO frequency change does not step over an LO divider boundary as explained in the Synthesizer Configuration section. Note the table that describes the "Div by" settings.

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

- The LO frequency change is greater than 100 MHz.
- The LO frequency change steps over an LO divider boundary.

Type 1 Frequency Change Procedure

If the LO frequency change falls into Type 1 described in the Calibration Guidelines after PLL Frequency Changes section, implement the following procedure:

1. Disable all tracking calibrations
2. Disable all RF channels. If TX_EN/RX_EN/ORX_CTRL pins cannot stop toggling, put the device into command control mode via adi_adrv9025_RadioCtrlCfgSet(…), then call adi_adrv9025_RxTxEnaleSet(…) to disable all channels.
3. Rerun the following initial calibrations. Make sure to follow system considerations as described in System Considerations for Initial Calibrations. Please ensure that INTERNAL_PATH_DELAY is run prior to TX_QEC_INIT if calibrations are run one at a time. The Arm sequences the calibrations properly when
   a. ADI_ADRV9025_INTERNAL_PATH_DELAY (if Tx QEC Tracking is used)
   b. ADI_ADRV9025_LO_LEAKAGE_EXTERNAL. This step is optional but highly recommended. The PA must be enabled in this step. Ensure that the external calibration is run for all Tx to ORx mappings used in the application.
      If the previous step is not executed, it is mandatory to call adi_adrv9025_ExtTxLolChannelReset(…) command for each Tx channel. It must be called one Tx channel at a time. Then a special procedure must be followed to relearn the channel estimate described in the Tx LOL Tracking Calibration section.
   c. Enable relevant tracking calibrations.
   d. Transition back to pin control mode, if necessary.

Type 2 Frequency Change Procedure

If the LO frequency change falls into Type 2 described in the Calibration Guidelines after PLL Frequency Changes section, implement a similar procedure to the Type 1 frequency change procedure while adding the ADI_ADRV9025_LOOPBACK_RX_LO_DELAY and ADI_ADRV9025_TX_QEC_INIT calibrations.

Initialization Calibrations Durations

In order to achieve best performance, the device features autonomous internal calibrations that are performed during device initialization. The calibrations are run in the Post-MCS section of device initialization. The majority of the calibrations are run with a single API call once the calibration structure is set. These are the internal calibrations that utilize internal loopback paths. Those that utilize external paths (such as External Tx LOL calibration) are run separately afterward.

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

In the following diagram the slices show the relative timing of each common initialization calibration relative to the total time. Some of the calibrations are very short and mostly involve for example loading coefficients and initializing for operation, or measuring the delay of the calibration path. Some others require observation of either internally generated calibration tones or pseudo-random noise to calculate the required coefficients that are used to define the characteristics of the channel. Still others for example the Tx QEC calibration use an algorithm to determine the correction factors which can be influenced by the actual load conditions the transmitter is connected to. For these reasons, the amount of time each of the calibrations needs to complete may vary slightly.

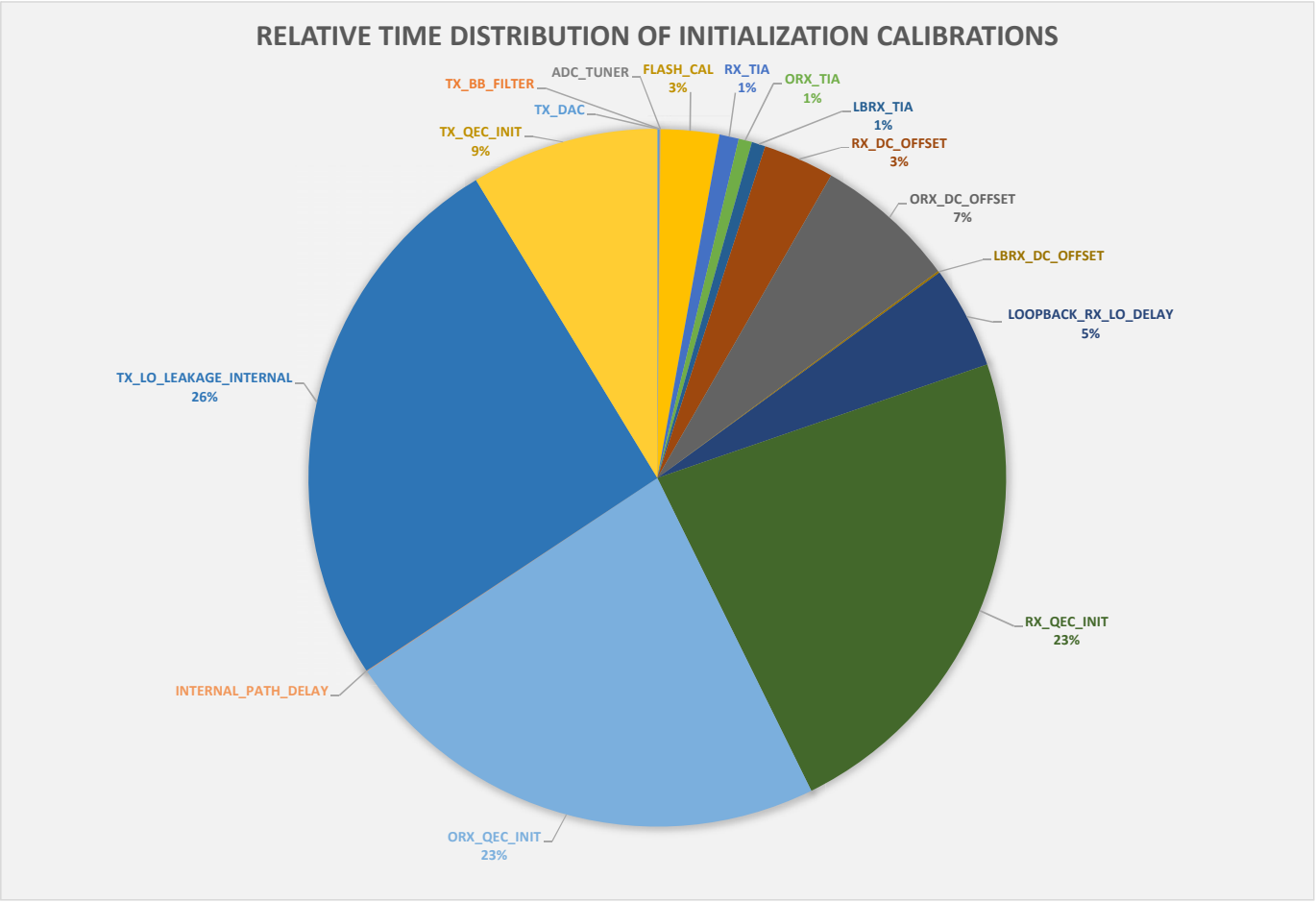


Figure 56. Relative Time Distribution of Initialization Calibrations

The following tables are measured calibration times of the device for a number of different use cases using the standard calibration mask of 0xD73FF. These results can be used as guidelines as to what the typical expected times are for a particular configuration. The columns in the table show the calibration timing results in milliseconds for 1, 2, 3, and 4 enabled Rx or Tx channels. In the case of ORx calibrations, because there are just two shared paths, the entries for ORX_DC_OFFSET are different for 1 and 2 channels enabled, but remain the same for 3 and 4 enabled channels. Other ORx calibrations do show differences from 1 to 4 channels because the paths from each of the transmitters is calibrated individually.

Table 78. ADRV9025Init_StdUseCase13_nonLinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC13_NLS	225 MHz	245.76 MHz	1.966 GHz	225 MHz	245.76 MHz	4.915 GHz	100 MHz	122.88 MHz	1.966 GHz

Table 79.

Calibration	1 Channel	2 CHANNELS	3 Channels	4 Channels
TX_DAC	4	8	12	17
TX_BB_FILTER	2	2	4	5
ADC_TUNER	1	1	1	1
FLASH_CAL	219	263	324	365
RX_TIA	84	125	166	207
ORX_TIA	64	86	108	128
LBRX_TIA	64	86	107	129
RX_DC_OFFSET	451	451	451	451
ORX_DC_OFFSET	451	899	899	899
LBRX_DC_OFFSET	8	14	14	14
LOOPBACK_RX_LO_DELAY	175	345	510	679
RX_QEC_INIT	756	1508	2262	3013
ORX_QEC_INIT	787	1570	2354	3137
INTERNAL_PATH_DELAY	1	1	3	3
TX_LO_LEAKAGE_INTERNAL	892	1781	2671	3560
TX_QEC_INIT	584	1162	1730	2323
Total Calibration time (ms)	4545	8302	11616	14932

Table 80. ADRV9025Init_StdUseCase14_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC14_LS	450 MHz	491.52 MHz	1.966 GHz	450 MHz	491.52 MHz	4.915 GHz	200 MHz	245.76 MHz	4.915 GHz

Table 81.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	4	9	12	17
TX_BB_FILTER	1	2	4	4
ADC_TUNER	1	1	1	1
FLASH_CAL	219	263	324	365
RX_TIA	64	85	106	126
ORX_TIA	54	65	76	88
LBRX_TIA	54	65	77	87
RX_DC_OFFSET	451	451	451	450
ORX_DC_OFFSET	467	899	898	899
LBRX_DC_OFFSET	7	14	14	14
LOOPBACK_RX_LO_DELAY	163	324	484	644
RX_QEC_INIT	787	1570	2354	3138
ORX_QEC_INIT	785	1566	2347	3127
INTERNAL_PATH_DELAY	1	2	2	3
TX_LO_LEAKAGE_INTERNAL	876	1748	2622	3494
TX_QEC_INIT	293	595	908	1197
Total Calibration time (ms)	4226	7658	10680	13654

Table 82. ADRV9025Init_StdUseCase14C_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC14C_LS	450 MHz	491.52 MHz	1.966 GHz	450 MHz	491.52 MHz	4.915GHz	200 MHz	245.76 MHz	4.915 GHz

Table 83.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	3	6	11	15
TX_BB_FILTER	2	3	3	4
ADC_TUNER	1	1	1	1
FLASH_CAL	219	262	325	368
RX_TIA	64	85	105	126
ORX_TIA	54	65	76	87
LBRX_TIA	55	65	76	87
RX_DC_OFFSET	451	451	450	451
ORX_DC_OFFSET	450	899	899	899
LBRX_DC_OFFSET	7	15	14	14
LOOPBACK_RX_LO_DELAY	166	325	484	645
RX_QEC_INIT	786	1569	2354	3138
ORX_QEC_INIT	784	1567	2350	3130
INTERNAL_PATH_DELAY	1	2	2	2
TX_LO_LEAKAGE_INTERNAL	876	1749	2622	3495
TX_QEC_INIT	307	593	902	1188
Total Calibration time (ms)	4226	7657	10674	13649

Table 84. ADRV9025Init_StdUseCase23C_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC23C_LS	337.5 MHz	368.64 MHz	1.475 GHz	337.5 MHz	368.64 MHz	3.686 GHz	150 MHz	184.32 MHz	3.686 GHz

Table 85.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	4	5	9	11
TX_BB_FILTER	1	2	3	5
ADC_TUNER	1	1	1	1
FLASH_CAL	285	343	426	482
RX_TIA	84	111	139	172
ORX_TIA	72	86	100	114
LBRX_TIA	72	85	100	115
RX_DC_OFFSET	450	451	451	451
ORX_DC_OFFSET	451	899	898	898
LBRX_DC_OFFSET	7	14	14	15
LOOPBACK_RX_LO_DELAY	210	419	628	835
RX_QEC_INIT	863	1728	2583	3443
ORX_QEC_INIT	861	1718	2574	3430
INTERNAL_PATH_DELAY	1	2	3	4
TX_LO_LEAKAGE_INTERNAL	883	1765	2645	3526
TX_QEC_INIT	401	800	1192	1607
Total Calibration time (ms)	4645	8429	11767	15108

Table 86. ADRV9025Init_StdUseCase26C_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC26C_LS	450 MHz	491.52 MHz	1.966 GHz	450 MHz	491.52 MHz	4.915 GHz	200 MHz	245.76 MHz	4.915 GHz

Table 87.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	4	7	10	15
TX_BB_FILTER	1	2	3	4
ADC_TUNER	1	1	1	1
FLASH_CAL	220	263	324	367
RX_TIA	64	84	106	125
ORX_TIA	54	66	76	88
LBRX_TIA	55	65	75	86
RX_DC_OFFSET	451	450	451	450
ORX_DC_OFFSET	451	900	899	899
LBRX_DC_OFFSET	7	14	14	14
LOOPBACK_RX_LO_DELAY	163	323	485	645
RX_QEC_INIT	787	1571	2354	3137
ORX_QEC_INIT	783	1564	2346	3125
INTERNAL_PATH_DELAY	1	2	2	3
TX_LO_LEAKAGE_INTERNAL	876	1748	2622	3494
TX_QEC_INIT	291	597	892	1201
Total Calibration time (ms)	4210	7657	10660	13654

Table 88. ADRV9025Init_StdUseCase26C_nonLinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC26C_NLS	450 MHz	491.52 MHz	1.966 GHz	450 MHz	491.52 MHz	4.915 GHz	200 MHz	245.76 MHz	4.915 GHz

Table 89.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	3	7	13	14
TX_BB_FILTER	1	2	3	4
ADC_TUNER	1	1	1	1
FLASH_CAL	220	261	324	368
RX_TIA	64	85	105	125
ORX_TIA	54	65	77	87
LBRX_TIA	54	65	76	87
RX_DC_OFFSET	451	451	451	451
ORX_DC_OFFSET	450	899	899	899
LBRX_DC_OFFSET	7	14	14	15
LOOPBACK_RX_LO_DELAY	164	325	485	645
RX_QEC_INIT	786	1570	2355	3138
ORX_QEC_INIT	785	1565	2346	3128
INTERNAL_PATH_DELAY	1	1	2	2
TX_LO_LEAKAGE_INTERNAL	876	1749	2621	3494
TX_QEC_INIT	293	598	891	1192
Total Calibration time (ms)	4211	7658	10663	13649

Table 90. ADRV9025Init_StdUseCase50_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC50_LS	450 MHz	122.88 MHz	1.966 GHz	450 MHz	245.76 MHz	4.915 GHz	100 MHz	122.88 MHz	1.966 GHz

Table 91.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	5	7	12	13
TX_BB_FILTER	1	2	3	4
ADC_TUNER	1	1	1	1
FLASH_CAL	218	261	324	369
RX_TIA	85	126	166	207
ORX_TIA	54	65	76	88
LBRX_TIA	54	66	75	87
RX_DC_OFFSET	451	452	451	452
ORX_DC_OFFSET	450	899	899	899
LBRX_DC_OFFSET	7	15	14	14
LOOPBACK_RX_LO_DELAY	172	342	506	663
RX_QEC_INIT	793	1583	2373	3161
ORX_QEC_INIT	784	1564	2347	3126
INTERNAL_PATH_DELAY	1	2	2	3
TX_LO_LEAKAGE_INTERNAL	876	1749	2621	3494
TX_QEC_INIT	299	588	899	1186
Total Calibration time (ms)	4251	7721	10771	13766

Table 92. ADRV9025Init_StdUseCase50_nonLinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC50_LS	450 MHz	122.88 MHz	1.966 GHz	450 MHz	245.76 MHz	4.915 GHz	100 MHz	122.88 MHz	1.966 GHz

Table 93.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	4	8	10	15
TX_BB_FILTER	1	3	3	4
ADC_TUNER	1	1	1	1
FLASH_CAL	219	262	324	367
RX_TIA	84	125	167	208
ORX_TIA	54	65	77	87
LBRX_TIA	54	66	76	87
RX_DC_OFFSET	451	451	451	452
ORX_DC_OFFSET	451	899	899	898
LBRX_DC_OFFSET	7	14	15	14
LOOPBACK_RX_LO_DELAY	174	343	507	663
RX_QEC_INIT	757	1508	2261	3012
ORX_QEC_INIT	785	1564	2347	3129
INTERNAL_PATH_DELAY	1	1	2	2
TX_LO_LEAKAGE_INTERNAL	875	1749	2622	3494
TX_QEC_INIT	301	605	890	1200
Total Calibration time (ms)	4219	7663	10651	13634

Table 94. ADRV9025Init_StdUseCase51_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC51_LS	450 MHz	245.76 MHz	1.966 GHz	450 MHz	245.76 MHz	4.915 GHz	200 MHz	245.76 MHz	4.915 GHz

Table 95.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	3	9	11	13
TX_BB_FILTER	1	3	3	4
ADC_TUNER	1	1	1	1
FLASH_CAL	219	262	324	367
RX_TIA	63	84	105	126
ORX_TIA	54	65	76	87
LBRX_TIA	54	64	75	87
RX_DC_OFFSET	450	451	451	451
ORX_DC_OFFSET	451	898	899	899
LBRX_DC_OFFSET	7	14	14	14
LOOPBACK_RX_LO_DELAY	165	328	488	647
RX_QEC_INIT	786	1571	2353	3139
ORX_QEC_INIT	783	1564	2347	3126
INTERNAL_PATH_DELAY	1	1	2	2
TX_LO_LEAKAGE_INTERNAL	873	1742	2612	3482
TX_QEC_INIT	291	598	921	1191
Total Calibration time (ms)	4203	7656	10682	13637

Table 96. ADRV9025Init_StdUseCase51_nonLinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC51_NLS	450 MHz	245.76 MHz	1.966 GHz	450 MHz	245.76 MHz	4.915 GHz	200 MHz	245.76 MHz	4.915 GHz

Table 97.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	4	9	10	16
TX_BB_FILTER	2	2	3	5
ADC_TUNER	1	1	1	1
FLASH_CAL	219	263	325	367
RX_TIA	64	85	106	127
ORX_TIA	54	65	77	87
LBRX_TIA	54	65	76	87
RX_DC_OFFSET	451	450	451	450
ORX_DC_OFFSET	451	899	898	898
LBRX_DC_OFFSET	7	14	14	14
LOOPBACK_RX_LO_DELAY	164	326	486	646
RX_QEC_INIT	790	1573	2358	3142
ORX_QEC_INIT	783	1564	2346	3128
INTERNAL_PATH_DELAY	1	1	2	3
TX_LO_LEAKAGE_INTERNAL	872	1743	2612	3482
TX_QEC_INIT	293	600	921	1210
Total Calibration time (ms)	4210	7659	10686	13664

Table 98. ADRV9025Init_StdUseCase54_nonLinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC54_NLS	450 MHz	122.88 MHz	1.966 GHz	450 MHz	245.76 MHz	4.915 GHz	200 MHz	122.88 MHz	4.915 GHz

Table 99.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	86	89	92	97
TX_BB_FILTER	83	84	85	86
ADC_TUNER	82	81	82	81
FLASH_CAL	301	344	560	603
RX_TIA	146	167	228	250
ORX_TIA	136	147	199	210
LBRX_TIA	136	147	199	210
RX_DC_OFFSET	532	532	980	980
ORX_DC_OFFSET	532	980	1428	1877
LBRX_DC_OFFSET	89	96	102	109
LOOPBACK_RX_LO_DELAY	99	115	283	447
RX_QEC_INIT	870	1655	2440	3223
ORX_QEC_INIT	865	1647	2430	3211
INTERNAL_PATH_DELAY	82	83	85	84
TX_LO_LEAKAGE_INTERNAL	957	1850	2703	3576
TX_QEC_INIT	447	724	1022	1326
Total Calibration time (ms)	5443	8742	12917	16370

Table 100. ADRV9025Init_StdUseCase55_nonLinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC55_NLS	450 MHz	122.88 MHz	1.966 GHz	450 MHz	245.76 MHz	4.915 GHz	160 MHz	122.88 MHz	4.915 GHz

Table 101.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	86	89	93	119
TX_BB_FILTER	83	84	85	86
ADC_TUNER	81	82	81	81
FLASH_CAL	300	344	561	604
RX_TIA	146	166	227	248
ORX_TIA	136	160	199	209
LBRX_TIA	136	146	199	210
RX_DC_OFFSET	533	532	981	982
ORX_DC_OFFSET	532	980	1428	1877
LBRX_DC_OFFSET	89	96	103	110
LOOPBACK_RX_LO_DELAY	99	115	284	449
RX_QEC_INIT	656	1229	1799	2370
ORX_QEC_INIT	866	1658	2431	3211
INTERNAL_PATH_DELAY	82	83	84	85
TX_LO_LEAKAGE_INTERNAL	958	1831	2703	3576
TX_QEC_INIT	404	720	1017	1336
Total Calibration time (ms)	5186	8315	12274	15552

Table 102. ADRV9025Init_StdUseCase61_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC61_LS	300 MHz	368.64 MHz	1.843 GHz	337.5 MHz	368.64 MHz	3.686 GHz	300 MHz	368.64 MHz	3.686 GHz

Table 103.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	7	11	15	20
TX_BB_FILTER	4	5	6	8
ADC_TUNER	2	3	3	3
FLASH_CAL	292	342	435	491
RX_TIA	73	88	102	117
ORX_TIA	74	88	103	118
LBRX_TIA	74	88	103	117
RX_DC_OFFSET	453	453	452	453
ORX_DC_OFFSET	453	901	901	902
LBRX_DC_OFFSET	11	21	22	21
LOOPBACK_RX_LO_DELAY	242	480	721	967
RX_QEC_INIT	861	1718	2573	3430
ORX_QEC_INIT	862	1717	2574	3431
INTERNAL_PATH_DELAY	4	5	5	7
TX_LO_LEAKAGE_INTERNAL	885	1763	2642	3521
TX_QEC_INIT	403	807	1231	1631
Total Calibration time (ms)	4701	8489	11889	15236

Table 104. ADRV9025Init_StdUseCase82C_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC82C_LS	450 MHz	491.52 MHz	1.966 GHz	450 MHz	491.52 MHz	4.915 GHz	200 MHz	245.76 MHz	4.915 GHz

Table 105.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	5	7	10	15
TX_BB_FILTER	2	2	3	5
ADC_TUNER	1	1	1	1
FLASH_CAL	219	263	324	366
RX_TIA	63	84	105	127
ORX_TIA	54	65	76	87
LBRX_TIA	54	65	76	87
RX_DC_OFFSET	451	451	450	451
ORX_DC_OFFSET	451	899	899	898
LBRX_DC_OFFSET	7	14	14	14
LOOPBACK_RX_LO_DELAY	166	330	485	650
RX_QEC_INIT	787	1569	2354	3138
ORX_QEC_INIT	785	1564	2346	3126
INTERNAL_PATH_DELAY	1	2	2	3
TX_LO_LEAKAGE_INTERNAL	876	1749	2622	3495
TX_QEC_INIT	302	589	907	1188
Total Calibration time (ms)	4224	7654	10676	13652

Table 106. ADRV9025Init_StdUseCase83C_LinkSharing

Use Case	Tx Bandwidth	TX Input Rate	TX DAC Rate	ORx Bandwidth	ORX Output Rate	ORx ADC Rate	Rx Bandwidth	Rx Output Rate	Rx ADC Rate
UC83C_LS	337.5 MHz	368.64 MHz	1.475 GHz	337.5 MHz	368.64 MHz	3.686 GHz	200 MHz	368.64 MHz	3.686G Hz

Table 107.

Calibration	1 Channel	2 Channels	3 Channels	4 Channels
TX_DAC	2	5	8	13
TX_BB_FILTER	1	3	3	5
ADC_TUNER	1	1	1	1
FLASH_CAL	285	343	425	483
RX_TIA	71	84	99	112
ORX_TIA	71	85	99	115
LBRX_TIA	71	85	100	114
RX_DC_OFFSET	451	451	450	451
ORX_DC_OFFSET	450	899	898	899
LBRX_DC_OFFSET	7	14	14	14
LOOPBACK_RX_LO_DELAY	208	416	625	830
RX_QEC_INIT	547	1094	1638	2184
ORX_QEC_INIT	860	1717	2572	3437
INTERNAL_PATH_DELAY	1	2	3	4
TX_LO_LEAKAGE_INTERNAL	883	1764	2645	3525
TX_QEC_INIT	419	818	1202	1639
Total Calibration time (ms)	4328	7781	10782	13826

INITIALIZATION CALIBRATIONS TO BE RUN AFTER DEVICE INITIALIZATION

The device requires a few additional initialization calibrations to be run after the standard set because they require external signal routing. An External TXLOL initialization calibration is available where the observation point is moved from inside the device to the selected observation receiver input. In this case it is typically connected to a directional coupler after the PA in the antenna path. This configuration results in the best possible performance because the correction observation point is moved to the PA output. The calibration is run on each transmitter individually after the correct observation input path has been set. Similarly, CFR calibrations are also run separately and sequentially. Refer to Table 71 for the appropriate cal mask.

The following table addresses the typical times for these INIT calibrations. Note the CFR initialization calibration is mostly coefficient setting and, therefore, completes quickly.

Table 108.

INIT Calibration	Time
TX_LO_LEAKAGE_EXTERNAL 122.88 MHz IQ Rate	320 ms
TX_LO_LEAKAGE_EXTERNAL 245.76 MHz IQ Rate and higher	230 ms
CFR Initialization calibration	<1 ms

TRACKING CALIBRATION TIMING

Tracking calibrations are provided to maintain performance over the device operating conditions. The Arm processor periodically runs the enabled tracking calibrations according to the tracking calibration scheduler.

On the receive side, there are RX QEC, ORX QEC and on some devices RX HD2 tracking calibrations. These calibrations, when enabled constantly observe the RX (or ORX) spectrum and update the correction parameters as the computations are completed. They are triggered on a 7 ms schedule, but are essentially running continuously in the background whenever the channel is enabled.

The transmitter tracking calibrations include TXLOL, TXQEC and for some versions of the device also include CLGC tracking calibration. When the tracking calibrations are enabled on the transmitter, the spectrum is observed based upon the available observation path and correction parameters are applied to each transmitter as the computations are completed.

TXLOL tracking calibration runs on a 6 sec schedule. The samples are collected in batches of 20 µs duration for a total sample size of approximately 30 ms. TXQEC runs on a 30 sec schedule and also collects batches of 20 µs duration. TXQEC captures as many batches as necessary to obtain good correlator results. The time to finish can vary and can be from 100 µs up to 55 ms. However because they run in the background, the absolute time is not of concern to the user. Even though these cals run at fixed intervals (6 sec and 30 sec), any change in Tx attenuation causes both cals to be restarted. This is done to quickly correct any channel impairments.

The CLGC tracking calibration runs on a 1 sec schedule with similar batch sizes.

In the case of JESD204C, an additional tracking calibration is run to maintain the link parameters on a 60 sec schedule.

STREAM PROCESSOR AND SYSTEM CONTROL

A stream processor is a processor within the device tasked with performing a series of configuration tasks based on some event. Upon a request from the user, the stream processor performs a series of pre-defined actions which are loaded into the stream processor during device initialization. This processor takes full advantage of the speed of the internal register buses for efficient execution of commands. The stream processor can access and modify registers independently, avoiding the need for Arm interaction.

The stream processor executes streams, or series of tasks for the following:

- Tx1/Tx2/Tx3/Tx4 Enable/Disable
- Rx1/Rx2/Rx3/Rx4 Enable/Disable

ORx1/ORx2/ORx3/ORx4 Enable/Disable

The device flexibility is maintained by implementing the stream processors with similar flexibility. The stream processor image changes with configuration similar to how the initialization structures change with the selected profiles. For example, the stream that enables the receivers differs depending on the JESD configuration. For this reason, it is necessary to save a stream image for each device configuration. When the user saves the configuration files (.c) using the GUI, a stream binary image is generated automatically. Then use this stream file when initializing the device with the profile in question.

The following are examples of how the stream files can differ:

- The framer choices for ORx and Rx
- For link sharing purposes
- If floating point formatting is being used on Rx and ORx paths, the stream image can change

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

SLICE STREAM PROCESSORS

There are ten slice stream processors, one each for the four Tx, Rx data paths, and two for the ORx data paths. Note that even though there are four distinct RF front ends for the ORx, the device only supports two digital data paths—one shared between ORx1/ORx2 and another shared between ORx3/ORx4. These ORx data paths are also shared with the internal Tx channel loopback paths in order to facilitate data collection during the various Tx calibrations. The existence of individual slice stream processors for each data path enables true real-time parallel operation of all unique Tx and Rx data paths. The ORx data paths still need to be managed based on the various system operation use cases detailed in this section.

Since each slice stream processor is limited to some dedicated part of the transceiver, a given slice stream processor may only access the digital register sub maps corresponding to its specific functionality. For example, the Tx slice stream processors can only access the Tx digital sub maps.

Core Stream Processor

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

SYSTEM CONTROL

The signal paths within the device can be controlled by either the API or through pin control. In the case of API control, control relies on the SPI communication bus and its inherent unpredictable timing with respect to register access. For critical time alignment when powering on/off signal chains, pin control is recommended. The device defaults to API mode upon power up.

API Control

The following API function is used to control the data paths when the device is in API control:

adi_adrv9025_RxTxEnableSet

```
adi_adrv9025_RxTxEnableSet(adi_adrv9025_Device_t *device, uint32_t rxChannelMask, uint32_t
  txChannelMask)
```

Description

Controls and configures the Tx and Rx data paths.

Parameters

Table 109.

Parameter	Description
*device	Pointer to device structure.
rxChannelMask	The desired Rx/ORx signal chain to power up. See Table 110 for list of enums.
txChannelMask	The desired Tx signal chain to power up. See Table 111 for list of enums.

The enums are used (OR'ed) to create a value for the channel masks that determine the paths enabled when this API is called. The selected channels remain active until further instruction from this API command. It is important to note that if an ORx channel is enabled continuously and not returned to ADI_ADRV9025_RXOFF for any time, then the Tx tracking calibrations are able to function.

Table 110. adi_adrv9025_RxChannels_e Enum Definition

adi_adrv9025_RxChannels_e Enum	Enabled Channels
ADI_ADRV9025_RXOFF	No Rx or ORx channels enabled
ADI_ADRV9025_RX1	Rx1 Enabled
ADI_ADRV9025_RX2	Rx2 Enabled
ADI_ADRV9025_RX3	Rx3 Enabled
ADI_ADRV9025_RX4	Rx4 Enabled
ADI_ADRV9025_ORX1	ORx1 Enabled
ADI_ADRV9025_ORX2	ORx2 Enabled
ADI_ADRV9025_ORX3	ORx3 Enabled
ADI_ADRV9025_ORX4	ORx4 Enabled
ADI_ADRV9025_LB12	Tx1 or Tx2 internal loopback into ORx1/2 channel enabled
ADI_ADRV9025_LB34	Tx3 or Tx4 internal loopback into ORx3/4 channel enabled

Table 111. adi_adrv9025_TxChannels_e Enum Definition

adi_adrv9025_TxChannels_e Enum	Enabled Channels
ADI_ADRV9025_TXOFF	No Tx channels enabled
ADI_ADRV9025_TX1	Tx1 Enabled
ADI_ADRV9025_TX2	Tx2 Enabled
ADI_ADRV9025_TX3	Tx3 Enabled
ADI_ADRV9025_TX4	Tx4 Enabled
ADI_ADRV9025_TXALL	All Tx Enabled

Pin Control

The individual channels can also be controlled using a series of enable pins. In pin control mode, the Rx and Tx signal chains are controlled using dedicated pins, one RX_ENABLE pin per receiver and one TX_ENABLE pin per transmitter. When these pins are toggled high, the relevant signal chain is enabled. When these pins are toggled low, the relevant signal chain is disabled.

The ORx paths can be controlled in various modes, as indicated in Table 112.

Table 112. ORx Select Mechanisms

ORx Pin Mode	ORx Select Mechanism		
Single Channel 1-Pin Mode	In this mode a single channel is selected through the API (over SPI). ORX_CTRL_A is the enable/disable control pin, when high the selected ORx is enabled, when low, all ORx paths are disabled. Figure 57 illustrates Single Channel 1 Pin Mode. Note that ORx1 has been shown in this example, however any of ORx1 to ORx4 can be chosen.		
Single Channel 2-Pin Mode	In this mode, ORX_CTRL_A is the enable/disable control pin, when high the selected ORx is enabled, when low, all ORx paths are disabled. The ORX_CTRL_B pin is used for to select the ORx path, allowing the user to choose between two different ORx paths. These paths are predetermined through the API (over SPI), with one path selected when ORX_CTRL_B is high and another when it is low. This mode is illustrated in Figure 58. Note where ORx2 on and ORx3 on are shown. Any of the other ORx can be configured to turn on at this time instead of ORx2 or ORx3.		
Single Channel 3-Pin Mode	ORX_CTRL_A is the enable/disable control. ORx select is accomplished by ORX_CTRL_B and ORX_CTRL_C. The mapping of which path is selected is as follows:		
	ORX_CTRL_C	**ORX_CTRL_B**	**Path Selected**
	0	0	ORx1
	0	1	ORx2
	1	0	ORx3
	1	1	ORx4
	This mode is illustrated in Figure 59.		
Dual Channel 2-Pin Mode	In this mode ORX_CTRL_A and ORX_CTRL_C are the enable/disable control allowing the user to choose between two different ORx paths. These paths are predetermined through the API (over SPI). This mode is illustrated in Figure 60.		
Dual Channel 4-Pin Mode	In this mode In this mode ORX_CTRL_A and ORX_CTRL_C are the enable/disable control while ORX_CTRL_B and ORX_CTRL_D selects which channel is to be enabled allowing the user to choose between four different ORx paths. This mode is illustrated in Figure 61.		

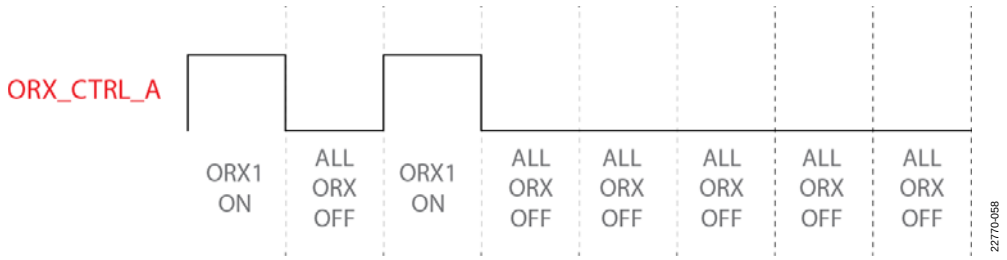


Figure 57. Single Channel 1-Pin Mode

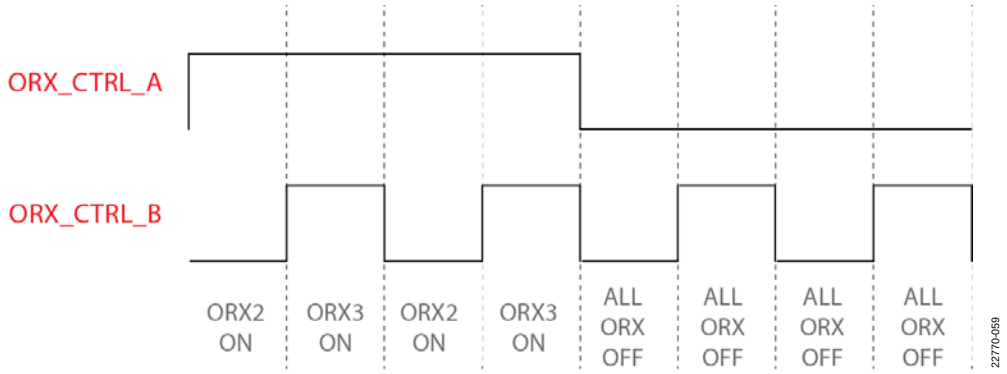


Figure 58. Single Channel 2-Pin Mode

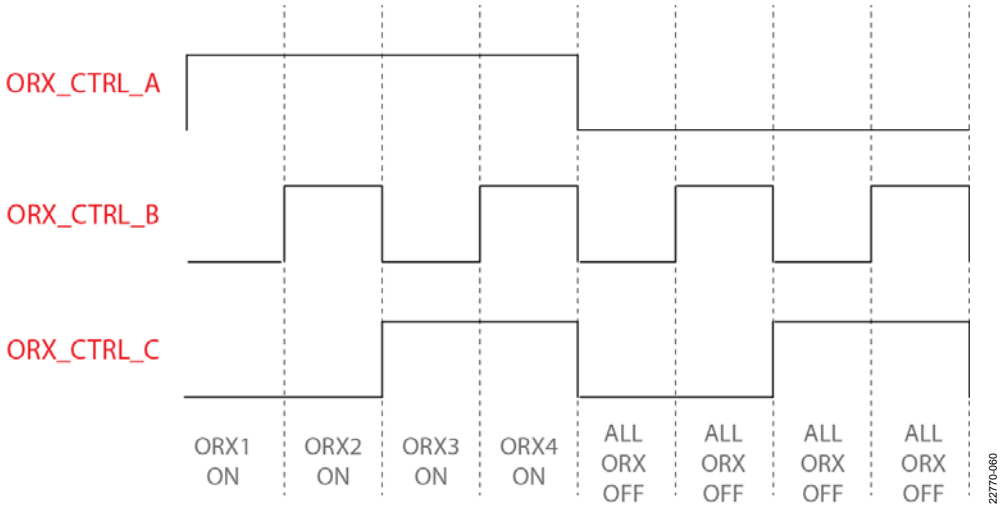


Figure 59. Single Channel 3-Pin Mode

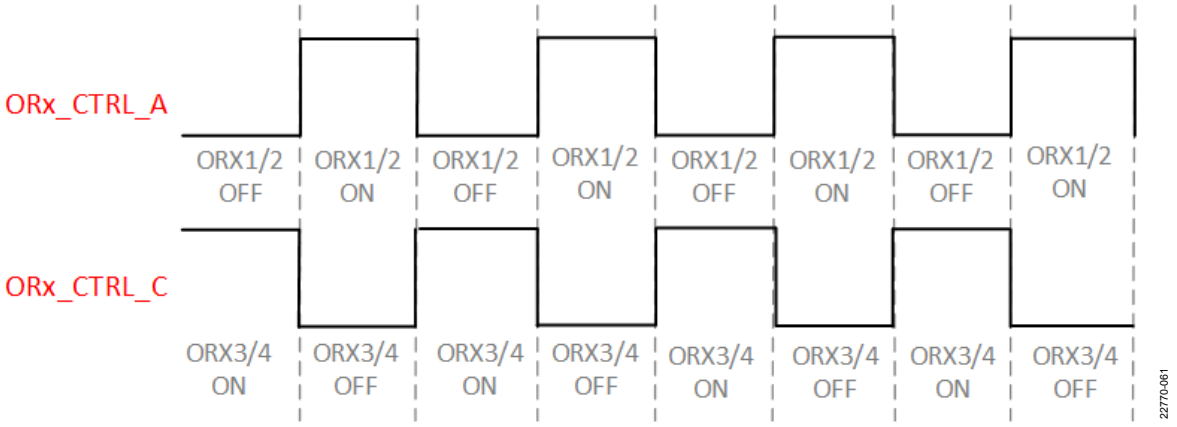


Figure 60. Dual Channel 2-Pin Mode

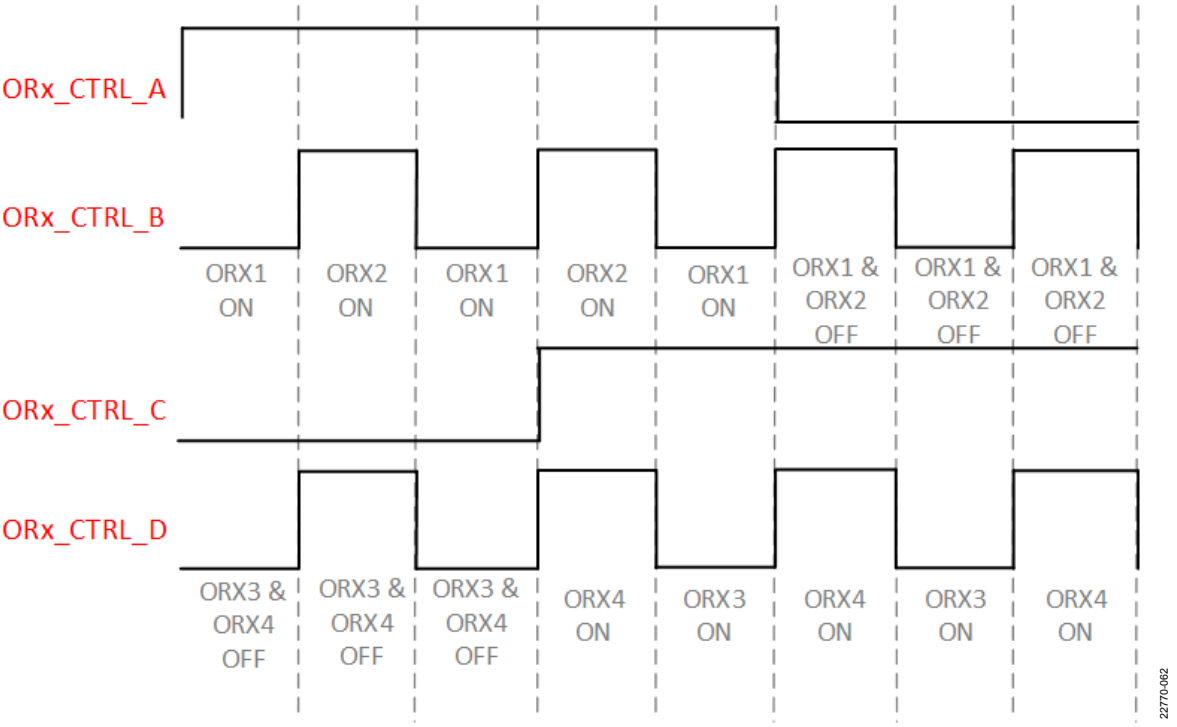


Figure 61. Dual Channel 4-Pin Mode

The user can set the channel control mode (API/Pin) with the post multi-chip sequence API function:

adi_adrv9025_PostMcsInit

```
adi_adrv9025_PostMcsInit(adi_adrv9025_Device_t *device, adi_adrv9025_PostMcsInit_t *utilityInit)
```

Description

Sets the channel control mode (API or Pin).

Parameters

Table 113.

Parameter	Description
*device	Pointer to device structure.
*utilityInit	Structure of type adi_adrv9025_PostMcsInit_t containing all relevant settings for the post MCS initialization routines.

This command contains a structure of type adi_adrv9025_ RadioctrlInit_t for setting up how the device is controlled. Inside this structure is the structure adi_adrv9025_ RadioCtrlModeCfg_t that contains the radio control mode configuration for the Tx, Rx and ORx channels.

This structure is defined in Table 114 and, depending on how the user configures this structure before the call to adi_adrv9025_PostMcsInit(), the part is configured in either pin or API mode.

Table 114. adi_adrv9025_RadioCtrlModeCfg_t Definition

Name	Description
txRadioCtrlModeCfg	Tx signal path enable mode configuration. See Table 115 for description.
rxRadioCtrlModeCfg	Rx signal path enable mode configuration. See Table 116 for description.
orxRadioCtrlModeCfg	ORx signal path enable mode configuration. See Table 117 for description.

Table 115. adi_adrv9025_TxRadioCtrlModeCfg_t Definition

Name	Value	Description
txEnableMode	A value of type adi_adrv9025_TxEnableMode_e, options are:	
	ADI_ADRV9025_TX_EN_SPI_MODE	Setting this mode selects API (or SPI) mode to control the Tx signal path
	ADI_ADRV9025_TX_EN_PIN_MODE	Setting this mode does not modify the currently set mode to control the Tx signal path
	ADI_ADRV9025_TX_EN_INVALID_MODE	Setting this mode selects no mode to control the Tx signal path
txChannelMask	Bit mask, one bit per channel ([D0] = Tx1, [D1] = Tx2, [D2] = Tx3, [D3] = Tx4). For example, to apply this to all four transmitters, txChannelMask is set to 15.	Set this to the Tx channels you want to configure with the selected txEnableMode

Table 116. adi_adrv9025_RxRadioCtrlModeCfg_t Definition

Name	Value	Description
rxEnableMode	A value of type adi_adrv9025_RxEnableMode_e, options are:	
	ADI_ADRV9025_RX_EN_SPI_MODE	Setting this mode selects API (or SPI) mode to control the Rx signal path
	ADI_ADRV9025_RX_EN_PIN_MODE	Setting this mode selects the Pin mode to control the Rx signal path
	ADI_ADRV9025_RX_EN_INVALID_MODE	Setting this mode does not modify the currently set mode to control the Rx signal path
rxChannelMask	Bit mask, one bit per channel ([D0] = Rx1, [D1] = Rx2, [D2] = Rx3, [D3] = Rx4). For example, to apply this to all four receivers, rxChannelMask is set to 15.	Set this to the Rx channels you want to configure with the selected rxEnableMode

Table 117. adi_adrv9025_ORxRadioCtrlModeCfg_t Definition

Name	Value	Description
orxEnableMode	A value of type adi_adrv9025_OrxEnableMode_e, options are:	
	ADI_ADRV9025_ORX_EN_SPI_MODE	Setting this mode selects API (or SPI) mode to control the ORx signal path
	ADI_ADRV9025_ORX_EN_SINGLE_CH_3PIN_MODE	Setting this mode puts the device in Single Channel 3 pin mode as described in Table 112
	ADI_ADRV9025_ORX_EN_SINGLE_CH_2PIN_MODE	Setting this mode puts the device in Single Channel 2 pin mode as described in Table 112
	ADI_ADRV9025_ORX_EN_SINGLE_CH_1PIN_MODE	Setting this mode puts the device in Single Channel 1 pin mode as described in Table 112
	ADI_ADRV9025_ORX_EN_DUAL_CH_4PIN_MODE	Setting this mode puts the device in Dual Channel 4 pin mode as described in Table 112
	ADI_ADRV9025_ORX_EN_DUAL_CH_2PIN_MODE	Setting this mode puts the device in Dual Channel 2 pin mode as described in Table 112
	ADI_ADRV9025_ORX_EN_INVALID_MODE	Setting this mode does not modify the currently set mode to control the ORx signal path
orxPinSelectSettlingDelay_armClkCycles	MinValue: 0, MaxValue: 16	Amount of time for the firmware to wait before sampling pins used for ORx selection; minimum is 2 Arm clock cycles, maximum is 18 Arm clock cycles
singleChannel1PinModeOrxSel	A value of type adi_adrv9025_ SingleChannelPinModeOrxSel_e, options are:	
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX1_FE	Selects ORx1 when in single channel 1 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX2_FE	Selects ORx2 when in single channel 1 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX3_FE	Selects ORx3 when in single channel 1 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX4_FE	Selects ORx4 when in single channel 1 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_INVALID_ORX_SEL	Does not modify the current mode of ORx when in single channel 1 pin ORx enable mode

Name	Value	Description
singleChannel2PinModeLowOrxSel	A value of type adi_adrv9025_SingleChannelPinModeOrxSel_e, options are:	
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX1_FE	Selects ORx1 when the ORX_CTRL_B pin is low in single channel 2 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX2_FE	Selects ORx2 when the ORX_CTRL_B pin is low in single channel 2 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX3_FE	Selects ORx3 when the ORX_CTRL_B pin is low in single channel 2 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX4_FE	Selects ORx4 when the ORX_CTRL_B pin is low in single channel 2 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_INVALID_ORX_SEL	Does not modify the current mode of ORx when the ORX_CTRL_B pin is low in single channel 2 pin ORx enable mode
singleChannel2PinModeHighOrxSel	A value of type adi_adrv9025_SingleChannelPinModeOrxSel_e, options are:	
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX1_FE	Selects ORx1 when the ORX_CTRL_B pin is high in single channel 2 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX2_FE	Selects ORx2 when the ORX_CTRL_B pin is high in single channel 2 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX3_FE	Selects ORx3 when the ORX_CTRL_B pin is high in single channel 2 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX4_FE	Selects ORx4 when the ORX_CTRL_B pin is high in single channel 2 pin ORx enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_INVALID_ORX_SEL	Does not modify the current mode of ORx when the ORX_CTRL_B pin is high in single channel 2 pin ORx enable mode
dualChannel2PinModeOrxSel	A value of type adi_adrv9025_DualChannelPinModeOrxSel_e, options are:	
	ADI_ADRV9025_DUAL_CH_PIN_MODE_ORX1_ORX3_SEL	Selects ORx1 and ORx3 when the part is in dual channel 2 pin mode
	ADI_ADRV9025_DUAL_CH_PIN_MODE_ORX1_ORX4_SEL	Selects ORx1 and ORx4 when the part is in dual channel 2 pin mode
	ADI_ADRV9025_DUAL_CH_PIN_MODE_ORX2_ORX3_SEL	Selects ORx2 and ORx3 when the part is in dual channel 2 pin mode
	ADI_ADRV9025_DUAL_CH_PIN_MODE_ORX2_ORX4_SEL	Selects ORx2 and ORx4 when the part is in dual channel 2 pin mode
	ADI_ADRV9025_DUAL_CH_PIN_MODE_INVALID_ORX_SEL	Does not modify the current mode of ORx when the part is in dual channel 2 pin mode

ADC Crossbar Control

There are two control modes for the ADC crossbar (Xbar) switches that feed the JESD serializers during link sharing mode. In the default mode, the Rx channel is connected to the serializer when the enable pin of the channel is active, and the ORx channel is connected to the serializer when the ORX_CTRL pins are driven to select the ORx channel. A second mode called ADC Xbar toggling exists that assigns the path control solely to the ORx channel control signals.

When ADC Xbar toggling is enabled, the ADC sample crossbar connects the desired ORx channel to the serializer when that channel is enabled using the ORX_CTRL pins. When the ORX_CTRL pins disable the ORx channel, the Rx channel is automatically connected to the serializer. This allows the system to keep the Rx channel enabled during link sharing operation and limit toggling to the ORX_CTRL inputs.

This feature can be enabled in a stream file by selecting ADC Xbar control in the TES Stream Settings window before generating the stream. The appropriate selection is shown in Figure 62.

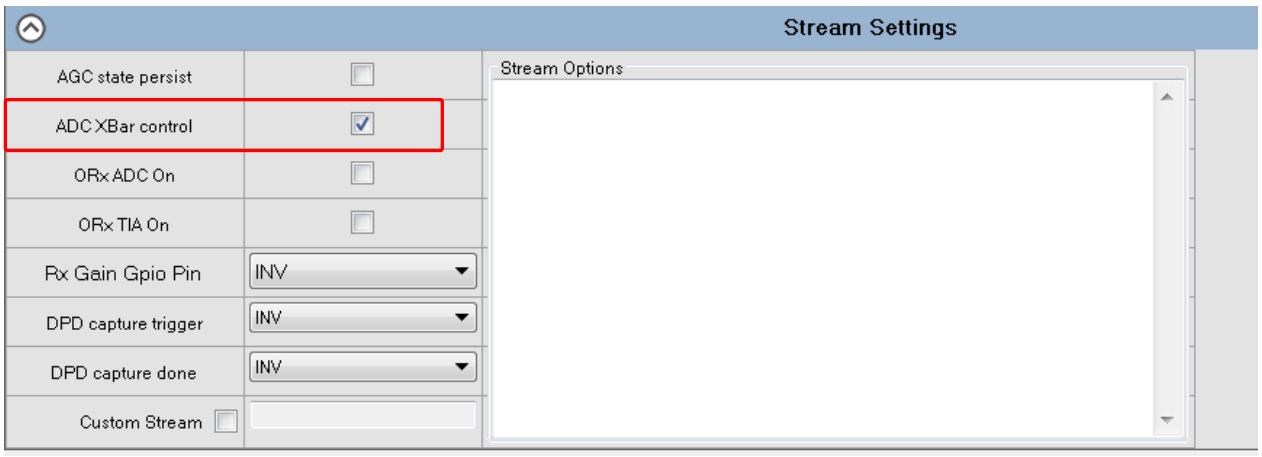


Figure 62. Stream Settings Window for Selecting ADC Xbar Control Mode

USE CASES

This section details example use cases for the device that show how the device is typically operated to ensure that calibrations are run.

4 Tx/4 Rx/2 ORx Input Use Case

In this use case the device is configured such that two Tx feed back into one ORx for each side of the device. The ORX_CTRL signals are configured in single channel 2 pin mode, with ORX_CTRL_A and ORX_CTRL_B used to determine which ORx is enabled and selected for the observation purposes of the user. ORX_CTRL_A is high at all the time, as an ORx path is always being used. When ORX_CTRL_A goes low, regardless of the state of ORX_CTRL_B, no ORx channel is enabled. ORX_CTRL_B determines which ORx channel the user is observing. For this example, ORx2 and ORx3 are being used. Note that ORx1 can be used in place of ORx2 or ORx4 in place of ORx3. At least one ORx from each side of the device must be used; that is, either ORx1 or ORx2 must be used for calibrations on Tx1 and Tx2. The ORx from one side of the device cannot be used to calibrate the Tx on the other side of the device. That is, ORx1 or ORx2 cannot be used to calibrate Tx3 and Tx4.

The ORX_TX_SEL and ORX2_TX_EN signals are used to indicate the external routing of the feedback paths, allowing the arm to know which transmitter is being looped back to which observation receiver at a given time and whether a calibration may be run or not. As a transmitter is always available at an observation receiver on its own side of the chip, ORX2_TX_EN and ORX3_TX_EN are defaulted high over SPI as they remain fixed. ORX2_TX_SEL and ORX3_TX_SEL indicate the external routing of a transmitter to a given observation receiver. When ORX2_TX_SEL is low, it indicates the Tx1 path is routed back to ORx2. Likewise, when ORX2_TX_SEL is high, this indicates PA2 is available at the ORx2 input. This is similar for ORX3_TX_SEL, such that when this signal is low it indicates PA3 is available at the ORx3 input, and, likewise, when it is high PA4 is available at the observation receiver input.

For this use case, internal calibrations can be performed on the inactive ORx channel while an external calibration is running on the active channel. In the first time slot of the timing diagram in Figure 64, it can be seen that ORx2 is enabled by the user. PA1 and PA3 have been routed back to ORx2 and ORx3, respectively. The device can perform an external LOL tracking calibration for Tx3 via ORx3, or a QEC tracking calibration on Tx3 or Tx4, while the system is performing calculations for PA1. The QEC tracking calibration is performed via an internal routing between each Tx channel and its corresponding ORx channel. The external LOL tracking calibration, however, can only be performed when an external loopback path is available. In the second time slot in Figure 64, ORx2 is still enabled for the user

with PA2 and PA4 being made available to ORx2 and ORx3. The system can perform calculations for PA2 via ORx2 while performing a QEC tracking calibrations on Tx3 or Tx4, or an external LOL tracking calibration on Tx4.

Note that calibrations are not automatically run in a designated time slot. The Arm scheduler of the device schedules which calibrations run at any given time. For more information on the scheduler, refer to the Arm Processor and Device Calibrations section of this user guide. In addition, the same JESD link can be used for ORx2 and ORx3 in the scenario above because only one ORx is used at any given time.

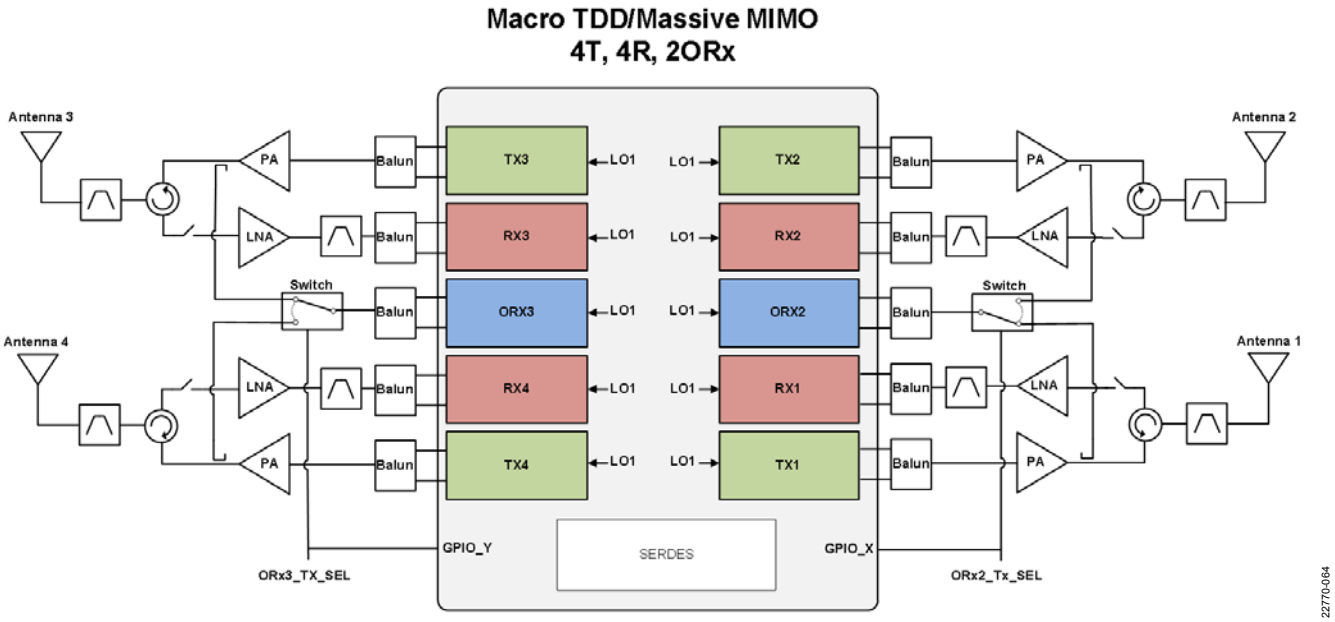


Figure 63. 4 Tx, 4 Rx, 2 ORx Configuration

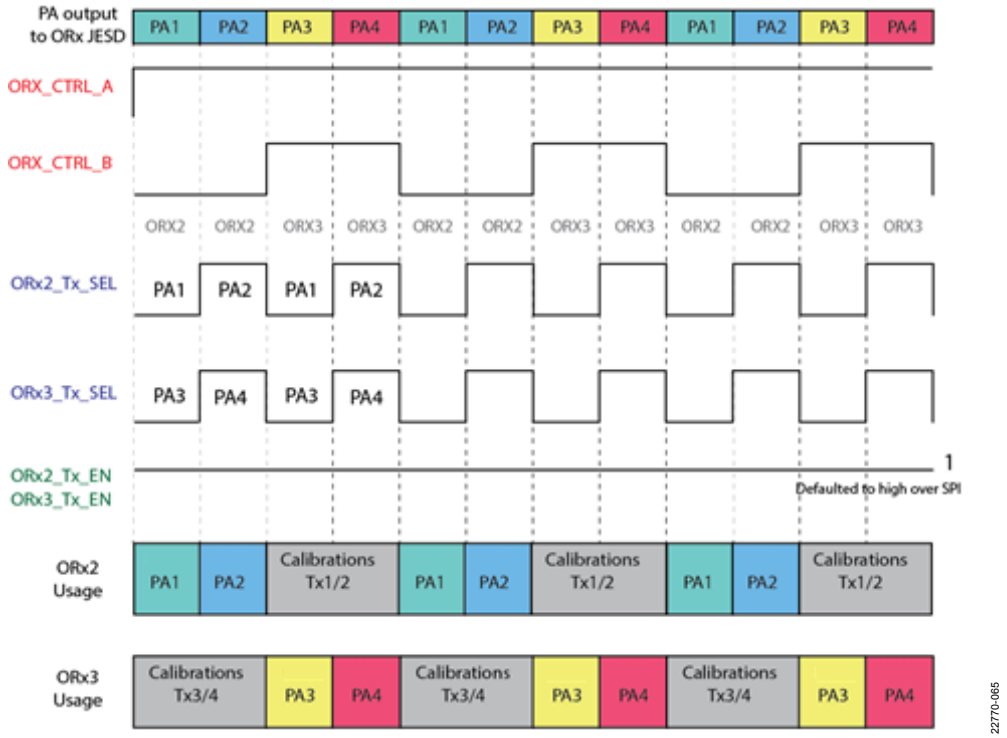


Figure 64. ORx Enable and Tx Select Signals: 4 Tx, 4 Rx, 2 ORx Configuration

4 Tx/4 Rx/4 ORx Input Use Case

In this use case, each Tx is routed back to its own ORx input. The device is configured in Single Channel 3 Pin Mode for this use case. ORX_CTRL_A is principally high all the time, meaning an ORx path is always being used. ORX_CTRL_B and ORX_CTRL_C determine what ORx channel is enabled and selected for the observation purposes of the user. Refer to Table 118 for how each ORx is selected via the two ORx select signals.

Table 118. ORx Select Logic

Logic of ORX_CTRL_C (MSB) and ORX_CTRL_B (LSB)	ORx Selected
00	ORx1
01	ORx2
10	ORx3
11	ORx4

Because each Tx is routed back to a separate ORx input, there is no need for external switching in this use case and each of the ORX_TX_SEL signals can be set to a default value via the SPI. ORX2_TX_SEL and ORX4_TX_SEL are both defaulted to a high state, and ORX1_TX_SEL and ORX3_TX_SEL are both defaulted to a low state. ORX1_TX_EN, ORX2_TX_EN, ORX3_TX_EN, and ORX4_TX_EN are all defaulted to a high state.

The first time slot in the timing diagram in Figure 66 shows that the ORX_CTRL_B and ORX_CTRL_C signals are set to a 00 value, enabling ORx1 to the user. In this scenario, calculations can be performed on PA1. ORx2 is on this side of the chip, so the device cannot use it for any calibrations during this time slot. The other side of the chip can be utilized via ORx3/ORx4 for calibrations. Note that calibrations can be performed on either Tx3 or Tx4 and it is up to the scheduler to determine what calibration for which Tx is to run in a given time slot. Because each Tx is permanently routed back to its own ORx, the external path always exists for external LOL tracking to run.

Because only one ORx is used at any given time, the same JESD link for ORx1, ORx2, ORx3 and ORx4 can be used in this scenario.

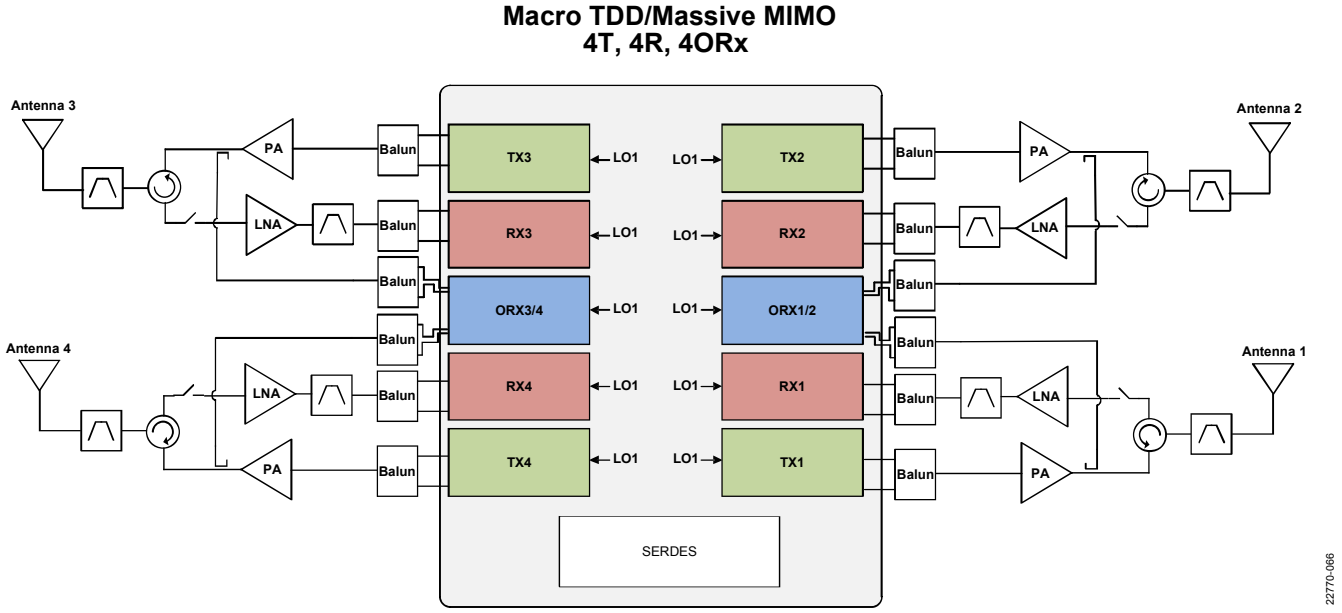


Figure 65. 4 Tx, 4 Rx, 4 ORx Configuration

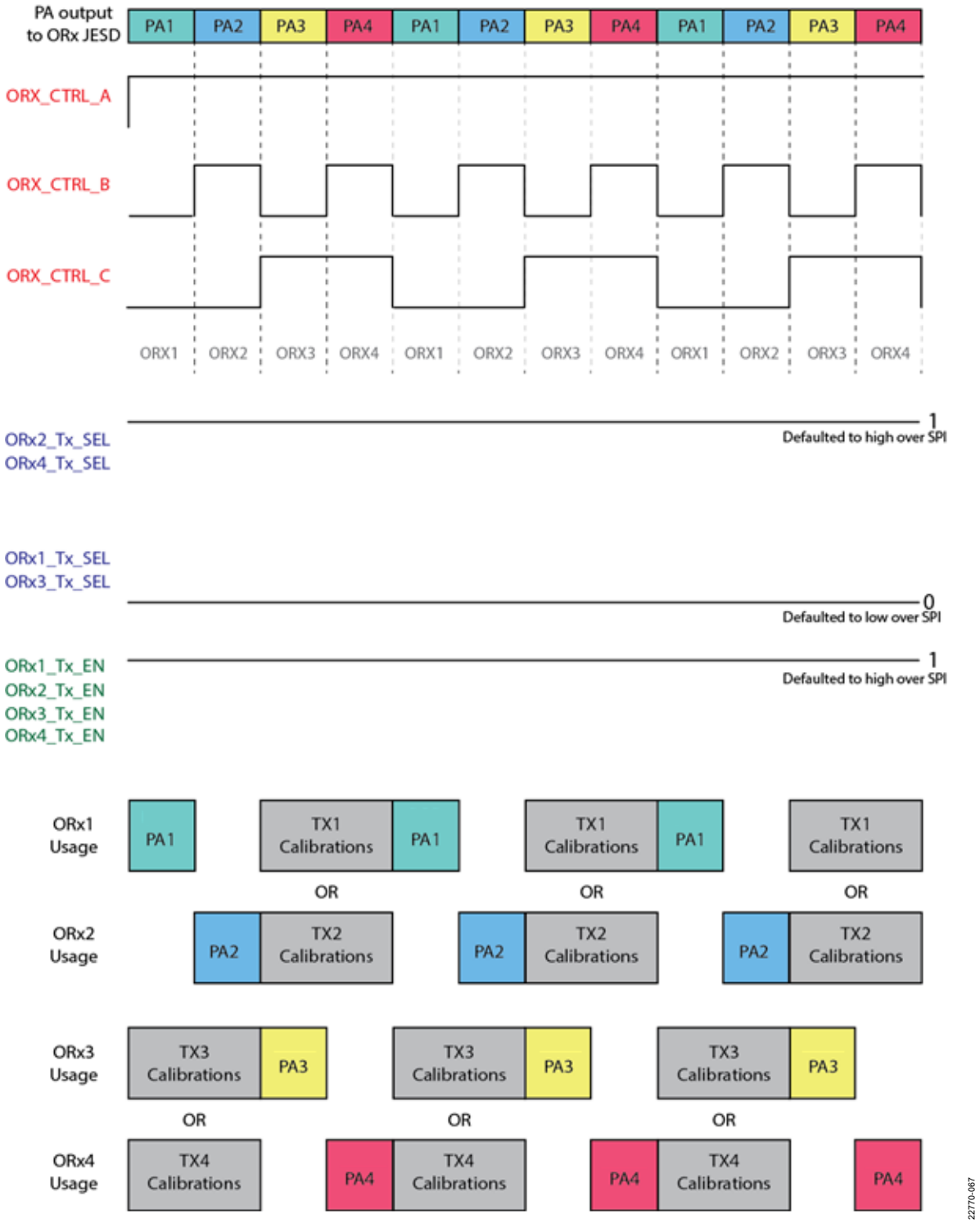


Figure 66. ORx Enable and Tx Select Signals: 4 Tx, 4 Rx, 4 ORx Configuration

4 Tx/4 Rx/2 ORx input – Single Point of Feedback from 4 Tx to ORx Use Case

This use case shows an example where all the ORx paths are shared through one common feedback point. As there are two sides to the device from a calibration perspective, the user must route Tx1 and Tx2 to either ORx1 or ORx2. Similarly, Tx3 and Tx4 need a path back to ORx3 or ORx4 for the purpose of calibrations. To allow calibrations to run in parallel with PA observation captures, the opposite side of the device to that required for calibrations is used to capture observation data; that is, if Tx2 is being fed back through this single feedback point, then ORx2 is used for device calibrations while ORx3 can be used to capture observation data. A resistive splitter is used to route the signal to both sides of the device.

For this use case, we use single channel 2 pin mode. ORX_CTRL_A is set high all the time because an ORx path is always being used. ORX_CTRL_B selects which ORx the user is observing in a given time slot. For this example, ORx2 and ORx3 are used. ORx3 is selected for observation when ORX_CTRL_B is high and ORx2 is selected for observation when ORX_CTRL_B is low.

ORX2_TX_SEL and ORX2_TX_EN together tell the Arm which external path (either Tx1 or Tx2) is routed back to ORx2. When ORX2_TX_SEL and ORX2_TX_EN are both high, the PA2 path is routed back to both ORx2 and ORx3. When ORX2_TX_SEL is low and ORX2_TX_EN is high, the PA1 path is routed back to both ORx2 and ORx3. When ORX2_TX_EN is low, this tells the device that there is no external feedback path between this ORx input and a Tx on the same side of the device. In this scenario, the external LOL calibration cannot be performed. Likewise, the ORX3_TX_SEL and ORX3_TX_EN perform the same function for the Tx paths on the other side of the chip. If ORX3_TX_SEL is low and ORX3_TX_EN is high, the PA3 path is routed back to both ORx2 and ORx3. If ORX3_TX_SEL and ORX3_TX_EN are both high, the PA4 path is routed back to both ORx2 and ORx3. Finally, if ORX3_TX_EN is low, this tells the device that there is no external feedback path between this ORx input and a Tx on the same side of the device. In this scenario, the external LOL calibration cannot be performed.

Unlike the other use cases previously described, the device can perform both calculations on a given PA and calibrations with the other ORx input for the same side of the chip. While the Tx calibrations have to be performed with an ORx from the same side of the chip, the PA calculations do not have that constraint. The first time slot in Figure 68 shows that calculations are being performed on PA1 via ORx3 while calibrations are performed on Tx1/Tx2 via ORx2. Note at the first time slot in this diagram that the external LOL calibration can be performed for Tx1 as the path is routed back to ORx2. In time slot two, the external LOL calibration can be performed for Tx2 but not Tx1 as there is no external feedback path. QEC calibrations are performed though an internal feedback path and do not require an external feedback path to run. It is up to the Arm scheduler to determine what calibration is due to run in any given slot. The same JESD link can be used for ORx2 and ORx3 in this scenario because only one ORx is used at any given time.

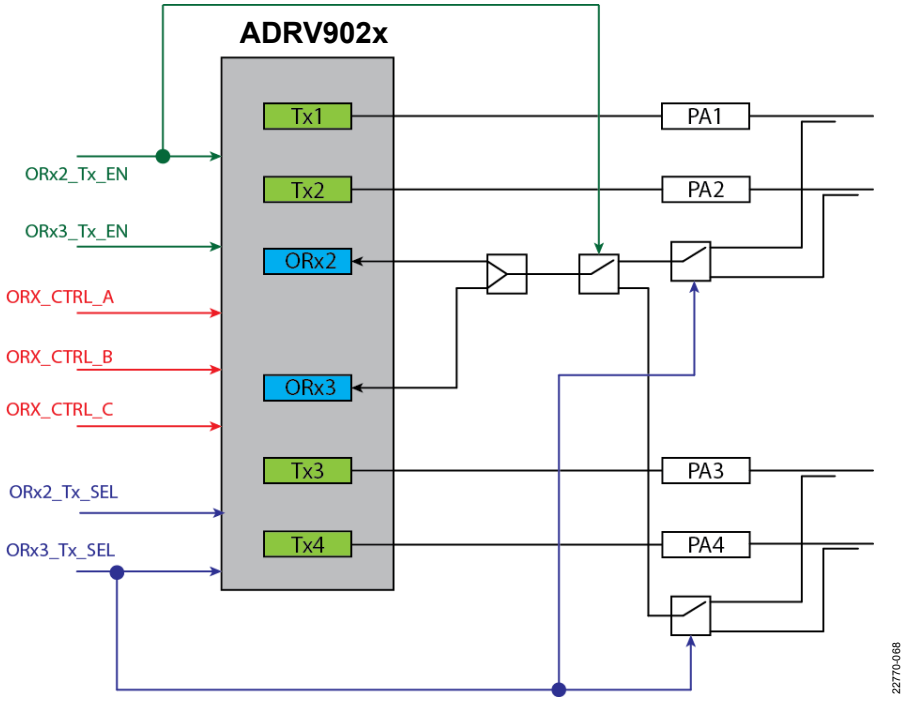


Figure 67. Observation Channel Routing: 4 Tx to 2 ORx Channels

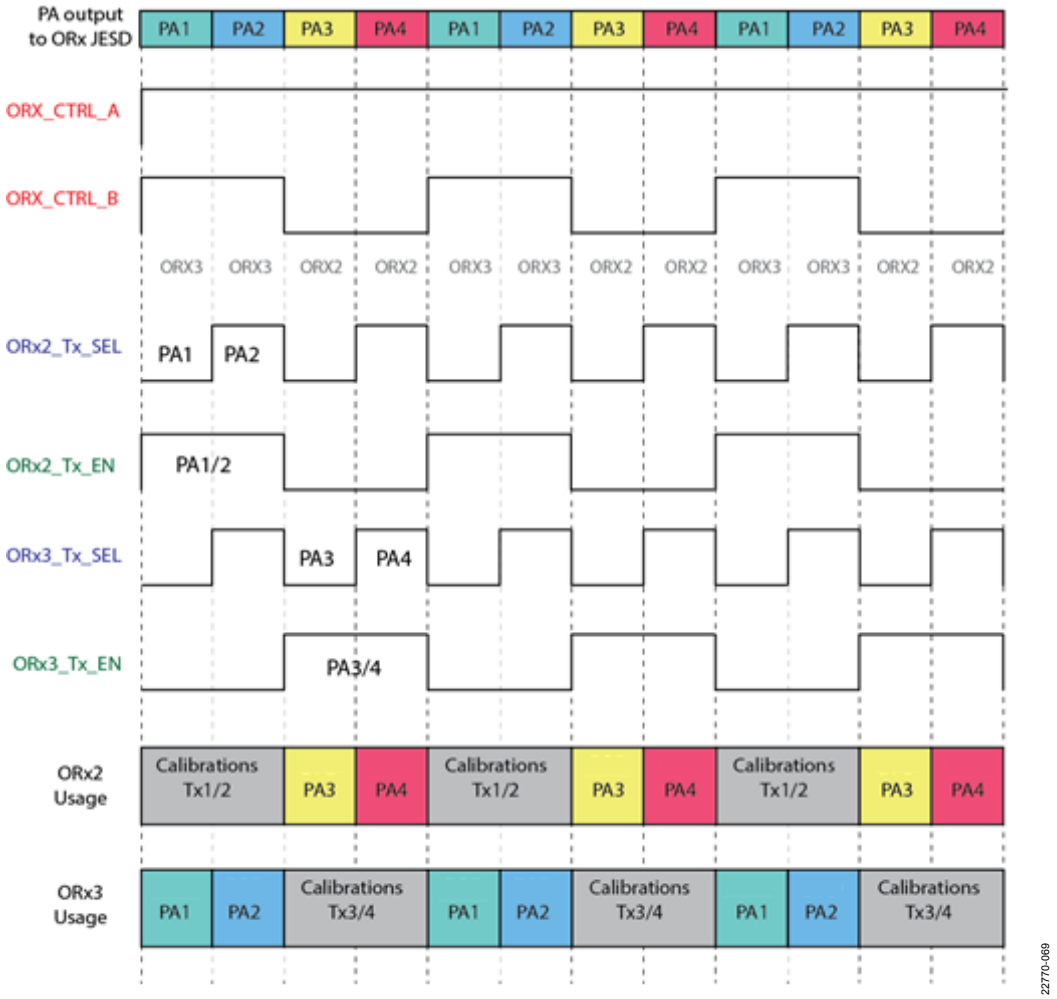


Figure 68. ORx Enable and Tx Select Signals: 4 Tx to 2 ORx Multiplexed Configuration

TRANSMITTER OVERVIEW AND PATH CONTROL

The ADRV9026 uses an accurate and efficient method of transmit power control (Tx attenuation control) that involves a minimum of interaction with the baseband processor. The power control in the transmit chain is implemented with two variable attenuations, one in the digital domain and one in the analog domain. Furthermore, the maximum output level of the transmitter can be adjusted between two levels, allowing a tradeoff between linearity and LOL performance.

There are three different modes available to control the attenuation setting of the transmitter. The attenuation can be set immediately via the API, incremented or decremented using GPIO pins to trigger the increment or decrement, or set through a SPI2 mode that enables real time operation using a GPIO pin. The choice of attenuation mode is set by attenMode.

The attenuation is controlled via a lookup table, which is programmed into the product during initialization. The lookup table maps a desired value in dB to the appropriate analog and digital attenuation settings to be applied in the data path. The default table provides a range of 0 dB to 41.95 dB of attenuation, with a step size of 0.05 dB, resulting in 840 available attenuation settings.

The Tx path allows the maximum output of the DAC to be increased by 3 dB adjusting the parameter dacFullScale. This results in the baseband signal (the desired signal) increasing by 3 dB while RF output components (such as LO leakage) remain unchanged, giving a net improvement of 3 dB in LOL performance. There is a reduction in linearity performance in this mode. Therefore, the setting is a trade-off based on the system requirements of the user.

The Tx datapath can be configured to automatically ramp the attenuation to the maximum level under certain conditions, such as the JESD link dropping (rampJesdDfrm) or the Tx PLL unlocking (disTxDataIfPllUnlock), to prevent spurious transmission in the event of these types of system errors.

Test tones may be generated digitally in the Tx baseband path. This function is useful for testing/debugging before the JESD link has been established. The frequency can be set from −TxInputRate/2 to +TxInputRate/2. The Tx attenuation is manually overridden when this function is enabled. When test tones are selected as the Tx input, the analog portion of the Tx attenuation is set to 0 dB (max output power), and the digital portion is set by txToneGain.

API COMMANDS

Several API commands are available to adjust the Tx paths after initialization and during normal operation. The following descriptions detail these commands and how they are used.

adi_adrv9025_TxAttenCfgSet

```
adi_adrv9025_TxAttenCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_TxAttenCfg_t
  txAttenCfg[], uint8_t attenCfgs);
```

Description

Configures Tx power control.

Parameters

Table 119.

Parameter	Description
*device	Pointer to device structure.
txAttenCfg[]	An array of structures of type adi_adrv9025_TxAttenCfg_t detailed in Table 120.
attenCfgs	The number of configurations passed in the array.

Table 120. adi_adrv9025_TxAttenCfg_t Parameters

Parameter	Comments	
txChannelMask	This selects the channels upon which the API acts. It is a bit mask with each bit corresponding to a channel. The desired mask can be generated by OR'ing the desired channel enums as listed below. Data type: uint32_t	
	Parameter	**Tx Channel**
	ADI_ADRV9025_TXOFF	No Tx channels selected
	ADI_ADRV9025_TX1	Tx1 channel selected
	ADI_ADRV9025_TX2	Tx2 channel selected
	ADI_ADRV9025_TX3	Tx3 channel selected
	ADI_ADRV9025_TX4	Tx4 channel selected
	ADI_ADRV9025_TXALL	All Tx channels selected
txAttenStepSize	This parameter sets the attenuation step size; Data type: adi_adrv9025_TxAttenStepSize_e	
	Parameter	**Step Size (dB)**
	ADI_ADRV9025_TXATTEN_0P05_DB	0.05
	ADI_ADRV9025_TXATTEN_0P1_DB	0.1
	ADI_ADRV9025_TXATTEN_0P2_DB	0.2
	ADI_ADRV9025_TXATTEN_0P4_DB	0.4
disTxDataIfPllUnlock	Option to ramp Transmit attenuation to max if the RFPLL unlocks ; Data type: adi_adrv9025_TxDataIfUnlock_e	
	Parameter	**Action**
	ADI_ADRV9025_TXUNLOCK_TX_NOT_DISABLED	Do not alter Tx attenuation in an unlock event.
	ADI_ADRV9025_TXUNLOCK_TX_RAMP_DOWN_TO_MIN_ATTEN	Ramp Tx attenuation to maximum in an unlock event.
rampJesdDfrm	Ramp up attenuation when a deframer link unlocks. Note this field is not being used actively. If user enables at least one deframer event with adi_adrv9025_PaPllDfrmEventRampDownEnableSet, the gain ramp down on the deframer event is automatically enabled. Data type: adi_adrv9025_TxDataIfUnlock_e	
attenMode	Selects the Tx attenuation mode; Data type: adi_adrv9025_TxAttenMode_e	
	Parameter	**Mode**
	ADI_ADRV9025_TXATTEN_BYPASS_MODE	Tx attenuation mode Bypass: zero total attenuation
	ADI_ADRV9025_TXATTEN_SPI_MODE	Tx attenuation set by 10-bit index programmed over SPI
	ADI_ADRV9025_TXATTEN_GPIO_MODE	Tx attenuation is incremented/decremented using GPIO pins
	ADI_ADRV9025_TXATTEN_SPI2_MODE	Attenuation is controlled using the SPI2 mode
dacFullScale	Sets the full scale of the Tx DAC; Data type: adi_adrv9025_DacFullScale_e	
	Parameter	**Description**
	ADI_ADRV9025_TX_DACFS_0DB	No Full Scale Boost
	ADI_ADRV9025_TX_DACFS_3DB	Full scale boost = 3 dB

adi_adrv9025_TxAttenCfgGet

```
adi_adrv9025_TxAttenCfgGet(adi_adrv9025_Device_t* device, adi_adrv9025_TxChannels_e txChannel,
  adi_adrv9025_TxAttenCfg_t *txAttenCfg)
```

Description

Reads Tx power control configuration one channel at a time.

Parameters

Table 121.

Parameter	Description
*device	Pointer to device structure.
txChannel	The Tx channel to be read back using an enum as described in Table 120.
*txAttenCfgs	The pointer to the readback structure of the queried Tx channel as defined in Table 120.

adi_adrv9025_TxAttenSet

```
adi_adrv9025_TxAttenSet(adi_adrv9025_Device_t* device, adi_adrv9025_TxAtten_t txAttenuation[],
  uint8_t numTxAttenConfigs);
```

Description

Sets Tx attenuation when Tx attenuation mode is set to ADI_ADRV9025_TXATTEN_SPI_MODE.

Parameters

Table 122.

Parameter	Description
*device	Pointer to device structure.
txAttenuation[]	An array of structures of type adi_adrv9025_TxAtten_t detailed in Table 123.
numTxAttenConfigs	The number of configurations passed in the array.

Table 123. adi_adrv9025_TxAtten_t Parameters

Parameter	Comments	
txChannelMask	This selects the channels upon which the API acts. It is a bit mask with each bit corresponding to a channel. The desired mask can be generated by ORing the desired channel enums as listed below. Data type: uint32_t	
	Parameter	**Tx Channel**
	ADI_ADRV9025_TXOFF	No Tx channels selected
	ADI_ADRV9025_TX1	Tx1 channel selected
	ADI_ADRV9025_TX2	Tx2 channel selected
	ADI_ADRV9025_TX3	Tx3 channel selected
	ADI_ADRV9025_TX4	Tx4 channel selected
	ADI_ADRV9025_TXALL	All Tx channels selected
txAttenuation_mdB	This parameter specifies the attenuation in mdB. Data type: uint16_t	

adi_adrv9025_TxAttenGet

```
adi_adrv9025_TxAttenGet(adi_adrv9025_Device_t* device, adi_adrv9025_TxChannels_e txChannel,
  adi_adrv9025_TxAtten_t* txAttenuation)
```

Description

Reads Tx attenuation when the Tx attenuation mode is set to ADI_ADRV9025_TXATTEN_SPI_MODE or ADI_ADRV9025_TXATTEN_GPIO_MODE.

Parameters

Table 124.

Parameter	Description
*device	Pointer to device structure.
txChannel	The Tx channel to be read back using an enum as described in Table 123.
*txAttenuation	Pointer to the readback structure of the queried Tx channel as defined in Table 123.

adi_adrv9025_TxAttenModeSet

```
adi_adrv9025_TxAttenModeSet(adi_adrv9025_Device_t* device, adi_adrv9025_TxChannels_e txChannel,
  adi_adrv9025_TxAttenMode_e *txAttenMode);
```

Description

Sets the Tx attenuation mode independent of the init structure.

Parameters

Table 125.

Parameter	Description
*device	Pointer to device structure.
txChannel	Tx channel upon which the API acts as described in Table 126.
*txAttenMode	Pointer to the desired mode of attenuation using an enum as described in Table 126.

Table 126. adi_adrv9025_TxAttenModeSet Parameters

Parameter	Comments	
txChannelMask	This selects the channels upon which the API acts. It is a bit mask with each bit corresponding to a channel. The desired mask can be generated by ORing the desired channel enums as listed below. Data type: uint32_t	
	Parameter	**Tx Channel**
	ADI_ADRV9025_TXOFF	No Tx channels selected
	ADI_ADRV9025_TX1	Tx1 channel selected
	ADI_ADRV9025_TX2	Tx2 channel selected
	ADI_ADRV9025_TX3	Tx3 channel selected
	ADI_ADRV9025_TX4	Tx4 channel selected
	ADI_ADRV9025_TXALL	All Tx channels selected
txAttenMode	Selects the Tx attenuation mode; Data type: adi_adrv9025_TxAttenMode_e	
	Parameter	**Mode**
	ADI_ADRV9025_TXATTEN_BYPASS_MODE	Tx attenuation mode Bypass: zero total attenuation
	ADI_ADRV9025_TXATTEN_SPI_MODE	Tx attenuation set by 10-bit index programmed over SPI
	ADI_ADRV9025_TXATTEN_GPIO_MODE	Tx attenuation is incremented/decremented using GPIO pins
	ADI_ADRV9025_TXATTEN_SPI2_MODE	Attenuation is controlled using the SPI2 mode

Adi_adrv9025_TxTestToneSet

```
adi_adrv9025_TxTestToneSet(adi_adrv9025_Device_t* device, adi_adrv9025_TxTestToneCfg_t
  txNcoTestToneCfg[], uint8_t arraySize);
```

Description

Generates test tones in the Tx baseband path.

Parameters

Table 127.

Parameter	Description
*device	Pointer to device structure.
txNcoTestToneCfg[]	An array of structures of type adi_adrv9025_TxAttenCfg_t as detailed in Table 128.
arraySize	The number of configurations passed in the array.

Table 128. adi_adrv9025_TxTestToneCfg_t Parameters

Parameter	Comments	
txChannelMask	This selects the channels upon which the API acts. It is a bit mask with each bit corresponding to a channel. The desired mask can be generated by OR'ing the desired channel enums as listed below. Data type: uint_8	
	Parameter	**Tx channel**
	ADI_ADRV9025_TXOFF	No Tx channels selected
	ADI_ADRV9025_TX1	Tx1 channel selected
	ADI_ADRV9025_TX2	Tx2 channel selected
	ADI_ADRV9025_TX3	Tx3 channel selected
	ADI_ADRV9025_TX4	Tx4 channel selected
	ADI_ADRV9025_TXALL	All Tx channels selected
enable	Sets whether the test tones are enabled or disabled; Data type: uint_8	
	Parameter	**Mode**
	0	Test tones disabled
	1	Test tones enabled
txToneFreq_Hz	Sets the frequency of the test tone in Hz. Range is ±245.76 MHz. Data type: uint_32	
txToneGain	Sets the amplitude of the test tone in dBFS; Data type: adi_adrv9025_TxNcoGain_e	
	Parameter	**Gain**
	ADI_ADRV9025_TX_NCO_NEG18_DB	−18 dBFS test tone
	ADI_ADRV9025_TX_NCO_NEG12_DB	−12 dBFS test tone
	ADI_ADRV9025_TX_NCO_NEG6_DB	−6 dBFS test tone
	ADI_ADRV9025_TX_NCO_0_DB	0 dBFS test tone

DAC FULL-SCALE FUNCTION (DAC BOOST)

The DAC Full Scale function is an analog 3 dB gain stage that can be used primarily to help systems that have marginal system performance to the TX LO leakage (Tx LOL) specification. As shown in the figure, the gain is realized in the DAC output but before the Tx predistortion (LPF) filters which is where the majority of the flicker noise observed on Tx LOL is added to the signal chain. When enabled, it provides an additional 3 dB of signal gain. By increasing the signal level by 3 dB it provides an additional 3 dB of separation to the noise/Tx LOL. The 3 dB gain factor is achieved by shifting the bias point of the DAC.

Increasing the signal level through the chain can potentially result in reduced linearity and spurious. The user is cautioned when transmitting signals with very low PAR for these reasons. When the mode is enabled, signal PAR does not allow the DAC to be driven above −3 dBFS. Normally, however for LTE signals or similar with PAR of about 12 dB, the signal chain has enough headroom for minimal performance impact. The measurements that follow show this.

Since the Tx signal level is increased when enabled, the configuration must be done prior to device initialization so that the internal calibrations see the appropriate gain through the signal chain. It is not possible to change it after the part has been configured.

The Tx predistortion (LPF) filters are the main contributors of flicker noise to the Tx signal chain. Because the gain occurs before them, the amount of Tx LOL emitted from the device is not changed by enabling the 3 dB mode. The transmitter Tx attenuators follow the filters in the signal chain. For this reason Tx LOL reduces at the output with each attenuator step, dB for dB. The Tx LOL measurement for both enabled and disabled modes along with the margin gained when the function is enabled is presented in Figure 69.

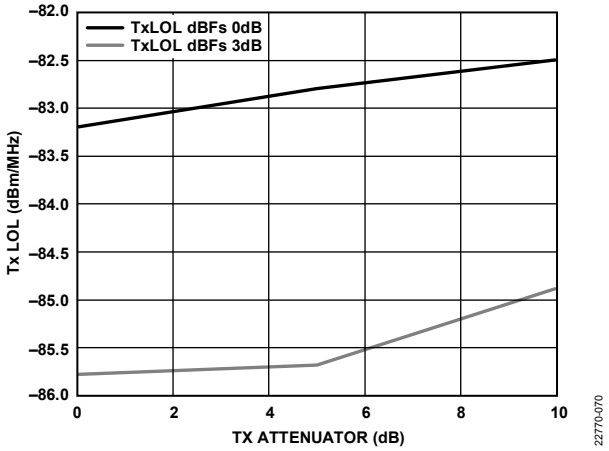


Figure 69. TxLO Leakage with dacFullScale Enabled/Disabled

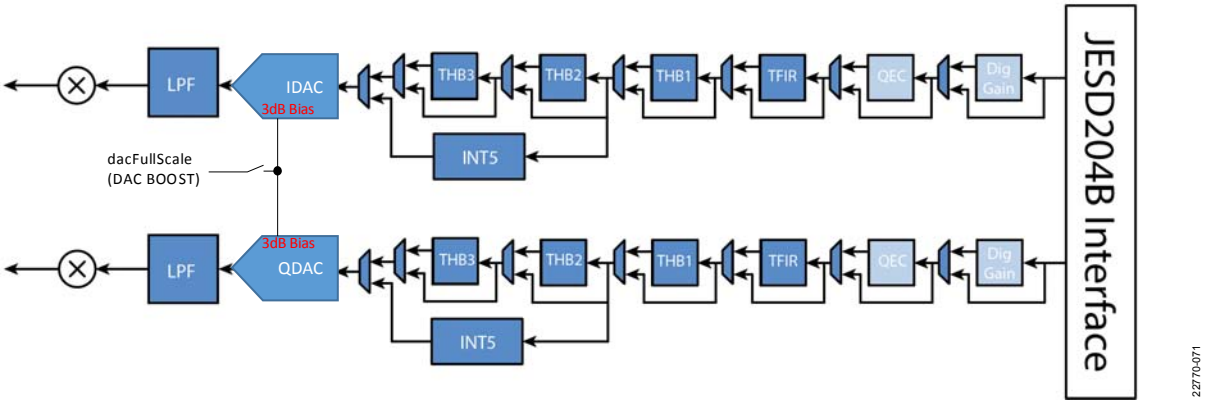


Figure 70. TX Datapath with dacFullScale Function

The Tx LOL specification is defined in terms of dBFS and is measured in a 1 MHz bandwidth. Tx LOL in dBFS is determined by applying a known signal level (-12 dBFS tone in this case) then measuring the resulting output power to determine 0 dBFs. Then the difference in power levels results in Tx LOL (dBFS).

The improvement shown in Table 85 is close to the 3 dB gain added. There is a small amount of variability due to the effects of flicker noise and the stability/accuracy of measuring the noise.

To meet data sheet performance levels, adjust the input signal to compensate for the 3 dB increase so that the resulting power levels are equivalent and therefore the OIP3 is equivalent as well. This is presented in the second OIP3 column in Table 130. In general, the OIP3 is only slightly affected by enabling the Boost with the same input tone levels (both tones level= −15 dBFS as per the data sheet). Typical performance shown on the data sheet is approximately 30 dBm, and measurements of the device in both modes are consistent. Impact on linearity is shown in Table 130.

EVM was measured with Boost in 0 dB mode and with Boost in 3 dB mode. There is no significant impact to EVM as a result of enabling the 3 dB mode. The impact on EVM is presented in Table 131.

System requirement: The desired DAC Boost mode must be configured prior to device initialization. The Tx signal level is increased which impacts internal calibrations, therefore it is not able to be modified during device operation.

Table 129. dacFullScale TxLOL and Tx Output Power Comparison 0dB Mode and 3dB Mode

DAC Full Scale Setting	TX Attn (dB)	TxLOL (dBm/MHz)	Tone Power (dBm)	0 dBFS in dBm	Tx LOL (dBFS)	Improvement (dB)
dacFullScale 0dB Mode	0	−77.2	−6	6	−83.2	
	5	−81.6	−10.8	1.2	−82.8	
	10	−86.5	−16	−4	−82.5	
dacFullScale 3dB Mode	0	−76.6	−2.8	9.2	−85.8	2.6
	5	−81.5	−7.8	4.2	−85.7	2.9
	10	−85.9	−13	−1	−84.9	2.4

Table 130. dacFullScale Tx Linearity 0 dB Mode and 3 dB Mode

F1 Tone MHz	F2 Tone MHz, (F1 + 5 MHz)	OIP3 dBm, 0 dB Mode, Tones = −15 dBFS	OIP3 dBm, 3 dB Mode, Tones = −18 dBFS (Data Sheet Equivalent Output Power)	OIP3 dBm, 3 dB mode, (Tones = −15 dBFs)
10	15	32.6	36.0	33.1
30	35	35.3	34.9	36.0
50	55	38.0	37.7	40.0
70	70	33.9	34.8	33.2
90	95	35.7	31.1	30.0

Table 131. dacFullScale EVM vs. Mode Selection

	0 dB mode		3 dB mode	
TX Attenuator (dB)	Signal Power (dBm)	EVM (dB)	Signal Power (dBm)	EVM (dB)
0	−17.9	−45.28	−15.0	−45.86
5	−22.9	−45.09	−20.0	−45.72
10	−27.9	−43.73	−25.0	−44.97
15	−32.8	−43.38	−30.0	−43.06
20	−37.8	−43.08	−34.9	−43.64

ADI_ADRV9025_TXCHANNELCFG API STRUCTURE

The dacFullScale enum is stored in the adi_adrv9025_TxChannelCfg structure. This structure is stored within the adi_adrv9025_TxSettings_t structure, which is stored in the overall device initialization structure (adi_adrv9025_Init_t). The parameters are described in Table 132 and Table 133. The dacFullScale parameter is also found in the json (profile) file.

Table 132. adi_adrv9025_TxChannelCfg Structure Parameters

Data Fields	Description
adi_adrv9025_TxProfile_t	profile
adi_adrv9025_DacFullScale_e	dacFullScale

Table 133. adi_adrv9025_DacFullScale_e enum Parameters

Data Fields	Description	Value
ADI_ADRV9025_TX_DACFS_0DB	DAC full scale = 0 dB (default mode)	0x0
ADI_ADRV9025_TX_DACFS_3DB	DAC full scale = 3 dB	0x1

TRANSMITTER POWER AMPLIFIER PROTECTION

The ADRV9026 features four transmitters with independent power amplifier (PA) protection circuitry. The PA protection circuitry operates in conjunction with other interrupt sources within the transceiver. This section describes both PA protection and the other interrupt sources that can trigger a Tx attenuation ramp to set the Tx attenuation to 40 dB to protect the PA device.

Note that it is recommended to use these features in conjunction with the GP_INTERRUPT feature so that the baseband processor receives information over GP_INTERRUPT pins that an attenuation ramp down may have occurred. This is achieved by unmasked relevant GP_INTERRUPT sources described within this document.

PA PROTECTION DESCRIPTION

The PA protection circuitry is designed to alert the user that the digital signal power within the Tx datapath exceeds a programmable threshold. The GPINT1 and GPINT2 pins can be configured to assert when the PA protection block detects an 'error.' In this context, error means that a power threshold has been exceeded. If PA protection is used, it is recommended that the user unmask the PA protection interrupts for one of the GPINT pins to give the baseband processor an indication that a PA protection error has occurred. Set up the power thresholds at a level appropriate for their system given the PA damage power level and Tx RF attenuation.

There are two types of thresholds in the PA protection circuit: peak power threshold and average power threshold.

- Peak Power Threshold: When the peak signals detected by PA protection exceed the peak power threshold (peakThreshold) a programmable number of times (peakCount) within a period (peakDuration), this leads to a peak power threshold error (peakPowerErr = 1).
- Average Power Threshold: When the signal power calculated by PA protection exceeds the programmable average power threshold (powerThreshold) within a period (avgDuration), this leads to an average power threshold error (avgPowerErr = 1)

When PA protection is enabled and a PA protection error occurs, a ramp down of the Tx attenuation can be executed. The attenuation is set to 40 dB after the ramp down, if enabled. This feature can be used to protect PA devices in scenarios where the baseband processor executes algorithms that affect the power of the transmitted signal. The attenuation ramp down is configured with the adi_adrv9025_PaPllDfrmEventRamp DownEnableSet(…) command.

PA Protection Configuration

The PA protection feature is setup with the API command adi_adrv9025_TxPaProtectionCfgSet(…).

adi_adrv9025_TxPaProtectionCfgSet

```
adi_adrv9025_TxPaProtectionCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_TxPaProtectCfg_t
  txPaProtectCfg[], uint8_t arraySize);
```

Description

Sets up the PA protection feature.

Parameters

Table 134.

Parameter	Description
*device	Pointer to device structure.
txPaProtectCfg[]	An array of PA protection configurations of data type adi_adrv9025_TxPaProtectCfg_t. This data structure is explained in further detail in Table 135.
arraySize	The array length of txPaProtectCfg[].

Table 135. adi_adrv9025_TxPaProtectCfg_t Data Structure Parameters

Data Type	Parameter Name	Parameter Description
adi_adrv9025_TxChannels_e	txChannel	Tx channel select based on adi_adrv9025_TxChannel_e. PA protection configuration is applied to channels selected by this parameter
uint8_t	avgDuration	Sets the duration for which average power is accumulated and compared with powerThreshold. Range = 0 to 15. Duration in time is given by (sample rate in Hz, duration in seconds): $t_{avgDuration} = \frac{1}{txSampleRate} \times 2^{avgDuration+5}$

Data Type	Parameter Name	Parameter Description
uint8_t	peakDuration	Sets the duration for which peaks are compared against peakThreshold. At the end of this duration, the number of counted peaks resets to zero. Range = 0 to 15. Duration in time is given by (sample rate in Hz, duration in seconds): $t_{peakDuration} = \frac{1}{txSampleRate} \times 2^{peakDuration+5}$
uint16_t	powerThreshold	Sets the powerThreshold for average power measurements. If the average power exceeds this threshold, the avgPowerErr signal is asserted. $powerThreshold_{dBFS} = 10\log\left(\frac{powerThreshold}{8192}\right)$
uint8_t	peakCount	Sets a limit for the number of peaks detected within a peakDuration. When this limit is exceeded, the PA protection peakPowerErr signal is asserted.
uint16_t	peakThreshold	Sets the peak threshold power limit for counting a peak. If a peak exceeds this threshold, it is counted. When this counter value exceeds peakCount, peakPowerErr signal is asserted. $peakThreshold_{dBFS} = 10\log\left(\frac{peakThreshold}{8192}\right)$
uint8_t	avgPowerEnable	When set = 1, the PA protection average power measurement block is enabled. Allows avgPowerErr signal assertion. When set = 0, the PA protection average power measurement block is disabled.
uint8_t	peakPowerEnable	When set = 1, the PA protection peak power measurement block is enabled. Allows peakPowerErr signal assertion. When set = 0, the PA protection peak power measurement block is disabled.
adi_adrv9025_PaProtectionInputSel_e	inputSel	Determines the data path location for peak and average power measurement. Options are given by the enumeration described in Table 136.
uint8_t	avgPeakRatioEnable	When set = 1, this enables the average to peak power ratio block. avgPowerEnable and peakPowerEnable need to be enabled. When set = 0, average to peak power calculations are not performed.

Table 136 describes the adi_adrv9025_PaProtectionInputSel_e enumeration. These measurement locations are shown in Figure 71.

Table 136. adi_adrv9025_PaProtectionInputSel_e Enumeration Options

Enumeration	Enum Value	Description
ADI_ADRV9025_COMPLEX_MULT_OUTPUT	0	Input data to PA protection block comes from the complex multiplier output.
ADI_ADRV9025_TXQEC_ACTUATOR_OUTPUT	1	Input data to PA protection block comes from the Tx QEC actuator output.

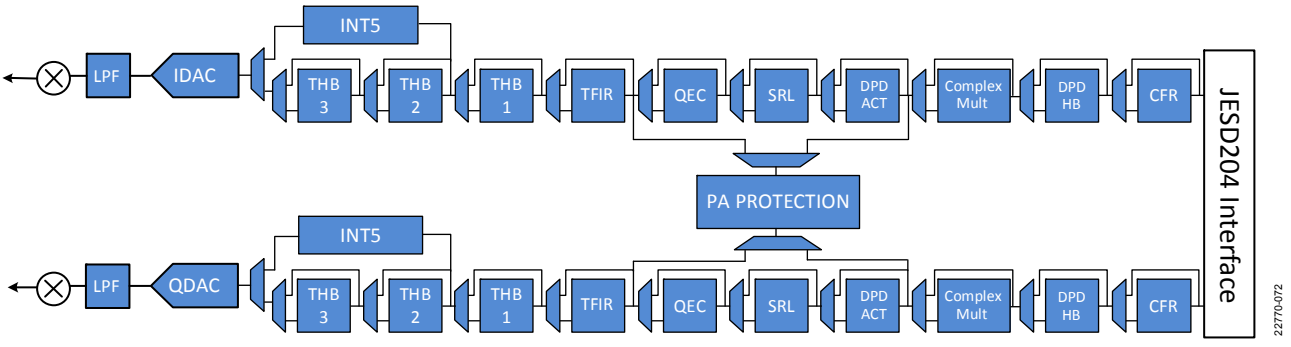


Figure 71. Tx Datapath Showing PA Protection Measurement Locations

PA Protection Runtime Commands

This section describes commands that can be used to check the status of the PA protection blocks. The GP_INTERRUPT represents a real time interface to notify the baseband processor that a PA protection error has occurred. When the interrupt asserts, call the GP_INTERRUPT handler command. If it is indicated that a PA protection error has occurred, the commands in this section describe what the user can do to acquire more information or clear the error.

adi_adrv9025_TxPaProtectionErrFlagsGet

```
adi_adrv9025_TxPaProtectionErrFlagsGet(adi_adrv9025_Device_t* device, adi_adrv9025_TxChannels_e
  txChannel, adi_adrv9025_TxPaProtectionErr_t* errorFlags);
```

Description

Gets information about which PA protection error flag has been asserted and the associated power level. Do not call this command before adi_adrv9025_TxPaProtectionCfgSet(…).

Parameters

Table 137.

Parameter	Description
*device	Pointer to device structure.
txChannel	The Tx channel mask that selects which transmitter to retrieve error flag information from.
errorFlags	A data structure containing the error flag information for selected Tx channel.

Table 138. adi_adrv9025_TxPaProtectionErr_t Data Structure Parameters

Data Type	Parameter Name	Parameter Description
uint8_t	peakPowerErr	If value = 1, then the peak power error bit has been asserted. If value = 0, the peak power error has not been asserted. This bit is sticky depending on configuration applied in adi_adrv9025_TxAttenuationRampUpStickyModeEnable(…)
uint8_t	avgPowerErr	If value = 1, then the average power error bit has been asserted. If value = 0, the average power error has not been asserted. This bit is sticky depending on configuration applied in adi_adrv9025_TxAttenuationRampUpStickyModeEnable(…)
uint16_t	powerErr	When avgPowerErr asserts, this parameter contains the average power level that triggered the error condition.

Clearing PA Protection Error Flags

In the case a PA protection error has occurred, it is useful to obtain specific information whether it is a peak power error or an average power error. To obtain information about which PA protection error flag has been asserted use adi_adrv9025_TxPaProtectionStatusGet(…). Once this information has been obtained and the cause of the error has been resolved, the user must clear the error flag manually when the errors are configured in sticky mode. This can be done with the adi_adrv9025_PaPllDfrmEventClear(…) command or the command described below. Note that adi_adrv9025_PaPllDfrmEventClear(…) can clear a PA protection error, a PLL unlock interrupt, or a deframer interrupt. The following command is specific only to PA protection errors.

adi_adrv9025_TxPaProtectionErrFlagsReset

```
adi_adrv9025_TxPaProtectionErrFlagsReset(adi_adrv9025_Device_t* device,
  adi_adrv9025_TxChannels_e txChannel, adi_adrv9025_TxPaProtectErrFlags_e errorFlags);
```

Description

Clears PA Protection error flags for specified channels.

Parameters

Table 139.

Parameter	Description
*device	Pointer to device structure.
txChannel	The Tx channel mask that selects which transmitter to clear/reset PA protection errors.
errorFlags	An enumerated data type describing which error flags need to be cleared.

Table 140 describes the adi_adrv9025_TxPaProtectErrFlags_e enumeration.

Table 140. adi_adrv9025_TxPaProtectErrFlags_e Enumeration Options

Enumeration	Enum value	Meaning
ADI_ADRV9025_TXPA_PROTECT_FLAGS_AVG_POWER_ERR	1	Reset average power error flag
ADI_ADRV9025_TXPA_PROTECT_FLAGS_PEAK_POWER_ERR	2	Reset peak power error flag
ADI_ADRV9025_TXPA_PROTECT_FLAGS_ALL	3	Reset both average and peak power error flags

Adi_adrv9025_TxPaProtectionStatusGet

The PA protection status data structure provides information regarding the power in the data path. After the PA protection configuration has been applied, the following command can be called.

```
adi_adrv9025_TxPaProtectionStatusGet(adi_adrv9025_Device_t* device, adi_adrv9025_TxChannels_e
  txChannel, adi_adrv9025_TxPaProtectStatus_t* status);
```

Description

Reads back the Tx average IQ sample power.

Parameters

Table 141.

Parameter	Description
*device	Pointer to device structure.
txChannel	The Tx channel mask that selects from which transmitter to retrieve PA protection status information.
status	A data structure containing the PA protection status information for selected Tx channel.

The data structure type adi_adrv9025_TxPaProtectionStatus_t is described in Table 142.

Table 142. adi_adrv9025_TxPaProtectionStatus_t Data Structure Parameters

Data Type	Parameter Name	Parameter Description
uint16_t	avgPower	Result of the most recently completed average power measurement. Result in dBFS is provided by the formula: $P_{AVG} = 10\log\left(\frac{avgPower}{2^{16}}\right)$
uint16_t	avgPeakRatio	Measurement describing the average to peak ratio as measured by PA protection. Enable peak and average power measurement for meaningful results. $PAR = 10\log\left(\frac{avgPeakRatio}{2^{16}}\right)$
uint16_t	avgErrorPower	When avgPowerErr asserts, this parameter contains the average power level that triggered the error condition. This parameter only updates when an average power error occurs. $P_{avgErrPow} = 10\log\left(\frac{avgErrorPower}{2^{16}}\right)$

adi_adrv9025_PaPllDfrmEventRampDownEnableSet

```
adi_adrv9025_ PaPllDfrmEventRampDownEnableSet(adi_adrv9025_Device_t* device, uint32_t
txChannelMask, uint32_t irqMask, uint8_t enable);
```

Description

Configures which interrupts can trigger a Tx attenuation ramp down event.

Parameters

Table 143.

Parameter	Description
*device	Pointer to device structure.
txChannelMask	The Tx channel mask for selecting which transmitters to configure based on adi_adrv9025_TxChannels_e enumeration.
irqMask	The bit mask that selects which interrupts are enabled or disabled based on the enable parameter. If a bit within this mask is set, then the value of enable is applied for each bit set. The value must not be zero. A description of the irqMask bit field is provided in Table 144.
enable	Bit that controls ramp down for the events selected by irqMask. If set to 0, the function is disabled for all selections.

Table 144. Bitwise Description of irqMask

Bit Position	Description	Command to Clear Interrupt
D7	PA protection error flag has been asserted. If Slew Rate Limiter (SRL) interrupt (IRQ) has been enabled, this bit also allows attenuation ramp down based on the SRL IRQ.	adi_adrv9025_adrv9025_PaPllDfrmEventClear(…) or adi_adrv9025_adrv9025_TxPaProtectionErrFlagsReset(…)
D6	SERDES PLL Unlock	adi_adrv9025_adrv9025_PaPllDfrmEventClear(…)
D5	RF PLL 2 Unlock	
D4	RF PLL 1 Unlock	
D3	AUX PLL Unlock	
D2	CLK PLL Unlock	
D1	Deframer 1 Interrupt/IRQ	
D0	Deframer 0 Interrupt/IRQ	

While the irqMask is a uint32_t data type value, the enumeration adi_adrv9025_PaPllDfrmRampDownEnSel_e can be used to form the irqMask.

Sticky Control for Tx Attenuation Ramp Down

If a Tx attenuation ramp down interrupt is asserted, there are two modes of interrupt behavior pertaining to when attenuation is restored. These modes of behavior control how the attenuation level ramp up is performed.

- Sticky Interrupt (Default operation): The attenuation ramp down remains in effect until the API command adi_adrv9025_PaPllDfrmEventClear(…) is called and the interrupt is no longer asserted. These two conditions need to be true for attenuation to return to its former level before the interrupt. This mode requires user intervention.
- Auto clear Interrupt: The attenuation ramp down remains in effect until the interrupt is no longer asserted. This mode only depends on the status of the interrupt.

The user can select between these modes through the following API command.

adi_adrv9025_TxAttenuationRampUpStickyModeEnable

```
adi_adrv9025_ TxAttenuationRampUpStickyModeEnable(adi_adrv9025_Device_t* device, uint32_t
channelMask, uint8_t txPllJesdProtClrReqd, uint8_t txPaProtectionAvgpowerErrorClearRequired,
uint8_t txPaProtectionPeakpowerErrorClearRequired)
```

Description

Configures Tx attenuation ramp up sticky mode for the selected Tx channel.

Parameters

Table 145.

Parameter	Description
*device	Pointer to device structure.
channelMask	The Tx channel mask for selecting which transmitters to configure based on adi_adrv9025_TxChannels_e enumeration.
txPllJesdProtClrReqd	Determines if the user is required to manually clear PLL/deframer attenuation ramp down events after assertion. Setting 1 requires user to clear, 0 does not require user to clear.
txPaProtectionAvgpowerErrorClearRequired	Determines if the user is required to manually clear PA protection average power error flag after assertion. Setting 1 requires user to clear, 0 does not require user to clear.
txPaProtectionPeakpowerErrorClearRequired	Determines if the user is required to manually clear PA protection peak power error flag after assertion. Setting 1 requires user to clear, 0 does not require user to clear.

The command adi_adrv9025_adrv9025_PaPllDfrmEventClear(…) can be used to clear the error.

Determining the Interrupt Source of an Attenuation Ramp Down

The GPINT1 and GPINT2 pins can be configured to alert the baseband processor that a PA protection error, PLL unlock event, or deframer interrupt has occurred. When the interrupt has occurred, the user is expected to call adi_adrv9025_GpInt1Handler or adi_adrv9025_GpInt0Handler depending on which GPINT pin has asserted. GpInt1Handler is linked to the GPINT2 pin and GPInt0Handler is linked to the GPINT1 pin. The handler returns information relevant to which interrupts have been asserted. This is one method to determine which interrupts have asserted. However, note that the GP_INTERRUPT bitmask description does not specify whether a peak or average power PA protection error has occurred. To obtain more specificity regarding the error source, call adi_adrv9025_PaPllDfrmEventGet(…).

adi_adrv9025_PaPllDfrmEventGet

```
adi_adrv9025_PaPllDfrmEventGet (adi_adrv9025_Device_t* device, adi_adrv9025_TxChannels_e
txChannelSelect, uint8_t eventBits);
```

Description

Reads the status of events causing Tx attenuation ramp down rather than any signal that has asserted GP_INTERRUPT.

Parameters

Table 146.

Parameter	Description
*device	Pointer to device structure.
txChannelSelect	The Tx channel mask for selecting which transmitters to configure based on adi_adrv9025_TxChannels_e enumeration.
eventBits	Selects which interrupt source to clear based on the bit description in Table 147. If a bit position in this value is set high, the associated interrupt has asserted to cause a Tx attenuation ramp down.

The command adi_adrv9025_adrv9025_PaPllDfrmEventClear(…) can be used to clear the error.

Table 147. Bitwise Description of eventBits Parameter

Bit Position	Description
D3 to D7	Unused
D2	Any PLL unlock or deframer error
D1	PA protection peak power error
D0	PA protection average power error

Clearing Tx Attenuation Ramp Down Events

There are two commands available to clear attenuation ramp down events. In the case that the interrupts are configured as sticky interrupts, the user needs to call the appropriate function to clear the error. Note that these commands do not execute corrective measures to remove the error source. For example, calling adi_adrv9025_TxPaProtectionErrFlagsReset(…) after a PA protection average power error does not mean that the cause of the error is gone. If the datapath power is still greater than the PA protection average power threshold after this command is called, then the interrupt persists. In some cases, the baseband processor must take an action to resolve the interrupt/error. The following command can be used to clear such interrupts.

adi_adrv9025_PaPllDfrmEventClear

```
adi_adrv9025_PaPllDfrmEventClear(adi_adrv9025_Device_t* device, adi_adrv9025_TxChannels_e
txChannelSelect, uint8_t eventBits);
```

Description

Clears the Tx attenuation ramp down interrupts caused by the deframer or PLL unlock events.

Parameters

Table 148.

Parameter	Description
*device	Pointer to device structure.
txChannelSelect	The Tx channel mask for selecting which transmitters to configure based on adi_adrv9025_TxChannels_e enumeration.
eventBits	Selects which interrupt source to clear based on the bit description in Table 147. If a bit position in this value is set high, the command attempts to clear the interrupt.

RECEIVER GAIN CONTROL AND GAIN COMPENSATION

OVERVIEW

The ADRV9026 receivers (Rx1/Rx2/Rx3/Rx4) feature automatic and manual gain control modes allowing for flexible gain control in a wide array of applications. Automatic gain control (AGC) allows for receivers to autonomously adjust the receiver gain depending on variations of the input signal, such as the onset of a strong interferer that can overload the receiver data path. It controls the gain of the device based on the information from a number of signal detectors (peak/power detectors). The AGC can control the gain with very fine resolution if required. The receivers are also capable of operating in Manual Gain Control (MGC) mode where changes in gain are initiated by the baseband processor. The gain control blocks are configured by means of the API data structures and several API functions exist to allow for user interaction with the gain control mechanisms.

The AGC is highly flexible and can be configured in a number of ways. 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 typically do not occur very often for typical 3G/4G operation. Therefore, the Peak Detect Mode operation is sufficient. Nevertheless, if an asynchronous blocker does appear, a fast attack mode exists that is able to reduce the gain at a fast rate.

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

This section contains the following functional descriptions:

Receiver Data Path: This section outlines the gain control and signal observation elements of the receiver chain. It then describes the concept of the receiver gain table.

Gain Control Modes: This section advises how to select between the gain control modes.

Manual Gain Control (MGC): This section describes how to operate the device in manual gain control mode.

Automatic Gain Control (AGC): This section describes the two principal modes of AGC operation, peak detect mode and peak/power detect mode.

AGC Clock and Gain Block Timing: This section describes the speed of the AGC clock and the various gain event and delay timers.

Peak and Power Detectors: This section outlines the operation and configuration of the gain control detectors in the device.

API Programming: This section outlines how to configure the AGC using the API, explaining each parameter of the AGC API structures. It also provides details of gain control utility functions within the API.

Sample Scripts: Sample scripts are provided in this section which can be used in the Iron Python tab of the TES, allowing AGC to be tested on the evaluation platform.

Gain Compensation, Floating Point Formatter and Slicer: This section outlines the various forms of gain compensation available in the ADRV9026.

This document contains a full description of the gain control functionality available in the device. Some features may not be available depending on the software version.

Glossary of Important Terms

Automatic Gain Control (AGC): This term is used to refer to the internal AGC of the device, where the device is in control of the receiver gain settings. If the user does not use the internal AGC, then it is expected that an AGC runs in the baseband processor.

Manual Gain Control (MGC): This term is used to refer to a use case when the user is in control of the currently applied gain settings in the receiver chain.

Gain Attack: This term is used to indicate the reduction of the receiver gain due to an overloaded signal path.

Gain Recovery: This term is used to indicate the increase of the receiver gain due to a reduction in the power of the signal being received.

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

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

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

Threshold Overload: When a threshold is exceeded in a peak detector, this is referred to as an overload.

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

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

RECEIVER DATA PATH

Figure 72 shows the Rx data path and gain control blocks. The receivers have front end attenuators prior to the mixer stage that are used to attenuate the signal in the RF domain to ensure the signal does not overload the receiver chain. In the digital domain, there is the option of digital attenuation or digital gain. This digital gain block is also utilized for gain compensation.

The receiver chain also has multiple observation elements that can monitor the incoming signal. These can be used in either MGC or AGC modes. Firstly, an Analog Peak Detector (APD) exists prior to the ADC. Being in the analog baseband, this peak detector sees signals first, and also has blocker signal visibility, which can overload the ADC but be filtered as they progress through the digital chain. Note that the APD is located after the TIA filter. The second peak detector is called the HB2 overload detector, so called because it monitors the data at the HB2 filter in the receiver chain.

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

This device can also control an external gain element through use of the receiver gain table and the GPIO_ANA pins.

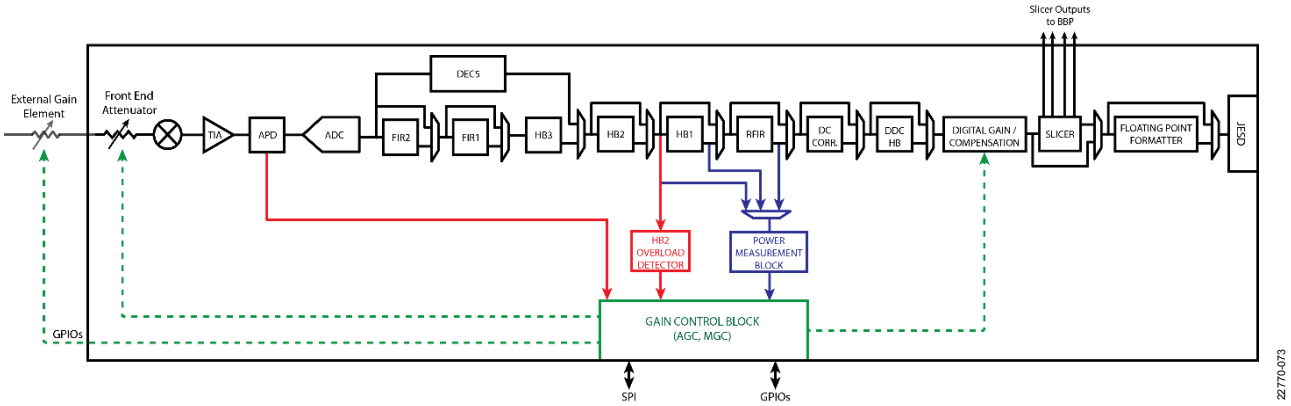


Figure 72. Rx Data Path and Gain Control Blocks

The gain control block is shown with multiple inputs providing information. Overload (Peak) detectors are shown in red, while the power measurement block is shown in blue. The gain control block controls the gain of the signal chain using a gain table.

This table is user programmable, and each row of the table provides a combination of front end attenuator, external gain element (if used) and digital gain settings. Based on the row of this table selected, either by the user in MGC mode, or automatically by the device in AGC mode, the gain control block updates the variable gain elements depicted by the green arrows. Finally, the user can control the gain control block using the SPI bus (configuration of AGC, MGC) and GPIOs.

Table 149 shows a sample gain table.

Table 149. Sample Rows from the Default Rx Gain Table

Gain Table Index	Front-End Attenuator[7:0]	External Gain Control[3:0]	TIA/ADC Gain	Signed Digital Gain/Attenuation[10:0]	Phase Offset
255	0	0	0	0	0
254	14	0	0	0	0
253	28	0	0	0	0

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

The front end attenuator has an 8-bit control word. The amount of attenuation applied depends on the value set in the front-end attenuator column of the selected gain table index. The following equation provides an approximate relationship between the internal attenuator and the front-end attenuation value programmed in the gain table, N:

$$Attenuation\,(dB) = 20\log_{10}\left(\frac{256 - N}{256}\right)$$

The external gain control column controls two analog GPIOs for each Rx. Table 150 shows which analog GPIOs are used for which Rx.

Table 150. Analog GPIOs for External Gain Element Control

Receiver	GPIO Pins to Control External Gain Element
Rx1	GPIO_ANA[1:0]
Rx2	GPIO_ANA[3:2]
Rx3	GPIO_ANA[5:4]
Rx4	GPIO_ANA[7:6]

These analog GPIOs must be enabled as outputs and set for external gain functionality. The 2-bit value programmed is directly related to the status of these GPIO pins, for example if the external gain word of the Rx1 gain table is programmed to 3 in selected gain index, then analog GPIO0 and 1 is high.

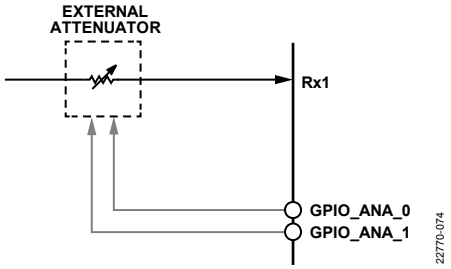


Figure 73. GPIO Control of an External Gain Element to Rx1

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

The TIA/ADC gain must be zero in all rows because this functionality is not used.

GAIN CONTROL MODES

The gain control mode is selected with the following API function.

adi_adrv9025_RxGainCtrlModeSet

```
adi_adrv9025_RxGainCtrlModeSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxAgcMode_t
gainMode[], uint8_t arraySize)
```

Description

Selects the gain control mode.

Parameters

Table 151.

Parameter	Description
*device	Pointer to device structure.
gainMode	An array of type adi_adrv9025_RxAgcMode_t indicating which gain mode is to be used for which Rx channel
arraySize	The size of the array

Each adi_adrv9025_RxAgcMode_t instance contains agcMode, an enum selecting the chosen gain mode. The possible options are shown in Table 152.

Table 152. Definition of adi_adrv9025_RxAgcMode_e

Enum	Gain Mode
ADI_ADRV9025_MGC	Manual Gain Mode
ADI_ADRV9025_AGCSLOW	Automatic Gain Control Mode
ADI_ADRV9025_HYBRID	Not currently supported

rxChannelMask

This selects the channels upon which to enable this gain control mode. It is a bit mask with each bit corresponding to a channel, [D0] = Rx1, [D1] = Rx2, [D2] = Rx3, [D3] = Rx4. Setting the rxChannelMask = 15 means that all Rxx are configured with the same agcMode.

MANUAL GAIN CONTROL (MGC)

The gain control block applies the settings from the selected gain index in the gain table. In MGC mode, the baseband processor is in control of the selecting the gain index. There are two options: 1) API commands; and 2) pin control. By default, if MGC is chosen the part is configured for API commands. The following commands can be used when in API command mode.

adi_adrv9025_RxGainSet

```
adi_adrv9025_RxGainSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxGain_t rxGain[], uint8_t
arraySize)
```

Description

Selects the gain index in the gain table when in API command mode.

Parameters

Table 153.

Parameter	Description
*device	Pointer to device structure.
rxGain	An array of type adi_adrv9025_RxGain_t that determines the gain setting and the channels using the chosen setting
arraySize	The size of the array

Each adi_adrv9025_RxGain_t instance contains:

- gainIndex—the selected gain index from the gain table
- rxChannelMask—this selects the channels upon which to apply the gainIndex setting. It is a bit mask with each bit corresponding to a channel, [D0] = Rx1, [D1] = Rx2, [D2] = Rx3, [D3] = Rx4. Setting the rxChannelMask = 15 applies this gain index to all four receivers.

adi_adrv9025_RxGainGet

```
adi_adrv9025_RxGainGet(adi_adrv9025_Device_t* device, adi_adrv9025_RxChannels_e rxChannel,
adi_adrv9025_RxGain_t *rxGain)
```

Description

Reads back the gain index in the gain table for the selected channel when in API command mode.

Parameters

Table 154.

Parameter	Description
*device	Pointer to device structure.
rxChannel	An enum as shown in Table 155.
*rxGain	Of type adi_adrv9025_RxGain_t, pointer to the current gain of the channel and a mask indicating which gain of the channel is contained within the structure.

Table 155. Definition of adi_adrv9025_RxChannels_e

Receiver	ENUM
Rx1	ADI_ADRV9025_RX1
Rx2	ADI_ADRV9025_RX2
Rx3	ADI_ADRV9025_RX3
Rx4	ADI_ADRV9025_RX4

The pin control MGC mode is useful when real time control of gain is required. In this mode 2 GPIO pins per receiver are used, two for each receiver, one increasing, the other decreasing the gain table index. The user specifies both the increment and decrement step size in terms of number of gain indices. A pulse is applied to the relevant GPIO pin to trigger an increment of decrement in gain as shown in Figure 74. This pulse must be held high for at least 2 AGC clock cycles for a gain change to occur (see the AGC Clock and Gain Block Timing section for details).

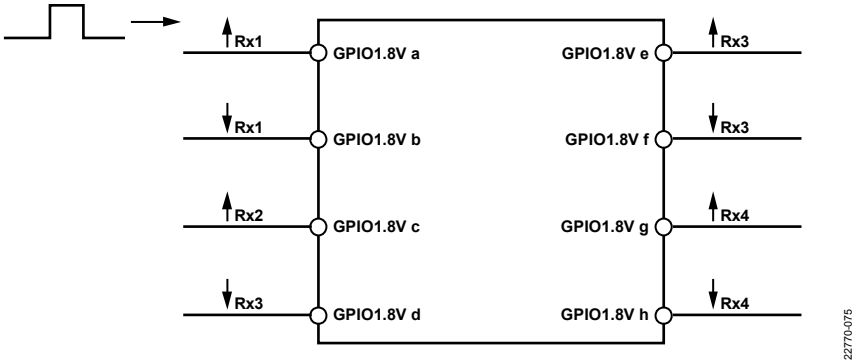


Figure 74. MGC Pin Mode: GPIO1.8V(a-h) Represent Any of GPIO0-15

adi_adrv9025_RxGainPinCtrlCfgSet

```
adi_adrv9025_RxGainPinCtrlCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxGainPinCfg_t
*rxGainPinCtrlCfg, adi_adrv9025_RxChannels_e rxChannel)
```

Description

Configures pin control MGC mode.

Parameters

Table 156.

Parameter	Description
*device	Pointer to device structure.
rxChannel	An enum indicating which Rx channel to configure as shown in Table 155.
*rxGainPinCtrlCfg	A configuration structure pointer for the pin control MGC mode containing members shown in Table 157.

Table 157. Definition of ADRV9025_RxGainCtrlPin_t

Member	Description
uint8_t incStep	Increment in gain index applied when the increment gain is pulsed. Acceptable values for this parameter are 0 to 7, however one is added to what is programmed into this parameter, resulting in step sizes of 1 to 8.
uint8_t decStep	Decrement in gain index applied when the decrement gain is pulsed. Acceptable values for this parameter are 0 to 7, however one is added to what is programmed into this parameter, resulting in step sizes of 1 to 8.
adi_adrv9025_GpioPinSel_e rxGainIncPin	GPIO used to increment gain. Any of GPIO0-15 can be used. Acceptable values: ADI_ADRV_9025_GPIO00 to ADI_ADRV9025_GPIO15.
adi_adrv9025_GpioPinSel_e rxGainDecPin	GPIO used to decrement gain. Any of GPIO0-15 can be used. Acceptable values: ADI_ADRV_9025_GPIO00 to ADI_ADRV9025_GPIO15.

The peak detector outputs can be monitored using GPIO pins by configuring them as outputs that are activated when an upper or lower threshold has been exceeded by the APD or HB2 detectors. More details can be found in the General-Purpose Input/Output Configuration section of this document. More details on what causes an over-range condition are provided in the Peak Detect Mode section of this guide.

AUTOMATIC GAIN CONTROL

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

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

The agcPeakThreshGainControlMode parameter of the AGC configuration structure adi_adrv9025_AgcCfg_t is used to select the individual modes of the AGC operation as shown in Table 158.

Table 158. agcPeakThreshGainControlMode Settings

agcPeakThreshGainControlMode	Description
0	AGC in peak/power mode
1	AGC in peak detect mode

Peak Detect Mode

In this mode, the peak detectors alone are used to inform the AGC to make gain changes. The APD and HB2 detector both have a high threshold and a low threshold. These are set with the parameters apdHighThresh, apdLowThresh, hb2HighTresh and hb2UnderRangeHighThresh. These levels are user programmable, as is the limit for the number of times a threshold needs to be crossed for an over range or under range condition to be flagged. The high thresholds are used as limits on the incoming signal level and typically are set based on the maximum input of the ADC. When an over range condition occurs, the AGC reduces the gain (gain attack).

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

Each overrange/underrange condition has its own attack and recovery gain step as shown in Table 159.

Table 159. Peak Detector Gain Steps

Overload/Under Range	Gain Step
apdHighThresh over range	Reduce gain by apdGainStepAttack
apdLowThresh under range	Increase gain by apdGainStepRecovery
hb2HighThresh over range	Reduce gain by hb2GainStepAttack
hb2UnderRangeHighThresh under range	Increase gain by hb2GainStepHighRecovery

An overrange condition occurs when the high thresholds have been exceeded a configurable number of times within a configurable period. An under range condition occurs when the low thresholds have not been exceeded a configurable number of times within the same configurable period. These counters make the AGC more or less sensitive to peaks in the input signal, ensuring that a single peak exceeding a threshold does not necessarily cause the AGC to react, allowing the user to trade off bit-error rate with signal to noise ratio. Table 160 outlines the counter parameters for the individual overload/under range conditions.

Table 160. Peak Detector Counter Values

Overload/Under Range	Counter
apdHighTresh over range	apdUpperThreshPeakExceededCnt
apdLowThresh under range	apdLowerThreshPeakExceededCnt
hb2HighThresh over range	hb2UpperThreshPeakExceededCnt
hb2UnderRangeHighThresh under range	hb2UnderRangeHighThreshExceededCnt

The AGC uses a gain update counter to time gain changes, with gain changes made when the counter expires. The counter value, and therefore the time spacing between possible gain changes, is user programmable through the agcGainUpdateCounter parameter. The user specifies the period, in AGC clock cycles, that gain changes can be made. Typically, this might be set to frame or sub-frame boundary periods. The total time between gain updates is the combination of the agcSlowLoopSettlingDelay and the agcGainUpdateCounter.

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 75 shows an example of the AGC response to a signal vs. the APD threshold levels. For ease of explanation, the APD is considered in isolation. The green 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 obtain the signal peaks within the threshold levels, and thus 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 interfering signal is removed, and the peaks of the signal are below the apdLowThresh, resulting in an under range condition. 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.

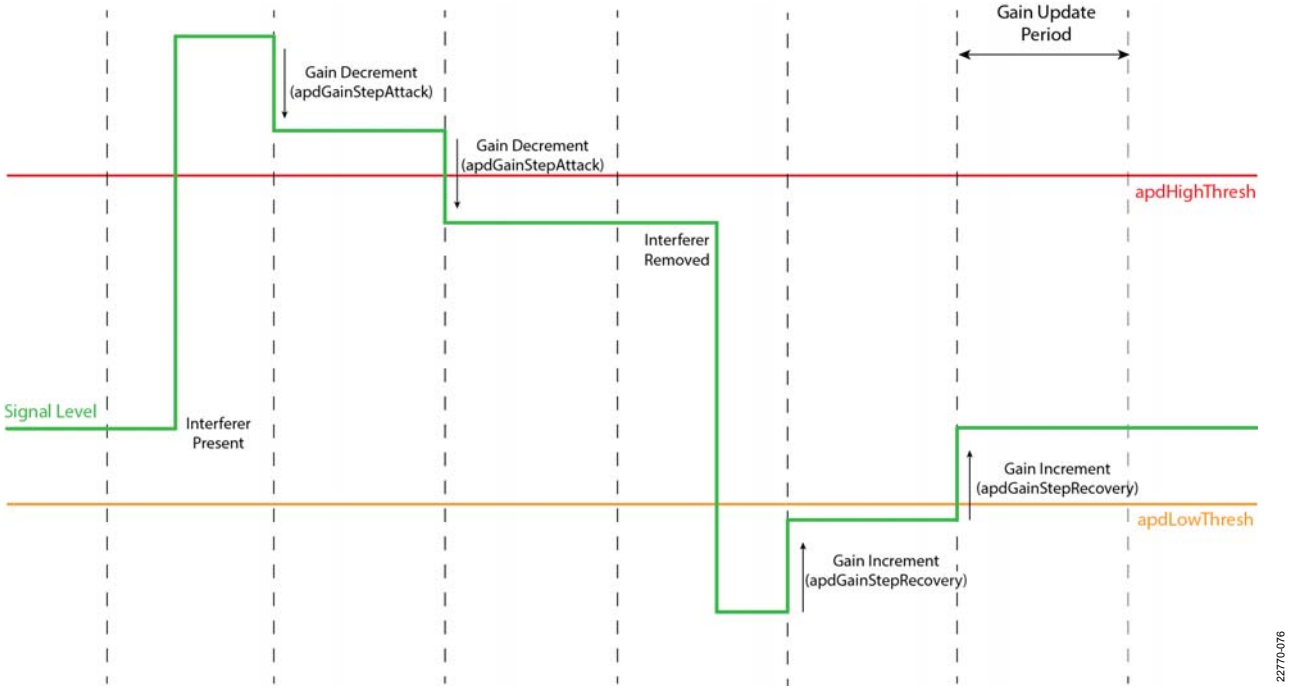


Figure 75. APD Thresholds and Gain Changes Associated with Underrange and Overrange Conditions

Figure 76 shows the same scenario but from the viewpoint of the HB2 detector considered in isolation.

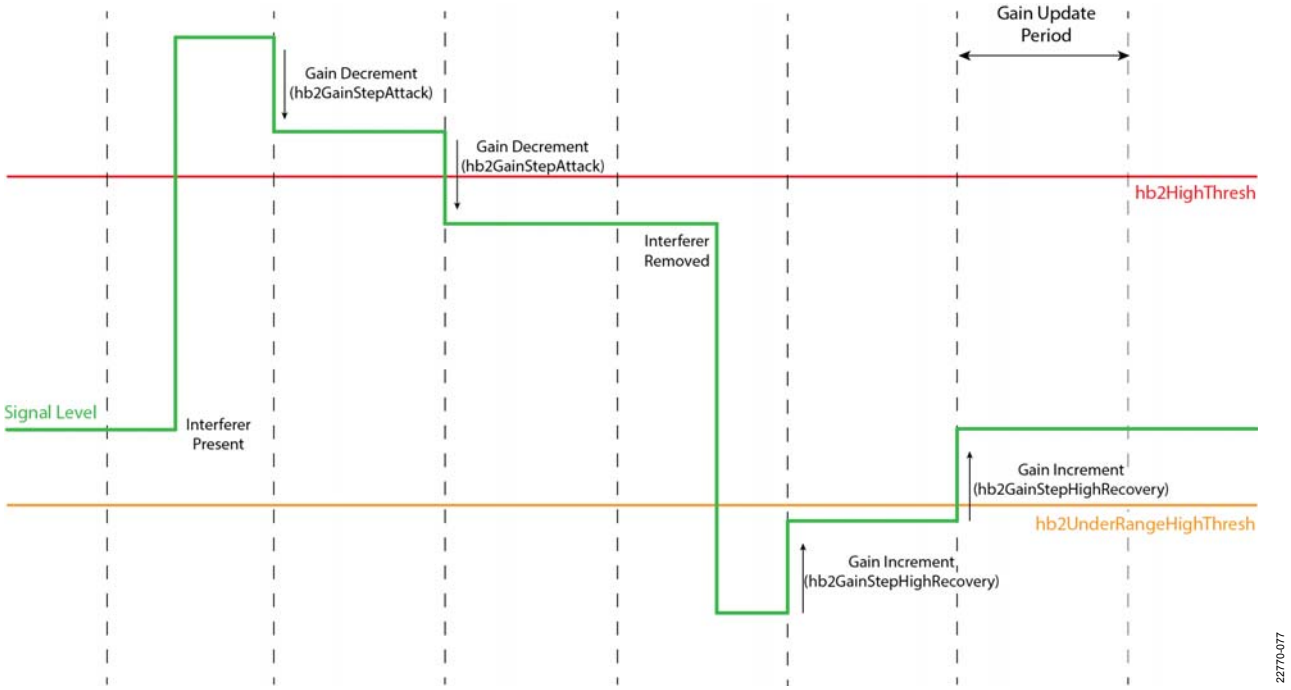


Figure 76. HB2 Thresholds and Gain Changes Associated with Under Range and Over Range Conditions

It is possible to enable a fast attack mode whereby the AGC is instructed to reduce gain immediately when an over range condition occurs, instead of waiting until the next expiry of the gain update counter using agcGainChangeIfThreshHigh. This parameter has independent controls for the APD and HB2 detectors. Values from 0-3 are valid as shown in Table 161.

Table 161. agcGainChangeIfThreshHigh Settings

agcChangeGainIfThreshHigh[1:0]	Gain Change Following APD Overrange	Gain Change Following HB2 Overrange
00	After expiry of agcGainUpdateCounter	After expiry of agcGainUpdateCounter
01	Immediately	After expiry of agcGainUpdateCounter
10	After expiry of agcGainUpdateCounter	Immediately
11	Immediately	Immediately

Figure 77 shows how the AGC reacts when the agcChangeGainIfThreshHigh is set for APD. In this case when the interferer appears, the gain is updated as soon as the number of peaks exceed the peak counter. It does not wait for the next expiry of the gain update counter. A number of gain changes can be 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 in order to change the gain.

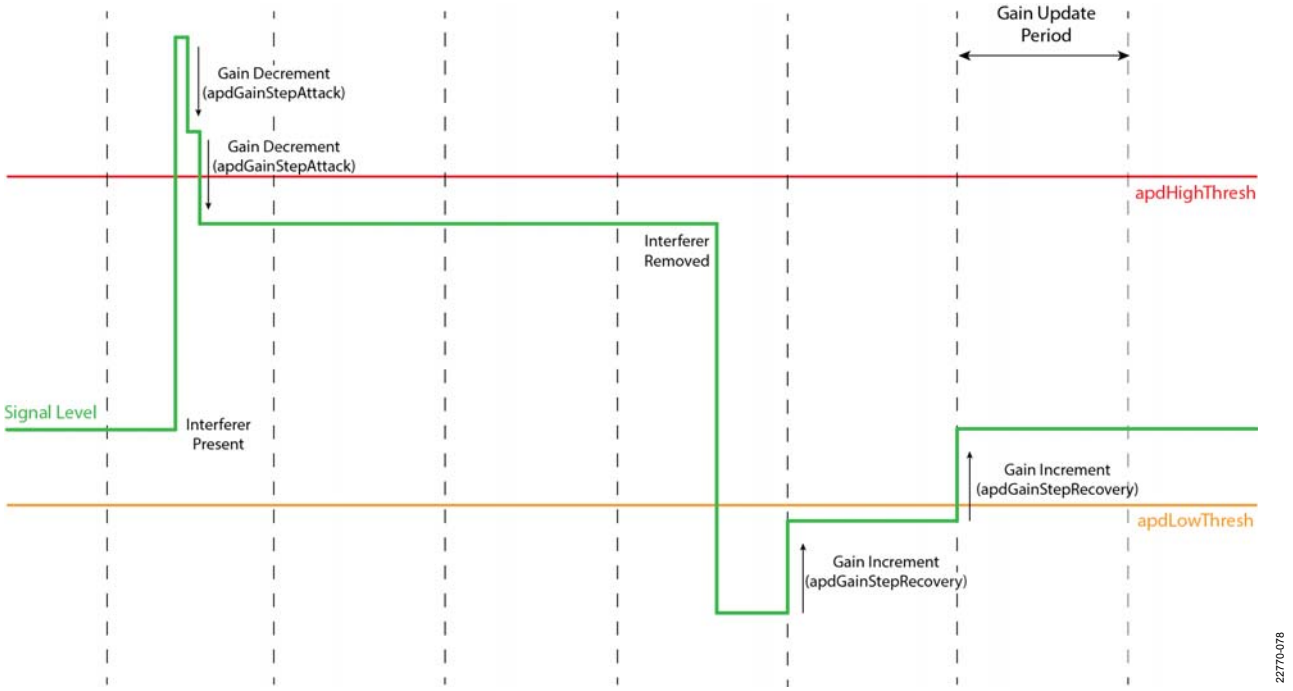


Figure 77. APD Gain Changes with Fast Attack Enabled

Figure 78 shows the same scenario but from the viewpoint of agcChangeGainIfThreshHigh being set for HB2.

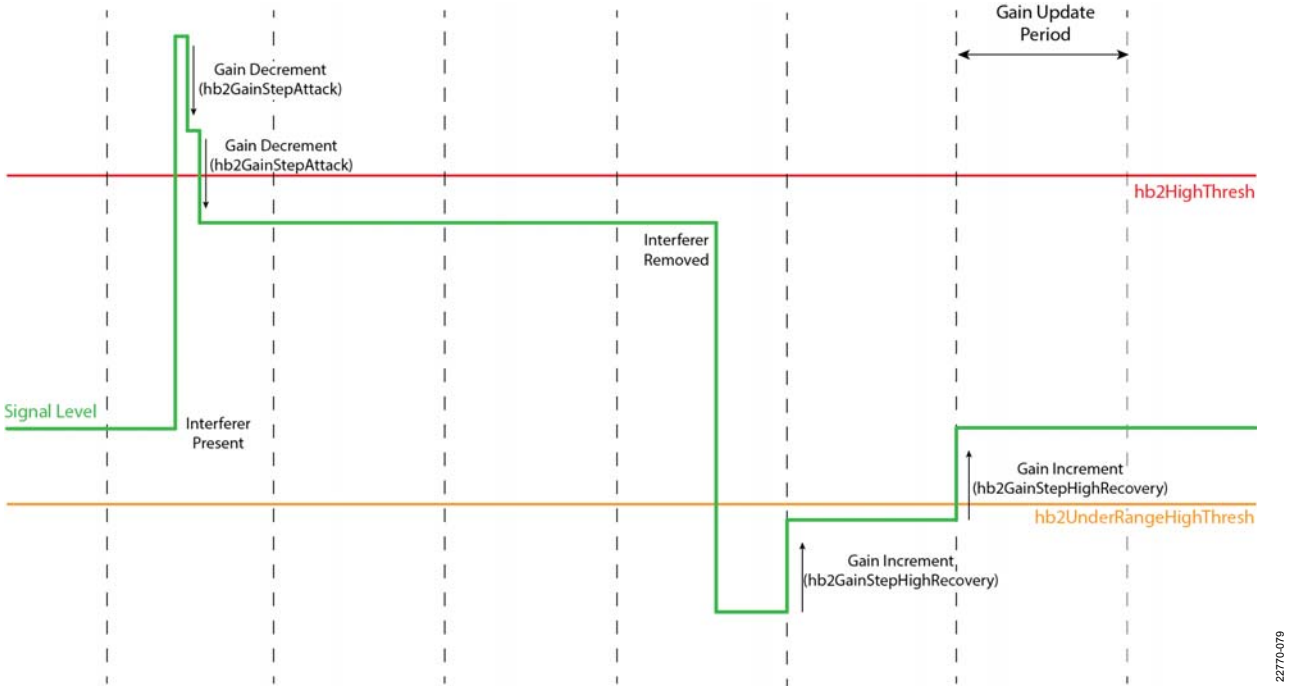


Figure 78. HB2 Gain Changes with Fast Attack Enabled

It is also possible to enable a fast recovery mode whereby a gain recovery event occurs at the expiry of the gain update period as shown in Figure 79. This functionality is enabled with the ableFastRecoveryLoop parameter. This fast recovery mode enables the HB2 overload detector. The operation is illustrated in Figure 80. When the signal level falls below hb2UnderRangeLowThresh, the gain is incremented by hb2GainStepLowRecovery following the expiry of the 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. After sufficient gain increases are implemented to bring the signal level above hb2UnderRangeLowThresh, the gain is incremented by hb2GainStepMidRecovery after the expiry of a number of gain update periods as set by hb2GainStepMidRecovery. Finally, when the signal level is increased above hb2UnderRangeMidThresh, the gain is incremented by hb2GainStepHighRecovery following the expiry of a number of gain update periods as set by agcUnderRangeHighInterval. The multiple threshold and interval parameters allow for a gain recovery whereby as the wanted signal level is approached, the magnitude of the gain adjustments is reduced and the time interval between gain changes is increased. However recovery events remain periodic as shown in Figure 79 because all gain updates occur at the expiry of the gain update period.

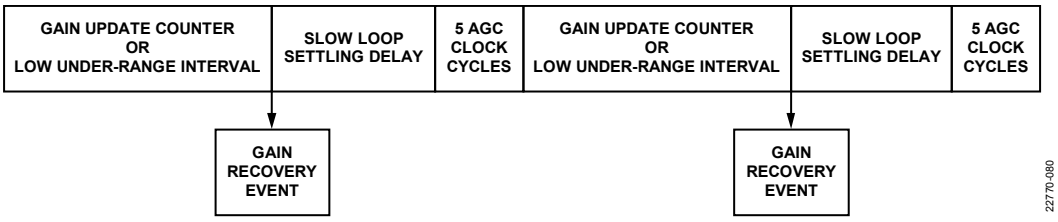


Figure 79. AGC Sequence with HB2 Detector in Fast Recovery Mode

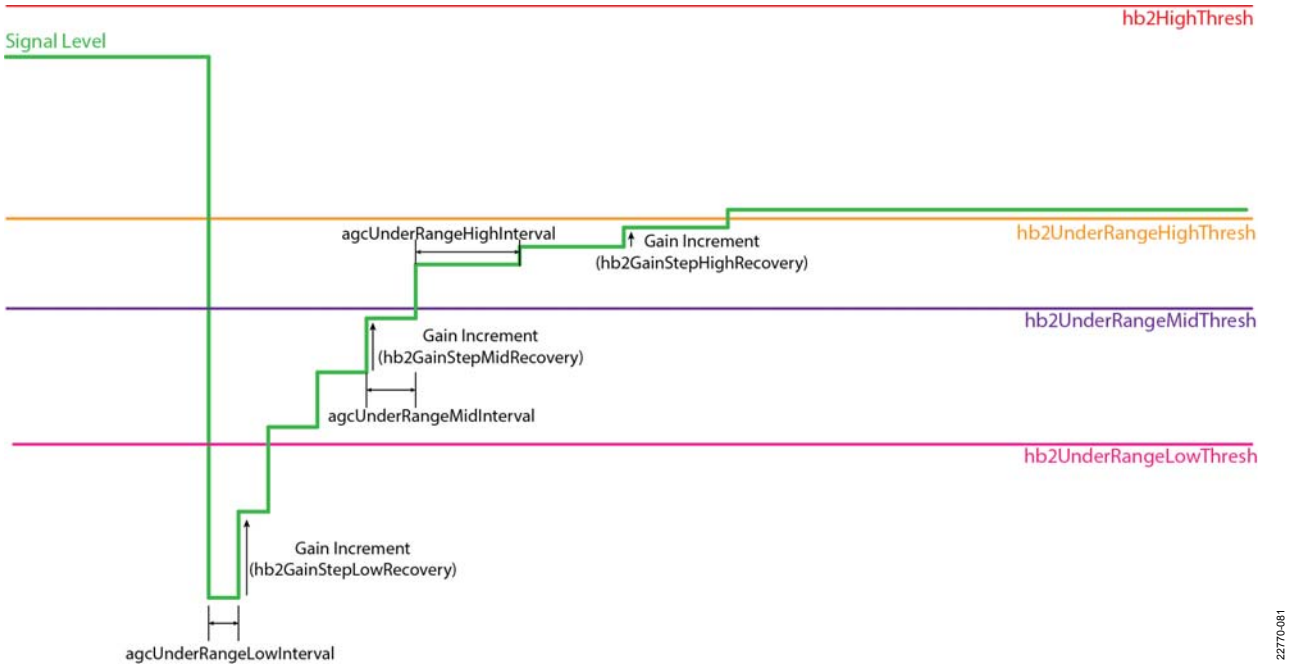


Figure 80. AGC operation with HB2 Detector in Fast Recovery Mode

Priorities and Overall Operation

It is highly recommended that the apdHighThresh and hb2HighThresh are set to an equivalent dBFS value. Likewise, it is highly recommended that the apdLowThresh and the hb2UnderRangeHighThresh are set to equivalent values. This equivalence is approximate because these thresholds have unique threshold settings that are not exactly equal. This section discusses the relevant priorities between the detectors and how the AGC reacts when multiple threshold detectors have been exceeded. Table 162 shows the priorities between the detectors when multiple overranges occur.

Table 162. Priorities of Attack Gain Steps

apdHighThresh Over Range	hb2HighThresh Over Range	Gain Change
No	No	No Gain Change
No	Yes	Gain Change by hb2GainStepAttack
Yes	No	Gain Change by apdGainStepAttack
Yes	Yes	Gain Change by apdGainStepAttack

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

1. hb2UnderRangeLowThresh underrange condition
2. hb2UnderRangeMidThresh underrange condition
3. hb2UnderRangeHighThresh underrange condition
4. apdLowThresh underrange condition

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

The apdLowThresh has priority in terms of preventing recovery. If apdLowThresh reports an over range condition (sufficient signal peaks have exceeded its threshold in a gain update counter period), then no further recovery is allowed. Configure apdLowThresh and hb2UnderRangeHighThresh to be as close to the same value of dBFS. However, assuming some small difference between the thresholds, then as soon as apdLowThresh is exceeded, recovery no longer occurs. The reverse is not true, hb2UnderRangeHighThresh does not prevent the gain recovery towards the apdLowThresh. Given the strong recommendation that apdLowThresh and hb2UnderRangeHighThresh be set equally, then a condition whereby apdLowThresh is at a lower dBFS level to hb2UnderRangeLowThresh or hb2UnderRangeMidThresh does 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 81 provides a flow diagram of the decisions of the AGC when recovering the gain in peak detect mode.

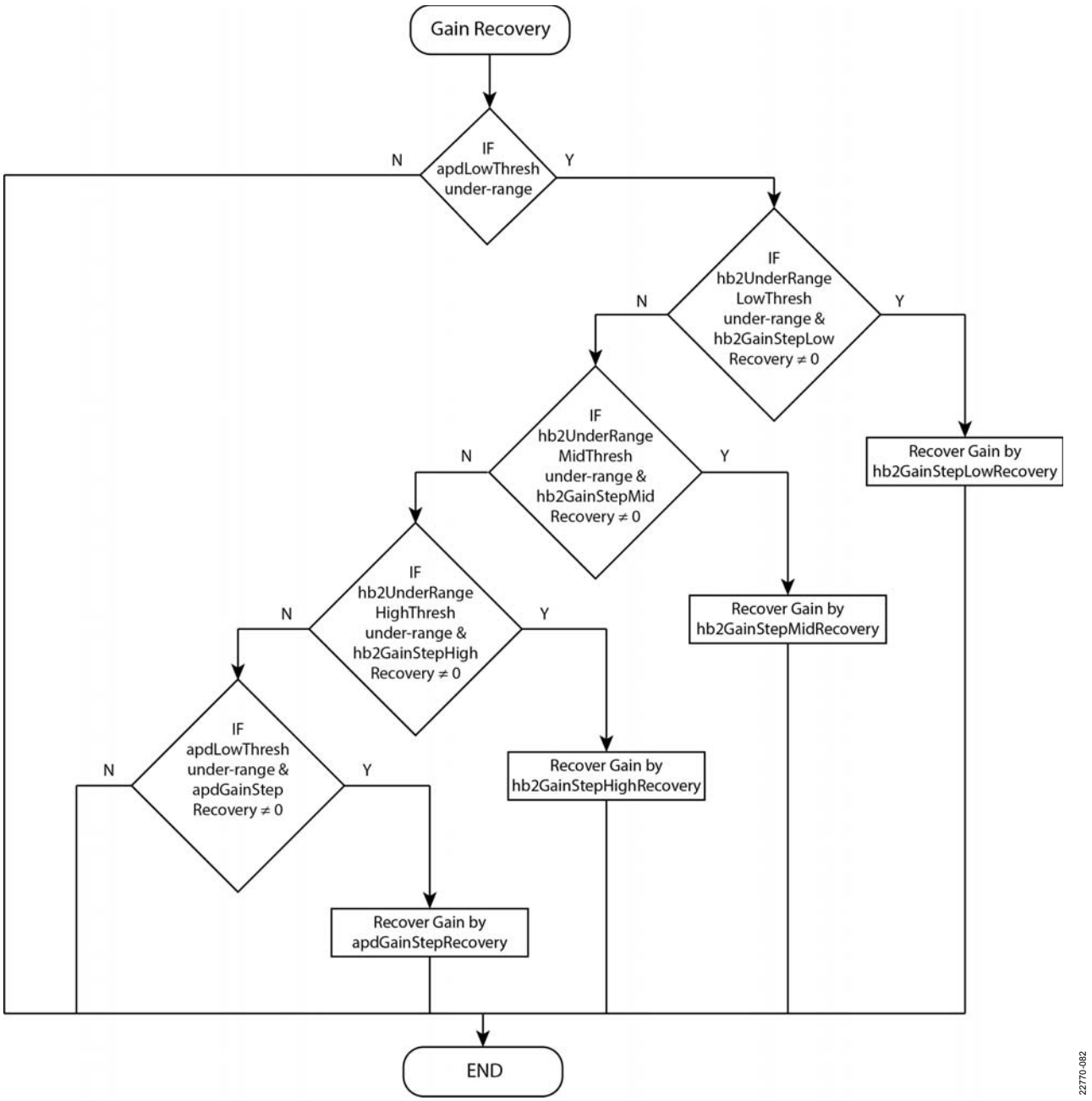


Figure 81. Flow Diagram for AGC Recovery in Peak Detect AGC Mode

Power Detect Mode

In this mode, the power detector measurement is also used to control the gain of the Rx chain. In the event of an over-range condition, then both the peak detectors and the power detector can instantiate a gain decrement. In the event of an under-range, only the power detector can increment the gain. The power detector changes gain solely at the expiry of the gain update counter. The peak detectors can be set in one of two modes (depending on the setting of agcGainChangeIfThreshHigh) whereby the AGC waits for the gain update counter to expire before initiating a gain change, or immediately updates the gain as soon as the overrange condition occurs (see Figure 75 to Figure 80).

The power measurement block provides the RMS power of the receiver data at the measurement location. It can be configured to monitor the signal in one of three locations as shown in Figure 72. In power detect mode, the AGC compares the measured signal level to programmable thresholds which provide a second-order control loop, whereby gain can be changed by larger amounts when the signal level is further from the target level, and make smaller gain changes when the signal is closer to the target level.

Figure 82 shows the operation of the AGC when using the power measurement detector. Considering the power measurement detector in isolation from the peak detectors, the AGC does not modify the gain when the signal level is between overRangeLowPowerThresh and underRangeHighPowerThresh. This range is the target range for the power measurement.

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.

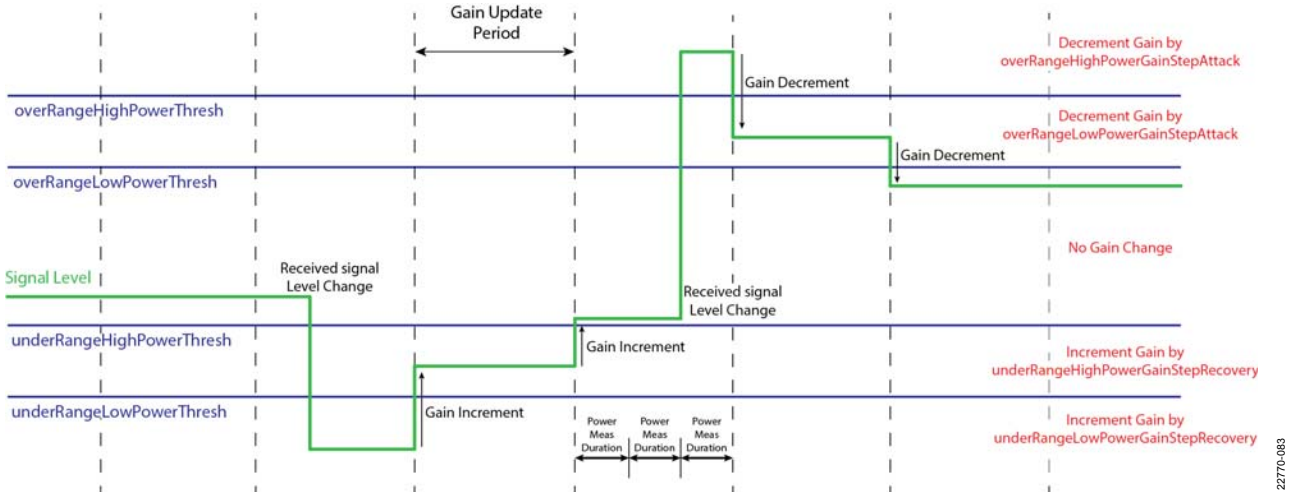


Figure 82. PMD Thresholds and Gain Changes for Under-range and Over-range Conditions

It is possible for the AGC to get contrasting requests from the power and peak detectors. An example is a blocker that is visible to the analog peak detector but is quite significantly attenuated by the power measurement block. In this case the APD can be requesting a gain decrement, while the power measurement block can be requesting a gain increment. The AGC has the following priority scheme in power detect mode:

1. APD Overrange (upper level)
2. HB2 Overrange (upper level)
3. APD lower level peak exceeded
4. HB2 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. Of note are the APD and HB2 lower level overloads. In peak detect mode, the lower level thresholds for these detectors were used to indicate an under-range condition which caused the AGC to increase the gain. In power detect more, these detectors are not used for gain recovery, but can be used to control gain recovery by setting the API parameter, agcLowThreshPreventGain. If this parameter is set, and if the signal level is exceeding a lower level threshold, the AGC is prevented from increasing the gain regardless of the power measurement.

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

AGC CLOCK AND GAIN BLOCK TIMING

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

The AGC state machine contains 3 states: Gain Update Counter, followed by the Slow Loop Settling (SLS) delay, and a constant 5 AGC clock cycles delay. The total time between gain updates (gain update period) is a combination of agcSlowLoopSettlingDelay and 5 AGC clock cycles.

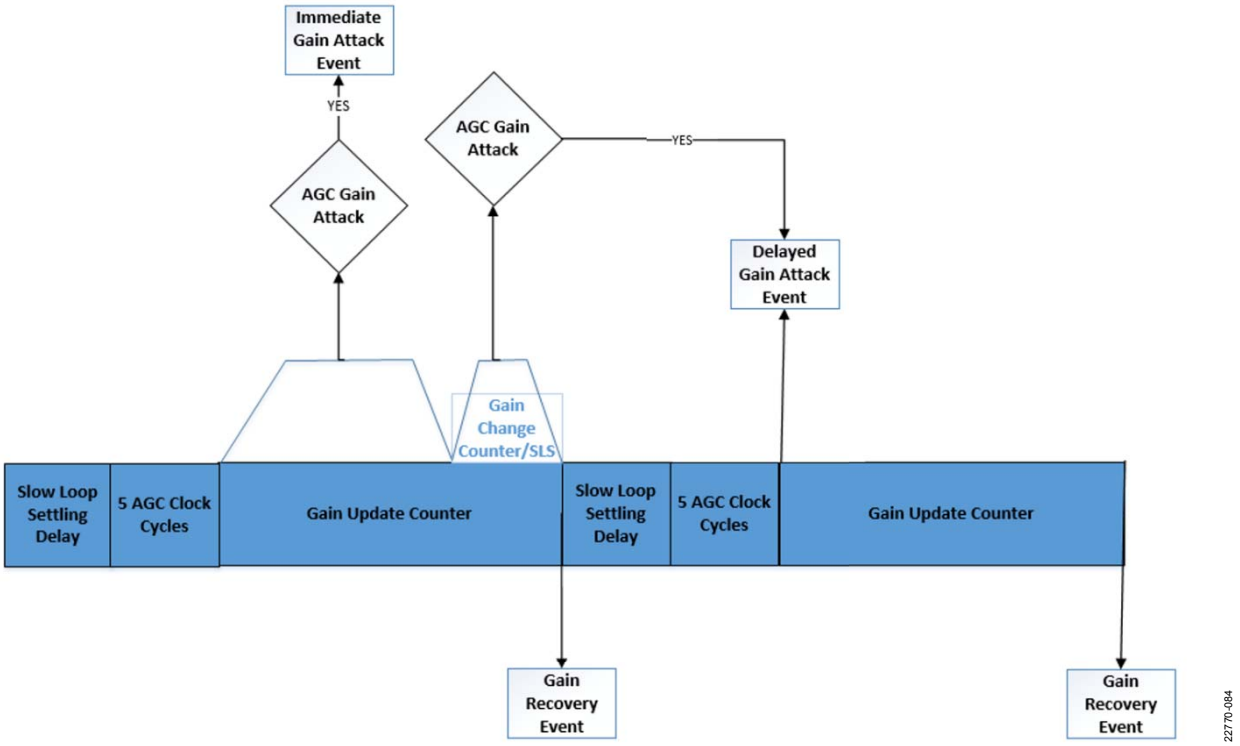


Figure 83. Gain Update Period

Figure 83 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 slow loop settling (SLS) is typically several orders of magnitude smaller than gain update counter, this is the most common gain decrement scenario.
- AGC Gain Attack within gain update counter, but within a SLS delay before the gain update counter expiry. This is a special case, but rarely occurs in applications per the reasoning described in the previous scenario).
- AGC Gain Recovery at the end of the gain update counter. Note that when fast recovery is enabled, the gain update counter is substituted with the low under range interval, per Figure 79.

A gain attack may occur within the gain update counter period when fast attack is enabled. A gain recovery event may only occur at the end of gain update counter period. After a gain attack, a gain change counter with a value equal to the SLS delay is started. No further gain attacks are possible while in this counter is running. This allows the minimum time to be set 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. Therefore if a gain attack occurred very close to end of the gain update counter state, the gain change counter delays the start of the SLS state and shift the gain recovery event. To prevent this happening and maintain a perfectly periodic gain recovery event, gain attacks are prevented from happening towards the end of gain update counter state as shown in Figure 83. 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 delay, therefore it is recommended to keep 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.

At the expiry of the gain update counter, all measurement blocks are reset and any peak detector counts is reset back to zero. When the Rx is enabled, the counter begins. This may mean that its expiry is at an arbitrary phase to the slot boundaries of the signal. The expiry of the counter can be aligned to the slot boundaries by setting the parameter agcEnableSyncPulseForGainCounter. While this bit is set, the AGC monitors a GPIO pin to find a synchronization pulse. This pulse causes the reset of the counter at this point in time. Therefore, if the user supplies a GPIO pulse time aligned to these slot boundaries, the expiry of the counter is aligned to slot boundaries. Any of GPIO_0-15 can be used for this purpose.

For example, considering 100 µs gain update period and a 491.52 MHz AGC clock, a total of 49,152 AGC clocks exist in the gain update period:

Gain Update Period (AGC Clocks) = 491.52 MHz ×100 µs = 49,152

As noted, the full gain update period is the sum of the agcGainUpdateCounter, the agcSlowLoopSettlingDelay and 5 clock cycles. If the agcSlowLoopSettlingDelay is set to 4, the gain update counter must be set to 49,139.

Gain Update Period (AGC Clocks) = *agcGainUpdateCounter* ×2(*agcSlowLoopSettingDelay*) + 5

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

When Rx is enabled, the AGC can be kept inactive for a number of AGC clock cycles by using agcRxAttackDelay. This means the user can specify one delay for AGC reaction when entering Rx mode, and another for after a gain change occurs (agcSlowLoopSettlingDelay).

ANALOG PEAK DETECTOR (APD)

The analog peak detector is located in the analog domain following the TIA filter and prior to the ADC input (see Figure 72). It functions by comparing the signal level to programmable thresholds. When a threshold has been exceeded a programmable number of times in a gain update period, then the detector flags that the threshold has been overloaded.

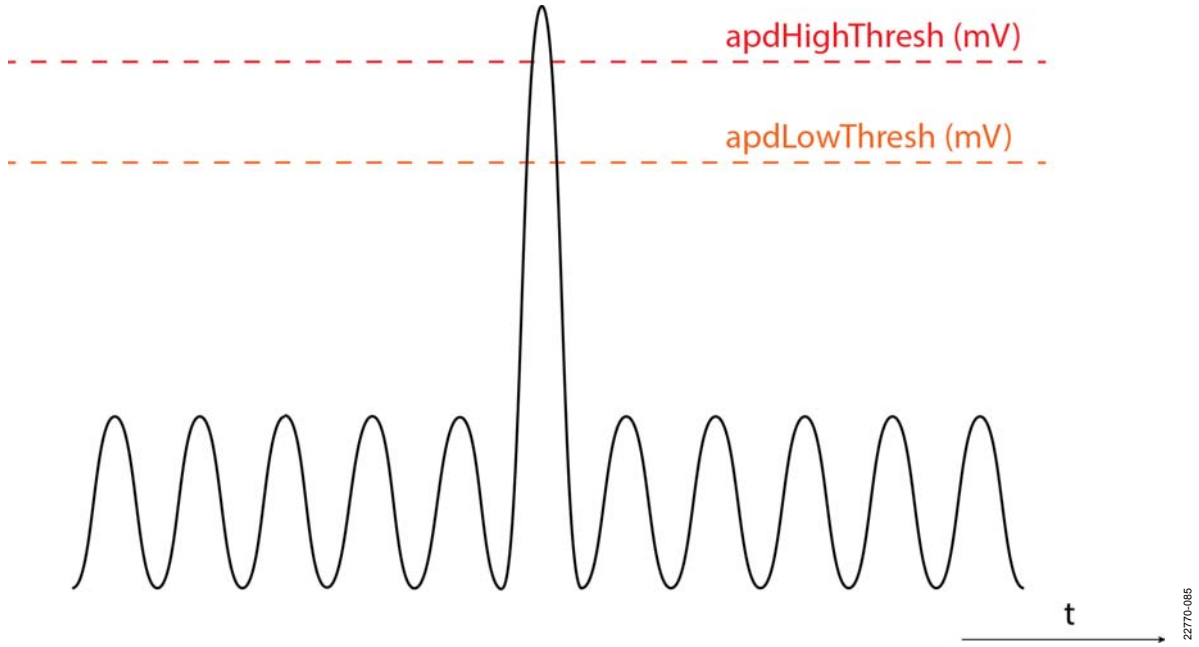


Figure 84. Analog Peak Detector Thresholds

There are two APD thresholds as shown in Figure 84. These thresholds are contained in the agcPeak API structure, apdHighThresh and apdLowThresh, respectively.

To determine the setting of the APD thresholds in terms of the closest possible setting in terms of dBFS of the ADC (ADCdBFS), the following equations can be used:

$$apdHighThresh = \text{round}\left(\frac{850\times10\frac{ADCdBFS}{20}-16}{16}\right)$$

$$apdLowThresh = \text{round}\left(\frac{850\times10\frac{ADCdBFS}{20}-16}{16}\right)$$

Note that the APD is an analog circuit located after the TIA filter. The equations above assume that the TIA does not attenuate the signal, but the receiver path is typically configured to have some analog roll-off within the pass band compensated by the programmable FIR filter. The TIA provides filtering that attenuates the signal seen at the APD, which means that a larger signal is required to assert the APD. There is a known issue with the APD where it is more sensitive to signals near dc (<5 MHz, generally). This increased sensitivity (typically on the order of 1 dB to 2 dB) is accounted for with the introduction of a secondary digital threshold that prevents the APD from making a gain change when the input signal is detected in-band. This prevents the sensitivity from causing unnecessary changes to the gain index. The APD acts mostly as an out-of-band blocker detector.

The APD threshold must be exceeded a programmable number of times within a gain update counter period before an over range condition occurs. Both the upper and lower thresholds have a programmable counter in the agcPeak API structure, as indicated in Table 163.

Table 163. APD Programmable Threshold Counters

Threshold	Counter
Upper Threshold (apdHighThresh)	apdUpperThreshPeakExceededCnt
Lower Threshold (apdLowThresh)	apdLowerThresPeakExceededCnt

As described in the earlier section on AGC control, the APD is used for both gain attack and gain recovery in peak detect mode. In power detect mode, the APD is used for gain attack, and is used to prevent overloading during gain recovery. For more details, refer to the relevant sections of this document.

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

Table 164. APD Attack and Recovery Step Sizes

Gain Change	Step Size
Gain Attack	apdGainStepAttack
Gain Recovery	apdGainStepRecovery

Step size refers to the number of indices of the gain table the gain is changed. As explained 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 the APD gain attack step size was programmed to 6, then this means that the gain index is reduced by 6 when the apdHighThresh has been exceeded more than apdUpperThreshPeakExceededCnt times. For example, if the gain index had been 255 before this over range condition, then the gain index is reduced 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 over range condition.

The APD is held in reset for a configurable amount of time following a gain change to ensure that the receiver path is settled at the new gain setting before monitoring the paths for over-ranges. This is configured using the agcPeakWaitTime API parameter.

HALF-BAND 2 PEAK DETECTOR

The HB2 peak detector is located in the digital domain at the output of the Half-Band filter 2. 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 $I^2 + Q^2$ samples (or peak I and peak Q if hb2OverloadPowerMode = 0) 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. The duration of the HB2 measurement is controlled by hb2OverloadDurationCnt, whereas the number of samples that exceeds the threshold in that period is controlled by hb2OverloadThreshCnt.

Once the required number of samples exceed the threshold in the duration required, then the detector records that the threshold was exceeded. Like the APD detector, the HB2 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 85 shows the two-level approach. It shows the gain update counter period, with the time being broken into subsets of time based on the setting of hb2OverloadDurationCnt. Each of these periods of time is considered separately, and hb2OverladThreshCnt individual samples must exceed the threshold within hb2OverloadDurationCnt for an overload to be declared. These individual samples greater than the threshold are shown in red, while those less than the threshold are shown in green. Two examples are shown, one where the number of samples exceeding the threshold was sufficient for the HB2 peak detector to declare an overload (this time period is shown as red in the gain update counter timeline), and a second example where the number of samples exceeding the threshold was not sufficient to declare an overload (this time period is shown as green in the gain update counter timeline). The number of overloads are counted, and if the number of overloads of the hb2HighThresh exceed hb2UpperThreshPeakExceededCnt in a gain update counter period, then an over range condition is called. Likewise, if the number of overloads of the hb2UnderRangeHighThresh does not exceed hb2LowerThreshPeakExceededCnt, then an under-range condition is called.

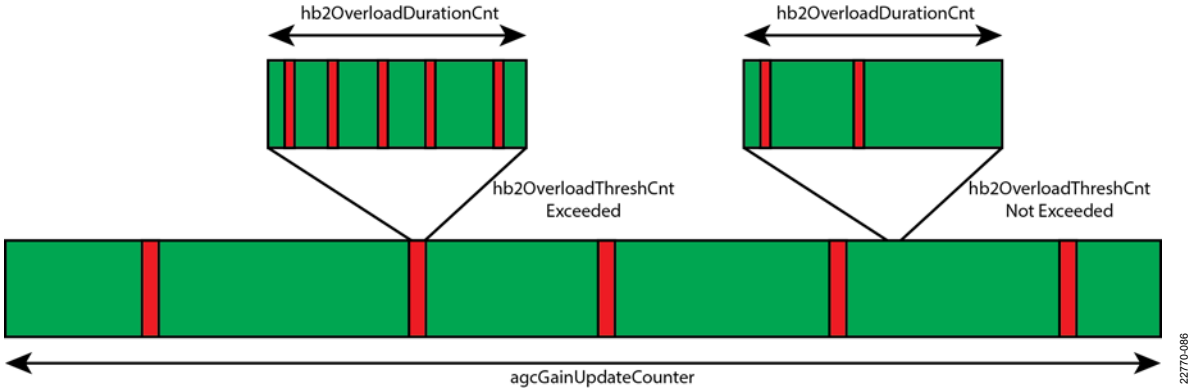


Figure 85. HB2 Detector Two-Level Approach for an Overload Condition

The HB2 detector has a number of programmable thresholds. Some of these thresholds are only used in the fast recovery mode of the peak detect AGC configuration, as summarized in Table 165.

Table 165. HB2 Overload Thresholds

HB2 Threshold	Usage
hb2HighThresh	Used for gain attack in both peak and power detect AGC modes.
hb2UnderRangeHighThresh	Used for gain recovery in peak detect AGC mode. In power detect AGC mode it is used to prevent overloads during gain recovery.
hb2UnderRangeMidThresh	Used only when the fast recovery option of the peak detect AGC mode is being utilized.
hb2UnderRangeLowThresh	Used only when the fast recovery option of the peak detect AGC mode is being utilized.

For more details of how these thresholds are used by the AGC, refer to the relevant sections of the AGC overview in this document (specifically Figure 76, Figure 78 and Figure 80).

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

$$hb2HighThresh = 16,384 \times 10^{\left(\frac{hb2HighdBFS}{20}\right)}$$

$$hb2UnderRangeHighThresh = 16,384 \times 10^{\left(\frac{hb2UnderRangeHighdBFS}{20}\right)}$$

$$hb2UnderRangeMidThresh = 16,384 \times 10^{\left(\frac{hb2UnderRangeMiddBFS}{20}\right)}$$

$$hb2UnderRangeLowThresh = 16,384 \times 10^{\left(\frac{hb2UnderRangeLowdBFS}{20}\right)}$$

Each threshold has an associated counter such that an over-range condition is not flagged until the threshold has been exceeded this amount of times in a gain update period.

Table 166. Gain Steps for HB2 Over-range and Underrange Conditions

HB2 Threshold	Counter
hb2HighThresh	hb2UpperThreshPeakExceededCnt
hb2UnderRangeHighThresh	hb2UnderRangeHighThreshExceededCnt
hb2UnderRangeMidThresh	Hb2UnderRangeMidThreshExceededCnt
hb2UnderRangeLowThresh	Hb2UnderRangeLowThreshExceededCnt

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

Table 167. HB2 Attack and Recovery Step Sizes

Gain Change	Step Size
Gain Attack	hb2GainStepAttack
Gain Recovery (hb2UnderRangeHighThresh)	hb2GainStepHighRecovery
Gain Recovery (hb2UnderRangeMidThresh)	hb2GainStepMidRecovery
Gain Recovery (hb2UnderRangeLowThresh)	hb2GainStepLowRecovery

The HB2 peak detector is held in reset for a configurable amount of time following a gain change to ensure that the receiver path is settled at the new gain setting before monitoring the paths for over-range conditions. This duration is configured using the agcPeakWaitTime API parameter.

POWER DETECTOR

The power measurement block measures the RMS power of the incoming signal. It can monitor the signal level at different locations, namely the HB2 output, the RFIR output and the output of the dc correction block. To choose a location, the powerInputSelect API parameter is utilized as described in Table 168.

Table 168. Location of the Decimated Power Measurement

powerInputSelect	Value
RFIR Output	0
HB1	1
HB2	2

The number of samples that are used in the power measurement calculation is configurable using the powerMeasurementDuration API parameter:

$$PowerMeasDuration\ (Rx\ Sample\ Clocks) = 8 \times 2^{powerMeasurementDuration}$$

where *Rx Sample Clocks* is the number of clocks at the power measurement location. It is important that this duration not exceed the gain update counter. The gain update counter resets the power measurement block and therefore a valid power measurement must be available before this event. In the case of multiple power measurements occurring in a gain update period, the AGC uses the last fully completed power measurement, any partial measurements being discarded.

The power measurement block has a dynamic range of 60 dB by default. Power measured in the receiver data path can be readback with the following command.

adi_adrv9025_RxDecPowerGet

```
adi_adrv9025_RxDecPowerGet(adi_adrv9025_Device_t* device, adi_adrv9025_RxChannels_e rxChannel,
uint16_t *rxDecPower_mdBFS)
```

Description

Readback for receiver power measurement.

Parameters

Table 169.

Parameter	Description
*device	Pointer to device structure.
rxChannel	An enum indicating which Rx channel to configure as shown in Table 155.
*rxDecPower_mdBFS	pointer to the variable through which the power measurement reading is returned.

API PROGRAMMING

The API programming sequence for the gain control blocks is detailed in Figure 86. The configuration of these blocks is one of the last steps before making the device operational. The structures are defined before initialization of the device begins. Once device initialization has proceeded up to the configuration of the JESD, then the gain control configuration begins.

The following API is used to configure the gain control blocks within the device such as the peak detectors, the power detector, and the AGC if used. It is required to call this command in applications that require AGC.

adi_adrv9025_AgcCfgSet

```
adi_adrv9025_AgcCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_AgcCfg_t agcConfig[], uint8_t
arraySize)
```

Description

Configures the gain control blocks within the device such as peak detectors, power detector, and AGC settings.

Parameters

Table 170.

Parameter	Description
*device	Pointer to device structure.
agcConfig	An array of gain control configuration structures of type adi_adrv9025_AgcCfg_t.
arraySize	The number of configuration structures in agcConfig[].

The composition of the gain control configuration structure is detailed in the next section. Once agcConfig[] has been configured, the desired gain control mode can be enabled by using the adi_adrv9025_RxGainCtrlModeSet( ) API function, which was detailed earlier in this user guide.

The final step is to configure any GPIOs as necessary, be it monitor outputs which allow real-time monitoring of the peak detector outputs, or GPIO inputs which allow the AGC gain update counter to be synchronized to a slot boundary, or GPIOs to directly control the gain index. The operation of these has been described above.

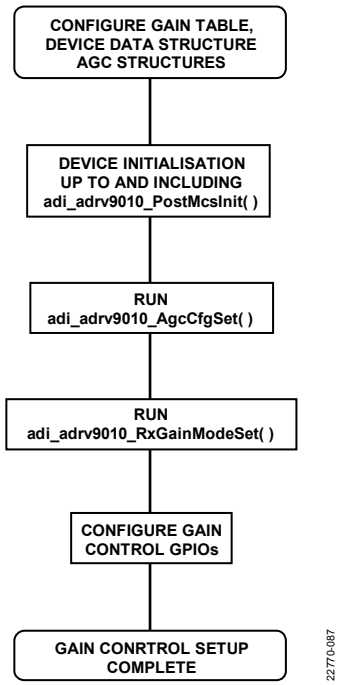


Figure 86. Gain Control Programming Flowchart

AGC HOLDOVER FUNCTION

The ADRV9026 AGC uses counters to keep track off any over range or under range events. These events are used to increment a counter that accumulates and triggers the AGC state machine if it exceeds the desired count value. For a TDD case, the counters get reset every time the Rx enable goes low. This reset of the over range and under range counters can potentially cause the AGC state machine to never trigger if the gainUpdateCounter is larger than the Rx TDD slot duration. The AGC holdover function has been implemented to avoid this situation by preventing the counters from getting reset when Rx enable is toggled.

To enable this function, the user needs to create a stream file using the Transceiver Evaluation Software with the AGC state persist box checked in the Stream Settings section as shown in Figure 89. Once this box is checked, a stream file can be created with the AGC holdover function enabled to prevent AGC counter resets during TDD operation.

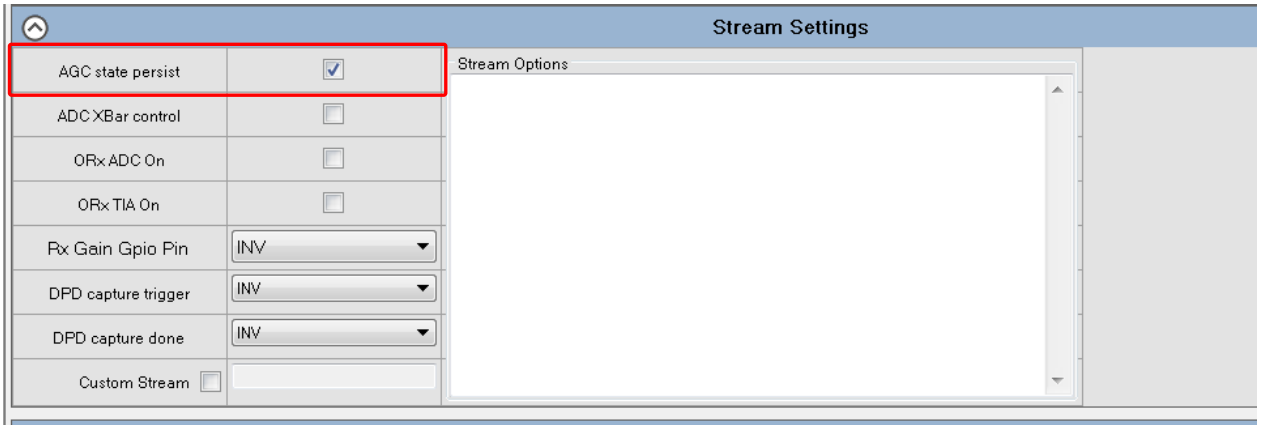


Figure 87. TES Stream Settings Control Window to Enable AGC Holdover

RX GAIN MODE SWITCHING USING GPIO

This feature allows use of a GPIO pin to force Rx gain index changes and move to MGC mode. This feature is beneficial if the user wants to run a quick RF calibration for the entire Rx signal chain. Such a cal requires a fixed Rx gain index, which is not possible to guarantee if the part is in AGC mode. The user can change the mode to MGC and then change the Rx gain index, but the duration of this switch is a few ms, which is not feasible in a 5G NR TDD platform.

When this feature is employed, the user can enable a GPIO pin to change the Rx gain index to a fixed predetermined value and move the Rx to MGC mode. This action sets the gain index and avoids the issue of the AGC state machine modifying the index. The user can then run the desired function (for example, RF calibration) and then toggle the GPIO low to restore the original Rx state. When the GPIO is low, the gain control mode is restored back to AGC to resume normal Rx operation.

To enable Rx GPIO gain mode switching, the use needs to create a stream file using the TES with the **Rx Gain Gpio Pin** set to the desired GPIO pin as shown in Figure 88.

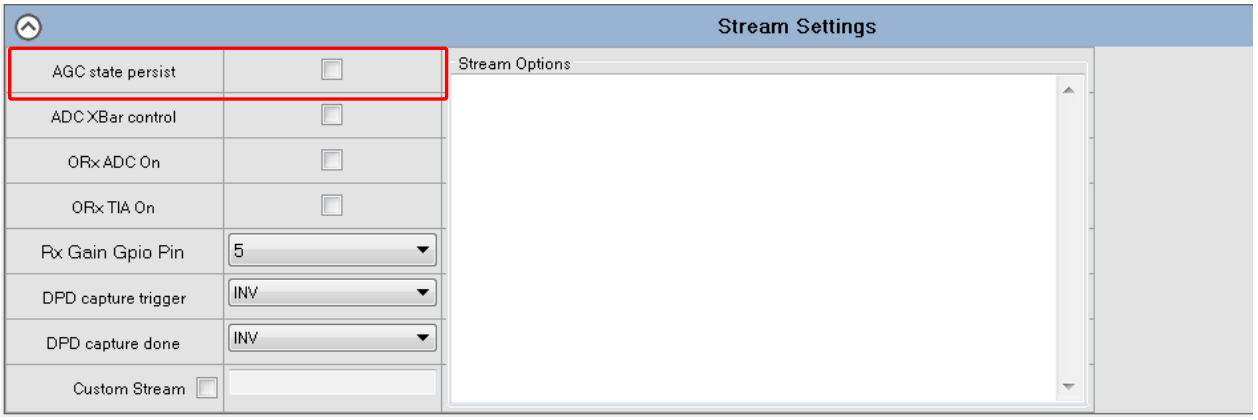


Figure 88. TES Stream Settings Control Window to Enable Rx Gain Mode Switching using GPIO

The user also needs to use the StreamGpioConfigSet API function to unmask the stream GPIO source so as to allow the stream to be triggered on the desired GPIO. The steps to set up this feature are the following:

1. Generate the stream with the correct GPIO set to the Rx Gain GPIO tag as shown in Figure 88.
2. Use StreamGpioConfigSet function (called during postMcsInit) with the correct GPIO pin selected as shown in the StreamGpioConfigSet Function section.
3. Set the Rx manual gain to the desired value to be used during the calibration.

By following these steps, the user can more Rx to MGC mode when the GPIO goes high and move back to AGC mode when the GPIO goes low. Not that this function affects all four Rx channels if utilized.

StreamGpioConfigSet Function

```
streamGpioCfg = Types.adi_adrv9025_StreamGpioPinCfg_t()

streamGpioCfg.streamGpInput0  = Types.adi_adrv9025_GpioPinSel_e. ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput1  = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput2  = Types.adi_adrv9025_GpioPinSel_e. ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput3  = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput4  = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput5  = Types.adi_adrv9025_GpioPinSel_e. ADI_ADRV9025_GPIO_05
streamGpioCfg.streamGpInput6  = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput7  = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput8  = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput9  = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput10 = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput11 = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput12 = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput13 = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput14 = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID
streamGpioCfg.streamGpInput15 = Types.adi_adrv9025_GpioPinSel_e.ADI_ADRV9025_GPIO_INVALID

link.platform.board.Adrv9025Device.RadioCtrl.StreamGpioConfigSet(streamGpioCfg)
```

GAIN CONTROL DATA STRUCTURES

Figure 89 shows the member structure of adi_adrv9025_AgcCfg_t, and of its substructures, adi_adrv9025_AgcPeak_t and adi_adrv9025_AgcPower_t. Each of the parameters are briefly explained in Table 171, Table 172, and Table 173, and the wider context of these parameter settings are outlined in the relevant gain control/peak detector sections.

Figure 89. Member Listing of adi_adrv9025_AgcCfg_t Data Structure

Table 171. ADRV9025_AgcCfg_t Structure Definition

Parameter	Description	Min Value	Max Value
rxChannelMask	This selects the channels upon which to enable this gain control mode. It is a bit mask with each bit corresponding to a channel, [D0] = Rx1, [D1] = Rx2, [D2] = Rx3, [D3] = Rx4. Therefore, setting the rxChannelMask = 15 means that all Rx channels are configured with the same AGC configuration.	0	15
agcPeakWaitTime	Number of AGC clock cycles to wait before enable peak/overload detectors after a gain change.	0	31
agcRxMaxGainIndex	Maximum gain index allowed in AGC mode. Must be greater than agcMinGainIndex and be a valid gain index.	0	255
agcRxMinGainIndex	Minimum gain index allowed in AGC mode. Must be less than agcRxMaxGainIndex and be a valid gain index.	0	255
agcGainUpdateCounter	Is used as a decision period, with the peak detectors reset on this period. Gain changes in AGC mode can also be synchronized to this period (the expiry of this counter). The full period is a combination of the agcGainUpdateCounter and agcSlowLoopSettlingDelay.	Depends on Overload Detector Settings	4194303 AGC_CLK Cycles
agcRxAttackDelay	Hold the duration the AGC should be held in reset when the Rx path is enabled.	0	63
agcSlowLoopSettlingDelay	Number of AGC clock cycles to wait after a gain change before the AGC changes gain again.	0	127
agcLowThreshPreventGain	Only relevant in Peak/Power Detect AGC operation. 1: If AGC is in Peak Power Detect Mode, then gain increments requested by the power detector are prevented if there are sufficient peaks (APD/HB2 Low Threshold Exceeded Count) above the apdLowThresh or hb2UnderRangeHighThresh. 0: apdLowThresh and hb2UnderRangeHighThresh are don't cares for gain recovery.	0	1
agcChangeGainIfThreshHigh	Applicable in both peak and peak/power detect modes. 0: Gain changes wait for the expiry of the gain update counter if a high threshold count has been exceeded on either the APD or HB2 detector. 1: Gain changes occur immediately when initiated by HB2. 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 HB2 want for the gain update to expire. 3: Gain changes occur immediately when initiated by APD or HB2 detectors.	0	3
agcPeakThreshGainControlMode	1: AGC in Peak AGC mode, power-based gain changes are disabled. 0: AGC in Peak/Power AGC mode where both Peak Detectors and Power Detectors are utilized.	0	1
agcResetOnRxon	1: AGC state machine is reset when Rx is disabled. The AGC gain setting is returned to the maximum gain. 0: AGC state machine maintains its state when Rx is disabled.	0	1
agcEnableSyncPulseForGainCounter	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
agcEnableFastRecoveryLoop	1: Enables the fast recovery AGC functionality using the HB2 overload detector. Only applicable in Peak Detect Mode. 0: AGC fast recovery is not enabled.	0	1
agcPower	Structure containing all the power detector settings.	N/A	N/A
agcPeak	Structure containing all the peak detector settings.	N/A	N/A

Table 172. ADRV9025_AgcPeak_t Structure Definition

Parameter	Description	Min Value	Max Value
agcUnderRangeLowInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hb2UnderRangeLowThresh. Only applicable when the fast recovery option is enabled in Peak Detect AGC mode.	Depends on HB2 detector settings	65535
agcUnderRangeMidInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hb2UnderRangeMidThresh. Calculated as (agcUnderRangeMidInterval + 1) × agcUnderRangeLowInterval Only applicable when the fast recovery option is enabled in Peak Detect AGC mode.	0	63
agcUnderRangeHighInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hb2UnderRangeHighThresh. Calculated as (agcUnderRangeHighInterval + 1) × agcUnderRangeMidInterval Only applicable when the fast recovery option is enabled in Peak Detect AGC mode.	0	63
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. *adpdHighThresh* (mV) = (*apdHighThresh* + 1) × 16 mV	apdLowThresh	63
apdLowGainModeHighThresh	This parameter is not utilized.		
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 Peak AGC mode, the gain is increased when this overload is not occurring. In Power AGC mode, this threshold can be used to prevent further gain increases if the agcLowThreshPreventGain bit is set. *adpdLowThresh* (mV) = (*apdLowThresh* + 1) × 16 mV	7	apdHighThresh
apdLowGainModeLowThresh	This parameter is not utilized.		
apdUpperThreshPeakExceededCnt	Sets number of peaks to detect above apdHighThresh to cause an APD High Over Range Event. In AGC modes, this results in a gain decrement set by apdGainStepAttack.	0	255
apdLowerThreshPeakExceededCnt	Sets number of peaks to detect above apdLowThresh to cause an APD Low Overload Event. In 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
apdGainStepAttack	The number of indices that the gain index pointer must be decreased in the event of an APD High Over Range 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
apdGainStepRecovery	The number of indices that the gain index pointer must be increased in the event of an APD Under range event occurring in 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
enableHb2Overload	1: HB2 Overload Detector enabled. 0: HB2 Overload Detector disabled	0	1
hb2OverloadDurationCnt	The number of clock cycles (at the HB2 output rate) within which hb2OverloadThreshCnt must be exceeded for an overload to occur. A HB2 overload flag is only raised when the number of these overloads exceeds hb2UpperThreshPeakExceededCnt or hb2LowerThreshPeakExceededCnt within a gain update period. The number of clocks is: $2^{(hb2OverloadDurationCnt + 1)}$	0	6
hb2OverloadThreshCnt	Sets the number of individual samples exceeding hb2HighThresh or hb2LowThresh necessary within hb2OverloadDurationCnt for an overload to occur. The HB2 overload flag is only raised when the number of these overloads exceeds hb2UpperThreshPeakExceededCnt or hb2LowerThreshPeakExceededCnt within a gain update period.	1	15

Parameter	Description	Min Value	Max Value
hb2HighThresh	This sets the upper threshold of the HB2 detector. $hb2HighThresh = 16,384 \times 10^{\left(\frac{hb2HighdBFS}{20}\right)}$	0	16383
hb2UnderRangeLowThresh	This sets the lower threshold of the HB2 under range threshold detectors. Used only when the fast recovery option of the peak detect AGC mode is being utilized. $hb2UnderRangeLowThresh = 16,384 \times 10^{\left(\frac{hb2UnderRangeLowdBFS}{20}\right)}$	0	16383
hb2UnderRangeMidThresh	This sets the middle threshold of the HB2 under range threshold detectors. Used only when the fast recovery option of the peak detect AGC mode is being utilized. $hb2UnderRangeMidThresh = 16,384 \times 10^{\left(\frac{hb2UnderRangeMiddBFS}{20}\right)}$	0	16383
hb2UnderRangeHighThresh;	Peak Detect Mode: Threshold used for gain recovery. Peak Detect with Fast Recovery Mode: This sets the highest threshold of the HB2 under range threshold detectors. Power Detect Mode: Threshold used to prevent further gain increases if agcLowThreshPreventGain is set. $hb2UnderRangeHighThresh = 16,384 \times 10^{\left(\frac{hb2UnderRangeHIghdBFS}{20}\right)}$	0	16383
hb2UpperThreshPeakExceededCnt	Sets number of individual overloads above hb2HighThresh (number of times hb2OverloadThreshCnt was exceeded in hb2OverloadDuractionCnt) to cause an HB2 High Over Range event. In AGC modes, this results in a gain decrement set by hb2GainStepAttack.	0	255
hb2UnderRangeHighThreshExceededCnt	Sets number of individual overloads above hb2UnderRangeHighThresh (number of times hb2OverloadThreshCnt was exceeded in hb2OverloadDurationCnt) to cause an HB2 Under Range High Threshold Overload Event. In Peak Detect AGC mode, not having sufficient peaks to cause the overload is flagged as an underrange event and the gain is recovered by hb2GainStepHighRecovery.	0	255
hb2GainStepHighRecovery	The number of indices that the gain index pointer must be increased in the event of an HB2 Under Range High Threshold Under Range Event.	0	31
hb2GainStepLowRecovery	Only applicable in fast recovery mode of peak detect AGC. This sets the number of indices that the gain index pointer must be increased in the event of an HB2 Under Range Low Threshold Under Range Event.	0	31
hb2GainStepMidRecovery	Only applicable in fast recovery mode of peak detect AGC. This sets the number of indices that the gain index pointer must be increased in the event of an HB2 Under Range Mid Threshold Under Range Event.	0	31
hb2GainStepAttack	The number of indices that the gain index pointer must be decreased in the event of an HB2 High Threshold Over Range event 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
hb2OverloadPowerMode	Sets the measurement mode of the HB2 detector.	0	1
hb2ThreshConfig	Set to 3.	3	3
hb2UnderRangeMidThreshExceededCnt	Only applicable in fast recovery mode of peak detect AGC. Sets number of individual overloads above hb2UnderRangeMidThresh (number of times hb2OverloadThreshCnt was exceeded in hb2OverloadDurationCnt) to cause an HB2 Under Range Mid Threshold Overload Event. In Peak Detect AGC mode, not having sufficient peaks to cause the overload is flagged as an under-range event and the gain is recovered by hb2GainStepMidRecovery.	0	255

Parameter	Description	Min Value	Max Value
hb2UnderRangeLowThreshExceededCnt	Only applicable in fast recovery mode of peak detect AGC. Sets number of individual overloads above hb2UnderRangeLowThresh (number of times hb2OverloadThreshCnt was exceeded in hb2OverloadDurationCnt) to cause an HB2 Under Range Low Threshold Overload Event. In Peak Detect AGC mode, not having sufficient peaks to cause the overload is flagged as an under-range event and the gain is recovered by hb2GainStepLowRecovery.	0	255

Table 173. ADRV9025_AgcPower_t Structure Definition

Parameter	Description	Min Value	Max Value
powerEnableMeasurement	1: Power Measurement block enabled. 0: Power Measurement block disabled.	0	1
powerInputSelect	This parameter sets the location of the power measurement. 0 = RFIR output; 1 = HB1 Output, 2 = HB2 Output.	0	3
underRangeHighPowerThresh	Threshold in dBFS (negative sign assumed) which defines the lower boundary on the stable region of the power detect gain control mode.	0	127
underRangeLowPowerThresh	Offset (negative sign assumed) from underRangeHighPowerThresh which defines the outer boundary of the power based AGC convergence. Typically, recovery is set to be larger steps than when the power measurement is less than this threshold.	0	31
underRangeHighPowerGainStepRecovery	The number of indices that the gain index pointer must be increased (gain increase) in the event of the power measurement being less than underRangeHighPowerThresh but greater than underRangeLowPowerThresh.	0	31
underRangeLowPowerGainStepRecovery	The number of indices that the gain index pointer must be increased (gain increase) in the event of the power measurement being less than underRangeLowPowerThresh.	0	31
powerMeasurementDuration	Number of IQ samples on which to perform the power measurement. The number of samples corresponding to the 4-bit word is $8 \times 2^{(pmdMeasDuration[3:0])}$. This value must be less than AGC Gain Update Counter.	0	31
rxTddPowerMeasDuration	Following an Rx Enable, the power measurement block can be requested to perform a power measurement for a specific period of a frame. This is 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.	0	65535 AGC clock cycles
rxTddPowerMeasDelay	Following an Rx Enable, the power measurement block can be 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 Rx1.	0	65535 AGC clock cycles
overRangeHighPowerThresh	Threshold in dBFS (negative sign assumed) which defines the upper boundary on the stable region (no gain change based on power measurement) of the power detect gain control mode.	0	127
overRangeLowPowerThresh	Offset (positive sign assumed) from upper0PowerThresh which defines the outer boundary of the power based AGC convergence. Typically attack is set to be larger steps than when the power measurement is greater than this threshold.	0	15
powerLogShift	Enable increase in dynamic range of the power measurement from 40 dB to ~60 dB.	0	1
overRangeHighPowerGainStepAttack	The number of indices that the gain index pointer must be decreased (gain reduction) in the event of the power measurement being greater than overRangeHighPowerThresh.	0	31
overRangeLowPowerGainStepAttack	The number of indices that the gain index pointer must be decreased (gain decrease) in the event of the power measurement being less than OverRangeHighPowerThresh but greater than OverRangeLowPowerThresh.	0	31

SAMPLE PYTHON SCRIPT—PEAK DETECT MODE WITH FAST ATTACK

The following is a sample python script to enable the AGC in peak detect mode. The user can use this sample script as a starting point to enable AGC on the evaluation platform.

```
#Import Reference to the DLL
import System
import clr
from System import Array
clr.AddReferenceToFileAndPath("C:\\Program Files (x86)\\Analog Devices\\ADRV9025 Transceiver Evaluation Software\\adrv9025_dll.dll")
from adrv9025_dll import AdiEvaluationSystem
from adrv9025_dll import Types

#Create an Instance of the Class
Link = AdiEvaluationSystem.Instance
connect = False

if (Link.IsConnected() == False):
    connect = True
    Link.Ads8.board.Client.Connect("192.168.1.10", 55556)
    print "Connecting"

if (Link.IsConnected()):
    adrv9025 = Link.Adrv9025Get(1)

    # Create an instance of the rxGainMode , agcConfig classes
    rxGainMode = Types.adi_adrv9025_RxAgcMode_t()
    agcConfig = Types.adi_adrv9025_AgcCfg_t()

    # General Rx Gain Mode Configuration
    rxGainMode.rxChannelMask = 0xF
    rxGainMode.agcMode = Types.adi_adrv9025_RxAgcMode_e.ADI_ADRV9025_AGCSLOW

    # General AGC Configuration
    agcConfig.rxChannelMask = 0xF
    agcConfig.agcPeakWaitTime = 4
    agcConfig.agcRxMaxGainIndex = 255
    agcConfig.agcRxMinGainIndex = 195
    agcConfig.agcGainUpdateCounter = 921600
    agcConfig.agcRxAttackDelay = 10
    agcConfig.agcSlowLoopSettlingDelay = 16
    agcConfig.agcLowThreshPreventGainInc = 1
    agcConfig.agcChangeGainIfThreshHigh = 1
    agcConfig.agcPeakThreshGainControlMode= 1
    agcConfig.agcResetOnRxon = 0
    agcConfig.agcEnableSyncPulseForGainCounter = 0
    agcConfig.agcEnableFastRecoveryLoop = 0
```

```
#adi_adrv9025_AgcPeak_t agcPeak;
agcConfig.agcPeak.agcUnderRangeLowInterval = 205000 / 245;
agcConfig.agcPeak.agcUnderRangeMidInterval = 2;
agcConfig.agcPeak.agcUnderRangeHighInterval = 4;
agcConfig.agcPeak.apdHighThresh = 38;
agcConfig.agcPeak.apdLowThresh = 25;
agcConfig.agcPeak.apdUpperThreshPeakExceededCnt = 10;
agcConfig.agcPeak.apdLowerThreshPeakExceededCnt = 3;
agcConfig.agcPeak.enableHb2Overload = 1;
agcConfig.agcPeak.hb2OverloadDurationCnt = 1;
agcConfig.agcPeak.hb2OverloadThreshCnt = 1;
agcConfig.agcPeak.hb2HighThresh = 11598; #-3dBFS
agcConfig.agcPeak.hb2UnderRangeLowThresh = 8211;
agcConfig.agcPeak.hb2UnderRangeMidThresh = 5813;
agcConfig.agcPeak.hb2UnderRangeHighThresh = 2913;
agcConfig.agcPeak.hb2UpperThreshPeakExceededCnt = 10;
agcConfig.agcPeak.hb2UnderRangeHighThreshExceededCnt = 3;
agcConfig.agcPeak.hb2UnderRangeMidThreshExceededCnt = 3;
agcConfig.agcPeak.hb2UnderRangeLowThreshExceededCnt = 3;
agcConfig.agcPeak.hb2OverloadPowerMode = 0;
agcConfig.agcPeak.hb2ThreshConfig = 3;

agcConfig.agcPeak.apdGainStepAttack = 4;
agcConfig.agcPeak.apdGainStepRecovery = 2;
agcConfig.agcPeak.hb2GainStepAttack = 4;
agcConfig.agcPeak.hb2GainStepHighRecovery =2;
agcConfig.agcPeak.hb2GainStepMidRecovery = 4;
agcConfig.agcPeak.hb2GainStepLowRecovery = 8;

#adi_adrv9025_AgcPower_t agcPower;
agcConfig.agcPower.powerEnableMeasurement = 0;
agcConfig.agcPower.powerInputSelect = 0;
agcConfig.agcPower.underRangeHighPowerThresh = 9;
agcConfig.agcPower.underRangeLowPowerThresh = 2;
agcConfig.agcPower.underRangeHighPowerGainStepRecovery = 0;
agcConfig.agcPower.underRangeLowPowerGainStepRecovery = 0;
agcConfig.agcPower.powerMeasurementDuration = 5;
agcConfig.agcPower.rxTddPowerMeasDuration = 5;
agcConfig.agcPower.rxTddPowerMeasDelay = 1;
agcConfig.agcPower.overRangeHighPowerThresh = 2;
agcConfig.agcPower.overRangeLowPowerThresh = 0;
agcConfig.agcPower.powerLogShift = 1;  # Force to 1
agcConfig.agcPower.overRangeHighPowerGainStepAttack = 0;
agcConfig.agcPower.overRangeLowPowerGainStepAttack = 0;

# Make agcConfig and rxGainMode into array types (necessary for syntax reasons)
agcConfigArr = Array[Types.adi_adrv9025_AgcCfg_t]([agcConfig])
```

```
    rxGainModeArr = Array[Types.adi_adrv9025_RxAgcMode_t]([rxGainMode])

    # Write settings to device
    adrv9025.Agc.AgcCfgSet(agcConfigArr, 1)

    # Enable AGC Mode
    adrv9025.Rx.RxGainCtrlModeSet(rxGainModeArr, 1)

    print "Finished Programming Device"
else:
    print "Not Connected"

if (connect == True):
    Link.Ads8.board.Client.Disconnect()
    print "Disconnecting"
```

GAIN COMPENSATION, FLOATING POINT FORMATTER AND SLICER

The user has the option of enabling gain compensation in the device. In gain compensation mode, the digital gain block is utilized to compensate for the analog front-end attenuation. The cumulative gain across the device is thus 0 dB; 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 that the digital data is representative of the RMS power of the signal at the Rx input port; any internal front-end attenuation changes in device in order to prevent ADC overloading are transparent to the baseband processor. In this way, the AGC of the device can be used to react quickly to incoming blockers without the need for the baseband processor to track the current gain index the level of the received signal at the input to the device for received signal strength measurements.

The digital gain block is controlled by the gain table, and a compensated gain table is required to operate in this mode. Analog Devices provides an example compensated gain table in the software package. Such a gain table has a unique front-end attenuator setting with a corresponding amount of digital gain programmed at each index of the table.

Gain compensation can be used in either AGC or MGC modes. The maximum amount of gain compensation is 50 dB. This allows for compensation of both the internal analog attenuator and an external gain component (such as a DSA or LNA).

Large amounts of digital gain increase the bit width of the path. There are a number of ways in which this expanded bit-width data can be sent to the baseband processor, which are detailed below. Figure 90 is a block diagram of the gain compensation portion of the Rx chain, showing the locations of the various blocks.

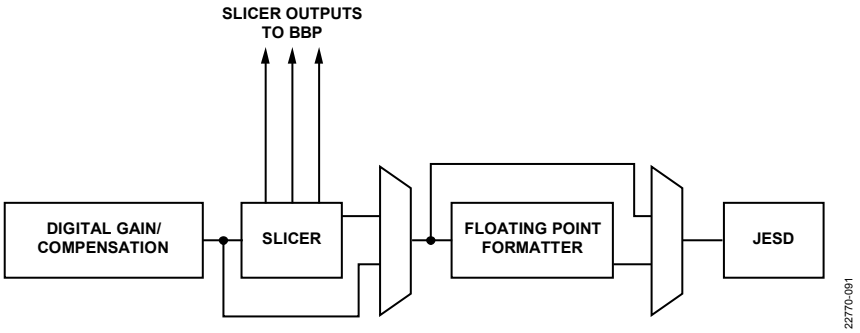


Figure 90. Gain Compensation, Floating Point Formatter, and Slicer Section of the Receiver Datapath

Mode 1: No Digital Gain Compensation

This is the mode that the chip is configured to by default. In this mode the digital gain block is not used for gain compensation. Instead the digital gain block may be utilized to apply small amounts of digital gain/attenuation to provide consistent gain steps in a gain table. The premise is that because the analog attenuator does not have consistent stops in dB across its range then the digital gain block can be utilized to even out the steps for consistency (the default table utilizes the digital gain block to provide consistent 0.5 dB steps).

Nether the slicer nor floating-point formatter block is utilized. As no gain compensation are applied, there is no bit-width expansion of the digital signal. The signal is provided to the JESD port which sends it to the baseband processor in either 12-bit, 16-bit or 24-bit format depending on the use case.

Mode 2: Digital Gain Compensation with Slicer GPIO Outputs

In this mode gain compensation is used. Load the device with a gain table that compensates for the analog front-end attenuation applied. Thus, as the analog front-end attenuation is increased, and equal amount of digital gain is applied. Considering 16-bit data at the input to the digital compensation block, then as more digital gain is applied the bit-width of the signal is increased. With every 6 dB of gain, the bit-width increases by 1. Figure 91 outlines this effect, with yellow boxes indicating the valid (used) bits in each case.

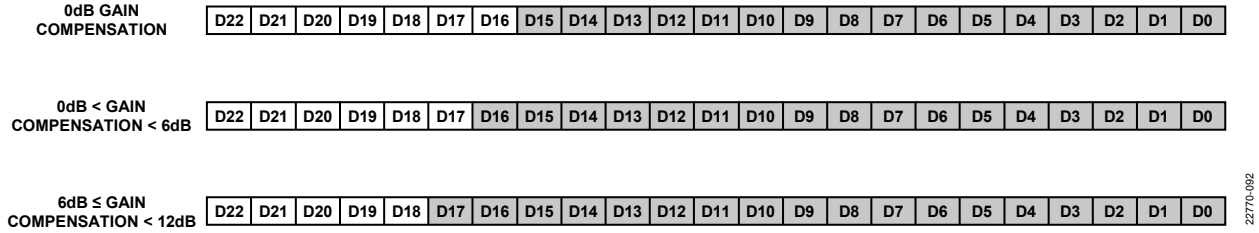


Figure 91. Bit Width of Received Signal for Increasing Gain Compensation

The slicer is used to attenuate the data after the digital gain block such that it can fit into the resolution of the JESD data path. It then advises the user how much attenuation is being applied in real time, so that the user can compensate on the baseband processor side. In this mode, the current slicer setting (amount of attenuation) is provided real time over GPIO pins.

Note that this slicer setting information is not necessarily time aligned to the data at the baseband processor side. As soon as the slicer value changes, this information is provided on the GPIO pins. However, there is some latency between this and when the corresponding data arrives across the JESD link. It is up to the user to determine an appropriate way of accounting for this latency.

This slicer can be configured for a number of attenuation resolutions, namely 1 dB, 2 dB, 3 dB, 4 dB, 6 dB or 8 dB steps. Higher resolution (smaller steps) allows the user to follow the actual signal amplitude with finer resolution, while lower resolution (larger steps) allows for more compensation range.

The slicer can use up to 4 GPIOs per receiver. The GPIOs used to output the slicer position are shown in Table 174. These require these pins to be enabled as outputs and configured for slicer output mode (see the General-Purpose Input/Output Configuration section).

Table 174. GPIOs Used for Slicer Output Mode

Receiver	GPIOs Utilized (MSB to LSB)
Rx1	GPIO11, GPIO10, GPIO9, GPIO8
Rx2	GPIO15, GPIO14, GPIO13, GPIO12
Rx3	GPIO7, GPIO6, GPIO5, GPIO4
Rx4	GPIO3, GPIO2, GPIO1, GPIO0

The following example explains the operation of the slicer in detail. In this use case, the JESD is configured for 16-bit data resolution. The slicer is configured to 6 dB resolution.

Figure 92 explains the operation. Initially the analog attenuator is applying no attenuation (0 dB) and thus there is 0 dB digital gain applied to the signal. The slicer is in its default (0000) position. As the attenuation increases (0 dB to 6 dB), a corresponding amount of digital gain is applied to the signal. With any digital gain applied to the signal, the bit-width of the signal has increased (the ADC can output 16-bits, further gain allows a maximum input to go beyond 16-bits). In this case the signal has now a bit-width of 17. The slicer therefore applies 6 dB of attenuation, and the slicer position information across the GPIOs is updated to advise the user of this change (in this case 0001). This 6 dB attenuation ensures that the bit-width of the signal is 16 once more; that is, the 16 MSBs have been selected (sliced) with the LSB dropped. When the compensation increases beyond 6 dB, it is now possible that the signal resolution in the digital path can be 18-bit. The slicer then attenuates by 12 dB (or slices the 16 MSBs dropping the 2 LSBs).

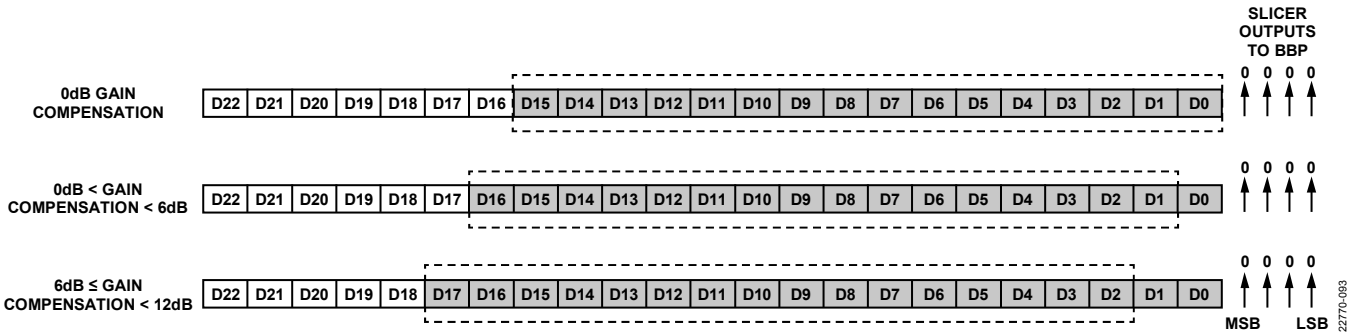


Figure 92. Slicer Bit Selection with Digital Gain

The baseband processor receives these 16-bits and uses the slicer output to scale the power of the received signal to determine the power at the input to the device (or at the input to an external gain element if considered part of the digital gain compensation).

The slicer position vs. digital gain for this 6 dB example is described in Table 175. Equivalent tables can be inferred for the other attenuation options.

Table 175. Slicer GPIO Output vs. Digital Gain Compensation

Digital Gain Compensation (dB)	Slicer Position (Value output on GPIOs)
0	0
0 < Dig_Gain < 6	1
6 ≤ Dig_Gain < 12	2
12 ≤ Dig_Gain < 18	3
18 ≤ Dig_Gain < 24	4
24 ≤ Dig_Gain < 30	5
30 ≤ Dig_Gain < 36	6
36 ≤ Dig_Gain < 42	7
42 ≤ Dig_Gain < 48	8
48 ≤ Dig_Gain ≤ 50	9

Mode 3: Digital Gain Compensation with Embedded Slicer Position

This mode is similar to Mode 2. The slicer is used to select the 16 MSBs based on the amount of digital gain used by the currently selected gain index in the gain table. However, in this mode the GPIO slicer outputs are not used. Instead the slicer position (or attenuation applied) is embedded into the data.

There are a number of permissible ways in which this can be configured, controlled by the intEmbeddedBits API parameter. The options are to place the slicer setting as 1 bit on both I and Q, or 2 bits on both I and Q. These can be placed at the MSBs or LSBs. For the case where 2 bits are embedded onto both I and Q data, there are further options of using 3 slicer bits or 4. If 3 is used, there is a further option of inserting a 0 to fill the 4th bit, or to insert a parity bit (by adjusting the intParity API parameter). Table 176 shows the various modes selectable by intEmbeddedBits.

Table 176. adi_adrv9025_RxSlicerEmbeddedBits_e Description

intEmbeddedBits	Description
ADI_ADRV9025_EMBED_1_SLICERBIT_AT_MSB	Embeds 1 slicer bit on both I and Q at the MSB position. See Figure 93.
ADI_ADRV9025_EMBED_1_SLICERBIT_AT_LSB	Embeds 1 slicer bit on both I and Q at the LSB position. See Figure 94.
ADI_ADRV9025_EMBED_2_SLICERBITS_AT_MSB_3_BIT_SLICER	Embeds 2 slicer bits on both I and Q at the MSB positions. See Figure 95. Given this is a 3-bit mode, an extra bit is inserted denoted as P in Figure 95. This can either be a parity bit or a zero can always be inserted alternatively.
ADI_ADRV9025_EMBED_2_SLICERBITS_AT_LSB_3_BIT_SLICER	Embeds 2 slicer bits on both I and Q at the LSB position. See Figure 96. Given this is a 3-bit mode, an extra bit is inserted denoted as P in Figure 96. This can either be a parity bit or a zero can always be inserted alternatively.
ADI_ADRV9025_EMBED_2_SLICERBITS_AT_MSB_4_BIT_SLICER	Embeds 2 slicer bits on both I and Q at the MSB positions. See Figure 97.
ADI_ADRV9025_EMBED_2_SLICERBITS_AT_LSB_4_BIT_SLICER	Embeds 2 slicer bits on both I and Q at the LSB positions. See Figure 98.

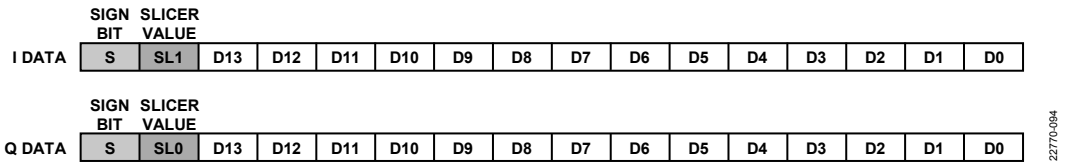


Figure 93. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_1_SLICERBIT_AT_MSB)

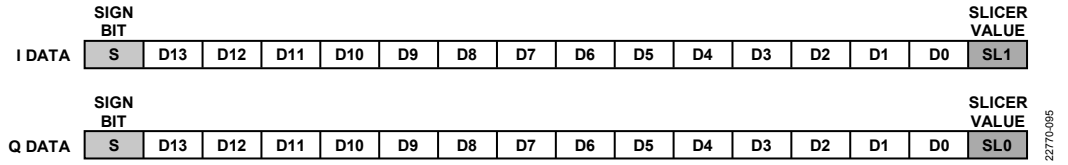


Figure 94. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025__EMBED_1_SLICERBIT_AT_LSB)

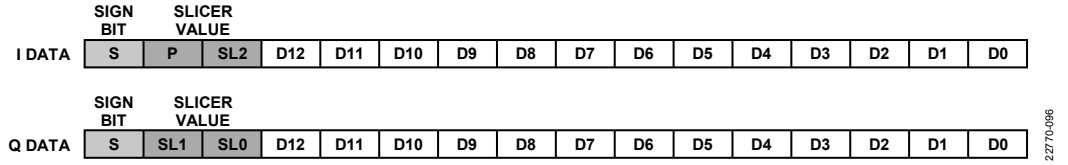


Figure 95. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_2_SLICERBITS_AT_MSB_3_BIT_SLICER)

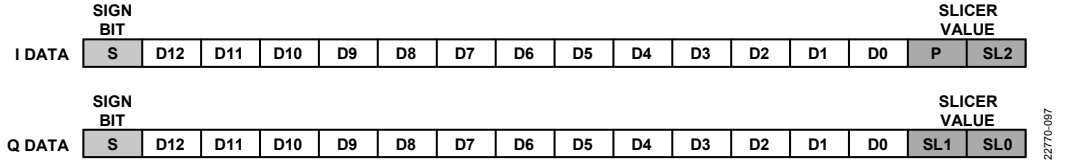


Figure 96. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_2_SLICERBITS_AT_LSB_3_BIT_SLICER)

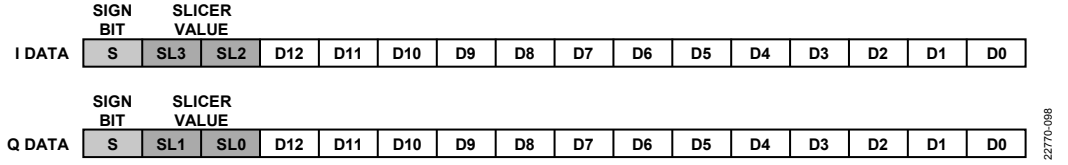


Figure 97. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_2_SLICERBITS_AT_MSB_4_BIT_SLICER)

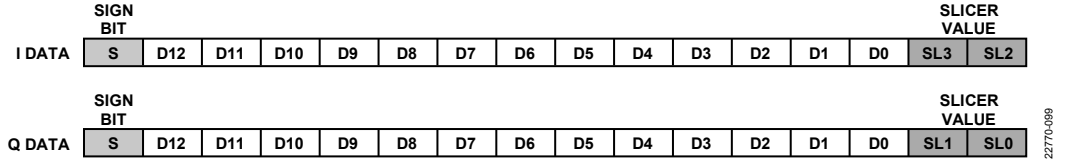


Figure 98. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_2_SLICERBITS_AT_LSB_4_BIT_SLICER)

Mode 4: Digital Gain Compensation and Slicer Input

In this mode, the slicer position is controlled by the user. In modes 2 and 3, the slicer can be viewed as an attenuator which reduces the signal level a certain dB with each slicer position step such that it can be sent across the JESD link. This mode is similar, except the position (amount of attenuation) is controlled externally. The valid step sizes are between 1 and 6 dB and controlled by the extPinStepSize API parameter as outlined in Table 177.

Table 177. adi_adrv9025_ExtSlicerStepSizes_e Description

extPinStepSize	Slicer Gain Step (dB)
ADI_ADRV9025_EXTSLICER_STEPSIZE_1DB	1
ADI_ADRV9025_TAL_EXTSLICER_STEPSIZE_2DB	2
ADI_ADRV9025_TAL_EXTSLICER_STEPSIZE_3DB	3
ADI_ADRV9025_TAL_EXTSLICER_STEPSIZE_4DB	4
ADI_ADRV9025_TAL_EXTSLICER_STEPSIZE_6DB	6

The slicer has 3 input pins. The valid options are shown in Table 178. Each channel can be set to any one of the options using the API parameters, rx1ExtSlicerGpioSelect, rx2ExtSlicerGpioSelect, rx3ExtSlicerGpioSelect, rx4ExtSlicerGpioSelect. The value of these pins and the step size chosen set the level of slicer attenuation applied to the data before transmission across the JESD link.

Slicer Attenuation = Slicer Input Pin Values × extPinStepSize

For example, if the value on the slicer input pins was 0'b111, and the step size was 2 dB, then the slicer applies 14 dB (7 × 2 dB) of attenuation to the data.

Table 178. adi_adrv9025_RxExtSlicerGpioSel_e Description

Value of RxExtSlicerGpioSelect	GPIOs Utilized (MSB to LSB)
ADI_ADRV9025_EXTSLICER_RX_GPIO_2_DOWNTO_0	GPIO2, GPIO1, GPIO0
ADI_ADRV9025_EXTSLICER_RX_GPIO_5_DOWNTO_3	GPIO5, GPIO4, GPIO3
ADI_ADRV9025_EXTSLICER_RX_GPIO_8_DOWNTO_6	GPIO8, GPIO7, GPIO6
ADI_ADRV9025_EXTSLICER_RX_GPIO_11_DOWNTO_9	GPIO11, GPIO10, GPIO9
ADI_ADRV9025_EXTSLICER_RX_GPIO_14_DOWNTO_12	GPIO14, GPIO13, GPIO12
ADI_ADRV9025_EXTSLICER_RX_GPIO_17_DOWNTO_15	GPIO17, GPIO16, GPIO15

Mode 5: Digital Gain Compensation and Floating-Point Formatting

The floating-point formatter offers an alternative way of encoding the digitally compensated data onto the JESD204B link. In this mode, the data is converted to IEEE754 half precision floating point format (binary 16). There is a slight loss in resolution when using the floating-point formatter, though resolution is distributed such that smaller numbers have higher resolution.

In binary 16 floating point format the number is composed on a sign-bit (S), an exponent (E) and a significand (T). There are a number of options in terms of the number of bits that can be assigned to the exponent. More bits in the exponent result in higher range, and thus can allow for more digital compensation to the represented, whereas more bits in the significand provides higher resolution. The available options for the floating point formatter of the device include the following:

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

It is also possible to pack the data in the following different formats (as shown in Figure 99):

- Sign, exponent, significand
- Sign, significand, exponent

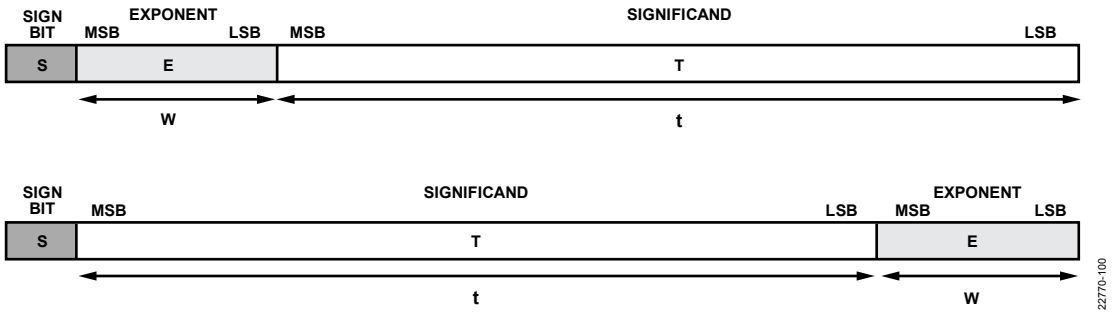


Figure 99. Floating Point Number Representation

In Figure 99, S is the sign bit, E is the value of the exponent, T is the value of the significand, w is the bit width of the exponent, and t is the bit width of the significand.

Upon receipt of an encoded floating-point formatter, the user breaks up the binary 16 number into its constituent parts. For the purposes of this explanation, consider a 3-bit exponent. In IEEE754, the maximum exponent (0'b111 in this case) is reserved for NaN. The minimum exponent (0'b000) is used for a signed zero (E = 0, T = 0) and subnormal numbers (E = 0, T ≠ 0). To decode a received floating-point sample, the following equations are used:

If E = 0 and T = 0,

$$Value = 0$$

If E = 0 and T ≠ 0:

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

If E ≠ 0:

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

where bias is used to convert the positive binary values to exponents which allow for values both less than and greater than the full-scale of the ADC and p is the precision of the mode (p = t + 1, because you have t significand bits coupled with a sign bit). Table 179 provides the values to use in these equations for the various IEEE754 supported modes.

Table 179. Floating Point Formatter—Supported IEEE 754 Modes

Exponent Bit Width (w)	Significand Bit Width (t)	Precision (p)	bias
5	10	11	15
4	11	12	7
3	12	13	3
2	13	14	1

Figure 100 provides a visual representation of how the values of a waveform are encoded in floating point format. In this case the maximum exponent (E-bias) is 3, meaning that data up to 24 dBFS of the ADC can be represented. As the signal reduces, the exponent required to represent each value differs. This is a different concept to the slicer that instead bit-shifted the data solely based on the applied digital attenuation and had a constant value for a constant digital gain. Instead the floating-point formatter interprets each data value after the digital gain compensation separately. Given the fixed precision of the significand and the sign bit, it can also be interpreted from this plot that there is higher resolution at lower signal levels then there is at higher signal levels, preserving SNR when the received signal strength is low.

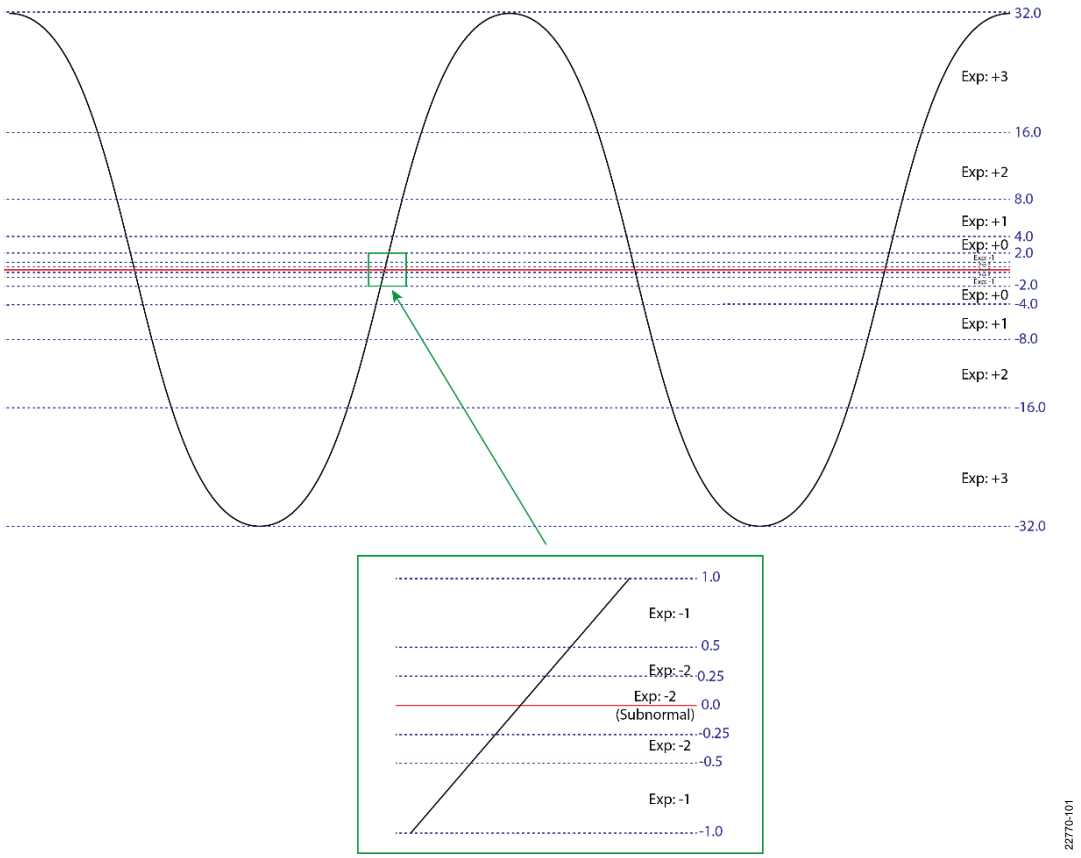


Figure 100. Visualization of the Floating-Point Formatter Values

The floating-point formatter also supports non-IEEE754 modes, referred to as Analog Devices modes, where the largest exponent is not used to express NaN in accordance with IEEE754. It is unnecessary for the device to encode NaN as none of the data values can be NaN, and therefore using this extra exponent value increases the largest value representable for a given exponent bit-width.

Table 180. Exponent Bit Widths of IEEE-754 and Analog Devices Modes

Exponent Bit Width (w)	IEE-754 Mode Exponent Range (After Unbiasing)	Analog Devices Mode Exponent Range (After Unbiasing)
5	+15 to −14	+16 to −14
4	+7 to −6	+8 to −6
3	+3 to −2	+4 to −2
2	+1 to −1	+2 to −1

In the default floating point format, the leading one is inferred and not encoded (for normal numbers). It is possible to enable a mode where the leading one is encoded and stored in the MSB of the significand. This reduces the precision of the values however.

If the user knows that the range of attenuation required for the worst case blocker (and therefore the digital gain required to compensate for it) exceeds the correction range allowed by the exponent width chosen, then it is also possible to enable a fixed digital attenuation (from 6 to 42 dB) prior to the floating point formatter to ensure that the signal never exceeds the maximum range encodable over the JESD link.

RECEIVER DATA FORMAT DATA STRUCTURE

The configuration parameters for the floating-point formatter and slicer are set up in a data structure of type adi_adrv9025_RxDataFormat_t.

Table 181. adi_adrv9025_RxDataFormat Definition

Parameter	Comments	
rxChannelMask	This selects the channels upon which to enable this gain control mode. It is a bit mask with each bit corresponding to a channel, [D0] = Rx1, [D1] = Rx2, [D2] = Rx3, [D3] = Rx4. Therefore, setting the rxChannelMask = 15 means that all Rxx are configured with the same agcMode. Data type: uint32	
formatSelect	This selects the format of the data received from the receive path. Data type: adirx9025_RxDataFormatModes_e	
	formatSelect	**Format**
	ADI_ADRV9025_GAIN_COMPENSATION_DISABLED	No gain compensation (Mode 1)
	ADI_ADRV9025_GAIN_WITH_FLOATING_POINT	Gain compensation and floating-point formatter enabled (Mode 5)
	ADI_ADRV9025_GAIN_WITH_INTERNAL_SLICER_NOGPIO	Gain Compensation and slicer bits embedded on JESD signal (mode 3)
	ADI_ADRV9025_GAIN_WITH_INTERNAL_SLICER	Gain compensation and slicer bits output on GPIOs (Mode 2)
	ADI_ADRV9025_GAIN_WITH_EXTERNAL_SLICER	Gain compensation and slicer position input from GPIOs (Mode 4)
floatingPointConfig	A configuration structure for floating point format. See Table 182. To be used when floating point formatter is utilized. Data type: adi_adrv9025_FloatingPointConfigSettings_t	
integerConfigSettings	A configuration structure for the data resolution across the JESD link. See Table 183. Data type: adi_adrv9025_IntegerConfigSettings_t	
slicerConfigSettings	A configuration structure for the slicer functionality. See Table 184. Data type: adi_adrv9025_SlicerConfigSettings_t	
externalLnaGain	For use in Dual Band Modes. Not currently supported.	
tempCompensationEnable	Not currently supported.	

Table 182. adi_adrv9025_FloatingPointCOnfigSettings_t

Parameter	Comments	
fpDataFormat	This parameter sets the format of the 16-bit output on the JESD204B interface. Data type: adi_adrv9025_FpFloatDataFormat_e	
	fpDataFormat	**Floating Point Data Format**
	ADI_ADRV9025_FP_FORMAT_SIGN_EXP_SIGNIFICAND	Sign, Exponent, Significand
	ADI_ADRV9025_FP_FORMAT_SIGN_SIGNIFICAND_EXP	Sign, Significand, Exponent
fpRoundMode	This parameter sets the round mode for the significand. The following settings are defined in the IEEE754 specification. For more information, consult Section 4.3 in IEEE 754-2008. Data type: adi_adrv9025_FpRoundModes_e	
	fpRoundMode	**Floating Point Rounding Mode**
	ADI_ADRV9025_ROUND_TO_EVEN	Floating point ties to an even value
	ADI_ADRV9025_ROUNDTOWARDS_POSITIVE	Round floating point toward the positive direction
	ADI_ADRV9025_ROUNDTOWARDS_NEGATIVE	Round floating point toward the negative direction
	ADI_ADRV9025_ROUNDTOWARDS_ZERO	Round floating point toward the zero direction
	ADI_ADRV9025_ROUND_FROM_EVEN	Round floating point away from the even value
fpNumExpBits	This parameter is used to indicate the number of exponent bits in the floating-point number. Data type: adi_adrv9025_FpExponentModes_e	
	fpNumExpBits	**No. of Exponent Bits**
	ADI_ADRV9025_2_EXPONENTBITS	2
	ADI_ADRV9025_3_EXPONENTBITS	3
	ADI_ADRV9025_4_EXPONENTBITS	4
	ADI_ADRV9025_5_EXPONENTBITS	5

Parameter	Comments	
fpAttenSteps	Attenuates integer data before floating point conversion when floating point mode enabled. Data type: adi_adrv9025_FpAttenSteps_e	
	fpRx1Atten	**Attenuation (dB)**
	ADI_ADRV9025_FPATTEN_0DB	0
	ADI_ADRV9025_FPATTEN_MINUS6DB	−6
	ADI_ADRV9025_FPATTEN_MINUS12DB	−12
	ADI_ADRV9025_FPATTEN_MINUS18DB	−18
	ADI_ADRV9025_FPATTEN_24DB	24
	ADI_ADRV9025_FPATTEN_18DB	18
	ADI_ADRV9025_FPATTEN_12DB	12
	ADI_ADRV9025_FPATTEN_6DB	6
fpHideLeadingOne	It is possible to hide the leading one in the significand to be compatible to the IEEE754 specification (IEEE mode). Alternatively, a leading one can be inserted at the MSB of the significand. Data type: adi_adrv9025_FpHideLeadingOne_e	
	fpHideLeadingOne	**Setting**
	ADI_ADRV9025_FP_FORMAT_HIDE_LEADING_ONE_DISABLE	Leading one at start of significand
	ADI_ADRV9025_FP_FORMAT_HIDE_LEADING_ONE_ENABLE	No leading one at start of the significand
fpEncodeNan	This parameter is used to configure whether the floating-point formatter reserves the highest value of exponent for NaN (not a number) to be compatible with the IEEE754 specification or whether to use the highest value of the exponent to extend the representable signal range. Data type: adi_adrv9025_FpNanEncode_e	
	fpHideLeadingOne	**Setting**
	ADI_ADRV9025_FP_FORMAT_NAN_ENCODE_DISABLE	Do not reserve the highest exponent for NaN
	ADI_ADRV9025_FP_FORMAT_NAN_ENCODE_ENABLE	Reserve highest exponent for NaN

Table 183. adi_adrv9025_IntegerConfigSettings_t Definition

Parameter	Comments	
intEmbdeddedBits	For use in slicer modes. This parameter sets the integer number of embedded slicer bits to embed in Rx data sample and bit position to embed them (see mode 3). Data type: adi_adrv9025_RxSlicerEmbeddedBits_e	
	intEmbeddedBits	**Slicer bit Embedded position in Data Frame**
	ADI_ADRV9025_NO_EMBEDDED_SLICER_BITS	Disabled all embedded slicer bits
	ADI_ADRV9025_EMBED_1_SLICERBIT_AT_MSB	Embeds 1 slicer bits on I and 1 slicer bits on Q and the MSB position
	ADI_ADRV9025_EMBED_1_SLICERBIT_AT_LSB	Embeds 1 slicer bits on I and 1 slicer bits on Q and the LSB position
	ADI_ADRV9025_EMBED_2_SLICERBITS_AT_MSB_3_BIT_SLICER	Embeds 2 slicer bits on I and 2 slicer bits on Q and the MSB position in 3-bit slicer mode
	ADI_ADRV9025_EMBED_2_SLICERBITS_AT_LSB_3_BIT_SLICER	Embeds 2 slicer bits on I and 2 slicer bits on Q and the LSB position in 3-bit slicer mode
	ADI_ADRV9025_EMBED_2_SLICERBITS_AT_MSB_4_BIT_SLICER	Embeds 2 slicer bits on I and 2 slicer bits on Q and the MSB position in 4-bit slicer mode
	ADI_ADRV9025_EMBED_2_SLICERBITS_AT_LSB_4_BIT_SLICER	Embeds 2 slicer bits on I and 2 slicer bits on Q and the LSB position in 4-bit slicer mode
intSampleResolution	This parameter sets the integer sample resolution selecting either 12, 16, 24 bits data with either twos complement or signed magnitude. Data type: adi_adrv9025_RxIntSampleResolution_e	
	intSampleResolution	**Resolution of Integer sample**
	ADI_ADRV9025_INTEGER_12BIT_2SCOMP	12-bit resolution with twos complement
	ADI_ADRV9025_INTEGER_12BIT_SIGNED	12-bit resolution with signed magnitude
	ADI_ADRV9025_INTEGER_16BIT_2SCOMP	16-bit resolution with twos complement
	ADI_ADRV9025_INTEGER_16BIT_SIGNED	16-bit resolution with signed magnitude
	ADI_ADRV9025_INTEGER_24BIT_2SCOMP	24-bit resolution with twos complement
	ADI_ADRV9025_INTEGER_24BIT_SIGNED	24-bit resolution with signed magnitude

Parameter	Comments	
intParity	In the embedded 3-bit slicer mode (mode 3), it is possible to enable a parity mode. The device can support even parity (whereby the number of 1s in the bit sequence is always even) or odd parity (whereby the number of 1s in the bit sequence is always odd). Data type: adi_adrv9025_RxIntParity_e	
	intParity	**Setting**
	ADI_ADRV9025_3BIT_SLICER_EVEN_PARITY	Even parity enabled
	ADI_ADRV9025_3BIT_SLICER_ODD_PARITY	Odd parity enabled
	ADI_ADRV9025_NO_PARITY	Parity disabled

Table 184. adi_adrv9025_SlicerConfigSettings_t Definition

Parameter	Comments	
extSlicerStepSize	This parameter is used in gain compensation with external slicer control (Mode 4). This parameter sets the slicer step value that is used with this external control mechanism. Data type: adi_adrv9025_ExtSlicerStepSizes_e	
	extSlicerStepSize	**Slicer Step Size**
	ADI_ADRV9025_EXTSLICER_STEPSIZE_1DB	1 dB
	ADI_ADRV9025_EXTSLICER_STEPSIZE_2DB	2 dB
	ADI_ADRV9025_EXTSLICER_STEPSIZE_3DB	3 dB
	ADI_ADRV9025_EXTSLICER_STEPSIZE_4DB	4 dB
	ADI_ADRV9025_EXTSLICER_STEPSIZE_6DB	6 dB
intSlicerStepSize	This parameter is used in gain compensation with internal (automatic) slicer control (Mode 2). This parameter sets the slicer step value. Data type: adi_adrv9025_IntSlicerStepSizes_e	
	intSlicerStepSize	**Slicer Step Size**
	ADI_ADRV9025_INTSLICER_STEPSIZE_1DB	1 dB
	ADI_ADRV9025_INTSLICER_STEPSIZE_2DB	2 dB
	ADI_ADRV9025_INTSLICER_STEPSIZE_3DB	3 dB
	ADI_ADRV9025_INTSLICER_STEPSIZE_4DB	4 dB
	ADI_ADRV9025_INTSLICER_STEPSIZE_6DB	6 dB
	ADI_ADRV9025_INTSLICER_STEPSIZE_8DB	8 dB
rx1ExtSlicerGpioSelect	This parameter selects the GPIOs used for external slicer control (Mode 4) on Rx1. The choice must be unique to Rx1. Data type: adi_adrv9025_RxExtSlicerGpioSel_e	
	rx1ExtSlicerGpioSelect	**GPIOs Utilized**
	ADI_ADRV9025_EXTSLICER_RX_GPIO_DISABLE	
	ADI_ADRV9025_EXTSLICER_RX_GPIO_2_DOWNTO_0	2, 1, 0
	ADI_ADRV9025_EXTSLICER_RX_GPIO_5_DOWNTO_3	5, 4, 3
	ADI_ADRV9025_EXTSLICER_RX_GPIO_8_DOWNTO_6	8, 7, 6
	ADI_ADRV9025_EXTSLICER_RX_GPIO_11_DOWNTO_9	11, 10, 9
	ADI_ADRV9025_EXTSLICER_RX_GPIO_14_DOWNTO_12	14, 13, 12
	ADI_ADRV9025_EXTSLICER_RX_GPIO_17_DOWNTO_15	17, 16, 15
	ADI_ADRV9025_EXTSLICER_RX_GPIO_INVALID	
rx2ExtSlicerGpioSelect	This parameter selects the GPIOs used for external slicer control (Mode 4) on Rx2. The choice must be unique to Rx2. Data type: adi_adrv9025_RxExtSlicerGpioSel_e	
	rx2ExtSlicerGpioSelect	**GPIOs Utilized**
	ADI_ADRV9025_EXTSLICER_RX_GPIO_DISABLE	
	ADI_ADRV9025_EXTSLICER_RX_GPIO_2_DOWNTO_0	2, 1, 0
	ADI_ADRV9025_EXTSLICER_RX_GPIO_5_DOWNTO_3	5, 4, 3
	ADI_ADRV9025_EXTSLICER_RX_GPIO_8_DOWNTO_6	8, 7, 6
	ADI_ADRV9025_EXTSLICER_RX_GPIO_11_DOWNTO_9	11, 10, 9
	ADI_ADRV9025_EXTSLICER_RX_GPIO_14_DOWNTO_12	14, 13, 12
	ADI_ADRV9025_EXTSLICER_RX_GPIO_17_DOWNTO_15	17, 16, 15
	ADI_ADRV9025_EXTSLICER_RX_GPIO_INVALID	

Parameter	Comments	
rx3ExtSlicerGpioSelect	This parameter selects the GPIOs used for external slicer control (Mode 4) on Rx3. The choice must be unique to Rx3. Data type: adi_adrv9025_RxExtSlicerGpioSel_e	
	rx3ExtSlicerGpioSelect	**GPIOs Utilized**
	ADI_ADRV9025_EXTSLICER_RX_GPIO_DISABLE	
	ADI_ADRV9025_EXTSLICER_RX_GPIO_2_DOWNTO_0	2, 1, 0
	ADI_ADRV9025_EXTSLICER_RX_GPIO_5_DOWNTO_3	5, 4, 3
	ADI_ADRV9025_EXTSLICER_RX_GPIO_8_DOWNTO_6	8, 7, 6
	ADI_ADRV9025_EXTSLICER_RX_GPIO_11_DOWNTO_9	11, 10, 9
	ADI_ADRV9025_EXTSLICER_RX_GPIO_14_DOWNTO_12	14, 13, 12
	ADI_ADRV9025_EXTSLICER_RX_GPIO_17_DOWNTO_15	17, 16, 15
	ADI_ADRV9025_EXTSLICER_RX_GPIO_INVALID	
rx4ExtSlicerGpioSelect	This parameter selects the GPIOs used for external slicer control (Mode 4) on Rx4. The choice must be unique to Rx4. Data type: adi_adrv9025_RxExtSlicerGpioSel_e	
	rx4ExtSlicerGpioSelect	**GPIOs Utilized**
	ADI_ADRV9025_EXTSLICER_RX_GPIO_DISABLE	
	ADI_ADRV9025_EXTSLICER_RX_GPIO_2_DOWNTO_0	2, 1, 0
	ADI_ADRV9025_EXTSLICER_RX_GPIO_5_DOWNTO_3	5, 4, 3
	ADI_ADRV9025_EXTSLICER_RX_GPIO_8_DOWNTO_6	8, 7, 6
	ADI_ADRV9025_EXTSLICER_RX_GPIO_11_DOWNTO_9	11, 10, 9
	ADI_ADRV9025_EXTSLICER_RX_GPIO_14_DOWNTO_12	14, 13, 12
	ADI_ADRV9025_EXTSLICER_RX_GPIO_17_DOWNTO_15	17, 16, 15

adi_adrv9025_RxDataFormatSet

```
adi_adrv9025_RxDataFormatSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxDataFormat_t
rxDataFormat[],uint8_t arraySize);
```

Description

Configure the Rx data format.

Parameters

Table 185.

Parameter	Description
*device	Pointer to device structure.
rxDataFormat[]	An array of Rx data format structures.
arraySize	The number of Rx data format structures in rxDataFormatarray length of txPaProtectCfg[].

DIGITAL FILTER CONFIGURATION

OVERVIEW

This section describes the digital filters within the transceiver. It provides a description of each of the filters in terms of their filter coefficients and position within the signal chain. The API structures are also described, and an example profile specific configuration is provided for each of the signal chains. Finally, the API functions that are used to configure the filters are discussed.

RECEIVER SIGNAL PATH

Each receiver input has an independent signal path including separate I/Q mixers that feed into programmable analog transimpedance amplifiers (TIA) that serve as low pass filters (LPF) in the analog data path. The signals are then converted by the sigma-delta ADCs and filtered in half-band decimation stages and the programmable finite impulse response filter (PFIR). The fixed coefficient half-band filters (FIR1, FIR2, RHB1(HR), RHB1(LP), RHB2, RHB3, DEC5) and the PFIR are designed to prevent data wrapping and over-range conditions.

Each receiver channel can convert signals down to zero-IF real data using the standard I/Q configuration or a low-IF complex data configuration. The digital filtering stage allows the configuration flexibility and decimation options to operate in either mode.

Figure 101 shows the signal path for the Rx1, Rx2, Rx3 and Rx4 signal chain. Blocks that are not discussed in this section are faded.

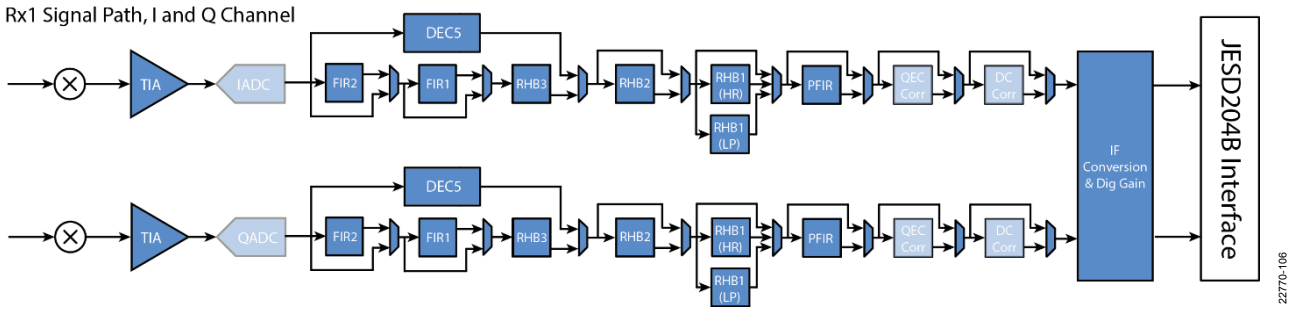


Figure 101. Rx Signal Path (Filter Blocks Highlighted in Blue)

Transimpedance Amplifier (TIA)

The Rx transimpedance amplifier is a low pass filter with a single real pole frequency response. The device supports bandwidths up to 200 MHz and thus each TIA supports a pass-band of 100 MHz on the I and Q paths. The TIA is calibrated during device initialization to ensure a consistent frequency corner across all devices. The TIA 3dB bandwidth is set within the device data structure and is profile dependent. Roll off within the Rx pass band is compensated by the PFIR to ensure a maximally flat pass-band frequency response.

Decimation Stages

The signal path can be configured so that either the decimate-by-5 filter (DEC5) or the combination of FIR2, FIR1, and RHB3 is used in the Rx digital path. The DEC5 decimates by a factor of 5 while the other filter combination can be configured to decimate by factors of 2, 4, or 8.

DEC5

DEC5 filter coefficients: 0.000976563, 0.001220703, 0.001953125, 0.001953125, −0.00390625, −0.0078125, −0.014648438, −0.018798828, −0.019042969, -0.007568359, 0.010742188, 0.041748047, 0.079101563, 0.1171875, 0.146972656, 0.165527344, 0.165527344, 0.146972656, 0.1171875, 0.079101563, 0.041748047, 0.010742188, −0.007568359, −0.019042969, −0.018798828, −0.014648438, −0.0078125, −0.00390625, 0.001220703, 0.001953125, 0.001953125, 0.001220703, 0.000976563

Finite Impulse Response 2 (FIR2)

The FIR2 filter is a fixed coefficient decimating filter. The FIR2 decimates by a factor of 2 or it may be bypassed.

FIR2 filter coefficients: 0.0625, 0.25, 0.375, 0.25, 0.0625

Finite Impulse Response 1 (FIR1)

The FIR1 filter is a fixed coefficient decimating filter. The FIR1 decimates by a factor of 2 or it may be bypassed.

FIR1 filter coefficients: 0.0625, 0.25, 0.375, 0.25, 0.0625

Receive Half Band 3 (RHB3)

The RHB3 filter is a fixed coefficient decimating filter. The RHB3 decimates by a factor of 2.

RHB3 filter coefficients: −0.033203125, 0, 0.28125, 0.49609375, 0.28125, 0, −0.033203125

Receive Half Band 2 (RHB2)

The RHB2 filter is a fixed coefficient decimating filter. The RHB2 decimates by a factor of 2 or it may be bypassed.

RHB2 filter coefficients: −0.000244141, 0, 0.001708984, 0, −0.0078125, 0, 0.026855469, 0, −0.078369141, 0, 0.30859375, 0.501220703, 0.30859375, 0, −0.078369141, 0, 0.026855469, 0, −0.0078125, 0, 0.001708984, 0, −0.000244141

Receive Half Band High Rejection 1 (RHB1 (HR))

The RHB1 (HR) filter is a fixed coefficient decimating filter. The RHB1 (HR) can decimate by a factor of 2, or it may be bypassed.

RHB1 (HR) filter coefficients: 0.000106812, 0, −0.000289917, 0, 0.00062561, 0, −0.001205444, 0, 0.002120972, 0, −0.003494263, 0, 0.005493164, 0, −0.008300781, 0, 0.012207031, 0, −0.01763916, 0, 0.025421143, 0, −0.03717041, 0, 0.057250977, 0, −0.101608276, 0, 0.314498901, 0.495956421, 0.314498901, 0, −0.101608276, 0, 0.057250977, 0, −0.03717041, 0, 0.025421143, 0, −0.01763916, 0, 0.012207031, 0, −0.008300781, 0, 0.005493164, 0, −0.003494263, 0, 0.002120972, 0, −0.001205444, 0, 0.00062561, 0, −0.000289917, 0, 0.000106812

Receive Half Band Low Power 1 (RHB1 (LP))

The RHB1 (LP) filter is a fixed coefficient decimating filter. The RHB1 (LP) can decimate by a factor of 2, or it may be bypassed.

RHB1 (LP) filter coefficients: −0.002685547, 0, 0.017333984, 0, −0.068359375, 0, 0.304443359, 0.501708984, 0.304443359, 0, −0.068359375, 0, 0.017333984, 0, −0.002685547

Rx Programmable Finite Impulse Response (PFIR)

The Rx PFIR filter acts as a decimating filter. The PFIR may decimate by a factor of 1, 2, or 4, or it can be bypassed. The RFIR is used to compensate for the roll-off of the analog TIA LPF. The PFIR can use either 24, 48, or 72 filter taps. The PFIR also has programmable gain settings of +6 dB, 0 dB, −6 dB or −12 dB.

The maximum number of taps is limited by the FIR Clock Rate (Data Processing Clock – DPCLK). The maximum DPCLK is 1 GHz. The DPCLK is the ADC Clock Rate divided by either 4 or 5. The divider is 4 when using the FIR2, FIR1 and HB3 filters, and it is 5 when using the DEC5 filter. The DPCLK affects the maximum number of PFIR filter taps that can be used according to the following:

$$Rx\ PFIR\ filter\ taps_{max} = \frac{DPCLK_{max}}{Rx\_IQ\_DataRate} \times 24$$

IF Conversion

The IF conversion stage provides the user with the ability to change how the received data is presented to the JESD port. Figure 102 shows a block diagram of the IF conversion stage. There are two parallel paths where data can be processed, referred to as Band A and Band B. In the circuitry of each band there are two mixer stages, allowing for upshifting or downshifting, interpolation and decimation stages, and a half band filter with a pass band of 0.4 × SampleRate. The coefficients of the HB filter in this IF conversion stage are as follows:

HB filter coefficients: -9.1553×10^{-5}, 0, 2.4414e-4, 0, −5.7983e-4, 0, 0.0012, 0, −0.0023, 0, 0.0040, 0, −0.0065, 0, 0.0103, 0, −0.0157, 0, 0.0236, 0, −0.0357, 0, 0.0563, 0, −0.1015, 0, 0.3168, 0.5000, 0.3168, 0, -0.1015, 0, 0.0563, 0, -0.0357, 0, 0.0236, 0, -0.0157, 0, 0.0103, 0, −0.0065, 0, 0.0040, 0, −0.0023, 0, 0.0012, 0, -5.7983×10^{-4}, 0, 2.4414×10^{-4}, 0, $-9.1553\ 10\times^{-5}$

The following use cases provide an example of the types of functionality supported by this block (note that currently only the Low-IF to Zero-IF conversion mode is supported in a released profile):

Complex Low-IF to Zero-IF

In this use case the received signal is offset from LO such that the entire signal of interest is on one side of the LO. The Band A NCO1 is used to downshift the signal such that it is centered at 0 Hz. There is a half-band filter and decimate by 2 stage, which if used, decreases the bandwidth and the subsequently the IQ rate. This reduces the number of JESD lanes required, or the rate that they need to be run at.

Figure 103 shows a conceptual case of a 200 MHz Rx bandwidth (IQ rate 245.76 MSPS) profile being used to receive a 75 MHz MC-GSM offset from the LO, the center frequency is 52.5 MHz offset from the LO, such that the band occupies from ±15 MHz to ±90 MHz. It then uses the IF conversion stage to shift the signal such that it is centered about 0 Hz, filters with the half-band filter, and decimates the output by two, such that the IQ rate sent over the JESD is 122.88 MSPS.

Complex Low-IF to Real-IF

In this use case the signal is shifted using NCO1 or NCO2 (or both/none) such that exists solely on one side of the LO. Once this is the case, the signal no longer needs to be complex represented and only I data is sent across the link, Q data being dropped. The interpolate by 2 stage may also need to be used to achieve this.

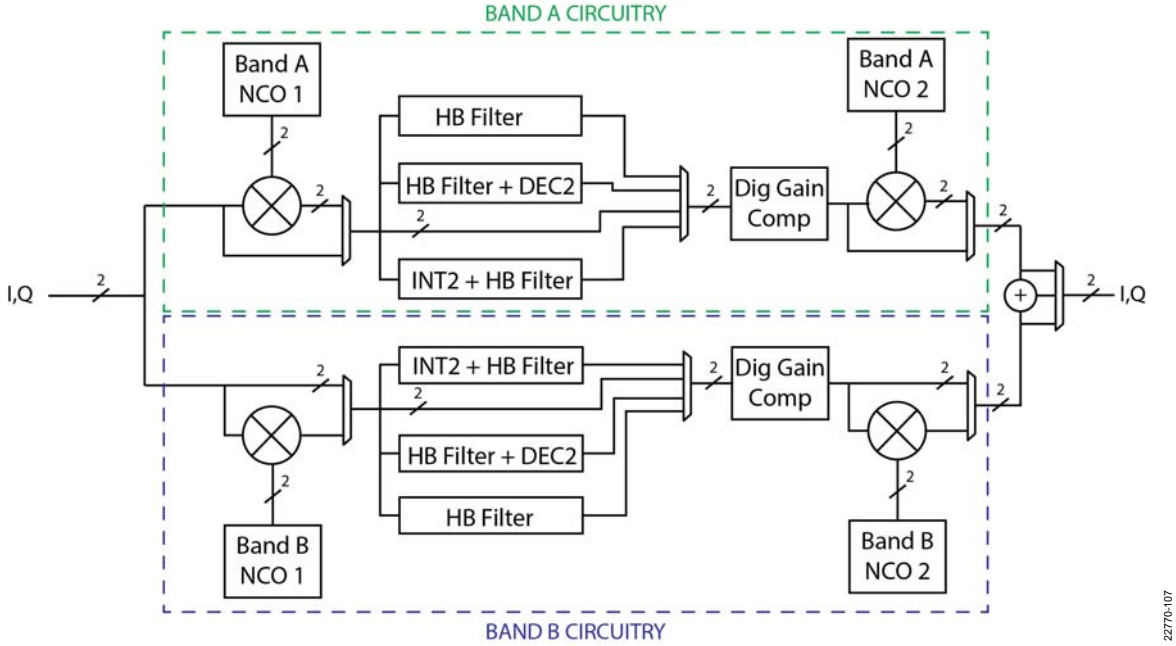


Figure 102. Block Diagram of the IF Conversion Stage (Note that all circuitry is implemented in quadrature as indicated)

Zero- IF to Real-IF

In this use case the received signal is centered around the LO. The signal is interpolated by 2, and half-band filtered. The Band A NCO2 is used to upshift or downshift the data to generate a signal that is symmetrical about 0 Hz. The result is that the spectrum no longer requires a complex representation, and only I data is sent across the link, Q data being dropped.

Dual Band Mode

In this use case there are multiple signals being received, referred to as Signal 1 and Signal 2. Band A circuitry can be used to process Signal 1, and Band B to process Signal 2. Band A NCO1 is used to shift Signal 1 such that it is placed within the pass band of the half band Filter such that it filters out signal 2. The decimate by 2 stage can also be used if the final composite bandwidth allows for a lower data rate across the JESD link. The Band A NCO2 stage is then used to offset the signal to the required position in the spectrum. Likewise, the same procedure is performed on Signal 2. The result is that the two signals originally located far apart in the spectrum, and thus requiring a high data rate, can be moved closer together with this IF conversion circuitry, and represented by a lower IQ rate.

Dual Band Mode (Real IF)

In this use case the signals are processed separately using Band A and Band B. The NCO2 stages are used to shift both signals so they exist on the same side of LO. At this point the spectrum no longer needs a complex representation and only I data can be sent across the link, Q data being dropped. The interpolate by 2 stage may also need to be used to achieve this.

HB Filter Only Mode

If there is a blocker to one side of the signal, it is possible to use the IF conversion stage to obtain further rejection of the blocker. Band A NCO1 is used to offset the signal such that the signal is positioned close to the edge of the pass band of the half-band filter, and that the blocker is positioned in the transition or stopband of the filter. The Band A NCO2 can be used to position the desired signal to its previous position within the spectrum if required.

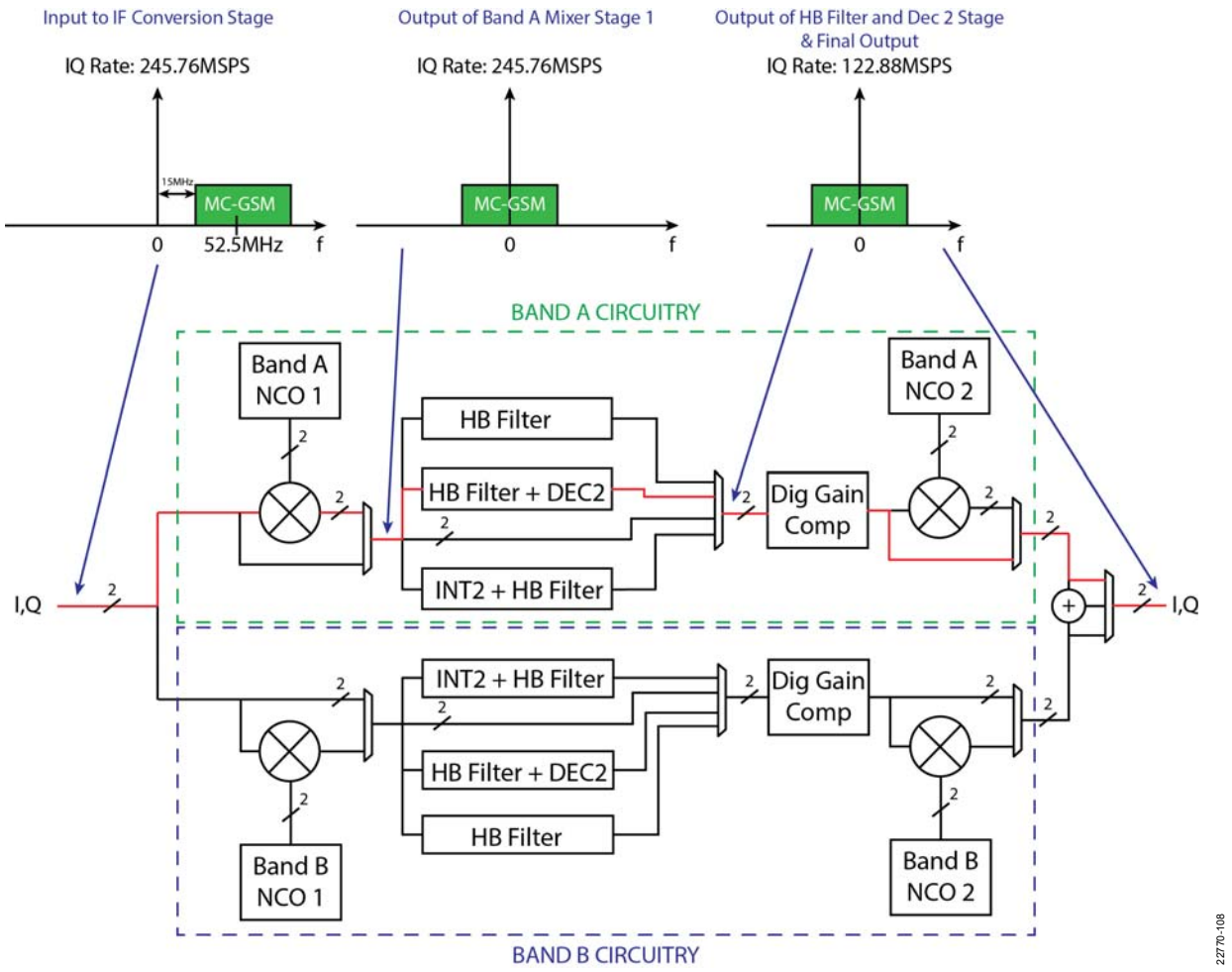


Figure 103. Block Diagram of the IF Conversion Stage in Zero-IF MC- GSM Configuration. The red line indicates the path of the signal through the IF conversion stage. The spectrums above show how the signal is shifted, filtered and decimated.

RECEIVER SIGNAL PATH EXAMPLE

The Transceiver Evaluation Software provides an example depicting how the baseband filtering stages are used in profile configurations for a signal pathway. In this example, the ADRV9025Init_StdUseCase26_nonLinkSharing profile is selected for the Rx channels. This is a 200 MHz profile with an IQ rate of 245.76 MSPS.

Figure 104 shows the filter configuration for this profile. The signal rate after the PFIR block is equal to the IQ rate of the profile.

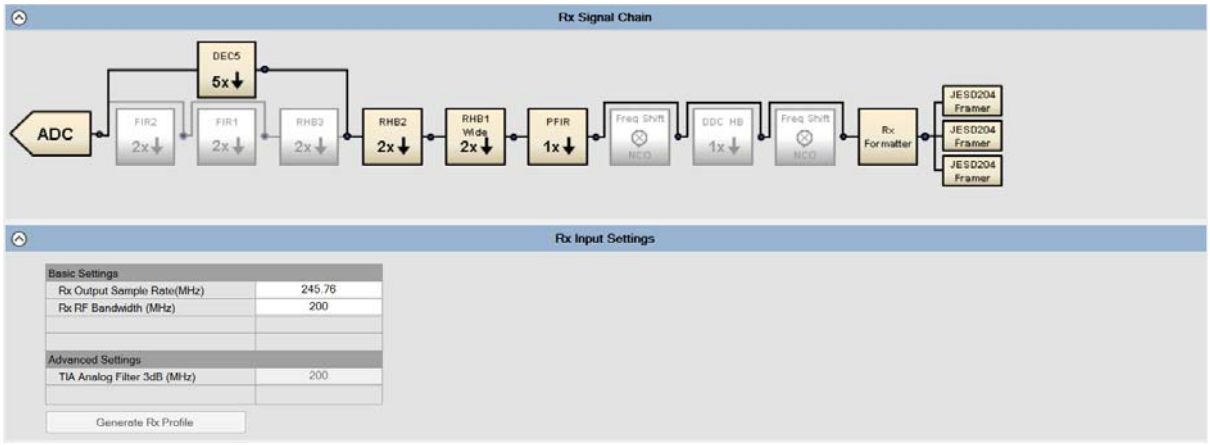


Figure 104. Filter Configuration for the Rx 200 MHz, IQ Rate 245.76 MSPS

The Transceiver Evaluation Software also provides a graph of the complete signal chain transfer function for this profile in the **Rx** tab under the **ChipConfig** dropdown. This is shown in Figure 105.

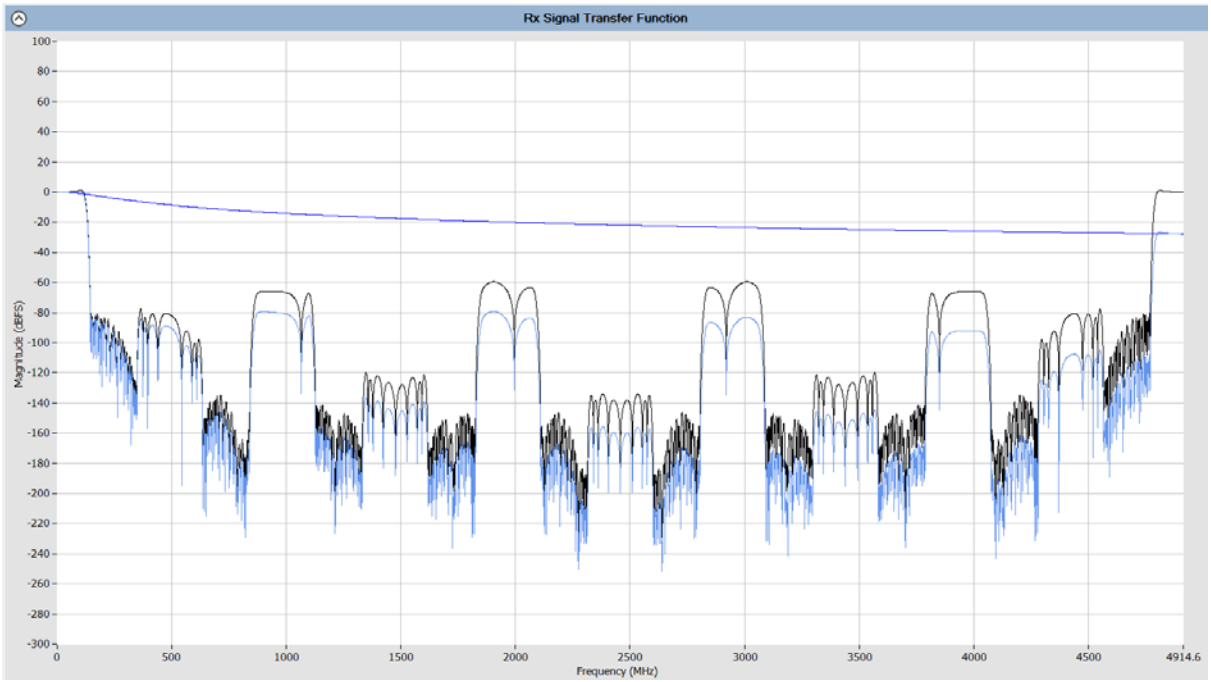


Figure 105. Rx Signal Transfer Function

RECEIVER FILTER API STRUCTURE

The filter configuration is stored in the adi_adrv9025_RxProfile_t structure. This structure is stored within the adi_adrv9025_RxSettings structure, which is stored in the overall device initialization structure (adi_adrv9025_Init_t). It contains the parameters listed in Table 186.

Table 186. adi_adrv9025_RxProfile_t Structure Parameters

Name	Value	Description
channelType	A value of type adi_adrv9025_RxChannels_e	Choose what channel to configure the filters described in Table 187
rxFirDecimation	1, 2, 4	Rx FIR Decimation setting
rxDec5Decimation	4 = Use combination of FIR1, FIR2, and/or RHB3; 5 = Use Dec5	Setting to use either the Dec5 or HB3 and HB2 in the ORx path
rhb1Decimation	1 = bypass , 2 = in use	Rx HB1 Decimation setting
rhb1WideBandMode	0 – HB1 is narrow, 1 – HB1 is wider	ORx and loopback profiles ignore this field
rhb2Decimation	1, 2	RX Half-Band 2 (HB2) decimation factor
rhb3Decimation	1, 2	RX Half-Band 3 (HB3) decimation factor
rxFir1Decimation	1, 2	Rx FIR1 decimation factor
rxFir2Decimation	1, 2	Rx FIR2 decimation factor; ORx and loopback profiles ignore this field
rxOutputRate_kHz	30720 to 368640 (based on currently defined use cases)	IQ data rate specified in kHz (to the input of the JESD block)
rfBandwidth_kHz	20000 to 200000 (based on currently defined use cases)	The RF bandwidth specified in kHz
rxBbf3dBCorner_kHz	20000 to 200000 (based on currently defined use cases)	The BBF 3 dB corner frequency specified in kHz
rxAdcBandWidth_kHz	10000 to 100000 (based on currently defined use cases)	Rx ADC bandwidth tunes the bandwidth of the pass band and noise transfer functions of the ADC
rxFir	A value of type adi_adrv9025_RxFir_t	The Rx FIR filter structure is described in Table 188
rxDdcMode	A value of type adi_adrv9025_RxDdc_e	The Rx DDC mode settings are described in Table 189
rxNcoShifterCfg	A value of type adi_adrv9025_RxNcoShifterCfg_t	The Rx NCO Shifter Configuration structure is described in Table 190
tiaPowerMode	0, 1, 2, 3	4 options for TIA power reduction modes (range 0-3)

Table 187. adi_adrv9025_RxChannels_e Enum Definition

adi_adrv9025_RxChannels_e Enum	Enabled Channels
ADI_ADRV9025_RXOFF	No Rx or ORx channels enabled
ADI_ADRV9025_RX1	Rx1 Enabled
ADI_ADRV9025_RX2	Rx2 Enabled
ADI_ADRV9025_RX3	Rx3 Enabled
ADI_ADRV9025_RX4	Rx4 Enabled
ADI_ADRV9025_ORX1	ORx1 Enabled
ADI_ADRV9025_ORX2	ORx2 Enabled
ADI_ADRV9025_ORX3	ORx3 Enabled
ADI_ADRV9025_ORX4	ORx4 Enabled
ADI_ADRV9025_LB12	Tx1 or Tx2 internal loopback into ORx1/2 channel enabled
ADI_ADRV9025_LB34	Tx3 or Tx4 internal loopback into ORx3/4 channel enabled

Rx PFIR Settings

The Rx PFIR is specified in signed coefficients from +32767 to -32768. The gain block allows for more flexibility when designing a digital filter. For example, a FIR can be designed with 6dB gain in the pass band, and then this block can be set to −6 dB gain to give an overall 0 dB gain in the pass band. The gain of the filter coefficients can be calculated as follows:

$$DC\ Gain = \frac{\sum FIR\ Coefficients}{2^{15}-1}$$

Table 188. adi_adrv9025_RxFir_t Structure Parameters

Name	Value	Description
gain_dB	−12, −6, 0, +6	The setting (in dB) for the gain block within the Rx FIR
numFirCoefs	24, 48, 72	Number of taps to be used in the Rx FIR
coefs[ADI_ADRV9025_MAX_RXPFIR_COEFS]	A pointer to an array of filter coefficients of size ADI_ADRV9025_MAX_RXPFIR_COEFS	

Rx DDC Mode

The Rx DDC Mode is defined within the adi_adrv9025_RxProfile_t structure as an enumerated type from the adi_adrv9025_RxDdc_e type definition. Permissible values are listed in Table 189.

Table 189. adi_adrv9025_RxDdc_e Enum Definition

adi_adrv9025_RxDdc_e Enum	Description
ADI_ADRV9025_RXDDC_BYPASS	In this mode, the half-band filter and interpolation/decimation stages are bypassed.
ADI_ADRV9025_RXDDC_FILTERONLY	In this mode, the half-band filter stage is used, but the interpolation and decimation stages are bypassed.
ADI_ADRV9025_RXDDC_INT2	In this mode, the interpolate by 2 and half-band filter stages are utilized.
ADI_ADRV9025_RXDDC_DEC2	In this mode, the half-band filter and decimate by 2 stages are utilized.
ADI_ADRV9025_RXDDC_BYPASS_REALIF	In this mode, the half-band filter and interpolation/decimation stages are bypassed. At the input to the JESD core, Q data is dropped.
ADI_ADRV9025_RXDDC_FILTERONLY_REALIF	In this mode, the half-band filter stage is used, but the interpolation and decimation stages are bypassed. At the input to the JESD core, Q data is dropped.
ADI_ADRV9025_RXDDC_INT2_REALIF	In this mode, the interpolate by 2 and half-band filter stages are utilized. At the input to the JESD core, Q data is dropped.
ADI_ADRV9025_RXDDC_DEC2_REALIF	In this mode, the half-band filter and decimate by 2 stages are utilized. At the input to the JESD code, Q data is dropped.

Rx NCO Shifter Configuration

The adi_adrv9025_RxNcoShifterCfg_t structure is contained within the adi_adrv9025_RxProfile_t structure. It contains the settings of the NCO stages of Band A and Band B, as well as the bandwidth and baseband center frequency of the desired signal(s). This allows the API to ensure that the IF conversion stage has been correctly setup, and that the signal(s) post NCO shifting is falling within the bandwidth provided by the IQ rate being utilized, and the pass-band bandwidth of the half-band filter if utilized.

The NCOs are able to be configured according to the following rules:

- bandwidthDiv2 = (bandAInputBandwidth_kHz/2) × 1000
- inputCenterFreq = (bandAInputCenterFreq_kHz) × 1000
- nco1OutputCenterFreq = (bandAInputCenterFreq_kHz + bandANco1Freq_kHz) × 1000
- nco2OutputCenterFreq = nco1OutputCenterFreq + (bandANco2Freq_kHz) × 1000
- outputRateHz = IQ Data rate of the Rx UseCase
- primaryBwHz = Primary Rx signal bandwidth of the Rx UseCase
- ddcHbCorner depends on the mode used:
  - If RXDDC_FILTERONLY, RXDDC_FILTERONLY_REALIF, RXDDC_INT2, RXDDC_INT2_REALIF at the ddcHbCorner = outputRateHz × 0.2
  - If RXDDC_DEC2, RXDDC_DEC2_REALIF at the ddcHbCorner = outputRateHz × 0.4

Range Checks (Total of 6 rules)

Rule 1: Input Center Frequency Setup

- inputCenterFreq + bandWidthDiv2 > primaryBwHz/2
- inputCenterFreq − bandWidthDiv2 < −primaryBwHz/2

Rule 2: Output Center Frequency Setup NCO1. If DDC HB is enabled,

- nco1OutputCenterFreq + bandWidthDiv2 > ddcHbCorner
- nco1OutputCenterFreq − bandWidthDiv2 < −ddcHbCorner

Rule 3: Output Center Frequency Setup NCO2

- nco2OutputCenterFreq + bandWidthDiv2 > outputRateHz/2
- nco2OutputCenterFreq − bandWidthDiv2 < −outputRateHz/2

Table 190. adi_adrv9025_RxNcoShifterCfg_t Structure Parameters

adi_adrv9025_RxNcoShifterCfg_t	Description
bandAInputBandWidth_kHz	The bandwidth of the received signal being processed in Band A specified in kHz.
bandAInputCenterFreq_kHz	The center frequency, in terms of baseband frequencies, of the received signal being process in Band A, specified in kHz.
bandANco1Freq_kHz	The frequency shift to be provided by NCO1 of Band A specified in kHz. Positive values shift the spectrum up in frequency; negative values shift the spectrum down in frequency.
bandANco2Freq_kHz	The frequency shift to be provided by NCO2 of Band B specified in kHz. Positive values shift the spectrum up in frequency; negative values shift the spectrum down in frequency.
bandBInputBandWidth_kHz	The bandwidth of the received signal being processed in Band B specified in kHz.
bandBInputCenterFreq_kHz	The center frequency, in terms of baseband frequencies, of the received signal being process in Band B, specified in kHz.
bandBNco1Freq_kHz	The frequency shift to be provided by NCO1 of Band B specified in kHz. Positive values shift the spectrum up in frequency; negative values shift the spectrum down in frequency.
bandBNco2Freq_kHz	The frequency shift to be provided by NCO2 of Band B specified in kHz. Positive values shift the spectrum up in frequency; negative values shift the spectrum down in frequency.
bandAbCombinedEnable	The frequency shift to be provided by the combination of Band A and Band B at output, 1 = combine dual-band AB, 0 = disable combine dualband on AB

Note that dual-band mode is selected when the input bandwidths of Band A and Band B are both specified (nonzero). In nondual band modes, specify Band A settings only, with Band B left with zero settings. Likewise, if the NCO stages of both Band A and Band B are not to be used, provide zero settings for all variables in the adi_adrv9025_RxNcoShifterCfg_t structure.

TRANSMITTER SIGNAL PATH

Each transmitter has an independent signal path including separate digital filters, DACs, analog low-pass filters, and I/Q mixers that drive the signal outputs. Data is input to the Tx signal path via the JESD204B high-speed serial data interface at the IQ data rate of the transmitter profile. The serial data is converted to parallel format through the JESD204B deframer into I and Q components. The data is processed through digital filtering and signal correction stages and input to I/Q DACs.

The DAC output is low pass filtered by the Tx low-pass filter (LPF) and input to the upconversion mixer. The I and Q paths are identical to one another. Over-ranging is detected in the Tx digital signal path at each stage and limited to the maximum code value to prevent data wrapping. A block diagram of a Tx signal path is shown in Figure 106. Blocks that are not discussed in this section are faded.

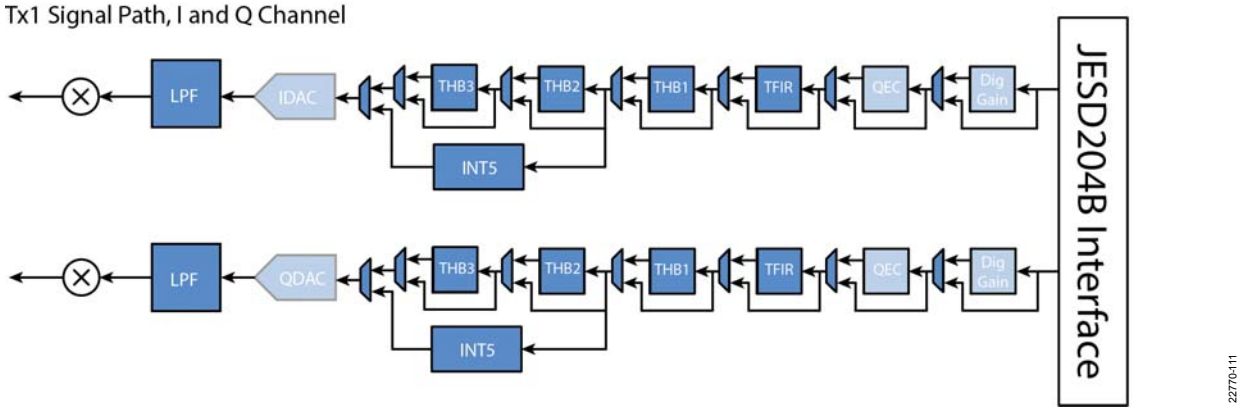


Figure 106. Tx Signal Path Diagram

Analog Low Pass Filter (LPF)

The LPF is a second-order analog Butterworth low pass filter with an adjustable 3dB corner. The Tx chains of the device can support pass-band bandwidths up to 225 MHz (on I and Q). The LPF is calibrated during device initialization, resulting in a consistent frequency corner across all devices. The LPF bandwidth is set within the device data structure and is profile dependent. Roll off within the analog LPF pass band is compensated by the TFIR to ensure a maximally flat pass-band frequency response.

Interpolation By 5 Filter (INT5)

Either the INT5 or any combination of THB3 and THB2 are used in the Tx digital path. The INT5 interpolates by a factor of 5. The INT5 coefficients are listed as follows:

INT5 filter coefficients: 0.002929688, 0.029052734, −0.029296875, 0.03125, −0.012207031, −0.005859375, −0.056640625, 0.051513672, −0.055664063, 0.025390625, 0.020996094, 0.081298828, −0.057617188, 0.072509766, −0.045166016, −0.047607422, −0.095947266, 0.030517578, −0.071289063, 0.068603516, 0.093994141, 0.113769531, 0.030761719, 0.055419922, −0.103759766, −0.185791016, −0.185302734, −0.136962891, −0.037353516, 0.227050781, 0.518554688, 0.717285156, 0.928466797, 1.019287109, 0.928466797, 0.717285156, 0.518554688, 0.227050781, −0.037353516, −0.136962891, −0.185302734, −0.185791016, -0.103759766, 0.055419922, 0.030761719, 0.113769531, 0.093994141, 0.068603516, −0.071289063, 0.030517578, −0.095947266, −0.047607422, −0.045166016, 0.072509766, −0.057617188, 0.081298828, 0.020996094, 0.025390625, −0.055664063, 0.051513672, −0.056640625, −0.005859375, −0.012207031, 0.03125, -−0.029296875, 0.029052734, 0.002929688

Transmit Half Band 3 THB3

The THB3 is a fixed coefficient half-band interpolating filter. THB3 can interpolate by a factor of 2 or it can be bypassed. The coefficients are listed as follows.

THB3 filter coefficients: 0.125, 0.5, 0.75, 0.5, 0.125

Transmit Half Band 2 (THB2)

The THB2 is a fixed coefficient half-band interpolating filter. THB2 can interpolate by a factor of 2 or it can be bypassed. The coefficients are listed below.

THB2 filter coefficients: −0.08203125, 0, 0.58203125, 1, 0.58203125, 0, −0.08203125

Transmit Half Band 1 (THB1)

The THB1 is a fixed coefficient half band interpolating filter. THB1 interpolates by a factor of 2 or it can be bypassed. The coefficients are listed as follows.

THB1 filter coefficients: −0.002319336, 0, 0.003601074, 0, −0.004058838, 0, 0.004119873, 0, −0.006439209, 0, 0.009613037, 0, −0.012023926, 0, 0.014404297, 0, −0.018737793, 0, 0.024291992, 0, -0.030059814, 0, 0.037353516, 0, −0.048156738, 0, 0.062927246, 0, −0.084350586, 0, 0.122283936, 0, −0.209564209, 0, 0.635925293, 1, 0.635925293, 0, −0.209564209, 0, 0.122283936, 0, −0.084350586, 0, 0.062927246, 0, −0.048156738, 0, 0.037353516, 0, −0.030059814, 0, 0.024291992, 0, −0.018737793, 0, 0.014404297, 0, −0.012023926, 0, 0.009613037, 0, −0.006439209, 0, 0.004119873, 0, −0.004058838, 0, 0.003601074, 0, −0.002319336

Programmable Transmitter Finite Impulse Response (TFIR)

The TFIR filter acts as an interpolating filter in the TX path. The TFIR may interpolate by a factor of 1, 2, or 4, or it can be bypassed. The TFIR is used to compensate for roll off caused by the post-DAC analog low pass filter. The TFIR has a configurable number of taps; either 20, 40, 60, or 80 taps can be used. The TFIR also has a programmable gain setting of +6 dB, 0 dB, −6 dB or −12 dB.

The maximum number of taps is limited by the TFIR Clock Rate (Data Processing Clock – DPCLK). The maximum DPCLK is 1 GHz. The DPCLK is the high speed digital clock (HSDIG_CLK) divided by either 4 or 5 depending on the HSDIG_CLK divider setting. The DPCLK affects the maximum number of TFIR filter taps that can be used according to the following relationship:

$$Tx\ PFIR\ filter\ taps_{max} = \frac{DPCLK_{max}}{Tx\_IQ\_DataRate} \times 20$$

TX SIGNAL PATH EXAMPLE

The Transceiver Evaluation Software provides an example depicting how the baseband filtering stages are used in profile configurations for a signal data path. In this example, the ADRV9025Init_StdUseCase26_nonLinkSharing profile is selected for the Tx channels. This is a 200 MHz/450 MHz profile with IQ Rate 491.52 MSPS.

To explain the terminology of the 200 MHz/450 MHz profile, the 200 MHz refers to the Tx primary signal bandwidth, whereas the 450 MHz refers to Tx RF bandwidth.

Figure 107 shows the filter configuration for this profile. The signal rate after the TFIR block is equal to the IQ rate of the profile.

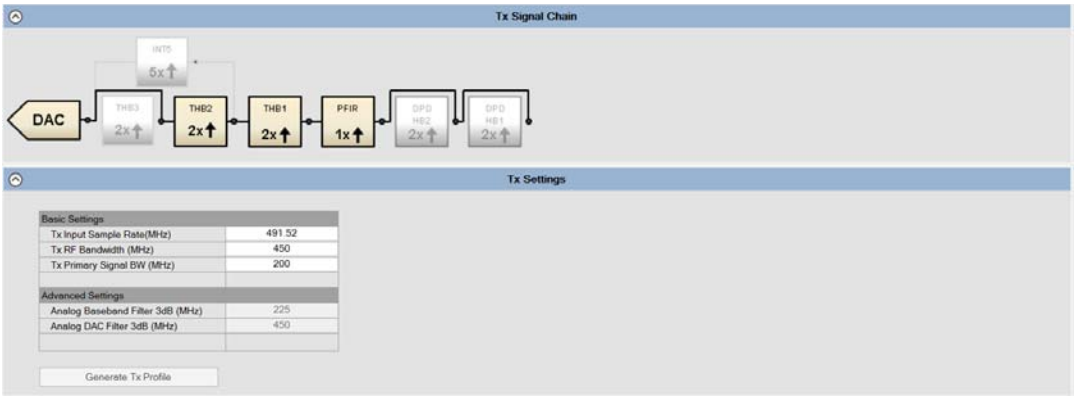


Figure 107. Filter Configuration for the Tx 200 MHz/450 MHz, 491.52 MSPS Profile

The combined Tx signal transfer function can be found in the **Tx** tab under the **ChipConfig** dropdown menu as shown in Figure 108.

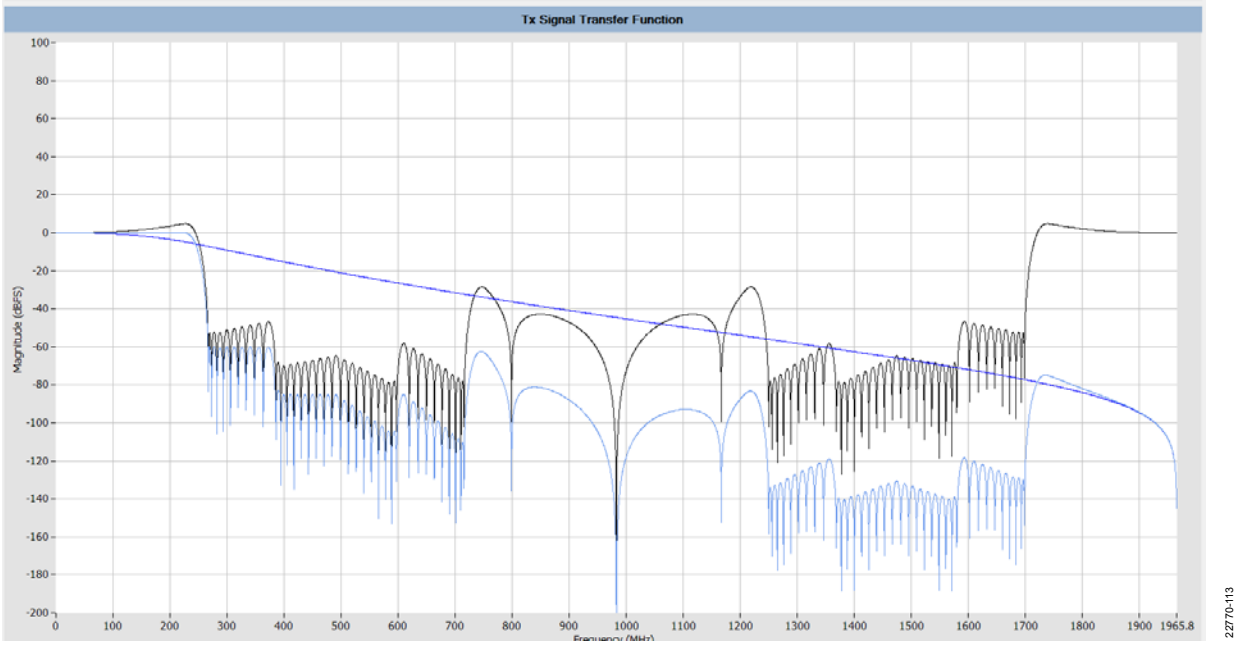


Figure 108. Tx Signal Transfer Function

TRANSMITTER FILTER API STRUCTURE

The filter configuration is stored in the adi_adrv9025_TxProfile_t structure. This structure is stored within the adi_adrv9025_TxSettings_t structure, which is stored in the overall device initialization structure (adi_adrv9025_Init_t). Its parameters are described in Table 191.

Table 191. adi_adrv9025_TxProfile_t Structure Parameters

Name	Value	Description
txInputRate_kHz	30720 to 491520 (based on currently defined use cases)	IQ data rate at the input to the TFIR specified in kHz
primarySigBandwidth_kHz	20000 to 200000 (based on currently defined use cases)	Primary signal bandwidth specified in kHz
rfBandwidth_kHz	100000 to 450000 (based on currently defined use cases)	The RF bandwidth specified in kHz
txDac3dBCorner_kHz	100000 to 450000 (based on currently defined use cases)	The DAC 3dB corner specified in kHz
txBbf3dBCorner_kHz	50000 to 225000 (based on currently defined use cases)	The BBF 3dB corner frequency specified in kHz
txFirInterpolation	1, 2, 4	Tx FIR interpolation setting
thb1Interpolation	1 = bypass, 2 = in use	Tx HB1 interpolation setting
thb2Interpolation	1 = bypass, 2 = in use	Tx HB2 interpolation setting
thb3Interpolation	1 = bypass, 2 = in use	Tx HB3 interpolation setting
txInt5Interpolation	1 = bypass, 5 = in use	Tx INT5 interpolation setting
txFir	A value of type adi_adrv9025_TxFir_t	txFir structure explained in detail in the Tx FIR Settings section
txBbfPowerMode	0 to 8	The Tx BBF power scaling mode selection between 0 and 8, where a value of 8 allows the Arm to set the power mode based on the LUT of power saving

Tx FIR Settings

The adi_adrv9025_TxFir_t structure is contained within the adi_adrv9025_TxProfile_t structure. Its parameters are described in Table 192.

Table 192. adi_adrv9025_TxFir_t Structure Parameters

Name	Value	Description
gain_dB	−12, −6, 0, +6	The setting (in dB) for the gain block within the Tx FIR
numFirCoefs	20, 40, 60, 80	Number of taps to be used in the Tx FIR
coefs[ADI_ADRV9025_MAX_TXPFIR_COEFS]	A pointer to an array of filter coefficients of size ADI_ADRV9025_MAX_TXPRIF_COEFS	

The Tx FIR is specified in signed coefficients from +32,767 to −32,768. The gain block allows for more flexibility when designing a digital filter. For example, a FIR can be designed with 6 dB gain in the pass band, and then this block can be set to −6 dB gain to give an overall 0 dB gain in the pass band. The gain of the filter coefficients can be calculated as follows:

$$DC\ Gain = \frac{\sum FIR\ Coefficients}{2^{15}-1}$$

OBSERVATION RECEIVERS SIGNAL PATH

The device has four observation receivers (ORx1, ORx2, ORx3 and ORx4) that can be used to capture data for digital pre-distortion (DPD) algorithms and other measurements/calibration that require monitoring the transmitter outputs. The observation receiver can serve as an external loopback path to loop back the output of a PA, provided input level to the ORx is below the full-scale level of the ADC.

The devices ORx1, ORx2, ORx3 and ORx4 channels have separate I/Q mixers. These mixers are identical to the mixers of the receivers, with the exception that the observation mixers include an LO mux. The LO mux allows either the RF PLL or the AUX PLL to provide the local oscillator signal source for the ORx1, ORx2, ORx3 and ORx4 mixers.

The mixer feeds into a programmable transimpedance amplifier (TIA) that serves as a low pass filter (LPF) in the analog data path. The signal is converted by the sigma-delta ADC and filtered in halfband decimation stages and the programmable finite impulse response (PFIR). The fixed coefficient halfband filters (FIR1, RHB1(HR), RHB1(LP), RHB2, RHB3, DEC5) and the PFIR are designed to prevent data wrapping and overrange conditions.

The IF conversion stage provides the ability frequency shift or upsample/downsample digital data. Configurations supported include real IF (real valued baseband data) configuration and low IF (complex data) configuration.

The diagram in Figure 109 shows the signal path for an ORx signal chain. Blocks that are not discussed in this section are faded.

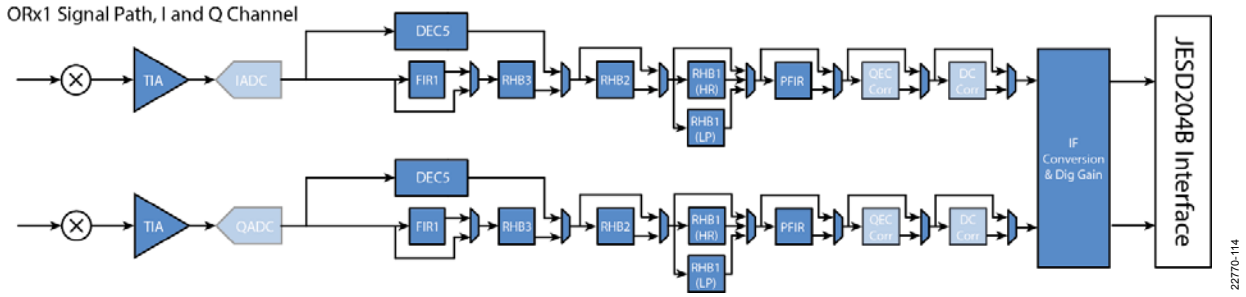


Figure 109. ORx Signal Path

Transimpedance Amplifier (TIA)

The ORx transimpedance amplifier is a low pass filter with a single real pole frequency response. The TIA can support pass-band bandwidths up to 225 MHz (for both I and Q). The TIA is calibrated during device initialization which ensures a consistent frequency corner across all devices. The TIA 3 dB bandwidth is set within the device data structure and is profile dependent. Roll-off within the ORx pass band is compensated by the PFIR to ensure a maximally flat pass-band frequency response.

DEC5

Either the DEC5, or the combination of RHB3 and FIR1 is used in the Rx digital path. The DEC5 decimates by a factor of 5 or it may be bypassed. The DEC5 coefficients are listed below.

DEC5 filter coefficients: 0.000732422, 0.001464844, 0.002441406, 0.003417969, 0.003173828, −0.000732422, −0.005615234, −0.013183594, −0.020507813, −0.022949219, −0.014648438, 0.003417969, 0.035400391, 0.077392578, 0.119873047, 0.154541016, 0.176269531, 0.176269531, 0.154541016, 0.119873047, 0.077392578, 0.035400391, 0.003417969, −0.014648438, −0.022949219, −0.020507813, −0.013183594, −0.005615234, −0.000732422, 0.003173828, 0.003417969, 0.002441406, 0.001464844, 0.000732422

Finite Impulse Response 1 (FIR1)

The FIR1 filter is a fixed coefficient decimating filter. The FIR1 decimates by a factor of 2 or it may be bypassed.

FIR1 filter coefficients: 0.25, 0.75, 0.75, 0.25

Receive Half Band 3 (RHB3)

The RHB3 filter is a fixed coefficient decimating filter. The RHB3 decimates by a factor of 2 or it may be bypassed. The RHB3 coefficients are listed below.

RHB3 filter coefficients: −0.0625, 0.0078125, 0.5625, 0.984375, 0.5625, 0.0078125, −0.0625

Receive Half Band 2 (RHB2)

The RHB2 filter is a fixed coefficient decimating filter. The RHB2 decimates by a factor of 2 or it may be bypassed. The RHB2 coefficients are listed below.

RHB2 filter coefficients: −0.002929688, 0, 0.018554688, 0, −0.0703125, 0, 0.3046875, 0.500976563, 0.3046875, 0, −0.0703125, 0, 0.018554688, 0, −0.002929688

Receive Half Band 1 High Rejection (RHB1 (HR))

The RHB1 (HR) filter is a fixed coefficient decimating filter. The RHB1 can decimate by a factor of 2, or it may be bypassed. The RHB1 coefficients are listed as follows.

RHB1 filter coefficients: −0.000732422, 0, 0.000732422, 0, −0.001098633, 0, 0.001586914, 0, −0.00213623, 0, 0.002929688, 0, −0.00378418, 0, 0.004882813, 0, −0.006225586, 0, 0.007873535, 0, −0.009887695, 0, 0.012329102, 0, −0.015380859, 0, 0.019226074, 0, −0.024353027, 0, 0.031555176, 0, -0.042419434, 0, 0.061462402, 0, −0.104797363, 0, 0.317871094, 0.5, 0.317871094, 0, −0.104797363, 0, 0.061462402, 0, −0.042419434, 0, 0.031555176, 0, −0.024353027, 0, 0.019226074, 0, -0.015380859, 0, 0.012329102, 0, −0.009887695, 0, 0.007873535, 0, −0.006225586, 0, 0.004882813, 0, −0.00378418, 0, 0.002929688, 0, -0.00213623, 0, 0.001586914, 0, −0.001098633, 0, 0.000732422, 0, −0.000732422

Receive Half Band 1 Low Power (RHB1 (LP))

The RHB1 (LP) filter is a fixed coefficient decimating filter. The RHB1 can decimate by a factor of 2, or it may be bypassed. The RHB1 coefficients are listed below.

RHB1 filter coefficients: −0.002685547, 0, 0.017333984, 0, −0.068359375, 0, 0.304443359, 0.501708984, 0.304443359, 0, −0.068359375, 0, 0.017333984, 0, −0.002685547

PFIR

The PFIR filter acts as a decimating filter. The PFIR may decimate by a factor of 1, 2, or 4, or it can be bypassed. The PFIR is used to compensate for the roll off of the analog TIA LPF. The PFIR can use either 24, 48, or 72 filter taps. The PFIR also has programmable gain settings of +6 dB, 0 dB, −6 dB or −12 dB.

The maximum number of taps is limited by the FIR Clock Rate (Data Processing Clock - DPCLK). The maximum DPCLK is 1 GHz. The DPCLK is the ADC Clock Rate divided by either 4 or 5. It is 4 when using the HB2 and HB3 filters. It is 5 when using the DEC5 filter. The DPCLK affects the maximum number of RFIR filter taps that can be used according to the following relationship:

$$ORx\ PFIR\ filter\ taps_{max} = \frac{DPCLK_{max}}{ORx\_IQ\_DataRate} \times 24$$

IF Conversion

Refer to the equivalent Receiver Signal Path section about the IF conversion stage.

OBSERVATION RECEIVER SIGNAL PATH EXAMPLE

The Transceiver Evaluation Software provides an example depicting how the baseband filtering stages are used in profile configurations for a signal pathway. In this example, the ORx 450 MHz, IQRate 491.52 MSPS profile is selected for the ORx channels. This profile is compatible with the other examples provided in this document.

Figure 110 shows the filter configuration for this profile. The clocking frequencies are noted in blue. The signal rate after the RFIR block is equal to the IQRate of the profile.

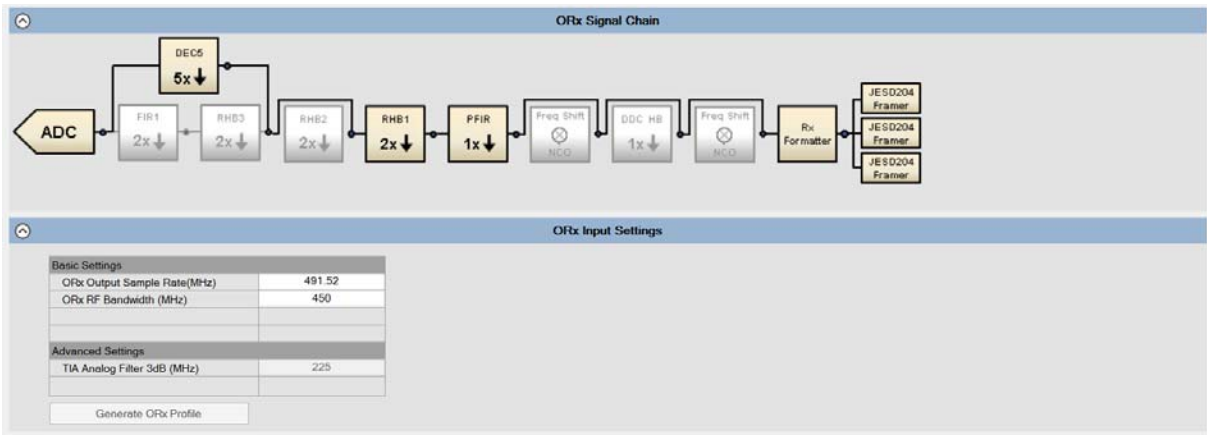


Figure 110. Filter Configuration for ORx 450 MHz, IQ Rate 491.52 MSPS

In the **ORx** tab under the **ChipConfig** dropdown menu the ORx signal transfer function of the signal chain can be found as shown in Figure 111.

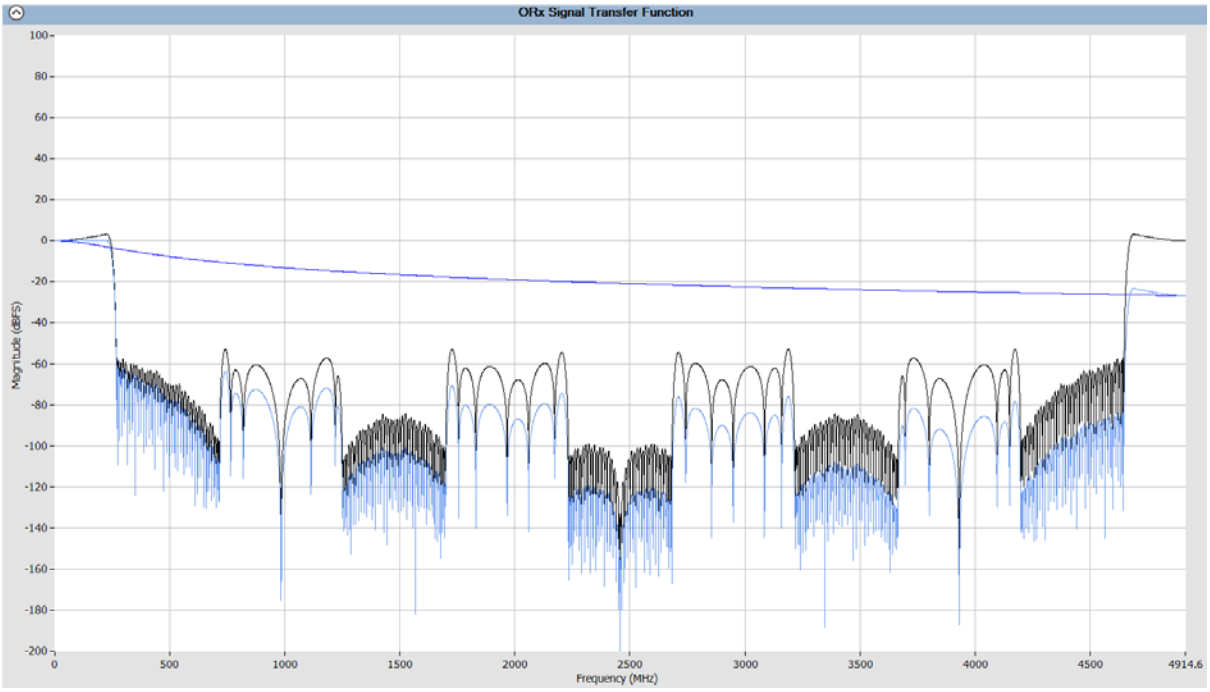


Figure 111. ORx Signal Transfer Function

OBSERVATION RECEIVER FILTER API STRUCTURE

The filter configuration is stored in the adi_adrv9025_RxProfile_t structure. This structure is stored within the adi_adrv9025_RxSettings structure, which is stored in the overall device initialization structure (adi_adrv9025_Init_t). It contains the parameters listed in Table 193. For further details refer to the Receiver Filter API Structure section.

Table 193. adi_adrv9025_RxProfile_t Structure Parameters

Name	Value	Description
channelType	A value of type adi_adrv9025_RxChannels_e	Choose what channel to configure the filters described in Table 187
rxFirDecimation	1, 2, 4	ORx FIR Decimation setting
rxDec5Decimation	4 = use combination of FIR1, FIR2, and/or RHB3, 5 = Use Dec5	Setting to use either the Dec5 or HB3 and HB2 in the ORx path
rhb1Decimation	1 = bypass, 2 = in use	ORx HB1 decimation setting
rhb1WideBandMode	0 = HB1 is narrow, 1 = HB1 is wider	ORx and loopback profiles ignore this field
rhb2Decimation	1, 2	ORx Half-Band2 (HB2) decimation factor
rhb3Decimation	1, 2	ORx Half-Band3 (HB3) decimation factor
rxFir1Decimation	1, 2	ORx FIR decimation factor
rxFir2Decimation	1, 2	Rx FIR decimation factor; ORx and loopback profiles ignore this field
rxOutputRate_kHz	122880 to 491520 (based on currently defined use cases)	IQ Data rate specified in kHz (to the input of the JESD block)
rfBandwidth_kHz	112500 to 450000 (based on currently defined use cases)	The RF bandwidth specified in kHz
rxBbf3dBCorner_kHz	112500 to 450000 (based on currently defined use cases)	The BBF 3 dB corner frequency specified in kHz
rxAdcBandWidth_kHz	56250 to 225000 (based on currently defined use cases)	Rx ADC bandwidth tunes the bandwidth of the pass band and noise transfer functions of the ADC
rxFir	A value of type adi_adrv9025_RxFir_t	The Rx FIR filter structure is described in Table 188
rxDdcMode	A value of type adi_adrv9025_RxDdc_e	The Rx DDC mode settings are described in Table 189
rxNcoShifterCfg	A value of type adi_adrv9025_RxNcoShifterCfg_t	The Rx NCO shifter configuration structure is described in Table 190
tiaPowerMode	0, 1, 2, 3	Four options for TIA power reduction modes (range 0 to 3)
rxDataFormat	A value of type adi_adrv9025_RxDataFormat_t	This structure is explained in the Gain Compensation, Floating Point Formatter and Slicer section and Table 181

GENERAL-PURPOSE INPUT/OUTPUT CONFIGURATION

The device features nineteen (19) digital General-Purpose Input/Output (GPIO) pins that can be used for a variety of functions. The device also features eight analog General-Purpose Input/Output (GPIO_ANA) pins. The GPIO/GPIO_ANA pins provide a real-time interface for the baseband processor to control the transceiver or for the transceiver to send information to the baseband processor. An example of baseband processor control uses rising edges sent by the baseband processor over user assigned GPIO pins to increase or decrease the transmitter attenuation. An example of the transceiver sending information to the baseband processor is the ability to send overload detection information from peak detectors in the receiver datapath to advise that input signal level is too high.

The GPIO_ANA pins serve as the output pins for 8 AuxDAC signals. The AuxDAC can be used for providing a control voltage. The AuxDAC is not a precision converter device and is recommended to be used in applications where high accuracy is not needed. It is best to use the AuxDAC in feedback systems rather than in open-loop control systems.

The digital GPIO supply is the VDD_IF supply voltage. The GPIO_ANA supply is the VDDA_1P8 supply voltage. IBIS models have been created to assist in the simulation of these interfaces.

DIGITAL GPIO OPERATION

Each digital GPIO pin can be set to either input or output mode. In this section, input and output mode are oriented with respect to the transceiver device. The input mode allows the baseband processor to drive pins on the transceiver to execute specific tasks. The output mode allows the device to output various signals.

The digital GPIO pin I/O direction can be set with the following API commands.

adi_adrv9025_GpioInputDirSet(…)

```
adi_adrv9025_GpioInputDirSet(adi_adrv9025_Device_t* device, uint32_t gpioInputMask)
```

Description

Configures pins for input direction.

Parameters

Table 194.

Parameter	Description
*device	Pointer to device structure.
gpioInputMask	Selects the device GPIO pins that are required to be set as input in the range 0x00000 - 0x7FFFF. If a bit is set high, the GPIO pin associated with the bit is set as an input (GPIO_0 corresponds to bit D0, GPIO_1 corresponds to bit D1, and so on).

adi_adrv9025_GpioOutputDirSet

```
adi_adrv9025_GpioOutputDirSet(adi_adrv9025_Device_t* device, uint32_t gpioOutputMask)
```

Description

Configures pins for output direction.

Parameters

Table 195.

Parameter	Description
*device	Pointer to device structure.
gpioInputMask	Selects the device GPIO pins that are required to be set as output in the range 0x00000 - 0x7FFFF. If a bit is set high, the GPIO pin associated with the bit is set as an output (GPIO_0 corresponds to bit D0, GPIO_1 corresponds to bit D1, and so on).

Note that conflicts regarding GPIO usage may occur when using combinations of certain features. Ensure that multiple functions are not assigned to the same GPIO pin.

Input GPIO Features

The following table provides a list of GPIO input features available that interact with datapath control elements on the device. For the GPIO features within Table 196, the API automatically sets the I/O direction of the GPIO pins assigned for the feature.

Table 196. Summary of Input GPIO Features

Feature	Description	GPIO Pins Available for Feature
SPI2	Secondary SPI channel for control and read back of receiver gain index and transmitter attenuation. API Configuration Command: adi_adrv9025_Spi2CfgSet(…) adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(…)	GPIO_0: SPI_DIO (input or output) GPIO_1: SPI_DO (output only) GPIO_2: SPI_CLK (input) GPIO_3: SPI_$\overline{CS}$ (input) GPIO_4 through GPIO_18: Tx Attenuation state select
Pin Controlled Rx/ORx Gain Index Increment and Decrement	Configure specific GPIO pins to increment or decrement the gain index on any Rx or ORx channel after a rising edge on the assigned pin. API Configuration Command: adi_adrv9025_RxGainPinCtrlCfgSet(…)	GPIO_0 through GPIO_15: Rx/ORx gain index increment pin select. GPIO_0 through GPIO_15: Rx/ORx gain index decrement pin select.
Pin Controlled Tx Attenuation Increment and Decrement	Configures specific GPIO pins to increment or decrement attenuation on any Tx channel after a rising edge on the assigned pin. API Configuration Command: adi_adrv9025_TxAttenPinCtrlCfgSet(…)	GPIO_0 through GPIO_15: Tx attenuation increment pin select. GPIO_0 through GPIO_15: Tx attenuation decrement pin select.
External Slicer Mode	A technique used in some gain compensation applications. The baseband processor instructs the slicer to attenuate the digital data to fit within a desired bit-width based on the value expressed on the slicer pins (up to 3 available in input mode). API Configuration Command: adi_adrv9025_RxDataFormatSet(…)	GPIO_[2:0] = Assign to any Rx GPIO_[5:3] = Assign to any Rx GPIO_[8:6] = Assign to any Rx GPIO_[11:9] = Assign to any Rx GPIO_[14:12] = Assign to any Rx GPIO_[17:15] = Assign to any Rx
Tx-Observation Receiver Select	When using fewer than 4 ORx channels, the ORx channel needs information about which Tx channel data is presented to the ORx. If a pin interface is required to indicate the Tx to ORx mapping, the following command sets up the pins, provided the stream file is generated with appropriate input settings. API Configuration Command: adi_adrv9025_StreamGpioConfigSet(…)	GPIO_0 through GPIO_15.

More details on these features are provided in the following subsections.

SPI2

A complete description, including descriptions of custom data types, for the SPI2 interface can be found in the SPI2 Description section of this user guide.

The SPI2 interface acts as a secondary SPI channel that operates on digital GPIO_[3:0]. An optional pin can be configured for toggling the Tx attenuation between attenuation state S1 and attenuation state S2 on GPIO_4 through GPIO_18. The SPI2 interface uses the same SPI configuration used on the primary SPI interface. SPI2 can be used to set the gain index on Rx/ORx channels, read back the gain index on Rx/ORx channels, and set up two distinct Tx attenuation states that the user can alternate between by toggling a GPIO pin. The SPI2 interface cannot access registers available to the primary SPI interface.

When the SPI2 feature is enabled, GPIO_[3:0] and the pin assigned for Tx Attenuation Select (can be GPIO_4 through GPIO_18 or leave unassigned) cannot be used for other purposes. When SPI2 is enabled, it overrides functionality previously assigned to digital GPIO_[3:0] pins. Refer to Table 196 for specific pin mapping details.

adi_adrv9025_Spi2CfgSet

```
adi_adrv9025_Spi2CfgSet(adi_adrv9025_Device_t* device, uint8_t spi2Enable)
```

Description

Enables the SPI2 feature.

Parameters

Table 197.

Parameter	Description
*device	Pointer to device structure.
Spi2Enable	used to set the state of the SPI2 bus: 1 = Enable, 0 = Disable.

adi_adrv9025_TxAttenSpi2PinCtrlCfgSet

```
adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(adi_adrv9025_Device_t* device,
adi_adrv9025_TxAttenSpi2PinCfg_t txAttenSpi2PinCfg[], uint8_t numTxAttenSpi2PinConfigs)
```

Description

Assigns the Tx attenuation select pin.

Parameters

Table 198.

Parameter	Description
*device	Pointer to device structure.
txAttenSpi2PinCfg[]	Pointer to an array of adi_adrv9025_TxAttenSpi2PinCfg_t structure that configures the Tx attenuation SPI2 pin control. Note that multiple transmitters can share an attenuation select pin if desired.
numTxAttenSpi2PinConfigs	Determines the number of channelized Tx attenuation SPI2 pin configurations passed in the array txAttenSpi2PinCfg.

Pin Based Rx Gain Control

A complete description of the pin based Rx gain control feature is provided in the Receiver Gain Control and Gain Compensation section of this user guide.

Pin based Rx gain control is relevant for applications which require Manual Gain Control (MGC) and precise timing for gain change events. The pin based control scheme offers lower latency than SPI based gain change operations. In pin-based gain control, specific GPIO pins are assigned “increment gain index” or “decrement gain index” functionality for a particular receiver channel. By applying a logic high pulse on the GPIO pin, the gain index for the corresponding channel is either incremented or decremented, depending on the assigned functionality. The pulse width requirement is 2 AGC clock cycles in the logic high state. The gain change due to gain index increment or decrement is programmable (ranges from 1 to 8 gain index steps). Increment and decrement functionality can be assigned to any digital GPIO from GPIO_15 to GPIO_0.

Note that if the user has programmed a gain table that operates in a subset of the full gain table range (that is, using Index 195 to Index 255), the pin-based Rx gain control does not have knowledge of this status. If the gain decrement pulse is applied when the gain index is 195, the gain index decrements off table. It is possible that the off-table gain indices (that is, gain indices below 195) correspond to maximum gain condition. It is recommended to exercise care when applying pulses when the gain index is at the edge of the useful section gain table, or design the gain table with this in mind.

adi_adrv9025_RxGainPinCtrlCfgSet

```
adi_adrv9025_RxGainPinCtrlCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxChannels_e
rxChannel, adi_adrv9025_RxGainPinCfg_t *rxGainPinCtrlCfg)
```

Description

This command configures the pin-based Rx Gain Control feature. the device must be in MGC for proper operation.

Parameters

Table 199.

Parameter	Description
*device	Pointer to device structure.
rxChannel	Selects which Rx channel for configuring pin-based Rx gain control.
*rxGainPinCtrlCfg	Pointer to adi_adrv9025_RxGainPinCfg_t structure containing configuration values for pin based Rx gain control.

Table 200 describes the adi_adrv9025_RxGainPinCfg_t data structure used in the above command.

Table 200. Description of adi_adrv9025_RxGainPinCfg_t Data Structure

Data Type	parameter name	Comments
uint8_t	incStep	Increment in gain index applied when the increment gain pin is pulsed. A value of 0 to 7 applies a step size of 1 to 8
uint8_t	decStep	Decrement in gain index applied when the increment gain pin is pulsed. A value of 0 to 7 applies a step size of 1 to 8
adi_adrv9025_GpioPinSel_e	rxGainIncPin	GPIO used for the increment gain input: ADI_ADRV9025_GPIO00 to ADI_ADRV9025_GPIO15 can be used
adi_adrv9025_GpioPinSel_e	rxGainDecPin	GPIO assigned for the decrement gain input: ADI_ADRV9025_GPIO00 to ADI_ADRV9025_GPIO15 can be used
uint8_t	enable	Enable (1) or disable (0) the gain pin control

Pin-Based Tx Attenuation Control

A complete description of Tx attenuation control is provided in the Tx Overview and Path Control section of this user guide.

Pin based Tx attenuation control, similar to the Tx attenuation select feature of SPI2, provides an interface to make attenuation adjustments with precise timing control. The pin based control scheme offers lower latency than SPI based attenuation change operations. In pin based attenuation control, certain GPIO pins are assigned “increment attenuation” or “decrement attenuation” functionality. By applying a high pulse on the assigned GPIO pin, the attenuation for a specific channel is either incremented or decremented, depending on the assigned functionality. Increment and decrement functionality can be assigned to any digital GPIO from GPIO_15 to GPIO_0.

A notable difference between SPI2 and pin based Tx attenuation control is that SPI2 allows toggling between programmed attenuation states (S1 and S2) while pin based Tx attenuation control allows for multiple increments or decrements of Tx attenuation.

adi_adrv9025_TxAttenPinCtrlCfgSet

```
adi_adrv9025_TxAttenPinCtrlCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_TxAttenPinCfg_t
txAttenPinCfg[],uint8_t numTxAttenPinConfigs)
```

Description

Configures the pin-based Tx attenuation control feature.

Parameters

Table 201.

Parameter	Description
*device	Pointer to device structure.
txAttenPinCfg[]	Pointer to an array of adi_adrv9025_TxAttenPinCfg_t structure that configures the Tx attenuation pin control.
numTxAttenPinConfigs	Determines the number of channelized Tx attenuation pin configuration passed in the array txAttenPinCfg.

Table 202 describes the adi_adrv9025_TxAttenPinCfg_t data structure used in the above command.

Table 202. Description of adi_adrv9025_TxAttenPinCfg_t Data Structure

Data Type	Parameter Name	Comments
uint32_t	txChannelMask	Bitwise channel mask that the Tx attenuation pin configuration settings are applied to. [D0] = Tx1, [D1] = Tx2, [D2] = Tx3, [D3] = Tx4.
uint8_t	stepSize	This parameter sets the change in Tx attenuation for each increment or decrement signal received in incr/decr mode. 0.5dB/LSB. Valid range is from 0 to 31
adi_adrv9025_GpioPinSel_e	txAttenIncPin	GPIO assigned for the increment attenuation input: ADI_ADRV9025_GPIO00 to ADI_ADRV9025_GPIO15 can be used
adi_adrv9025_GpioPinSel_e	txAttenDecPin	GPIO assigned for the decrement attenuation input: ADI_ADRV9025_GPIO00 to ADI_ADRV9025_GPIO15 can be used
uint8_t	enable	Enable (1) or disable (0) the gain pin control

External Slicer Mode

A complete description of the external slicer use case is provided in the Receiver Gain Control and Gain Compensation section of this user guide.

The Rx datapath features a GPIO based slicer used in conjunction with digital gain compensation to digitally attenuate data sent over the JESD204B/JESD204C interface. The digital gain compensation may expand the required number of bits to express data path samples beyond the interface bit width. The slicer attenuates the data to fit within the interface bit width.

The slicer can be used in a mode where the amount of digital gain compensation at a particular gain index determines the slicer position (internal slicer). Alternatively, the slicer can be used with GPIOs in an externally driven mode where the baseband processor determines the slicer position, which controls the amount of digital attenuation applied by the slicer. When using the slicer in the external mode, specific groups of GPIO pins are assigned to set the slicer position. 3 GPIO pins per Rx are utilized. See Table 206 for a list of the valid external slicer pins.

The following command configures the external slicer mode.

adi_adrv9025_RxDataFormatSet

```
adi_adrv9025_RxDataFormatSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxDataFormat_t
rxDataFormat[], uint8_t arraySize)
```

Description

Configures the external slicer mode.

Parameters

Table 203.

Parameter	Description
*device	Pointer to device structure.
rxDataFormat[]	Pointer to the Rx data format configuration structure.
arraySize	Determines the size of rxDataFormat array representing the number of configurations.

Table 204 describes the adi_adrv9025_RxDataFormat_t data structure.

Table 204. Description of adi_adrv9025_RxDataFormat_t Data Structure

Data Type	Parameter Name	Comments
uint32_t	rxChannelMask	Rx channel mask
adi_adrv9025_RxDataFormatModes_e	formatSelect	Rx Channel format mode selects
adi_adrv9025_FloatingPointConfigSettings_t	floatingPointConfig	Rx Channel floating point format configuration
adi_adrv9025_IntegerConfigSettings_t	integerConfigSettings	Rx Channel integer format configuration
adi_adrv9025_SlicerConfigSettings_t	slicerConfigSettings	Rx Channel integer slicer configuration
uint8_t	externalLnaGain	Selects Slicer to compensate for external dual band LNA (0= disabled, 1 = enabled)
uint8_t	tempCompensationEnable	Selects Slicer to compensate for temperature variations (0 = disabled, 1 = enabled)

For the external slicer mode, the formatSelect parameter must be set as ADI_ADRV9025_GAIN_WITH_EXTERNAL_SLICER.

Other settings relevant to the external slicer configuration include the adi_adrv9025_SlicerConfigSettings_t data structure described in Table 205.

Table 205. Description of adi_adrv9025_SlicerConfigSettings_t Data Structure

Data Type	Parameter Name	Comments
adi_adrv9025_ExtSlicerStepSizes_e	extSlicerStepSize	Enum selects the external pin gain step size
adi_adrv9025_IntSlicerStepSizes_e	intSlicerStepSize	Enum selects the internal pin gain step size
adi_adrv9025_RxExtSlicerGpioSel_e	rx1ExtSlicerGpioSelect	Enum selects the Rx1 Ext Ctrl GPIO Configuration
adi_adrv9025_RxExtSlicerGpioSel_e	rx2ExtSlicerGpioSelect	Enum selects the Rx2 Ext Ctrl GPIO Configuration
adi_adrv9025_RxExtSlicerGpioSel_e	rx3ExtSlicerGpioSelect	Enum selects the Rx3 Ext Ctrl GPIO Configuration
adi_adrv9025_RxExtSlicerGpioSel_e	rx4ExtSlicerGpioSelect	Enum selects the Rx4 Ext Ctrl GPIO Configuration

The enum adi_adrv9025_RxExtSlicerGpioSel_e provides the list of GPIO groupings available when using the external slicer mode as displayed in Table 206.

Table 206. Description of adi_adrv9025_RxExtSlicerGpioSel_e Enumeration

Enum Name	Enum Value	Comments
ADI_ADRV9025_EXTSLICER_RX_GPIO_DISABLE	0	No GPIO assigned to external slicer
ADI_ADRV9025_EXTSLICER_RX_GPIO_2_DOWNTO_0	1	Select Rx Gain Slicer External GPIO2, GPIO1, GPIO0
ADI_ADRV9025_EXTSLICER_RX_GPIO_5_DOWNTO_3	2	Select Rx Gain Slicer External GPIO5, GPIO4, GPIO3
ADI_ADRV9025_EXTSLICER_RX_GPIO_8_DOWNTO_6	3	Select Rx Gain Slicer External GPIO8, GPIO7, GPIO6
ADI_ADRV9025_EXTSLICER_RX_GPIO_11_DOWNTO_9	4	Select Rx Gain Slicer External GPIO11, GPIO10, GPIO9
ADI_ADRV9025_EXTSLICER_RX_GPIO_14_DOWNTO_12	5	Select Rx Gain Slicer External GPIO14, GPIO13, GPIO12
ADI_ADRV9025_EXTSLICER_RX_GPIO_17_DOWNTO_15	6	Select Rx Gain Slicer External GPIO17, GPIO16, GPIO15

Other members of the adi_adrv9025_RxDataFormatter_t are discussed in the Receiver Gain Control and Gain Compensation section.

Tx-ORx Mapping

A full description of Tx-ORx mapping is provided in the "Stream Processor and System Control" section.

For initial calibrations and tracking calibrations that require the use of an external Tx to ORx loopback channel for the algorithm, the Arm must understand the specific mapping of Tx to ORx at that time. In the 4 ORx use case, typically the mapping is static, and it is recommended to use the API command adi_adrv9025_TxToOrxMappingSet(…) to configure the mapping. In the 2 ORx use case, each ORx channel must know which Tx channel is provided as input. An alternative to the API command interface is to use a GPIO based interface to inform the Arm about the currently mapped Tx channels into the ORx. To clarify, the baseband processor informs the transceiver about the channel mapping state by signaling on GPIOs which executes a stream processor command. This stream processor command provides the mapping information to the Arm processor which executes the calibration routines.

The GPIO pins available for this feature range from GPIO_0 to GPIO_15. Up to four GPIO pins are required to fully implement pin based mapping controls. A partial implementation can be achieved with 2 GPIO pins. The partial implementation only indicates which Tx was mapped to the ORx (TX_SEL signal) and does not permit the baseband processor to inform the device that it must not perform tracking calibrations (TX_EN signal). This additional information is useful if antenna calibrations are performed while the tracking calibrations that depend on a constant external channel are still enabled.

To set up this feature, the GUI must generate a stream file with the desired GPIO pins of the user to use for the TX_SEL/TX_EN signals. With the proper steam file, the user can configure the stream processor to listen to the input GPIO pins with the following command. Note that this command is called as a part of the adrv9025_RadioctrlInit command which is called during adi_adrv9025_PostMcsInit(…).

adi_adrv9025_StreamGpioConfigSet(…)

```
adi_adrv9025_ StreamGpioConfigSet(adi_adrv9025_Device_t* device,
 adi_adrv9025_StreamGpioPinCfg_t* streamGpioPinCfg);
```

Description

This function associates a GPIO pin with stream processor GP inputs and enables stream trigger functionality if a valid GPIO (GPIO0 to GPIO15) is assigned to the streamGpInput pins.

There are 16 GPIO inputs available to trigger streams. These GPIO pins can be mapped to one of GPIOs[0:15].

To unmap a GPIO association with a stream GP input, please set the GPIO input to ADI_ADRV9025_GPIO_INVALID.

Parameters

Table 207.

Parameter	Description
*device	Pointer to device structure.
streamGpioPinCfg	A data structure containing the GPIO assignments for stream processor inputs.

Table 208. Description of the adi_adrv9025_StreamGpioPinCfg_t Data Structure

Member	Data Type	Description
streamGpInput0	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 0 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput1	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 1 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput2	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 2 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput3	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 3 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput4	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 4 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput5	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 5 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput6	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 6 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput7	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 7 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput8	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 8 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput9	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 9 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput10	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 10 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput11	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 11 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput12	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 12 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput13	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 13 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput14	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 14 (valid GPIO0 to GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput15	adi_adrv9025_GpioPinSel_e	Select desired GPIO pin input to stream processor GP Input 15 (valid GPIO0 GPIO15). To disable select ADI_ADRV9025_GPIO_INVALID.

It is recommended to use the GUI to determine the values required for this data structure. This helps ensure that features assigned the stream GPIO pins are properly assigned.

Output GPIO Features

This section outlines digital GPIO output features available on the device. Output GPIO features on the transceiver use a concept called source control. The source control describes the source of the signals routed to GPIO pins, whether they are from the monitor feature or the Arm. Table 211 summarizes the available source control selections. Source control is relevant for GPIO pins that are configured in the output mode. GPIO pins operating in the input mode do not require a source control setup.

adi_adrv9025_GpioOutSourceCtrlSet

```
adi_adrv9025_GpioOutSourceCtrlSet(adi_adrv9025_Device_t* device, uint32_t gpioSrcCtrl);
```

Description

Sets the source control.

Parameters

Table 209.

Parameter	Description
*device	Pointer to device structure.
gpioSrcCtrl	The nibble-based source control – this is a 32-bit value containing 5 nibbles that set the output source control for each set of four GPIO pins. This parameter is set in 4-bit nibble groupings as shown in Table 210.

Table 210. Description of the Nibble Groups Configured Via gpioSrcCtrl

gpioSrcCtrl[bits]	Description
gpioSrcCtrl[d3:d0]	GPIO output source for GPIO_[3:0] pins
gpioSrcCtrl[d7:d4]	GPIO output source for GPIO_[7:4] pins
gpioSrcCtrl[d11:d8]	GPIO output source for GPIO_[11:8] pins
gpioSrcCtrl[d15:d12]	GPIO output source for GPIO_[15:12] pins
gpioSrcCtrl[d19:d16]	GPIO output source for GPIO_[18:16] pins

The values for these nibble groupings can be formed with the adi_adrv9025_GpioOutputModes_e enumeration. This enum is described in Table 211.

Table 211. Description of adi_adrv9025_GpioOutputModes_e Enumeration

Enum Name	Enum Value	Comments
ADI_ADRV9025_GPIO_BITBANG_MODE	3	Manual mode, API function sets output pin levels and reads input pin levels
ADI_ADRV9025_GPIO_SLICER_OUT_MODE	10	Allows slicer position to be output on GPIO pins

Note that if a GPIO is not designated as an output pin that it can be set as an input pin. As an example, consider a use case where 3 pins in a 4-pin nibble group are dedicated for slicer output mode, the 4th pin in the group can be set as an input pin for gain control. As a constraint on customer applications, multiple source control selections cannot be used within a single 4 pin nibble group.

Manual Pin Toggle (Bitbang) Mode

This mode allows control of the logic level of individual GPIO pins.

adi_adrv9025_GpioOutPinLevelSet

```
adi_adrv9025_GpioOutPinLevelSet(adi_adrv9025_Device_t* device, uint32_t gpioOutPinLevel)
```

Description

Sets the output logic level of the GPIO pins (after configuring the I/O direction and source control).

Parameters

Table 212.

Parameter	Description
*device	Pointer to device structure.
gpioOutPinLevel	Determines the level to output on each GPIO pin. 0 = low output, 1 = high output.

Slicer Output Mode

A general description of this feature is provided in the Mode 2: Digital Gain Compensation with Slicer GPIO Outputs section.

GPIO_ANA OPERATION

The main purpose of the GPIO_ANA pins is to serve as control pins for an external control element, such as a Digital Step Attenuator (DSA) or Low Noise Amplifier (LNA). Other features may be exposed in future software releases. A high-level overview of the GPIO_ANA features are provided below.

Table 213. Summary of GPIO_ANA Features

Feature	Description	GPIO Pins Available for Feature
Rx Gain Table External Control Word Output	The Rx gain table includes a column for 2-bit control of an external gain element. Each Rx channel has 2 fixed GPIO_ANA pins associated with it. The 2-bit value expressed on the pins depends on the gain index and gain table column. API Function for Configuration: adi_adrv9025_RxGainTableExtCtrlPinsSet(…)	GPIO_ANA_[1:0]: Rx1 External Control Word, GPIO_ANA_[3:2]: Rx2 External Control Word, GPIO_ANA_[5:4]: Rx3 External Control Word, GPIO_ANA_[7:6]: Rx4 External Control Word

Gain Table External Control Word

For proper use of this feature, a custom gain table must be created that uses the external control column. When a gain index with a non-zero value in the external control column of the gain table is selected, the value of the external control column is output on a pair of GPIO_ANA pins. The configuration of the GPIO pins for gain table external control word is performed with the following API command.

adi_adrv9025_RxGainTableExtCtrlPinsSet

```
adi_adrv9025_RxGainTableExtCtrlPinsSet(adi_adrv9025_Device_t* device,
adi_adrv9025_RxExtCtrlPinOuputEnable_e extCtrlGpioChannelEn)
```

Description

Configures the GPIO pins for the gain table external control word.

Parameters

Table 214.

Parameter	Description
*device	Pointer to device structure.
extCtrlGpioChannelEnable	Determines the adi_adrv9025_RxChannels_e enum type to select which set of gain table external control words to output on analog GPIOs.

Table 215 describes the adi_adrv9025_RxExtCtrlPinOutputEnable_e enumeration.

Table 215. Description of adi_adrv9025_RxExtCtrlPinOuputEnable_e Enumeration

Enum Name	Comments
ADI_ADRV9025_DISABLE_RX1_RX2_EXT_CTRL_GPIOS	Disable Rx1 and Rx2 Ext Ctrl Word output on Analog GPIOs
ADI_ADRV9025_ENABLE_RX1_RX2_EXT_CTRL_GPIOS	Enable Rx1 and Rx2 Ext Ctrl Word output on Analog GPIOs
ADI_ADRV9025_DISABLE_RX3_RX4_EXT_CTRL_GPIOS	Disable Rx3 and Rx4 Ext Ctrl Word output on Analog GPIOs
ADI_ADRV9025_ENABLE_RX3_RX4_EXT_CTRL_GPIOS	Enable Rx3 and Rx4 Ext Ctrl Word output on Analog GPIOs
ADI_ADRV9025_DISABLE_RX1_RX2_RX3_RX4_EXT_CTRL_GPIOS	Disable Rx1, Rx2, Rx3 and Rx4 Ext Ctrl Word output on Analog GPIOs
ADI_ADRV9025_ENABLE_RX1_RX2_RX3_RX4_EXT_CTRL_GPIOS	Enable Rx1, Rx2, Rx3 and Rx4 Ext Ctrl Word output on Analog GPIOs

GENERAL-PURPOSE INTERRUPT

The device features two General Purpose Interrupt pins, GPINT1 and GPINT2 per the data sheet pinout. Note that the data sheet pinout conventions of GPINT1 and GPINT2 are referenced within the API as GPINT0 and GPINT1, respectively. In this section, references are made to the GPINT conventions on the data sheet pinout except when listed in an API code example. A summary of API commands relevant for the GPINT functionality is provided in the API Commands for GPINT section.

The GPINT pins provide an interface that allows the device to inform the baseband processor of an error in normal operation. Examples of the interrupt sources include PLL unlock events, SERDES link status, a stream processor error, or Arm exception. A full list of interrupt sources is provided in Table 216. The GPINT2 pin acts as the high priority interrupt pin and GPINT1 acts as the low priority interrupt pin. The pins can be configured with independent bitmasks that controls which signals can assert GPINT1 or GPINT2. A high-level block diagram of the GPINT operation is shown in Figure 112.

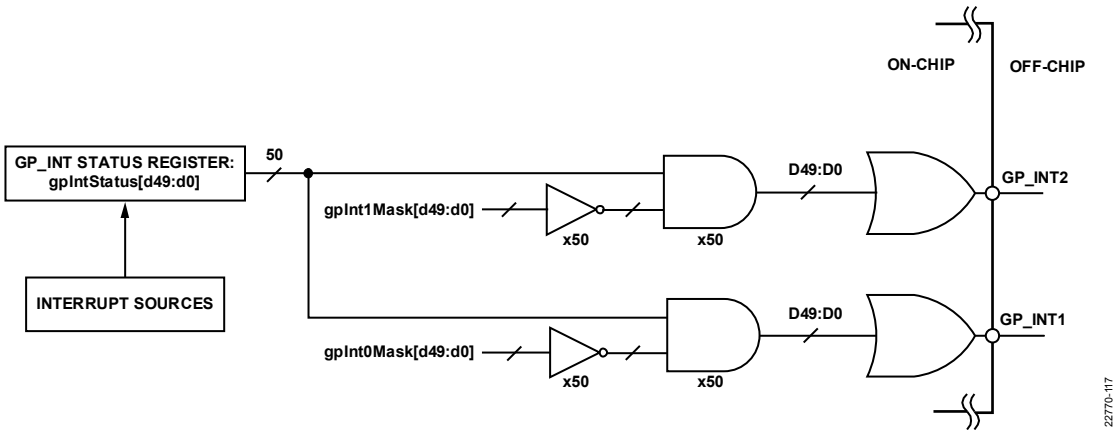


Figure 112. Block Diagram of General-Purpose Interrupt Outputs

The GPINT1 and GPINT2 pins are a bitwise OR of all unmasked GPINT sources. The status register represents all possible interrupt sources that can assert on the device. Any time the GPINT pin asserts, the GPINT status indicates what interrupt source(s) asserted the GPINT pin.

Note that the GPINT status and the GPINT pins have different behaviors. The GPINT pins are real-time indicators of error status. For example, if a PA protection error occurs when PA protection is configured in the autoclear mode, the GPINT pin deasserts after the power returns to normal. The GPINT status bit fields are sticky and remain asserted until the user clears the register. If the PA protection error occurs and disappears in autoclear mode, the GPINT status still indicates a PA protection error occurred until the user manually clears the GPINT status.

A description of the interrupt sources and their bit positions within the 50-bit general purpose interrupt mask is provided in Table 216.

Table 216. GP_INTERRUPT Bitmask Description

Bit Position	Brief Description	Sub-System	API Recovery Action
D49	Deframer IRQ 11: Deframer1 JESD204C CRC Error	Deframer	ADI_ADRV9025_ACT_ERR_BBIC_LOG_ERROR
D48	Deframer IRQ 10: Deframer1 JESD204C Loss of Sync		
D47	LO1 PLL Unlock	PLL	ADI_COMMON_ACT_ERR_RESET_MODULE
D46	LO2 PLL Unlock		ADI_COMMON_ACT_ERR_RESET_MODULE
D45	AUX PLL Unlock		ADI_COMMON_ACT_ERR_RESET_MODULE
D44	CLK PLL Unlock		ADI_COMMON_ACT_ERROR_RESET_FULL
D43	LO1 PLL Charge Pump Overrange		ADI_ADRV9025_ACT_ERR_BBIC_LOG_ERROR
D42	LO2 PLL Charge Pump Overrange		
D41	AUX PLL Charge Pump Overrange		
D40	CLK PLL Charge Pump Overrange		
D39	SERDES PLL Unlock		ADI_COMMON_ACT_ERROR_RESET_FULL

Bit Position	Brief Description	Sub-System	API Recovery Action
D38	Deframer IRQ 9: Deframer1 JESD204B QBD IRQ	Deframer	ADI_ADRV9025_ACT_ERR_BBIC_LOG_ERROR
D37	Deframer IRQ 8: Deframer1 SYSREF Out of Phase		
D36	Deframer IRQ 7: Deframer1 Elastic Buffer Error		
D35	Deframer IRQ 6: Deframer1 Lane FIFO Pointer Error		
D34	Deframer IRQ 5: Deframer0 JESD204C CRC Error		
D33	Deframer IRQ 4: Deframer0 JESD204C Loss of Sync		
D32	Deframer IRQ 3: Deframer0 JESD204B QBD IRQ		
D31	Deframer IRQ 2: Deframer0 SYSREF Out of Phase		
D30	Deframer IRQ 1: Deframer0 Elastic Buffer Error		
D29	Deframer IRQ 0: Deframer0 Lane FIFO Pointer Error		
D28	Framer IRQ 8: Framer2 Transport Not Sending Data	Framer	
D27	Framer IRQ 7: Framer2 SYSREF Out of Phase		
D26	Framer IRQ 6: Framer2 Lane FIFO Pointer Error		
D25	Framer IRQ 5: Framer1 Transport Layer Not Sending Data		
D24	Framer IRQ 4: Framer1 SYSREF Out of Phase		
D23	Framer IRQ 3: Framer1 Lane FIFO Pointer Error		
D22	Framer IRQ 2: Framer0 Transport Layer Not Sending Data		
D21	Framer IRQ 1: Framer0 SYSREF Out of Phase		
D20	Framer IRQ 0: Framer0 Lane FIFO Pointer Error		
D19	PA Protection Error (Threshold Exceeded) Tx4	Transmitter	ADI_ADRV9025_ACT_ERR_BBIC_LOG_ERROR
D18	PA Protection Error (Threshold Exceeded) Tx3		
D17	PA Protection Error (Threshold Exceeded) Tx2		
D16	PA Protection Error (Threshold Exceeded) Tx1		
D15	Arm Has Forced Interrupt	Arm	ADI_COMMON_ACT_ERROR_RESET_FULL
D14	Arm Watchdog Timer Timeout		ADI_COMMON_ACT_ERROR_RESET_FULL
D13	Slew Rate Limiter IRQ		ADI_ADRV9025_ACT_ERR_BBIC_LOG_ERROR
D12	Arm System Error		ADI_ADRV9025_ACT_ERR_BBIC_LOG_ERROR
D11	ORx3/4 Stream Processor Error	Stream Processor	ADI_COMMON_ACT_ERROR_RESET_FULL
D10	ORx1/2 Stream Processor Error		
D9	Tx4 Stream Processor Error		
D8	Tx3 Stream Processor Error		
D7	Tx2 Stream Processor Error		
D6	Tx1 Stream Processor Error		
D5	Rx4 Stream Processor Error		
D4	Rx3 Stream Processor Error		
D3	Rx2 Stream Processor Error		
D2	Rx1 Stream Processor Error		
D1	Core Stream Processor Error		
D0	Memory ECC Error	Arm	ADI_COMMON_ACT_ERROR_RESET_FULL

Table 216 can be used to form bitmasks for GPINT2 and GPINT1. Note that in the API, GPINT1 is linked to the GPINT2 pin and GPINT0 is linked to the GPINT1 pin.

Further description of these event sources is provided in the following sections.

PLL GPINT SOURCES

The PLL GPINT sources include two types of interrupt for the PLLs: PLL unlock events and PLL Charge Pump (CP) overrange events. It must be noted that if initial calibrations are run, it is expected that some PLLs are used during this time and a PLL unlock event may show up in the GPINT status register. PLL unlocks during successful runs of initialization calibrations are expected and is not a concern.

PLL Unlock Event Bits

The PLL unlock event bits, if asserted, indicate that a PLL has unlocked and is not operating properly. The PLLs are designed to maintain lock over the full temperature range and operation of the device. In extremely rare cases the PLL may unlock due to external or internal factors. There are two recovery procedures for PLL unlocks depending on the PLL that unlocks.

- If CLK PLL unlocks: Reset Device. It is not expected the device can recover from the loss of the primary clock within the transceiver.
- If LO2, LO1 or AUX PLL unlocks, call adi_adrv9025_PllFrequencySet(…) to see if the PLL relocks.
    - If the unlocked PLL re-locks, then follow procedures to re-run certain initialization calibrations as this is effectively a PLL frequency change procedure. If the user has configured attenuation ramp down/up events to occur based on PLL lock status, the attenuation ramp down/up event must be cleared prior to running initial calibrations.
    - If the unlocked PLL fails to achieve lock, then reset the device.

The real time lock status of the PLL can be verified with the command adi_adrv9025_PllStatusGet(…).

CP Overrange Event Bits

CP Overrange Event Bits must not be unmasked for the GPINT pins. These bits may assert intermittently but do not indicate a significant device issue.

JESD204B/JESD204C GPINT SOURCES

The deframer and framer, in both JESD204B and JESD204C modes of operation, may send information to the user regarding error events over the GPINT pin.

Note: Due to a hardware issue, the JESD204C CRC error can assert when the link is configured for JESD204B mode. Ignore the JESD204C CRC Error when detected in JESD204B use cases. Additionally, do not allow JESD204C errors assert the GPINT pins when configured in JESD204B mode since there is no value provided in this configuration.

Table 217 provides some additional detail regarding the deframer and framer interrupts that can assert the GPINT pin. In general, referring to JESD204B/JESD204C documentation can help explain these events in more detail and possible recovery mechanisms.

Table 217. Framer and Deframer Interrupt List

GP_INT Bits	Brief Description	Technical Description	Further Actions, If Necessary
D34, D49	Deframer JESD204 CRC Error	A cyclic redundancy check (CRC) error has been detected on one of the active deframer lanes. Tx data possibly corrupted.	Log event. Customer to decide how to react to the event.
D33, D48	Deframer JESD204C Loss of Sync	The JESD204C link layer has lost sync. This can be due to loss of sync header alignment, or multi-block alignment. Typically, the link has dropped and needs to be reestablished.	Log event. If link is down, reestablish link.
D32, D38	Deframer JESD204B QBD IRQ	The Quad Byte Deframer (QBD) Interrupt (IRQ) indicates that a deframer IRQ source has asserted. Deframer IRQ sources include Bad Disparity (BD), Not-in-Table (NIT), Unexpected K (UEK). Most errors are considered minor.	Log event. Call adi_adrv9025_DfrmIrqSourceGet(…) to retrieve the specific interrupt that asserted. Typically this is an informational interrupt, but some cases may require link reset.
D31, D37	Deframer SYSREF Out of Phase	SYSREF registered at the wrong phase in the link.	Log event. Something is likely incorrect in overall system timing and needs to be adjusted.

GP_INT Bits	Brief Description	Technical Description	Further Actions, If Necessary
D30, D36	Deframer Elastic Buffer Error	The phase of lane data in the link with respect to global LMFC has shifted such that the buffer is in "protect" mode to avoid corrupt data transfer. Deterministic latency is lost.	Log event. Reassess lmfcOffset value selection if deterministic latency is required.
D29, D35	Deframer Lane FIFO Pointer Error	Lane FIFO pointers have moved in the link. This may or may not be associated with SYNC going low.	Log event. Reset link.
D22, D25, D28	Framer Transport Layer Not Sending Data	The framer is not sending user data. This occurs if the LMFC from the link layer is out of phase with the transport layer LMFC. This forces a relink by taking SYNC low.	Log event.
D21, D24, D27	Framer SYSREF Out of Phase	SYSREF is registered at the wrong phase in the framer link. If JESD is configured to attempt re-link with the new phase, no action required.	Log event. Something is likely incorrect in overall system timing and needs to be adjusted.
D20, D23, D26	Framer Lane FIFO Pointer Error	The lane FIFO pointer has changed.	Log event.

These deframer interrupts can be used to assert the rampdown of Tx attenuation as described in the Transmitter Power Amplifier Protection section of this document.

PA PROTECTION GPINT SOURCES

The PA protection feature must be enabled for these interrupts to assert. The PA protection block refers specifically to the peak and average power measurement capabilities within the Tx data path and must not be misconstrued for the general Tx attenuation ramp features.

PA protection GPINT sources indicate to the user that a peak or average power measurement within the Tx data path has exceeded the thresholds as configured on the device. When the power measurement exceeds the threshold, this is also referred to as a PA protection error. Log this event and take appropriate action within their system to resolve the reason for the power increase in the Tx data path.

The user can configure the PA protection block to enforce a ramp (or increase) of Tx attenuation with the command adi_adrv9025_PaPllDfrmEventRampDownEnableSet(…). Control over whether the attenuation ramp is sticky or autoclears is determined by adi_adrv9025_TxAttenuationRampUpStickyModeEnable(…) command. Refer to the Transmitter Power Amplifier Protection section for more information.

ARM GPINT SOURCES

There are four Arm interrupt sources available.

Arm Has Forced Interrupt

The Arm asserts this interrupt in the case a fatal error occurs within the FW. If possible, acquire an Arm memory dump to assist in debug. Reset the device.

Arm Watchdog Timer Timeout

The Arm asserts this interrupt when the watchdog timer within the Arm reaches its timeout value. If the Arm was unable to reset this timer there is a fatal error within the Arm. If possible, acquire an Arm memory dump s to assist in debug. Reset the device.

Slew Rate Limiter IRQ

As of SW 2.0.5 versions, this bit represents the Slew Rate Limiter (SRL) error interrupt for the Tx datapaths. If this interrupt asserts, then it indicates an SRL error event has occurred. Check the SRL statistics for each channel to check which channel generated the interrupt.

Arm System Error

The Arm asserts this interrupt when the Arm detects an issue with any calibration or system related issue managed by the Arm. Some events may be fatal. To acquire more information about the error, call the API command adi_adrv9025_ArmSystemErrorGet(…). This bit also represents any issues with tracking calibrations.

STREAM PROCESSOR SOURCES

Assertion of any stream processor interrupt bits indicate that a significant problem has occurred within the stream processor. The stream processor does not have a way to recover from these events. Reset the device if stream processor errors are detected.

MEMORY ECC ERROR

A memory ECC error indicates that a bit error has occurred in a memory circuit within the chip. This is an extremely rare event. The device must be reset if this is detected.

SOFTWARE PROCEDURES FOR GPINT

Referring to the device programming sequence in adi_adrv9025_daughter_board.c, the GPINT feature setup is one of the last steps in device initialization, occurring after both adi_board_adrv9025_JesdBringup(…) and adi_adrv9025_TxRampDownInit(…). The GPINT masks for GPINT2/GPINT1 physical pins are stored in the adi_adrv9025_GpInterruptSettings_t structure and applied to the device during adi_adrv9025_GpIntInit(…). This configures both GPINT pins and no further action is needed for setup.

If it is necessary to reconfigure the GPINT masks after initialization, use the command adi_adrv9025_GpIntMaskSet(…). The primary difference between the two GPINT setup commands is that adi_adrv9025_GpIntMaskSet(…) allows selection regarding which pin bitmask to program.

The baseband processor monitors the status of the GPINT2 and GPINT1 pins after configuring the mask bits. If either pin asserts, this indicates that the transceiver has run into a problem that may require user intervention to resolve. The GPINT handler functions tries to resolve the error by reading back the status and then clearing the status bit fields. The bits in the status register are sticky, but the pin is not. The pin represents whether the interrupt source is active or not. The register indicates which interrupts have occurred since the status was last cleared.

The general setup and usage for the GPINT command is detailed as follows:

1. Initialize device (call adi_adrv9025_GpIntInit(…) or adi_adrv9025_GpIntMaskSet(…) to set up the GPINT feature).
2. Operate device. The baseband processor monitors the GPINT2 and/or GPINT1 pins for rising edges indicating an interrupt has occurred.
3. If the GPINT2 and/or GPINT1 pins assert, call their associated interrupt handler API command, either adi_adrv9025_GpInt1Handler(…) or adi_adrv9025_GpInt0Handler(…), respectively. The interrupt handler returns information related to the interrupt source to the user. Calling this command may be sufficient to clearing the error. Either handler function returns a recovery action which suggests further action if necessary.
    a. Alternatively, the user can call adi_adrv9025_GpIntStatusGet(…), which only returns the interrupt status bits. The status word is not maskable and indicates all errors since the previous clearing of the status word.
    b. If the device does not need to be reset and the error state has been eliminated, it is necessary to call adi_adrv9025_GPIntClearStatusRegister(…) to clear all error bits asserted in the GPINT status register.
4. Perform recovery action(s).

API COMMANDS FOR GPINT

The following section outlines API commands for configuring and using the GPINT feature.

adi_adrv9025_GpIntMaskSet

```
adi_adrv9025_GpIntMaskSet(adi_adrv9025_Device_t* device, adi_adrv9025_gpMaskSelect_e maskSelect,
adi_adrv9025_gp_MaskArray_t *maskArray)
```

Description

Applies the desired bitmasks to the device.

Parameters

Table 218.

Parameter	Description
*device	Pointer to device structure.
maskSelect	The enum indicating which GP_INTERRUPT bitmask (GPINT1 or GPINT0) to write.
*maskArray	Pointer to the data structure holding the GP_INTERRUPT bitmasks to write.

Table 219 describes the adi_adrv9025_gpMaskSelect_e enumeration. This parameter describes which pin to write the mask to.

Table 219. Description of adi_adrv9025_gpMaskSelect_e Enumeration

Enum Name	Comments
ADI_ADRV9025_GPINT0	GPINT1 Select (GPINT0 bitmask). Only adi_adrv9025_gp_MaskArray_t -> gpInt0Mask is programmed to the device.
ADI_ADRV9025_GPINT1	GPINT2 Select (GPINT1 bitmask). Only adi_adrv9025_gp_MaskArray_t -> gpInt1Mask is programmed to the device.
ADI_ADRV9025_GPINTALL	GPINT1 and GPINT2 Select. Both members of adi_adrv9025_gp_MaskArray_t are programmed to the device.

Table 220 describes the adi_adrv9025_gp_MaskArray_t data structure. Refer to Table 216 for a description of the bitmasks.

Table 220. Description of adi_adrv9025_gp_MaskArray_t Data Structure

Data Type	Parameter Name	Comments
uint64_t	gpInt0Mask	Bitmask for the GPINT1 pin. If a bit within the mask is set to 1, the associated interrupt source cannot assert the GPINT2 pin.
uint64_t	gpInt1Mask	Bitmask for the GPINT2 pin. If a bit within the mask is set to 1, the associated interrupt source cannot assert the GPINT1 pin.

When either GPINT pin asserts, there are interrupt handler API commands to assist with determining the error. The following commands are the GPINT2 and GPINT1 interrupt handlers.

adi_adrv9025_GpInt1Handler

```
adi_adrv9025_GpInt1Handler(adi_adrv9025_Device_t* device, adi_adrv9025_gpIntStatus_t
*gpInt1Status)
```

Description

Sets up the GPINT2 interrupt handler.

Parameters

Table 221.

Parameter	Description
*device	Pointer to device structure.
*gpInt1Status	Pointer to the status read-back word containing the GPINT2 source registers.

adi_adrv9025_GpInt0Handler

```
adi_adrv9025_GpInt0Handler(adi_adrv9025_Device_t* device, adi_adrv9025_gpIntStatus_t
*gpInt0Status)
```

Description

Sets up the GPINT1 interrupt handler.

Parameters

Table 222.

Parameter	Description
*device	Pointer to device structure.
*gpInt0Status	Pointer to the status read-back word containing the GPINT1 source registers.

When either handler command is called, the first step in the procedure is to temporarily modify the interrupt bitmask such that no other interrupts can assert GPINT2 or GPINT1 while the handler is invoked. This masking is followed by retrieval of the GPINT status. The final step in the handler is to restore initial bitmask for GPINT2/GPINT1. In some cases, reading the error is sufficient to clearing the error – this is the case for short-term, intermittent errors. If the error persists, then the status continues to indicate the interrupt and further intervention is necessary.

adi_adrv9025_GpIntStatusGet

```
adi_adrv9025_GpIntStatusGet(adi_adrv9025_Device_t* device, uint64_t *gpIntStatus)
```

Description

Provides direct readback of the GPINT status word.

Parameters

Table 223.

Parameter	Description
*device	Pointer to device structure.
*gpIntStatus	Pointer to the status read-back word. Refer to Table 216 for bitmask description.

adi_adrv9025_GPIntClearStatusRegister

```
adi_adrv9025_GPIntClearStatusRegister(adi_adrv9025_Device_t *device, uint64_t *gpIntStatus)
```

Description

Clears the GPINT status register.

Parameters

Table 224.

Parameter	Description
*device	Pointer to device structure.
*gpIntStatus	Pointer to the status read-back word. Refer to Table 216 for bitmask description.

AUXILIARY CONVERTERS AND TEMPERATURE SENSOR

The transceiver features auxiliary data converters including eight 12-bit auxiliary digital-to-analog converters (AuxDAC) and two 12-bit auxiliary analog-to-digital converters (AuxADC). An integrated diode-based temperature sensor is available to readback 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 transceiver. Example usage of the auxiliary converters include 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 API command for configuration and control.

The AuxDAC and AuxADC are not precision data converters. DC offset and gain/slope errors are present and may vary on different channels. Refer to specifications in data sheet. The AuxDAC and AuxADC are best used in feedback systems rather than in open-loop systems for precision voltage readback or control.

AUXILIARY DAC (AUXDAC)

There are eight independent 12-bit AuxDACs integrated on the transceiver. The voltage range of the AuxDAC is from ground (0 V) to 1.8 V. The AuxDACs use the enumeration adi_adrv9025_AuxDacs_e when referenced in the API. The pins used for the AuxDAC features are listed in Table 225.

Table 225. AuxDAC Pin Mapping and adi_adrv9025_AuxDacs_e Enum Description

Auxiliary DAC Number	Pin Name	Pin Number	Enum Name	Enum Value
AUXDAC[0]	GPIO_ANA_0	C4	ADI_ADRV9025_AUXDAC0	0x01
AUXDAC[1]	GPIO_ANA_1	C5	ADI_ADRV9025_AUXDAC1	0x02
AUXDAC[2]	GPIO_ANA_2	L1	ADI_ADRV9025_AUXDAC2	0x04
AUXDAC[3]	GPIO_ANA_3	L2	ADI_ADRV9025 _AUXDAC3	0x08
AUXDAC[4]	GPIO_ANA_4	L17	ADI_ADRV9025_AUXDAC4	0x10
AUXDAC[5]	GPIO_ANA_5	L16	ADI_ADRV9025_AUXDAC5	0x20
AUXDAC[6]	GPIO_ANA_6	C12	ADI_ADRV9025_AUXDAC6	0x40
AUXDAC[7]	GPIO_ANA_7	C13	ADI_ADRV9025_AUXDAC7	0x80

The capacitive load of the AuxDAC pins must not exceed more than 100 pF or stability issues may occur.

The AuxDAC uses the GPIO_ANA pins on the device. Conflicts between GPIO_ANA and AuxDAC functionality may occur. In case of these conflicts, the AuxDAC takes precedence over all other GPIO_ANA functionality when AuxDAC is enabled for a specific pin. When the AuxDAC is disabled, the configured GPIO_ANA functionality is applied. The AuxDAC can be enabled one pin at a time to allow flexibility between AuxDAC and GPIO_ANA 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 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:

$$V_{AuxDAC} = \frac{AuxDACValue}{4096} \times 1.8\ \text{V}$$

where parameter AuxDacValue is the 12-bit digital code applied to the AuxDAC.

As previously mentioned, the AuxDAC is not a precision converter. It is best used in feedback systems. Figure 113 shows AuxDAC output voltage vs. input codes for a full range code sweep of the AuxDAC. Channel to channel variability in slope and dc offset are expected.

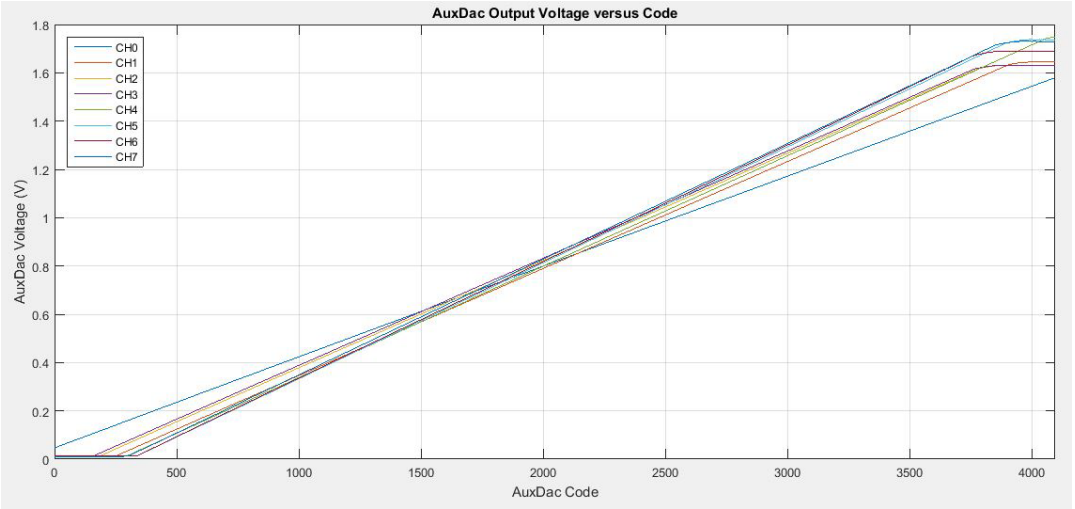


Figure 113. AuxDAC Channel Comparison over Full Range Code Sweep

AuxDAC Configuration

The AuxDAC is configured using the API command adi_adrv9025_AuxDacCfgSet(…). This command must be called after device initialization to use the AuxDACs.

```
adi_adrv9025_ AuxDacCfgSet(adi_adrv9025_Device_t *device, adi_adrv9025_AuxDacCfg_t
auxDacConfig[],
uint8_t numberOfCfg)
```

Table 226. AuxDAC Configuration Parameters

Parameter	Description
*device	Pointer to the device settings structure
auxDacConfig[]	The pointer to an array of AuxDAC configuration structure
numberOfCfg	The number of configurations at the auxDacConfig array

A data structure used in this command is the adi_adrv9025_AuxDacCfg_t data structure. The elements within this structure are described in Table 227.

Table 227. Description of adi_adrv9025_AuxDacCfg_t Data Structure

Data Type	Parameter Name	Comments
uint32_t	auxDacMask	AuxDAC selection. Bit 0 = AuxDAC0, Bit1 = AuxDAC1, …, Bit7 = AuxDAC7
uint8_t	enable	1 = Enable selected AuxDAC per auxDacMask. 0 = Disable selected AuxDAC.

AuxDAC Ouput Setup

After enabling the AuxDAC, the user can set the output value of one or more AuxDACs with the API command adi_adrv9025_AuxDacValueSet(…). This command is described below.

```
adi_adrv9025_ AuxDacValueSet(adi_adrv9025_Device_t* device,adi_adrv9025_AuxDacValue_t
auxDacValues[], uint8_t numberOfCfg)
```

Parameters

Table 228.

Parameter	Description
*device	Pointer to the device settings structure
auxDacValues[]	The array of DAC value data structures to set
numberOfCfg	The number of configurations at the auxDacValues array

A data structure used in this command is the adi_adrv9025_AuxDacValue_t data structure. The elements within this structure are described in Table 229.

Table 229. Description of adi_adrv9025_AuxDacValue_t Data Structure

Data Type	Parameter Name	Comments
uint32_t	auxDacMask	AuxDAC selection. Bit 0 = AuxDAC0, Bit1 = AuxDAC1, …, Bit7 = AuxDAC7
uint16_t	value	12-bit AuxDAC word to apply to AuxDACs selected by auxDacMask

AUXILIARY ADC (AUXADC)

There are two physical AuxADCs integrated on the device. Each AuxADC has two inputs for a total of four AuxADC pins. The different AuxADCs are designated ADI_ADRV9025_AUXADC_A and ADI_ADRV9025_AUXADC_B per the enumeration adi_adrv9025_AuxAdcSelect_e.

The AuxADC is a 12-bit Δ-Σ converter. It is most useful for relative voltage measurements rather than precision measurements due to slope and dc offset variability. The decimator state at the AuxADC output is linear to 10 bits. The input voltage range of the AuxADC is 50 mV to 950 mV. Readback of the AuxADC data word is performed using API commands. Accuracy of the AuxADC is dependent upon the supply voltages provided to VCONV1_1P0 for AUXADC_A and VCONV2_1P0 for AUXADC_B.

There are no on-chip calibrations executed or available for the AuxADC.

Each physical converter has two inputs providing four possible measurement channels (see Figure 114).

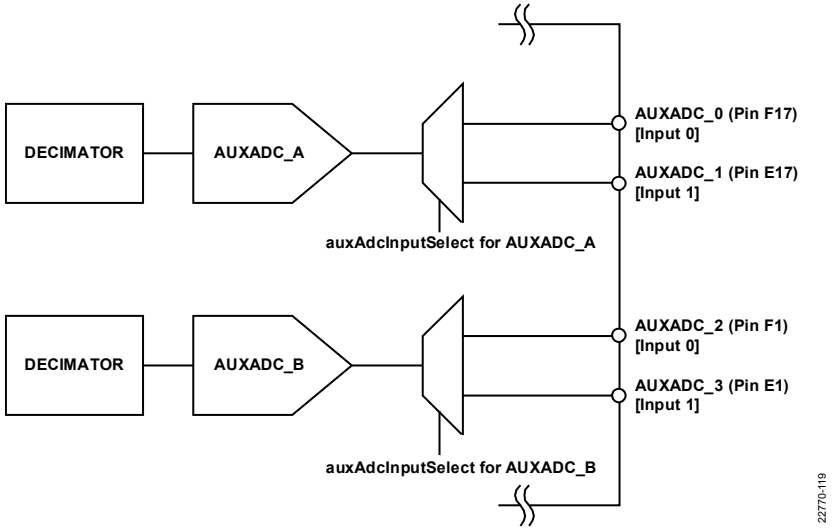


Figure 114. AuxADC On-Chip Block Diagram

The following (ideal) equation describes the output code in relation to an input voltage, V_{IN}. In practice, the AuxADC has slope and dc offset variability.

$$D_{OUT} = 4096(V_{IN} - 0.5\ V) + 2048$$

AuxADC Configuration

The AuxADC is configured with the following API command.

```
adi_adrv9025_AuxAdcCfgSet(adi_adrv9025_Device_t *device, adi_adrv9025_AuxAdcCfg_t *auxAdcConfig,
uint8_t arraySize)
```

Parameters

Table 230.

Parameter	Description
*device	Pointer to the device settings structure
*auxAdcConfig	Pointer to the supplied ADC configuration structure(s)
arraySize	The number of supplied configuration structures

An important data structure used in this command is adi_adrv9025_AuxAdcCfg_t. Table 231 describes the parameters used in this structure.

Table 231. Description of adi_adrv9025_AuxDacValueAuxDacAdcValueCfg_t Data Structure

Data Type	Parameter Name	Comments
AdiAadi_adrv9025_AuxAdcEnable_e	auxAdcEnable	Enable = 1, Disable = 0
AdiAadi_adrv9025_AuxAdcSelect_e	auxAdcSelect	Select which ADC to configure (AUXADC_A or AUXADC_B)
AdiAadi_adrv9025_AuxAdcInputSelect_e	auxAdcInputSelect	Select which input of the selected AuxADC to use (INPUT_0 or INPUT_1)
AdiAadi_adrv9025_AuxAdcClkDivide_e	auxAdcClkDivide	ADC CLK Divider Setting

The enumerations used in this structure are described further in the following tables.

Table 232. Description of adi_adrv9025_AuxAdcEnable_e Enumeration

Enum Name	Enum Value	Comments
ADI_ADRV9025_AUXADC_DISABLE	0	Aux ADC Disabled
ADI_ADRV9025_AUXADC_ENABLE	1	Aux ADC Enabled

Table 233 provides the enumerations describing the two physical converters on the device.

Table 233. Description of adi_adrv9025_AuxAdcSelect_e Enumeration

Enum Name	Enum Value	Comments
ADI_ADRV9025_AUXADC_A	0	Aux ADC A Selection
ADI_ADRV9025_AUXADC_B	1	Aux ADC B Selection

Table 234 provides the enumerations describing the two input selections that can be applied to each converter.

Table 234. Description of adi_adrv9025_AuxAdcInputSelect_e Enumeration

Enum Name	Enum Value	Comments
ADI_ADRV9025_AUXADC_INPUT_0	3	Aux ADC Input 0 Selection
ADI_ADRV9025_AUXADC_INPUT_1	2	Aux ADC Input 1 Selection

The AuxADC clock can be set based on a divider. The AuxADC input clock is supplied by the device clock input to the device (DEVCLK±). The valid options are provided in Table 235. Select the AuxADC divider setting such that the sampling clock frequency is set as low as possible without resulting in aliasing.

Table 235. Description of adi_adrv9025_AuxAdcClkDivide_e Enumeration

Enum Name	Enum Value	Comments
ADI_ADRV9025_AUXADC_CLKDIVIDE_32	0	Input clock divide by 32
ADI_ADRV9025_AUXADC_CLKDIVIDE_1	1	No Clock Divide
ADI_ADRV9025_AUXADC_CLKDIVIDE_2	2	Input Clock divide by 2
ADI_ADRV9025_AUXADC_CLKDIVIDE_3	3	Input Clock divide by 3
ADI_ADRV9025_AUXADC_CLKDIVIDE_4	4	Input Clock divide by 4
ADI_ADRV9025_AUXADC_CLKDIVIDE_5	5	Input Clock divide by 5
ADI_ADRV9025_AUXADC_CLKDIVIDE_6	6	Input Clock divide by 6
ADI_ADRV9025_AUXADC_CLKDIVIDE_7	7	Input Clock divide by 7
ADI_ADRV9025_AUXADC_CLKDIVIDE_8	8	Input Clock divide by 8
ADI_ADRV9025_AUXADC_CLKDIVIDE_9	9	Input Clock divide by 9
ADI_ADRV9025_AUXADC_CLKDIVIDE_10	10	Input Clock divide by 10
ADI_ADRV9025_AUXADC_CLKDIVIDE_11	11	Input Clock divide by 11
ADI_ADRV9025_AUXADC_CLKDIVIDE_12	12	Input Clock divide by 12
ADI_ADRV9025_AUXADC_CLKDIVIDE_13	13	Input Clock divide by 13
ADI_ADRV9025_AUXADC_CLKDIVIDE_14	14	Input Clock divide by 14
ADI_ADRV9025_AUXADC_CLKDIVIDE_15	15	Input Clock divide by 15

AuxADC Readback

After the AuxADC has been configured, the command that retrieves the AuxADC readback value is adi_adrv9025_AuxAdcValueGet(…). This command is described below.

```
adi_adrv9025_ AuxAdcValueGet(adi_adrv9025_Device_t *device, adi_adrv9025_AuxAdcSelect_e
auxAdcSelect, adi_adrv9025_AuxAdcValue_t *auxAdcValue)
```

Parameters

Table 236.

Parameter	Description
*device	Pointer to the device settings structure
auxAdcSelect	Selects the desired AuxADC to read a sample from
*auxAdcValue	Pointer to the supplied AuxADC value structure to populate

A data structure used in the above command is the adi_adrv9025_AuxAdcValue_t. Table 237 describes the members within this data structure.

Table 237. Description of adi_adrv9025_AuxAdcValue_t Data Structure

Data Type	Parameter Name	Comments
adi_adrv9025_AuxAdcSelect_e	auxAdcSelect	Selects which AuxADC to read from.
uint16_t	auxAdcValue	12-bit ADC sample from the selected AuxADC

TEMPERATURE SENSOR

The device features a 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 initiation of a temperature measurement is performed without user intervention by the Arm processor. The user can retrieve this measurement results in degrees C through an API command. The API command to readback the temperature sensor measurement is described below.

```
adi_adrv9025_ TemperatureGet(adi_adrv9025_Device_t *device, int16_t *temperatureDegC)
```

Parameters

Table 238.

Parameter	Description
*device	Pointer to the device settings structure
*temperatureDegC	Pointer to a single int16_t element that returns the current 12-bit temperature sensor in degrees C

SPI2 DESCRIPTION

The ADRV9026 uses the primary SPI port for nearly all SPI transactions needed during operation. The device also features a secondary SPI (SPI2) port that can be used to control Tx, Rx and ORx attenuation settings.

SPI2 CONFIGURATION

The SPI2 port can be enabled by calling the following API and setting spi2Enable to 1:

adi_adrv9025_Spi2CfgSet(adi_adrv9025_Device_t *device, uint8_t spi2Enable);

When this feature is enabled, the GPIO pins listed in Table 239 are configured automatically to the correct IO port direction to support the SPI Interface.

Table 239. SPI2 GPIO Pin Assignments

Pin Number	SPI2 Functionality	Pin Direction
GPIO_3	$\overline{CS}$	Input
GPIO_2	SCLK	Input
GPIO_1	SDO	Input/Output (depending on 3-wire or 4-wire wire mode)
GPIO_0	SDIO	Input/Output (depending on 3-wire or 4-wire wire mode)

The primary SPI and SPI2 share the same configuration: LSB first/MSB first, 3-wire/4-wire and single-instruction mode. Whichever configuration is selected for SPI is automatically assigned to SPI2.

TRANSMITTER CONTROL WITH SPI2

SPI2 provides the option to switch between two distinct attenuation states for the transmitters by toggling a single GPIO pin, bypassing the need to access the main SPI bus. The user can program four 10-bit attenuation words into registers designated State 1 (S1) and State 2 (S2). When the GPIO is low, the S1 registers set the attenuation values for the four transmitters. When the GPIO is high, the S2 registers set the attenuation values for the four transmitters. The user must select which GPIO is to be used to control the attenuation state. The valid selection values range from GPIO4 to GPIO18. The GPIO selection is performed by calling the following API

```
adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(adi_adrv9025_Device_t *device,
adi_adrv9025_TxAttenSpi2PinCfg_t txAttenSpi2PinCfg[], uint8_t numTxAttenSpi2PinConfigs);
```

Parameters

Table 240.

Parameter	Description
*device	Pointer to the device settings structure
txAttenSpi2PinCfg[]	An array of structures of type adi_adrv9025_TxAttenSpi2PinCfg_t detailed in Table 241
numTxAttenSpi2PinConfigs	The number of configurations passed in the array

Table 241. SPI2 Configuration Parameters

Parameter	Comments	
txChannelMask	This selects the channels upon which the API acts. It is a bit mask with each bit corresponding to a channel. The desired mask can be generated by OR'ing the desired channel enums as listed below. Data type: uint32_t	
	adi_adrv9025_TxChannels_e	**Tx Channel**
	ADI_ADRV9025_TXOFF	No Tx channels selected
	ADI_ADRV9025_TX1	Tx1 channel selected
	ADI_ADRV9025_TX2	Tx2 channel selected
	ADI_ADRV9025_TX3	Tx3 channel selected
	ADI_ADRV9025_TX4	Tx4 channel selected
txAttenSpi2Pin	This parameter selects which GPIO pin is used to select between Tx attenuation state 1 and state 2. Data type: adi_adrv9025_Spi2TxAttenGpioSel_e	
	txAttenSpi2Pin	**GPIO Selected**
	ADI_ADRV9025_SPI2_TXATTEN_DISABLE	Remove GPIO selection – this choice is only used if SPI2 is being disabled. This removes the previously selected GPIO from the list of used resources
	ADI_ADRV9025_SPI2_TXATTEN_GPIO4	Select GPIO 4 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO5	Select GPIO 5 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO6	Select GPIO 6 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO7	Select GPIO 7 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO8	Select GPIO 8 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO9	Select GPIO 9 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO10	Select GPIO 10 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO11	Select GPIO 11 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO12	Select GPIO 12 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO13	Select GPIO 13 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO14	Select GPIO 14 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO15	Select GPIO 15 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO16	Select GPIO 16 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO17	Select GPIO 17 for Tx state selection
	ADI_ADRV9025_SPI2_TXATTEN_GPIO18	Select GPIO 18 for Tx state selection

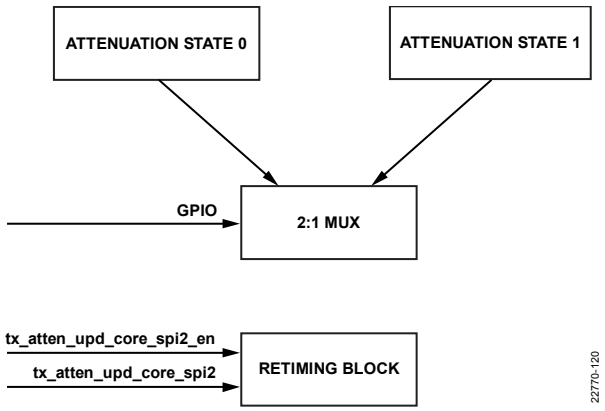


Figure 115. SPI2 Transmitter Attenuation Update Options

There are two update modes selectable for updating the attenuation applied to the transmitters, selected by bit D0 in SPI2 register 0x2A. When SPI2 0x2A[D0] is 0, updates to the attenuation state registers or MUX select GPIO take immediate effect. When SPI2 0x2A[D0] is 1, a retiming block is used to block updates to the transmit attenuation until a latch bit (one per transmitter channel) is set. The latch bits are in SPI2 register 0x2A, bits D4 to D1. Note that these bits are not self-clearing and must be written to zero before being used to latch new attenuation values.

Table 242. SPI2 Register 0x2A details

Register 0x2A	Comments
D4	Latch bit for Tx3 attenuation words (not self-clearing)
D3	Latch bit for Tx2 attenuation words (not self-clearing)
D2	Latch bit for Tx1 attenuation words (not self-clearing)
D1	Latch bit for Tx0 attenuation words (not self-clearing)
D0	Attenuation update mode selection bit. 0 = Update attenuation when LSB is written. 1 = Update attenuation when latch bit is set transitioned from low to high

It is generally preferred to synchronize the attenuation change of all the Tx channels in one device, or across an antenna array comprising many devices. An example of how to do this is given in the following steps:

1. Set 0x2A[D0] low to allow immediate updates of attenuation
2. Update the attenuation values of the attenuation state not in use
3. Toggle the selected attenuation state by toggling the GPIO pin which selects between states. The new attenuation values are now simultaneously applied to all transmitters in the product/antenna array

As this sequence is repeated, the Tx attenuation values of an entire antenna array can be adjusted simultaneously, with real time attenuation changes triggered by the GPIO transition.

The two different attenuation states for each transmitter can be stored in the SPI2 register map shown in Table 243. Values are written to these registers using the SPI protocol that is defined in the Serial Peripheral Interface (SPI) section.

RECEIVER AND OBSERVATION RECEIVER CONTROL WITH SPI2

SPI2 can also be used to control both receiver and observation receiver attenuation settings. Dual states like those used by the transmitters are not implemented for the receiver and observation receiver attenuation settings. When a new attenuation setting is written to one of the gain index registers shown in Table 243, an immediate update occurs. The value of each register can be written or read back using the SPI protocol that is defined in the Serial Peripheral Interface (SPI) section.

Table 243. SPI2 Register Map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0002	orx0_agc_manual_gain_index	agc_orx0_manual_gain_index							
0x0003	orx1_agc_manual_gain_index	agc_orx1_manual_gain_index							
0x0004	orx2_agc_manual_gain_index	agc_orx2_manual_gain_index							
0x0005	orx3_agc_manual_gain_index	agc_orx3_manual_gain_index							
0x0006	orx0_agc_gain_index_readback	agc_orx0_gain_index_readback							
0x0007	orx1_agc_gain_index_readback	agc_orx1_gain_index_readback							
0x0008	orx2_agc_gain_index_readback	agc_orx2_gain_index_readback							
0x0009	orx3_agc_gain_index_readback	agc_orx3_gain_index_readback							
0x000a	rx0_agc_manual_gain_index	agc_rx0_manual_gain_index							
0x000b	rx1_agc_manual_gain_index	agc_rx1_manual_gain_index							
0x000c	rx2_agc_manual_gain_index	agc_rx2_manual_gain_index							
0x000d	rx3_agc_manual_gain_index	agc_rx3_manual_gain_index							
0x000e	rx0_agc_gain_index_readback	agc_rx0_gain_index_readback							
0x000f	rx1_agc_gain_index_readback	agc_rx1_gain_index_readback							
0x0010	rx2_agc_gain_index_readback	agc_rx2_gain_index_readback							
0x0011	rx3_agc_gain_index_readback	agc_rx3_gain_index_readback							
0x0012	tx0_attenuation_readback_lsb	tx0_attenuation_readback[7:0]							
0x0013	tx0_attenuation_readback_msb	Reserved						tx0_attenuation_readback[9:8]	
0x0014	tx0_attenuation_s1_lsb	tx0_attenuation_s1[7:0]							
0x0015	tx0_attenuation_s1_msb	Reserved						tx0_attenuation_s1[9:8]	
0x0016	tx0_attenuation_s2_lsb	tx0_attenuation_s2[7:0]							
0x0017	tx0_attenuation_s2_msb	Reserved						tx0_attenuation_s2[9:8]	
0x0018	tx1_attenuation_readback_lsb	tx1_attenuation_readback[7:0]							
0x0019	tx1_attenuation_readback_msb	Reserved						tx1_attenuation_readback[9:8]	
0x001a	tx1_attenuation_s1_lsb	tx1_attenuation_s1[7:0]							
0x001b	tx1_attenuation_s1_msb	Reserved						tx1_attenuation_s1[9:8]	
0x001c	tx1_attenuation_s2_lsb	tx1_attenuation_s2[7:0]							
0x001d	tx1_attenuation_s2_msb	Reserved						tx1_attenuation_s2[9:8]	
0x001e	tx2_attenuation_readback_lsb	tx2_attenuation_readback[7:0]							
0x001f	tx2_attenuation_readback_msb	Reserved						tx2_attenuation_readback[9:8]	
0x0020	tx2_attenuation_s1_lsb	tx2_attenuation_s1[7:0]							
0x0021	tx2_attenuation_s1_msb	Reserved						tx2_attenuation_s1[9:8]	
0x0022	tx2_attenuation_s2_lsb	tx2_attenuation_s2[7:0]							
0x0023	tx2_attenuation_s2_msb	Reserved						tx2_attenuation_s2[9:8]	
0x0024	tx3_attenuation_readback_lsb	tx3_attenuation_readback[7:0]							
0x0025	tx3_attenuation_readback_msb	Reserved						tx3_attenuation_readback[9:8]	
0x0026	tx3_attenuation_s1_lsb	tx3_attenuation_s1[7:0]							
0x0027	tx3_attenuation_s1_msb	Reserved						tx3_attenuation_s1[9:8]	
0x0028	tx3_attenuation_s2_lsb	tx3_attenuation_s2[7:0]							
0x0029	tx3_attenuation_s2_msb	Reserved						tx3_attenuation_s2[9:8]	
0x002a	tx_atten_upd_spi2	Reserved			tx_atten_upd_core_spi2				tx_atten_upd_core_spi2_en

RF PORT INTERFACE OVERVIEW

This section describes the recommended RF transmitter and receiver interfaces to obtain optimal device performance. This section includes data regarding the expected RF port impedance values and examples of impedance matching networks used in the evaluation platform. Some reference is also provided regarding board layout techniques and balun selection guidelines.

The ADRV9026 is a highly integrated transceiver with transmit, receive and observation receive signal chains. External impedance matching networks are required on transmitter and receiver ports to achieve performance levels indicated on the data sheet. Analog Devices Inc. recommends the utilization of simulation tools in the design and optimization of impedance matching networks. To achieve best correlation from simulation to PCB, accurate models of the board environment, SMD components (for example, baluns and filters), and device port impedances are required.

RF PORT IMPEDANCE DATA

This section provides the port impedance data for all transmitters and receivers in the device. Note the following:

- Z_o is defined as 50 Ω for Tx and as 100 Ω for Rx/ORx.
- The reference plane for this data is the device ball pads.
- Single ended mode port impedance data is not available. However, a rough assessment is possible by taking the differential mode port impedance data and dividing both the real and imaginary components by 2.
- Contact Analog Devices Applications Engineering for the impedance data in Touchstone format.

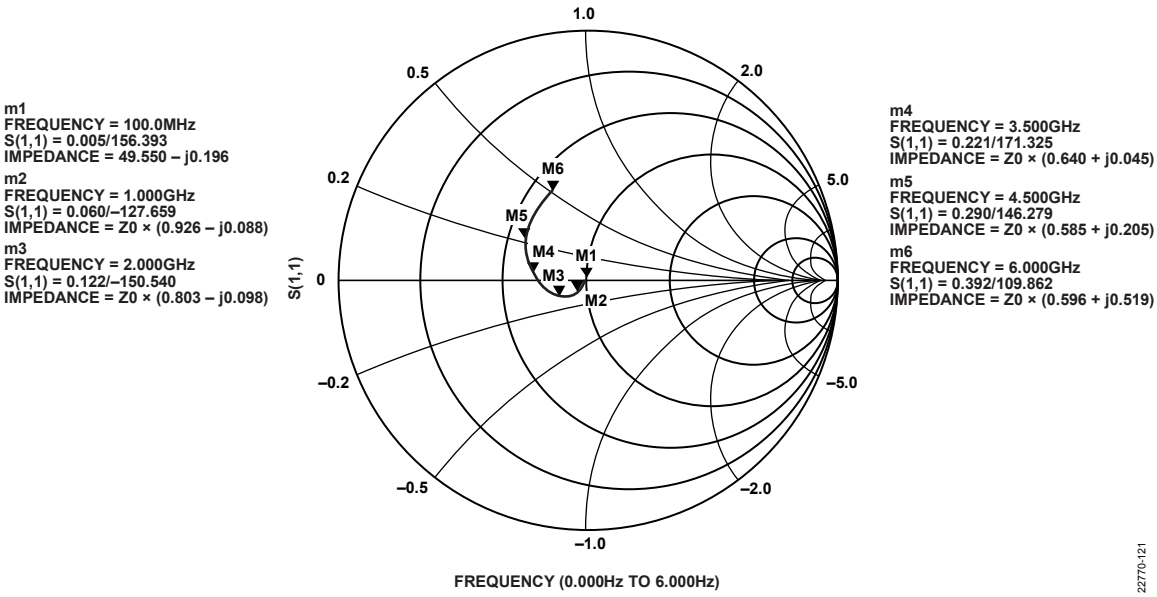


Figure 116. Tx1 and Tx4 SEDZ and PEDZ Data

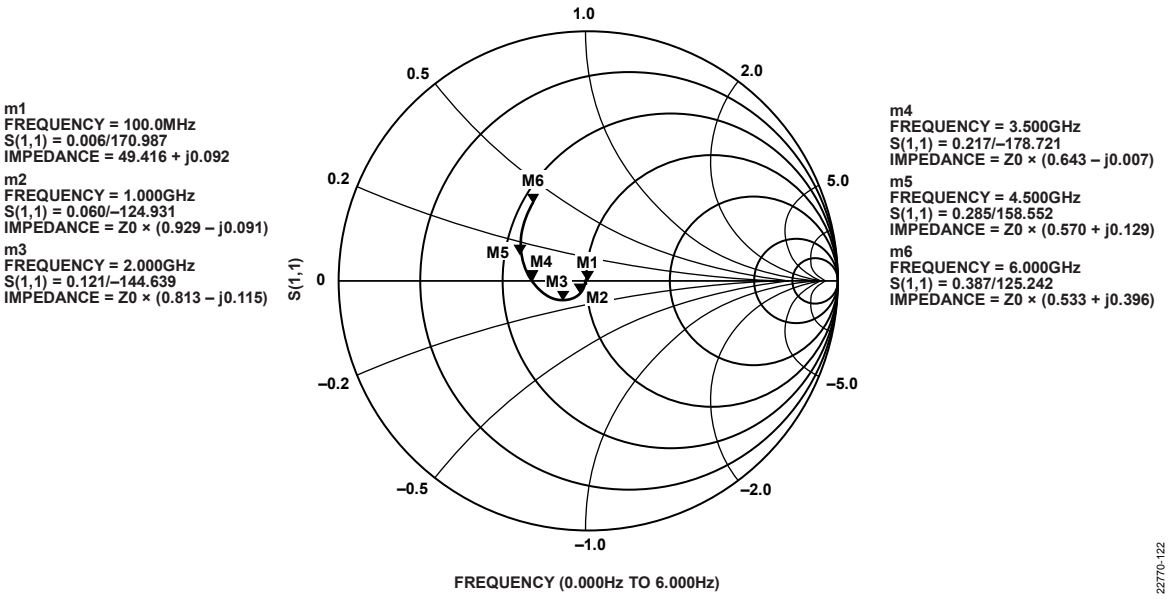


Figure 117. Tx2 and Tx3 SEDZ Data

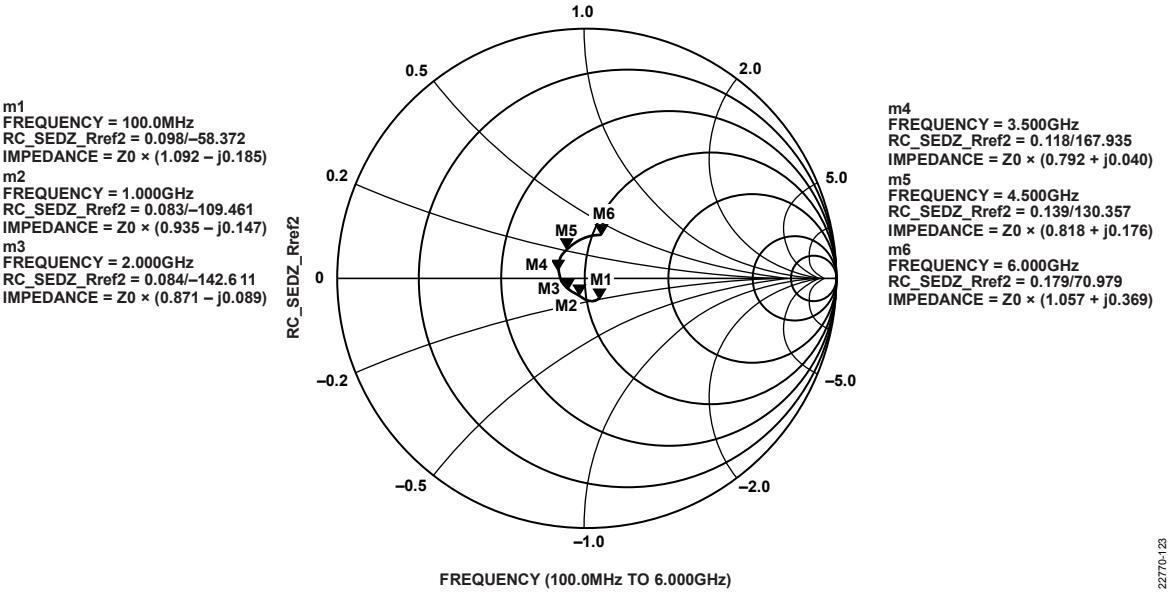


Figure 118. Rx1 and Rx4 SEDZ Data

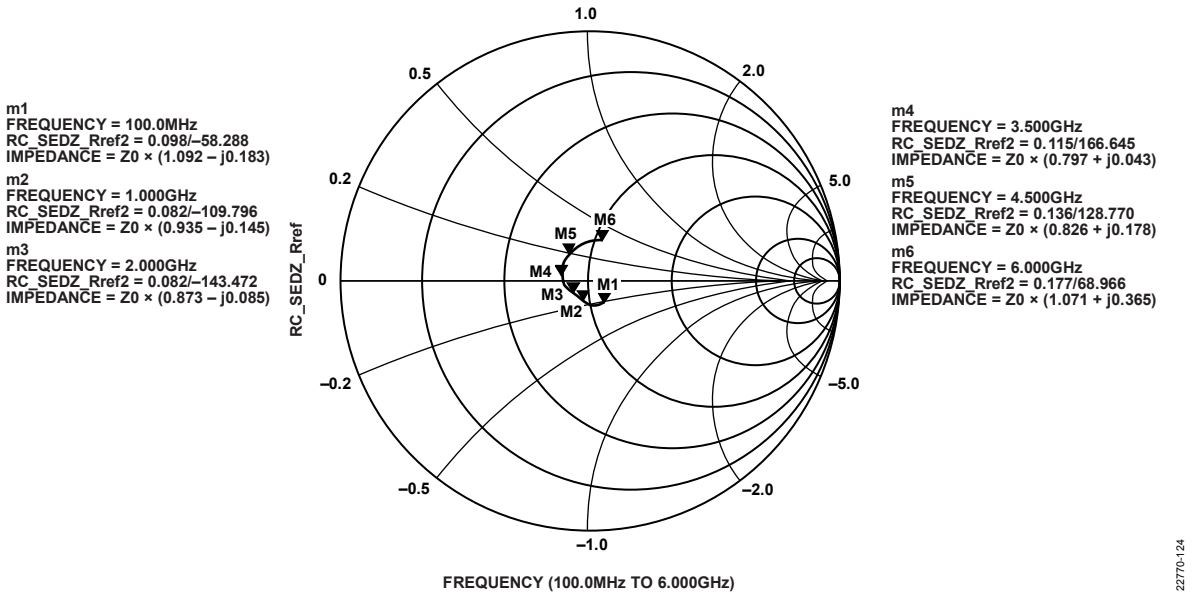


Figure 119. Rx2 and Rx3 SEDZ Data

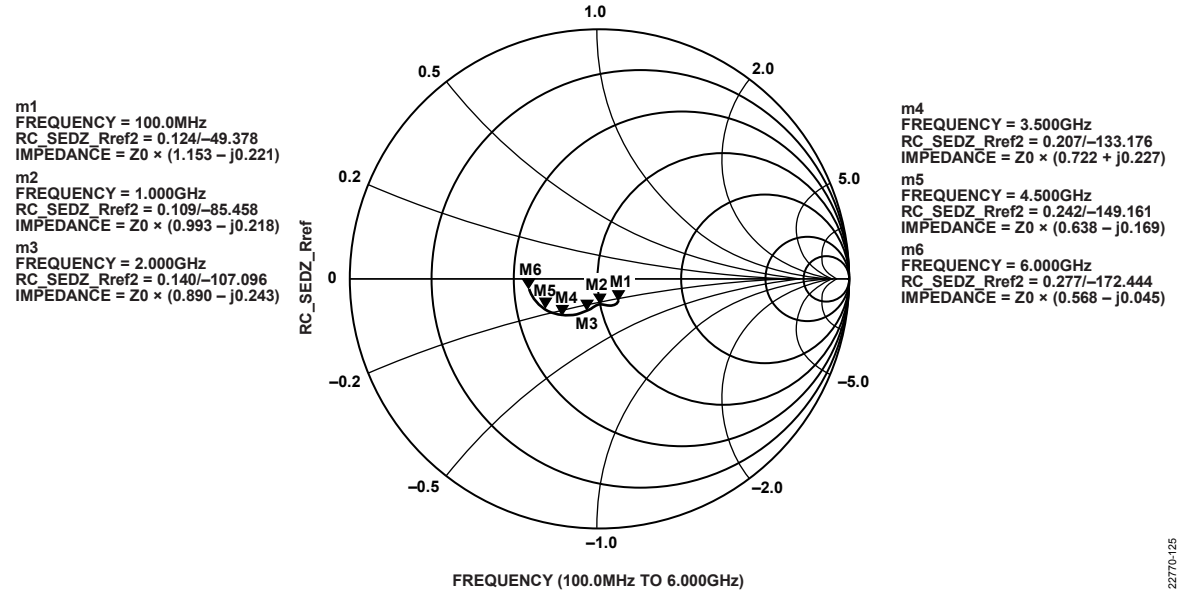


Figure 120. ORx1 and ORx4 SEDZ Data

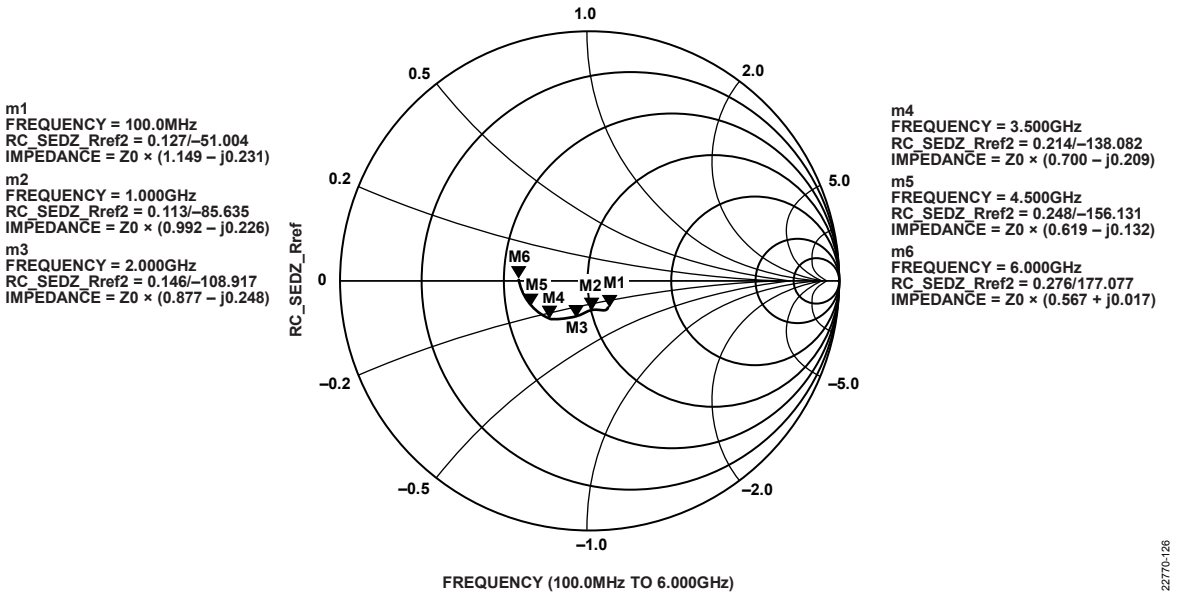


Figure 121. ORx2 and ORx3 SEDZ Data

ADS SETUP USING DAC (DATA ACCESS COMPONENT) AND SEDZ FILE

The port impedances are supplied as an *.s1p Series Equivalent Differential Z (impedance) file. This format allows simple interface to ADS by using the Data Access Component. In Figure 122, Term1 is the single ended input or output and Term2 represents the differential input or output RF port. The Pi network 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 appropriate component selection.

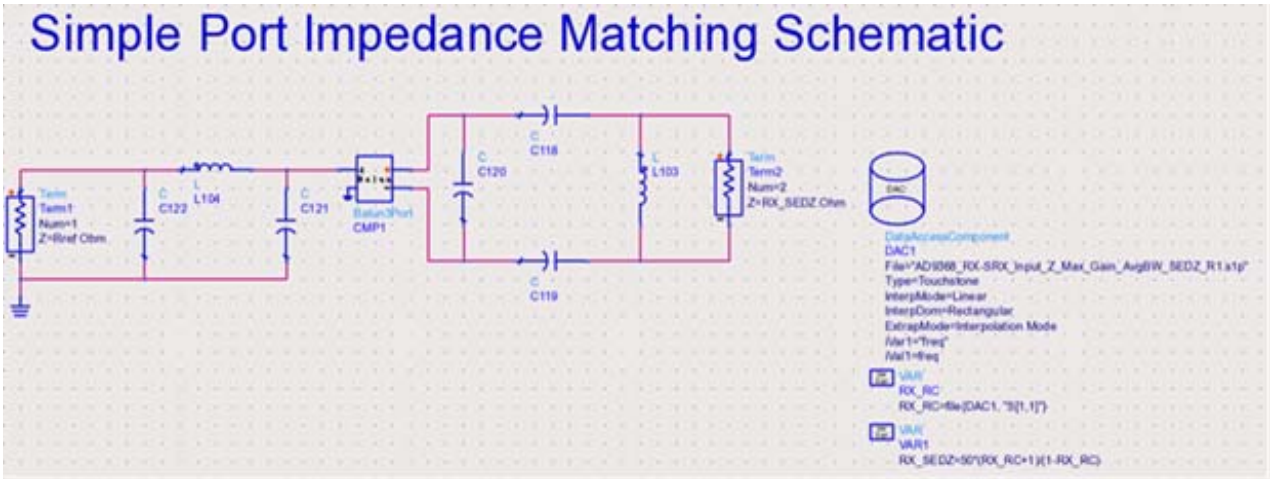


Figure 122. Simulation Setup in ADS with SEDZ s1p Files and DAC Ccomponent

Operation is as follows:

1. 1. The DAC bBlock reads the rf port *.s1p file. This is the device rf port reflection coefficient.
2. 2. The two equations convert the RF port reflection coefficient to a complex impedance. The end result is the RX_SEDZ variable.
3. 3. The RF port calculated complex impedance (RX_SEDZ) is utilized to define the Term2 impedance.

Term2 is used in a differential mode and Term1 is used in a single-ended mode. Setting up the simulation this way allows measurement of S11, S22, and S21 of the 3-port system without complex math operations within the display page.

For highest accuracy, use EM modeling results of the PCB artwork and S parameters of the matching components and balun in the simulations.

TRANSMITTER BIAS AND PORT INTERFACE

This section considers the dc biasing of the transmitter (Tx) outputs and how to interface to each Tx port. The transmitters operate over a range of frequencies. At full output power, each differential output side draws approximately 100 mA of dc bias current. The Tx outputs are dc biased to a 1.8 V supply voltage using either RF chokes (wire-wound inductors) or a transformer center tap connection.

Careful design of the dc bias network is required to ensure optimal RF performance levels. When designing the dc bias network, select components with low dc resistance (R_{DCR}) to minimize the voltage drop across the series parasitic resistance element with either of the suggested dc bias schemes suggested in Figure 123. The red resistors (R_DCR) indicate the parasitic elements. As the impedance of the parasitics increase, the voltage drop (ΔV) across the parasitic element increases, causing the transmitter RF performance (for example, $P_{O,1dB}$ and $P_{O,MAX}$) to degrade. Select the choke inductance (L_C) high enough relative to the load impedance such that it does not degrade the output power.

The recommended dc bias network is shown in Figure 124. This network has fewer parasitics and fewer total components.

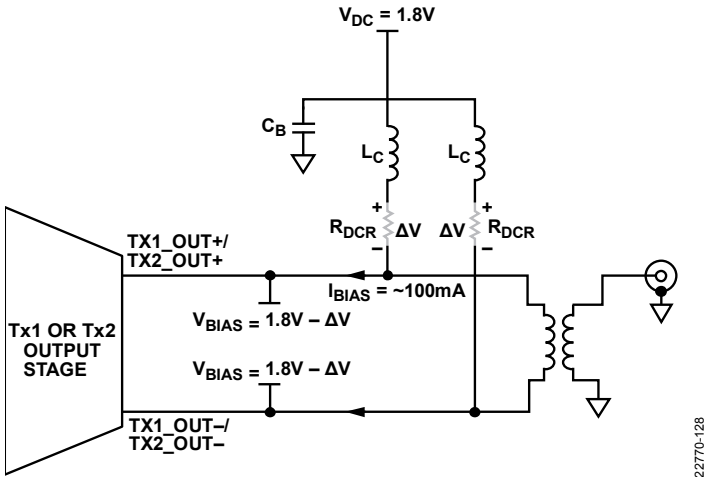


Figure 123. RF DC Bias Configurations Depicting Parasitic Losses Due toWire Wound Chokes

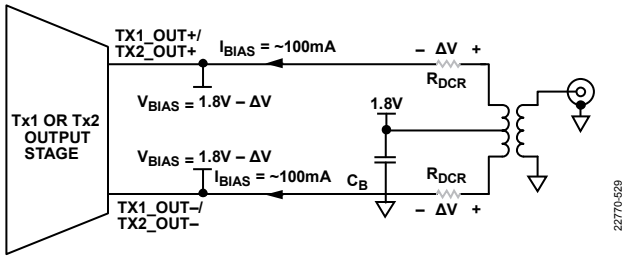


Figure 124. RF DC Bias Configurations Depicting Parasitic Losses Due to Center Tapped Transformers

Figure 125 to Figure 128 identifyfour basic differential transmitter output configurations. Impedance matching networks (balun single-ended port) are most likely required to achieve optimum device performance from the device. Also, the transmitter outputs must be ac-coupled in most applications due to the dc bias voltage applied to the differential output lines of the transmitter.

The recommended RF transmitter interface featuring a center tapped balun is shown in Figure 125. This configuration offers the lowest component count of the options presented.

Brief descriptions of the Tx port interface schemes are provided as follows:

- Center tapped transformer passes the bias voltage directly to the transmitter outputs
- RF chokes are used to bias the differential transmitter output lines. Additional coupling capacitors (C_C) are added in the creation of a transmission line balun
- RF chokes are used to bias the differential transmitter output lines and connect into a transformer
- RF chokes are used to bias the differential output lines that are ac-coupled into the input of a driver amplifier.

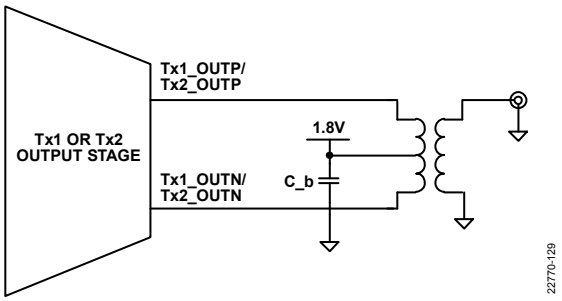


Figure 125. ADRV9026 RF Transmitter Interface Configuration A

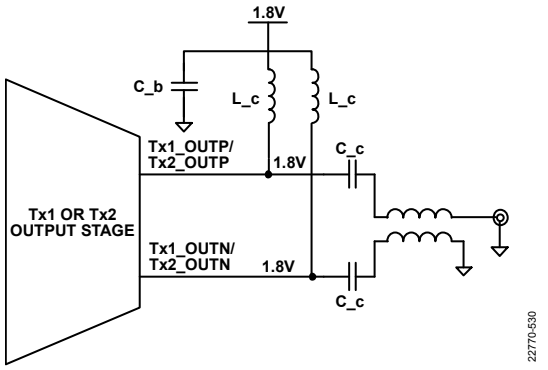


Figure 126. ADRV9026 RF Transmitter Interface Configuration B

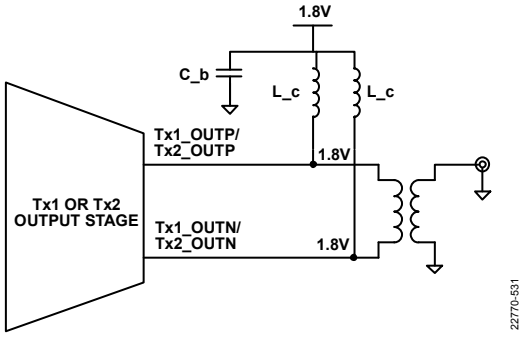


Figure 127. *ADRV9026 RF Transmitter Interface Configuration C*

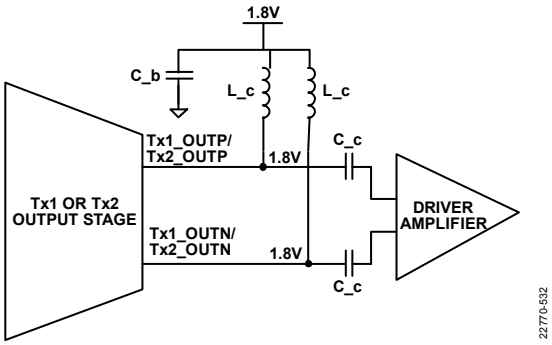


Figure 128. ADRV9026 RF Transmitter Interface Configuration D

If a Tx balun is selected that requires a set of external dc bias chokes, careful planning is required. It is necessary to find the optimum compromise between the choke physical size, choke dc resistance (R_{DCR}) and the balun low frequency insertion loss. In commercially available dc bias chokes, resistance decreases as size increases. However, as choke inductance increases, resistance increases. Therefore, it is undesirable to use physically small chokes with high inductance as they exhibit the greatest resistance. For example, the voltage drop of a 500 nH, 0603 choke at 100 mA is roughly 50 mV.

Table 244. Sample Wire-Wound DC Bias Choke Resistance vs. Size

Inductance (nH)	Resistance (Size: 0603)	Resistance (Size: 1206)
100	0.10	0.08
200	0.15	0.10
300	0.16	0.12
400	0.28	0.14
500	0.45	0.15
600	0.52	0.20

GENERAL RECEIVER PATH INTERFACE

The device has two types of receivers. These receivers include four main receive pathways (Rx1, Rx2, Rx3 and Rx4) and four observation receivers (ORx1, ORx2, ORx3 and ORx4). The Rx and ORx channels are designed for differential use only.

The receivers support a wide range of operation frequencies. In the case of the Rx and ORx channels, the differential signals interface to an integrated mixer. The mixer input pins have a dc bias of approximately 0.7 V present on them and may need to be ac-coupled depending on the common mode voltage level of the external circuit.

Important considerations for the receiver port interface are as follows:

- Device to be interfaced: filter, balun, T/R switch, external LNA, and external PA. Determine if this device represents a short to ground at dc.
- Rx and ORx maximum safe input power is +18 dBm (peak).
- Rx and ORx optimum dc bias voltage is 0.7 V bias to ground.
- Board Design: reference planes, transmission lines, and impedance matching.

Figure 129 shows possible differential receiver port interface circuits. The options in Figure 129 and Figure 130 are valid for all receiver inputs operating in differential mode, though only the Rx1 signal names are indicated. Impedance matching may be necessary to obtain data sheet performance levels.

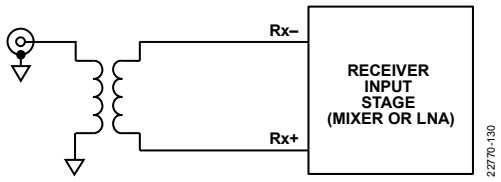


Figure 129. Differential Receiver Input Interface Circuits

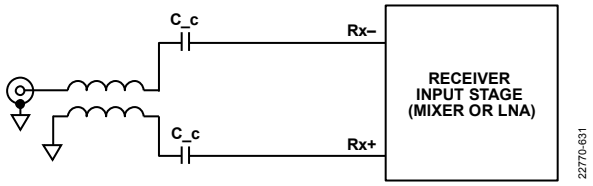


Figure 130. Differential Receiver Input Interface Circuits

Given wide RF bandwidth applications, SMD balun devices function well. Decent loss and differential balance are available in a relatively small (0603, 0805) package.

IMPEDANCE MATCHING NETWORK EXAMPLES

Impedance matching networks are required to achieve performance levels noted on the data sheet. This section provides example topologies and components used on the CE board.

Models of the devices, board, balun and SMD components are required to build an accurate system level simulation. The board layout model may be obtained from an EM (electro-magnetic: Momentum) simulator. The balun and SMD component models may be obtained from the device vendors or built locally. Contact Analog Devices Applications Engineering for device modeling details.

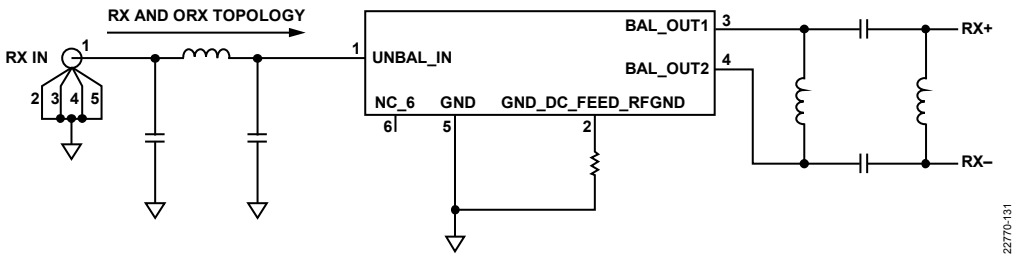


Figure 131. Impedance Matching Topology

The impedance matching networks provided in this section have not been evaluated in terms of Mean Time to Failure (MTTF) in high volume production. Consult with component vendors for long-term reliability concerns. Additionally, consult with balun vendors to determine appropriate conditions for dc biasing.

The schematics in Figure 132, Figure 133, and Figure 134 show two or three circuit elements in parallel marked DNI (Do Not Include). This was done on the evaluation board schematic to accommodate different component configurations for different frequency ranges. Only one set of SMD component pads are placed on the board to provide a physical location that can be used for the selected parallel circuit element. For example, R302, L302, and C302 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 may be either a resistor, inductor or a capacitor whereas the shunt elements may be either an inductor or a capacitor. Only one component of each parallel combination is placed in a practical application. Note that in some matching circuits, some shunt elements may not be required. All components for a given physical location remain DNI in those particular applications.

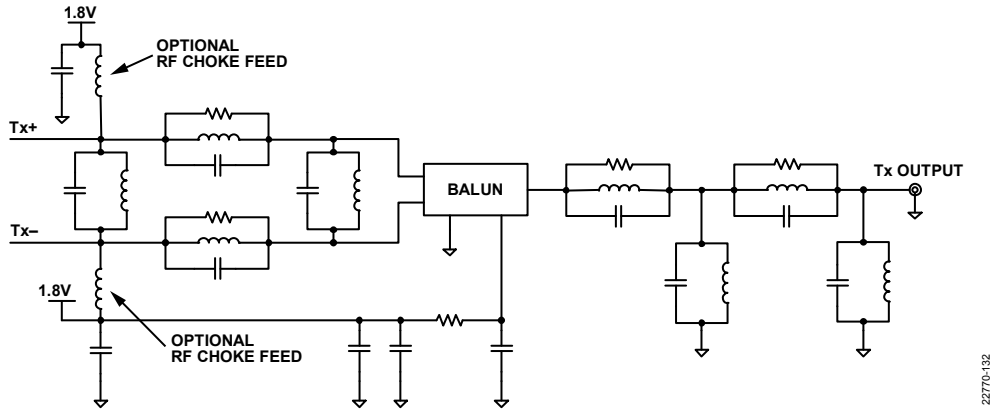


Figure 132. Transmitter Generic Matching Network Topology from CE Board

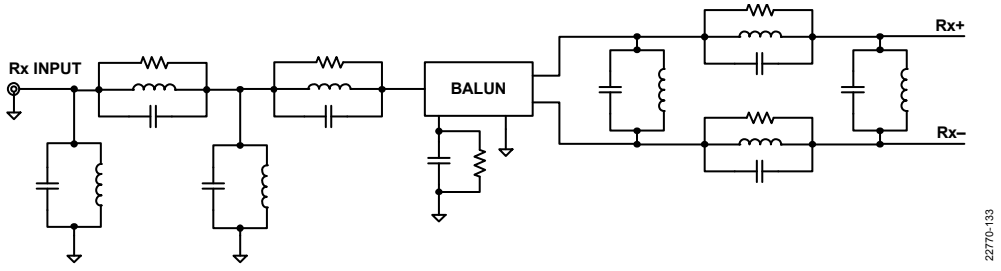


Figure 133. Receiver Generic Matching Network Topology from CE Board

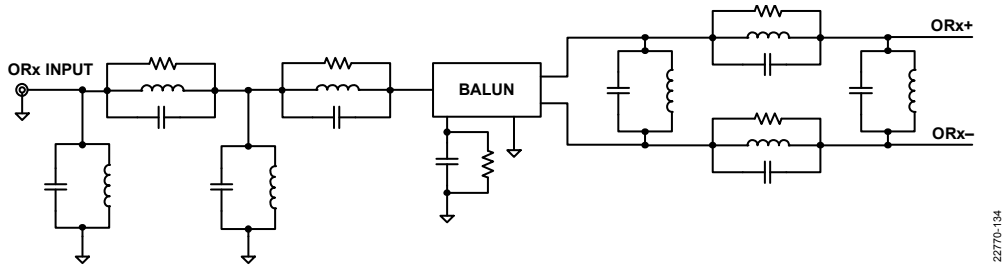


Figure 134. Observation Receiver Generic Matching Network Topology from CE Board

MATCHING COMPONENT RECOMMENDATIONS

Table 245 through Table 250 show the balun and matching components used on the CE boards. DNI stands for do not install (leave open). Note that all tolerances are ±3% unless listed. Tolerance notations are either shown as a percentage of the nominal value (%) or as a range in the units of the component. Component reference designators can be cross-referenced with the schematic drawings for the CE boards.

Table 245. Receiver Matching Components—Rx1 and Rx4

	Component Location on PCB (All Tolerances Are ±3% Unless Noted)									
Frequency Band (MHz)	**C301/ L301, C331/ L331**	**C302/L302/ R302, C332/ L332/ R332**	**C303/ L303, C333/ L333**	**C304/L304/ R304, C334/ L334/R334**	**C305/ R305, C335/ R335**	**C306/ L306, C336/ L336**	**C307/L307/ R307, C337/ L337/ R337**	**C308/L308/ R308, C338/ L338/R338**	**C309/ L309, C339/ L339**	**T301, T307**
650 to 2800	DNI	0 Ω	DNI	0 Ω	0 Ω	91 nH	3.9 pF ± 0.1 pF	3.9 pF ± 0.1 pF	47 nH	Johanson 1720BL15A0100
2800 to 6000	DNI	1.2 nH ± 0.1 nH	DNI	0 Ω	1.8 pF ± 0.1 pF	9.1 nH	0.7 nH ± 0.1 nH	0.7 nH ± 0.1 nH	30 nH	Johanson 4400BL15A0100E

Table 246. Receiver Matching Components—Rx2 and Rx3

	Component Location on PCB (All Tolerances Are ±3% Unless Noted)									
Frequency Band (MHz)	**C311/ L311, C321/ L321**	**C312/L312/ R312, C322/ L322/R322**	**C313/ L313, C323/ L323**	**C314/L314/ R314, C324/ L324/R324**	**C315/ R315, C325/ R325**	**C316/ L316, C326/ L326**	**C317/ L317/R317, C327/ L327/ R327**	**C318/L318/ R318, C328/ L328/R328**	**C319/ L319, C329/ L329**	**T303, T305**
650 to 2800	DNI	0 Ω	DNI	0 Ω	0 Ω	100 nH	3.9 pF ± 0.1 pF	3.9 pF ± 0.1 pF	43 nH	Johanson 1720BL15A0100
2800 to 6000	DNI	1.2 nH ± 0.1 nH	0.2 pF ± 0.05 pF	0 Ω	4.8 pF ± 0.1 pF	9.1 nH	0.7 nH ± 0.1 nH	0.7 nH ± 0.1 nH	30 nH	Johanson 4400BL15A0100E

Table 247. Observation Receiver Matching Components—ORx2 and ORx4

	Component Location on PCB (All Tolerances Are ±3% Unless Noted)									
Frequency Band (MHz)	**C401/ L401, C431/ L431**	**C402/L402/ R402, C432/ L432/R432**	**C403/ L403, C433/ L433**	**C404/L404/ R404, C434/ L434/R434**	**C405/ R405, C435/ R435**	**C406/ L406, C436/ L436**	**C407/ L407/ R407, C437/ L437/R437**	**C408/ L408/ R408, C438/ L438/ R438**	**C409/ L409, C439/ L439**	**T401, T407**
650 to 2800	DNI	0 Ω	DNI	0 Ω	0 Ω	82 nH	4.7 pF ± 0.1 pF	4.7 pF ± 0.1 pF	75 nH ± 5%	Johanson 1720BL15A0100
2800 to 6000	DNI	1.3 nH ± 0.1 nH	0.2 pF ± 0.05 pF	0 Ω	5.6 pF ± 0.1 pF	7.5 nH	0.6 nH ± 0.1 nH	0.6 nH ± 0.1 nH	0.1pF ± 0.05 pF	Johanson 4400BL15A0100E

Table 248. Observation Receiver Matching Components—ORx1 and ORx3

	Component Location on PCB (All Tolerances Are ±3% Unless Noted)									
Frequency Band (MHz)	**C411/ L411, C421/ L421**	**C432/ L412/ R412, C422/ L422/R422**	**C413/ L413, C423/ L423**	**C414/ L414/ R414, C424/ L424/R424**	**C415/ R415, C425/ R425**	**C416/ L416, C426/ L426**	**C417/L417/ R417, C427/ L427/R427**	**C418/L418/ R418, C428/ L428/R428**	**C419/ L419, C429/ L429**	**T403, T405**
650 to 2800	DNI	0 Ω	DNI	0 Ω	0 Ω	200 nH	10 pF ± 5%	10 pF ± 5%	200 nH	Johanson 1720BL15A0100
2800 to 6000	13 nH	0.5 nH ± 0.1 nH	DNI	0.3 nH ± 0.1 nH	1.6 pF ± 0.1 pF	11 nH	0.5 nH ± 0.1 nH	0.5 nH ± 0.1 nH	39 nH	Johanson 4400BL15A0100E

Table 249. Transmitter Matching Components—Tx1 and Tx4

	Component Location on PCB (All Tolerances Are ±3% Unless Noted)									
Frequency Band (MHz)	**C512/ L512, C572/ L572**	**C511/L511/ R511, C571/ L571/R571**	**C510/ L510, C570/ L570**	**C509/L509/ R509, C569/ L569/R569**	**C508/ L508, C568/ L568**	**C506/L506/ R506, C566/ L566/R566**	**C507/ L507/ R507, C567/ L567/ R567**	**C503/ L503, C563/ L563**	**C516, C576**	**T501, T507**
650 to 2800	DNI	0.8 nH ± 0.1 nH	33 nH	5.1 pF ± 0.1 pF	DNI	0 Ω	0 Ω	DNI	82 pF	Johanson 1720BL15A0100
2800 to 6000	3.2 nH ± 0.1 nH	8.2 pF ± 0.1 pF	DNI	2 pF ± 0.1 pF	16 nH	1.1 nH ± 0.1 nH	1.1 nH ± 0.1 nH	12 nH	6.2 pF ± 0.1 pF	Johanson 4400BL15A0100E

Table 250. Transmitter Matching Components—Tx2 and Tx3

	Component Location on PCB (All Tolerances Are ±3% Unless Noted)									
Frequency Band (MHz)	**C532/ L532, C552/ L552**	**C531/L531/ R531, C551/ L551/R551**	**C530/ L530, C550/ L550**	**C529/L529/ R529, C549/ L549/R549**	**C528/ L528, C548/ L548**	**C526/ L526/ R526, C546/ L546/ R546**	**C527/ L527/ R527, C547/ L547/ R547**	**C523/ L523, C543/ L543**	**C536, C556**	**T503, T505**
650 to 2800	DNI	0.9 nH ± 0.1 nH	200 nH	6.8 pF ± 0.1 pF	DNI	0 Ω	0 Ω	DNI	82 pF	Johanson 1720BL15A0100
2800 to 6000	62 nH	1.8 nH ± 0.1 nH	0.2 pF ± 0.05 pF	0.5 nH ± 0.1 nH	12 nH	1 nH ± 0.1 nH	1 nH ± 0.1 nH	20 nH	4.9 pF ± 0.1 pF	Johanson 4400BL15A0100E

POWER MANAGEMENT CONSIDERATIONS

The ADRV9026 requires five different power supply domains:

- 1.0 V digital: this supply is connected to the device through the three VDIG_1P0 pins. This is the supply that feeds all digital processing and clock generation. 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. This supply is a high-current input, so it is critical that the input traces for these three inputs be balance (same impedance for inputs) and as thick as possible to minimize the I × R drop.
- 1.0 V analog: these supplies are collectively referred to in the data sheet as the VANA_1P0 supply. This covers the VDES_1P0, VSER_1P0, VTT_DES, and VJSY_1P0 supplies. All of these inputs provide power for various functions in the JESD interface blocks. They can be connected directly to the same supply as VDIG_1P0 if the source has the current capability to supply the extra current needed for the JESD interface and if proper isolation is included to prevent digital noise from corrupting these inputs. Alternatively, these supply inputs can be connected to a separate 1.0 V regulator to keep them isolated from digital domains inside the device. This supply input also has a tolerance of ±5%.
- 1.3 V analog: these supplies connect to all functional blocks in the device through 14 different input pins. They are collectively referred to in the data sheet as the VANA_1P3 supply. Treat each input as a noise susceptible input, meaning proper decoupling and isolation techniques must be followed to avoid crosstalk between channels. The tolerance on these supply inputs is ±2.5%.
- 1.8 V analog: these supplies are primarily used to supply the transmitter outputs, but they also supply current for multiple transmitter, receiver, converter, and auxiliary converter blocks. They are collectively referred to in the data sheet as the VANA_1P8 supply. This supply has a tolerance of ±5%.
- Interface supply: the VIF supply is a separate power domain shared with the baseband processor interface. The nominal input voltage on this supply is 1.8 V with a tolerance of ±5%. This input serves as the voltage reference for the digital interface (SPI), GPIO, and digital control inputs.

IMPORTANT

During operation, supply currents can vary significantly, especially if operating in TDD mode. The supply needs to have adequate capacity to provide the necessary current (as indicated on the data sheet) so that performance criteria over all process and temperature variations are maintained. Analog Devices recommends adding 900 mA to the digital and 20% margin to all analog supply maximums to ensure proper operation under all conditions.

POWER SUPPLY SEQUENCE

The device requires a specific power-up sequence to avoid undesirable power-up currents. In the optimal sequence, the VDIG_1P0 supply must come up first. If the VANA_1P0 supplies are connected to the same source as the VDIG_1P0 supply, then it is acceptable for these inputs to power up at the same time as the VDIG_1P0 supply. After the VDIG_1P0 source is enabled, the other supplies can be enabled in any order or all together. Note that the VIF supply can be enabled at any time without affecting the other circuits in the device. In addition to this sequence, it is also recommended to toggle the $\overline{\text{RESET}}$ signal after power has stabilized prior to initializing the device.

The power-down sequence recommendation is similar to power-up. Disable all analog supplies in any order (or all together) before VDIG_1P0 is disabled. If such a sequence is not possible, then disable the sources of all supplies simultaneously to ensure there is no back feeding circuits that have been powered down.

POWER SUPPLY DOMAIN CONNECTIONS

lists the pin number, the pin name, the recommended routing technique for that pin from the main 1.3 V analog supply (if applicable), and a brief description of the block it powers in the chip.

The information listed in Table 251 shows which power supply pins must be powered by designated traces and which pins are tied together and share a common trace. In some cases, a separate trace from a common power plane is used to power up two to three 1.3 V power supply pins, wheras in other cases, there are power supply pins that are powered from a separate trace.

The recommendation for VDDA1P3_DES is to keep it separate from the VDDA1P3_SER supplies using a separate trace. It is acceptable to power this input from the other 1.3 V analog supply. Noise from this supply can affect the JESD link performance directly.

Table 251. Power Supply Pins and Functions

Pin Name	Pin No.	Type	Voltage (V)	Recommended Routing/Notes	Description
VDIG_1P0	G9, J9, L9	Digital	1.0	Ensure all connections are matched to avoid variations in voltage among the pins. Minimize total impedance to ensure as little voltage drop as possible.	Digital clocks and processing blocks
VIF	N9	Analog	1.8	CMOS/LVDS Interface Supply (routing typically not critical).	Supply for SPI interface, GPIO, control signals
TX1± (RF Choke Feed)	N17, P17	Analog	1.8	Star connect from the 1.8 V plane, isolated by ground from other transmitter supplies, connected to pins using RF chokes (part depends on frequency range)	Alternative Tx supply if power is not supplied via a center-tapped balun
TX2± (RF Choke Feed)	A13, A14	Analog	1.8	Star connect from the 1.8 V plane, isolated by ground from other transmitter supplies, connected to pins using RF chokes (part depends on frequency range)	Alternative Tx supply if power is not supplied via a center-tapped balun
TX3± (RF Choke Feed)	A4, A5	Analog	1.8	Star connect from the 1.8 V plane, isolated by ground from other transmitter supplies, connected to pins using RF chokes (part depends on frequency range)	Alternative Tx output supply if power is not supplied via a center-tapped balun
TX4± (RF Choke Feed)	P1, N1	Analog	1.8	Star connect from the 1.8 V plane, isolated by ground from other transmitter supplies, connected to pins using RF chokes (part depends on frequency range)	Alternative Tx output supply if power is not supplied via a center-tapped balun
VANA1_1P8	N16	Analog	1.8	Star connect from the 1.8 V plane. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.8V supply for: Tx1 analog output, Rx1 LO buffer, RF synth1, AUXADC_0, AUXADC_1, Rx1 TIA, ORx1 mixer, Converter1 LDO
VANA2_1P8	B14	Analog	1.8	Star connect from the 1.8 V plane. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.8V supply for: Tx2 analog output, Tx1_2 LO buffers and LO delay, Analog SPI, DEVCLK, AUX PLL and AUX LO generation, Rx LO mux and mbias, Rx2 LO buffer, RF PLL1 and LOGEN1, ORx1_2 LO buffers and TxLB1_2 LO buffer, Rx2 TIA, ORx2 mixer, ORx1 and ORx2 TIA
VANA3_1P8	B4	Analog	1.8	Star connect from the 1.8 V plane. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.8V supply for: Tx3 analog output, Aux synth, Tx3_4 LO, Analog SPI, Rx3 LO buffer, RF PLL2 and LOGEN2, ORx3_4 LO buffers and TxLB3_4 LO buffers, Rx3 TIA, ORx3 mixer, ORx3_4 TIA
VANA4_1P8	N2	Analog	1.8	Star connect from the 1.8 V plane. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.8V supply for: Tx4 analog output, RF synth2, Rx4 LO buffer, Clock PLL and CLKGEN, Clock synth, AUXADC_2, AUXADC_3, Rx4 TIA, ORx4 mixer, Converter2 LDO
VCONV1_1P8	H15	Analog	1.8	Star connect from the 1.8 V plane. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.8V supply for: Tx1,2 DACs, Rx1,2 ADCs, ORx1,2 ADCs
VCONV2_1P8	H3	Analog	1.8	Star connect from the 1.8 V plane. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.8V supply for: Tx3,4 DACs, Rx3,4 ADCs, ORx3,4 ADCs
VJVCO_1P8	P11	Analog	1.8	Star connect from the 1.8 V plane. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.8V supply for JESD VCO/PLL
VANA1_1P3	D15	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.3V supply for: Tx1 phase detector, BBF, Tx2 phase detector, BBF, Rx1, Rx2 TIA, ORx1_2 TIA, Analog SPI

Pin Name	Pin No.	Type	Voltage (V)	Recommended Routing/Notes	Description
VANA2_1P3	D3	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.3V supply for: Tx3 phase detector, BBF, Tx4 phase detector, BBF, Rx3,4 TIA, ORx3_4 TIA, Analog SPI
VCONV1_1P3	J15	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.3V supply for: Tx1,2 DACs, Rx1,2 ADCs, ORx1_2 ADCs
VCONV2_1P3	J3	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary.	1.3V supply for: Tx3,4 DACs, Rx3, Rx4 ADCs, ORx3_4 ADCs
VRFVCO1_1P3	G15	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for: RF VCO1, LOGEN1
VRFVCO2_1P3	G3	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for: RF VCO2, LOGEN2
VRFSYN1_1P3	J13	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for RF1 synth
VRFSYN2_1P3	J5	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for RF1 synth
VAUXVCO_1P3	C12	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for: Aux VCO, Aux LOGEN1_2, Aux LOGEN3_4
VAUXSYN_1P3	C6	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for Aux synth
VCLKSYN_1P3	R7	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for: DEVCLK, Clock synth
VCLKVCO_1P3	N5	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for: Clock VCO, Clock generation, Clock distribution
VRXLO_1P3	A9	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for: Rx1,2 LO mux; Rx3, Rx4 LO mux

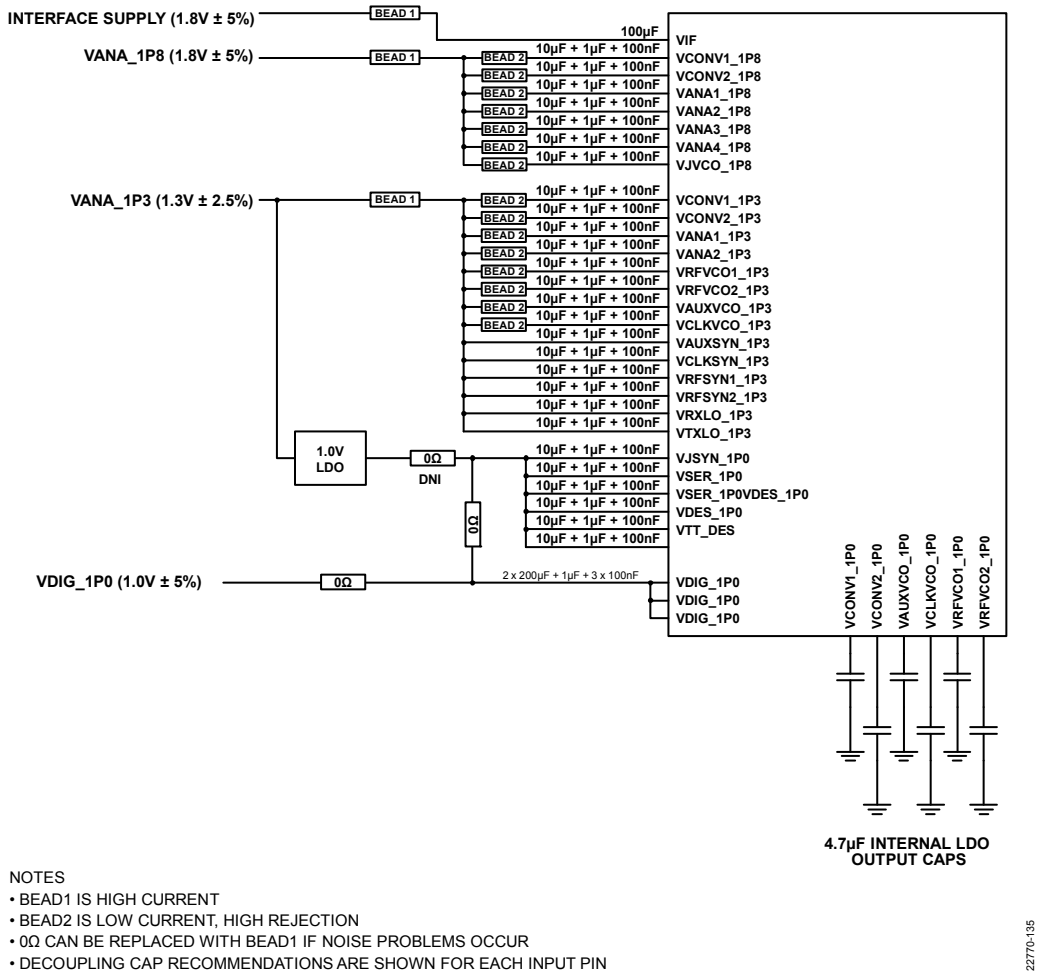
Pin Name	Pin No.	Type	Voltage (V)	Recommended Routing/Notes	Description
VTXLO_1P3	A7	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize trace resistance as much as possible. Isolate from other 1.8 V inputs with a ferrite bead if necessary. Very sensitive to aggressors.	1.3V supply for: Tx1, Tx2 LO mux; Tx3, Tx4 LO mux
VSER_1P0	R3, R4	Analog	1.0	Connect directly to VDIG_1P0 or to a 1.0 V regulator using a separate wide trace to minimize resistance as much as possible. Connect using a ferrite bead if concerned with digital noise.	1.0V supply for JESD serializer
VDES_1P0	P12, P13	Analog	1.0	Connect directly to VDIG_1P0 or to a 1.0 V regulator using a separate wide trace to minimize resistance as much as possible. Connect using a ferrite bead if concerned with digital noise.	1.0V supply for JESD deserializer
VTT_DES	P14	Analog	1.0	Connect directly to VDIG_1P0 or to a 1.0 V regulator. Connect using a ferrite bead if concerned with digital noise.	1.0V supply for JESD deserializer V_{TT}
VJSYN_1P0	R9	Analog	1.0	Connect directly to VDIG_1P0 or to a 1.0 V regulator. Connect using a ferrite bead if concerned with digital noise.	1.0V supply for the JESD synth
VCONV1_1P0	K12	Analog	1.0	Connect a 4.7 µF bypass capacitor to ground.	Bypass connection point for internal converter regulator.
VCONV2_1P0	K3	Analog	1.0	Connect a 4.7 µF bypass capacitor to ground.	Bypass connection point for internal converter regulator.
VAUXVCO_1P0	B11	Analog	1.0	Connect a 4.7 µF bypass capacitor to ground.	Bypass connection point for internal AUXVCO regulator.
VCLKVCO_1P0	P5	Analog	1.0	Connect a 4.7 µF bypass capacitor to ground.	Bypass connection point for internal CLKVCO regulator.
VRFVCO1_1P0	G13	Analog	1.0	Connect a 4.7 µF bypass capacitor to ground.	Bypass connection point for internal RFVCO1 regulator.
VRFVCO2_1P0	G5	Analog	1.0	Connect a 4.7 µF bypass capacitor to ground.	Bypass connection point for internal RFVCO2 regulator.

POWER SUPPLY ARCHITECTURE

The diagram in Figure 135 outlines the power supply configuration used on the CE board. This configuration follows the recommendations outlined in Table 251. This diagram includes the use of ferrite beads for additional RF isolation and 0 Ω resistors. The use of 0 Ω resistors accomplishes three goals.

- Serve as place holders for ferrite beads or other filter devices that may be needed when users encounter RF noise problems in their application and additional isolation is required.
- Ensure that layout follows power routing, forcing traces to be star connected to a central supply.
- Provide a place in the circuit where the current can be monitored and measured for debugging purposes. For this case, the 0 Ω components can be replaced by very low impedance shunt resistors and the voltage measured to determine total current to the specified input ball.

For more details on exact power supply implementation, refer to the ADRV9026 CE board schematic that is supplied by the Analog ADRV9026 design support package with this user guide.



NOTES
• BEAD1 IS HIGH CURRENT
• BEAD2 IS LOW CURRENT, HIGH REJECTION
• 0Ω CAN BE REPLACED WITH BEAD1 IF NOISE PROBLEMS OCCUR
• DECOUPLING CAP RECOMMENDATIONS ARE SHOWN FOR EACH INPUT PIN

Figure 135. Power Supply Connection Diagram

CURRENT CONSUMPTION

Current consumption in each block can vary depending on the device configuration for the profile in use. Clock frequencies, data rates, calibrations, and number of channels in operation all influence the amount of current required for transceiver operation. The following information is a sample of a typical use case profile and the resulting current consumption in different modes. Note that this is a typical example, but do not consider the values maximums for design purposes. Follow the design margins noted previously in this section when sizing power supplies.

Current Measurements: Use Case 26C-Link Sharing Profile

The setup parameters are as follows:

- Tx channels: 4
- Rx channels: 4
- ORx channels: 1
- Device clock: 491.52 MHz
- Tx/Rx primary signal bandwidth: 200 MHz
- Tx/ORx synthesis bandwidth: 450 MHz
- Rx data sample rate: 245.76 MSPS
- Tx/ORx data sample rate: 491.52 MSPS
- JESD lane rate: 16.22016 Gbps

Table 252. Typical Current Consumption—Use Case 26-NLS

				Measured Current (mA)		
Pin Name	**Pins**	**Type**	**Voltage (V)**	**Rx Enabled**	**Tx + ORx Enabled**	**Rx + Tx + ORx Enabled**
VDIG_1P0	G9, J9, L9	Digital	1.0	985	1193	1540
VSER_1P0	R3, R4	Analog	1.0	155	155	156
VDES_1P0	P12, P13	Analog	1.0	416	418	419
VTT_DES	P14	Analog	1.0	4	4	4
VJSYN_1P0	R9	Analog	1.0	8	8	8
VIF	N9	Analog	1.8	5	5	5
VANA1_1P8	N16	Analog	1.8	5	130	130
VANA2_1P8	B14	Analog	1.8	13	131	131
VANA3_1P8	B4	Analog	1.8	8	130	130
VANA4_1P8	N2	Analog	1.8	8	130	130
VCONV1_1P8	H15	Analog	1.8	102	64	141
VCONV2_1P8	H3	Analog	1.8	102	25	102
VJVCO_1P8	P11	Analog	1.8	43	43	43
VANA1_1P3	D15	Analog	1.3	321	359	475
VANA2_1P3	D3	Analog	1.3	317	310	417
VCONV1_1P3	J15	Analog	1.3	377	299	662
VCONV2_1P3	J3	Analog	1.3	372	116	476
VRFVCO1_1P3	G15	Analog	1.3	179	177	179
VRFVCO2_1P3	G3	Analog	1.3	177	176	179
VRFSYN1_1P3	J13	Analog	1.3	10	10	10
VRFSYN2_1P3	J5	Analog	1.3	10	10	10
VAUXVCO_1P3	C12	Analog	1.3	189	214	218
VAUXSYN_1P3	C6	Analog	1.3	7	8	8
VCLKSYN_1P3	R7	Analog	1.3	22	22	22
VCLKVCO_1P3	N5	Analog	1.3	103	103	132
VRXLO_1P3	A9	Analog	1.3	169	19	171
VTXLO_1P3	A7	Analog	1.3	11	184	187

Table 253. Total Current Consumption per Supply Rail

Mode of Operation	**1.8 V Source Current (mA)**	**1.3 V Source Current (mA)**	**1.0 V Source Current (mA)**
Rx Enabled	281	2264	1568
Tx + ORx Enabled	653	2007	1778
Rx + Tx + ORx Enabled	807	3146	2127

PCB LAYOUT CONSIDERATIONS

OVERVIEW

The ADRV9026 is a highly integrated RF agile transceiver with significant signal conditioning integrated onto one chip. Due to the high level of complexity of the device and its high pin count, careful printed circuit board (PCB) layout is important to obtain optimal performance. This document provides a checklist of issues to look for and general guidelines on how to optimize the PCB to mitigate performance issues. The goal of this document is to help achieve the best performance from the ADRV9026 while reducing board layout effort. This section assumes that the reader is an experienced analog/RF engineer who understands RF PCB layout as well as RF and high speed transmission lines.

The ADRV9026 evaluation board represents one of the most complex implementations of the device. All RF inputs and outputs, JESD serial data lanes, and digital control and monitoring signals are implemented in this design. As such, a high level of PCB technology is used to achieve maximum device performance while seeking to maintain a high level of performance in the face of constraints presented by the routing density. Depending on the intended application, users may not require all signals to be routed and can, therefore, use alternate PCB layout techniques to reach their design goals. This includes but is not limited to a traditional BGA fan-out, fewer layers, through hole vias only, and lower grade PCB materials.

This section discusses the following issues and provides guidelines for system designers to get the best performance out of the ADRV9026 device:

- PCB material and stack up selection
- Fan-out and trace-space layout guidelines
- Component placement and routing priorities
- RF and JESD transmission line layout
- Isolation techniques used on the ADRV9026 CE board
- Power management routing considerations
- Analog signal routing recommendations
- Digital signal routing recommendations
- Unused pin instructions

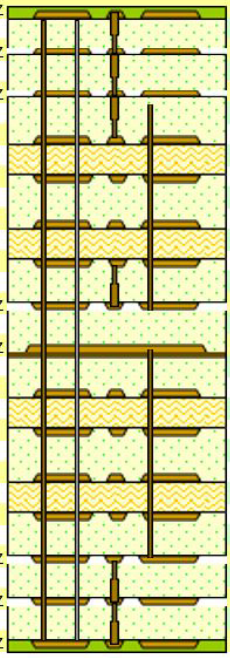
PCB MATERIAL AND STACK UP SELECTION

The ADRV9026 evaluation board utilizes Isola I-Speed dielectric material. It was selected for its low loss tangent and low dielectric constant characteristics. On previous evaluation systems, Analog Devices has chosen a combination of low loss, RF capable dielectric for the outer edge layers and standard FR4-370 HR dielectric for interior layers. RF signal routing on these boards was confined to the top and bottom layers. Therefore, the material mix was a good compromise to obtain optimum RF performance and low overall board cost. Given the need to route RF and high speed digital data lanes on multiple layers due to the increased number of RF channels and JESD lanes, I-Speed material was chosen for all layers on this board. There are several other material options on the market from other PCB material vendors that are also valid options for use with the ADRV9026 device. The key comparison metric for these materials is the dielectric constant and the loss tangent. Designers must also be careful to ensure that the thermal characteristics of the material are adequate to handle high reflow temperatures for short durations and expected operating temperatures for extended durations.

Figure 136 shows the PCB stack up used for the ADRV9026 evaluation board. Layer 1 and Layer 16 are primarily used for RF IO signal routing and I-Speed prepreg material was selected to support the required controlled impedance traces. Layer 2 and Layer 15 have uninterrupted ground copper flood beneath all RF routes on Layer 1 and Layer 16. Layer 2 is also used in combination with Layer 4 to route high speed digital JESD lanes. These signal layers use Layer 3 and Layer 4 as references. Clean reference planes are important to maintain signal integrity on sensitive RF and high speed digital signal paths. Layer 3, Layer 5, and Layer 7 are used to route analog power domains. Routing of analog power planes and traces are discussed in more detail in the power supply layout section. Layer 9 is a solid ground plane used to help isolate sensitive analog signal and power layers from potentially noisy digital signals routed in the lower half of the PCB. Layer 10 through Layer 14 are used to route a variety of digital power, GPIO, and control signals. Table 254 describes the drill table for via structures used in the evaluation board to route all signals from the transceiver. Note that the metal and dielectric thicknesses have been balanced to ensure that the thickness of each half of the PCB is relatively equal to avoid uneven flexing or deforming under pressure or temperature changes.

Via structures were selected based on signal routing requirements and manufacturing constraints. Ground planes are full copper floods with no splits except for vias, through-hole components, and isolation structures. Ground and power planes are all routed to the edge of the PCB with a 10 mil pullback from the edge to decrease the risk of a layer to layer shorts at the exposed board edge.

Layer	Cu Thick. (mils)	Cu Foil wt (oz)	DK	Lam. Thick. (mils)	Description
1	2.15	.375 oz			Foil .375 oz
			3.35	6.15	Prepreg I-Speed 1035(77)/1035(77) 18.25Gx24.25
2	0.55	.375 oz			Foil .375 oz
			3.56	3.20	Prepreg I-Speed 1086(66.5) 18.25Gx24.25
3	1.85	.375 oz			Foil .375 oz
			3.46	4.69	Prepreg I-Speed 1035(71.5)/1035(71.5) 18.25Gx24.25
4	0.60	0.5 oz			
			3.77	4.00	Core I-Speed 4.00mils 3313 0.5 oz / 1 oz VLP2 18.5Gx24.5
5	1.20	1 oz			
			3.50	6.88	Prepreg I-Speed 1078(69.5)/1078(69.5) 18.25Gx24.25
6	0.60	0.5 oz			
			3.77	4.00	Core I-Speed 4.00mils 3313 0.5 oz / 1 oz VLP2 18.5Gx24.5
7	1.20	1 oz			
			3.35	2.88	Prepreg I-Speed 1035(77) 18.25Gx24.25
8	1.85	.375 oz			Foil .375 oz
			3.46	4.25	Prepreg I-Speed 1035(71.5)/1035(71.5) 18.25Gx24.25
9	1.85	.375 oz			Foil .375 oz
			3.35	2.98	Prepreg I-Speed 1035(77) 18.25Gx24.25
10	0.60	0.5 oz			
			3.77	4.00	Core I-Speed 4.00mils 3313 0.5 oz / 0.5 oz VLP2 18.25Gx24.25
11	0.60	0.5 oz			
			3.50	6.96	Prepreg I-Speed 1078(69.5)/1078(69.5) 18.25Gx24.25
12	0.60	0.5 oz			
			3.77	4.00	Core I-Speed 4.00mils 3313 0.5 oz / 1 oz VLP2 18.5Gx24.5
13	1.20	1 oz			
			3.46	4.58	Prepreg I-Speed 1035(71.5)/1035(71.5) 18.25Gx24.25
14	1.85	.375 oz			Foil .375 oz
			3.56	3.20	Prepreg I-Speed 1086(66.5) 18.25Gx24.25
15	0.55	.375 oz			Foil .375 oz
			3.35	6.16	Prepreg I-Speed 1035(77)/1035(77) 18.25Gx24.25
16	2.15	.375 oz			Foil .375 oz



22770-136

Figure 136. PCB Material Stack Up Diagram

Table 254. Drill Table

Start Layer	End Layer	Drill Type	Plate Type	Via Fill	Drill Size (min)	Drill Depth	Pad Size(min)	Stacked Vias
1	16	Mech	PTH	Not applicable	45.30	83.35		
3	8	Mech	Via	Resin fill	11.80	26.96		
9	14	Mech	Via	Resin fill	11.80	26.42		
1	16	Mech	Via	Nonconductive via fill	7.90	83.35		
15	14	Laser	Microvia	CuVF_Button pattern	7.90	3.65		Y
16	15	Laser	Microvia	Nonconductive via fill	7.90	6.61		Y
8	7	Laser	Microvia	CuVF_Button pattern	11.80	3.33		Y
1	2	Laser	Microvia	Non-Conductive via fill	7.90	6.60		Y
2	3	Laser	Via	CuVF_Button pattern	7.90	3.65		Y
3	4	Laser	Microvia	CuVF_Button pattern	7.90	5.14		Y

Controlled impedance traces, single ended and differential, are required to obtain best RF performance. Impedances of 50 Ω and 100 Ω are required for RF, high speed digital, and clock signals. Table 255 describes details about trace impedance controls used in the ADRV9026 evaluation board and types of line structures used to obtain desired impedance and performance on and for given layers and impedances.

Table 255. Impedance Table

Layer	Structure Type	Target Impedance (Ω)	Impedance Tolerance (Ω)	Target Line Width (mils)	Edge Coupled Pitch (mils)	Reference Layers	Modeled Line Width (mils)	Modeled Impedance (Ω)	Coplanar Space (mils)
1	Single-ended	50.00	±5	11.00	0.00	(2)	11.50	51.48	9.75
1	Edge coupled differential	50.00	±5	27.00	32.00	(2)	27.50	50.99	9.75
1	Edge coupled differential	100.00	±10	7.50	14.50	(2)	8.25	102.70	9.62
2	Edge coupled differential	100.00	±10	4.25	12.00	(1, 3)	4.10	102.17	12.07
4	Edge coupled differential	100.00	±10	3.75	9.50	(3, 5)	3.75	100.60	12.00
4	Single-ended	50.00	±5	4.50	0.00	(3, 5)	4.25	50.00	12.13
12	Single-ended	50.00	±5	4.50	0.00	(11, 13)	4.75	51.77	11.88
12	Edge coupled differential	100.00	±10	4.00	9.00	(11, 13)	4.00	101.61	12.00
16	Single-Ended	50.00	±5	11.00	0.00	(15)	11.50	51.51	9.75
16	Edge coupled differential	100.00	±10	7.50	14.50	(15)	8.25	102.71	9.62

FANOUT AND TRACE SPACING GUIDELINES

The ADRV9026 uses a 289-ball BGA 14 mm × 14 mm package. The pitch between the pins is 0.8 mm. This small pitch makes it impractical to route all signals on a single layer. RF and high speed data pins have been placed on the perimeter rows of the BGA to minimize complexity of routing of these critical signals. Via in pad technology is used to escape all other signals to layers on which they are routed. The recommended via size includes an 8 mil drill hole with a 12 mil capture pad. A combination of stacked micro vias, buried vias, and through vias are used to route signals to appropriate inner layers for further routing. JESD interface signals are routed on two inner signal layers utilizing controlled impedance traces.

Figure 137 illustrates the fanout of RF differential channels from the device on the top layer of the PCB. Note that each signal pair is designed with the required characteristic impedance and isolation to minimize crosstalk between channels. The isolation structures include a series of ground balls around each RF channel and the digital interface section of the device. Connect these ground balls by traces to form a wall around each section, and then fill the area to make the ground as continuous as possible underneath the device.

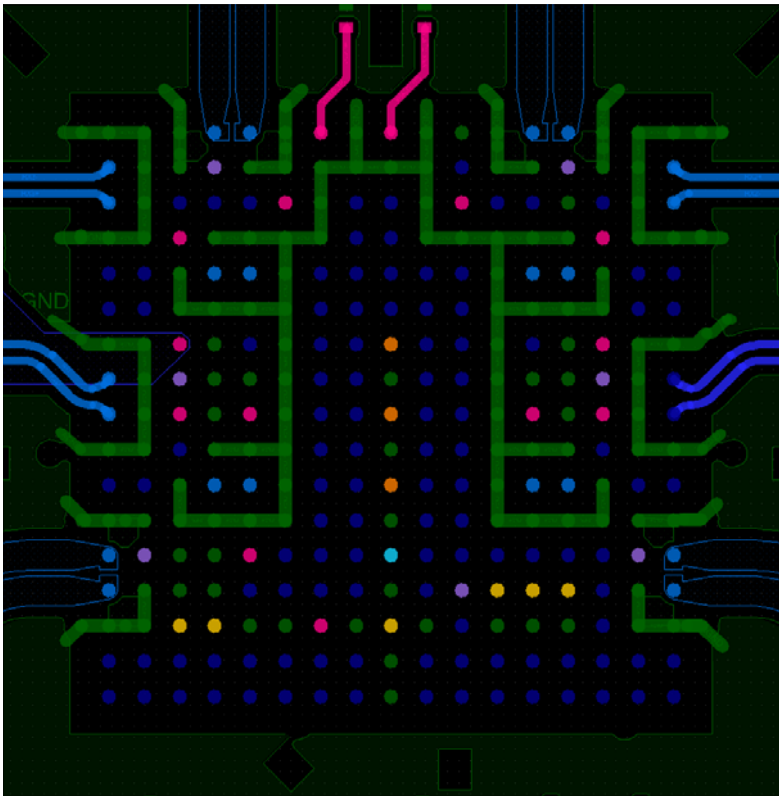


Figure 137. ADRV9026 CE Board RF Receiver and Transmitter Fanout and Layout

COMPONENT PLACEMENT AND ROUTING GUIDELINES

The ADRV9026 transceiver requires few external components to function. Those that are required must be carefully placed and routed to optimize performance. This section provides a checklist for properly placing and routing some of those critical signals and components.

Signals with Highest Routing Priority

RF inputs and outputs, clocks, and high speed digital signals are the most critical for optimizing performance and must be routed with the highest priority. Figure 138 shows the general directions in which each of the signals must be routed so that they can be effectively isolated from aggressor signals. It may be difficult to keep all RF channels on a single outer layer. In such cases, it is recommended to route the receiver and transmitter channels on the top PCB layer with adequate channel-to-channel isolation and the observation receivers on internal layers or on the bottom layers. Ensure that the trace impedance is properly designed to 100 Ω differential including the vias needed to transfer the signals between PCB layers.

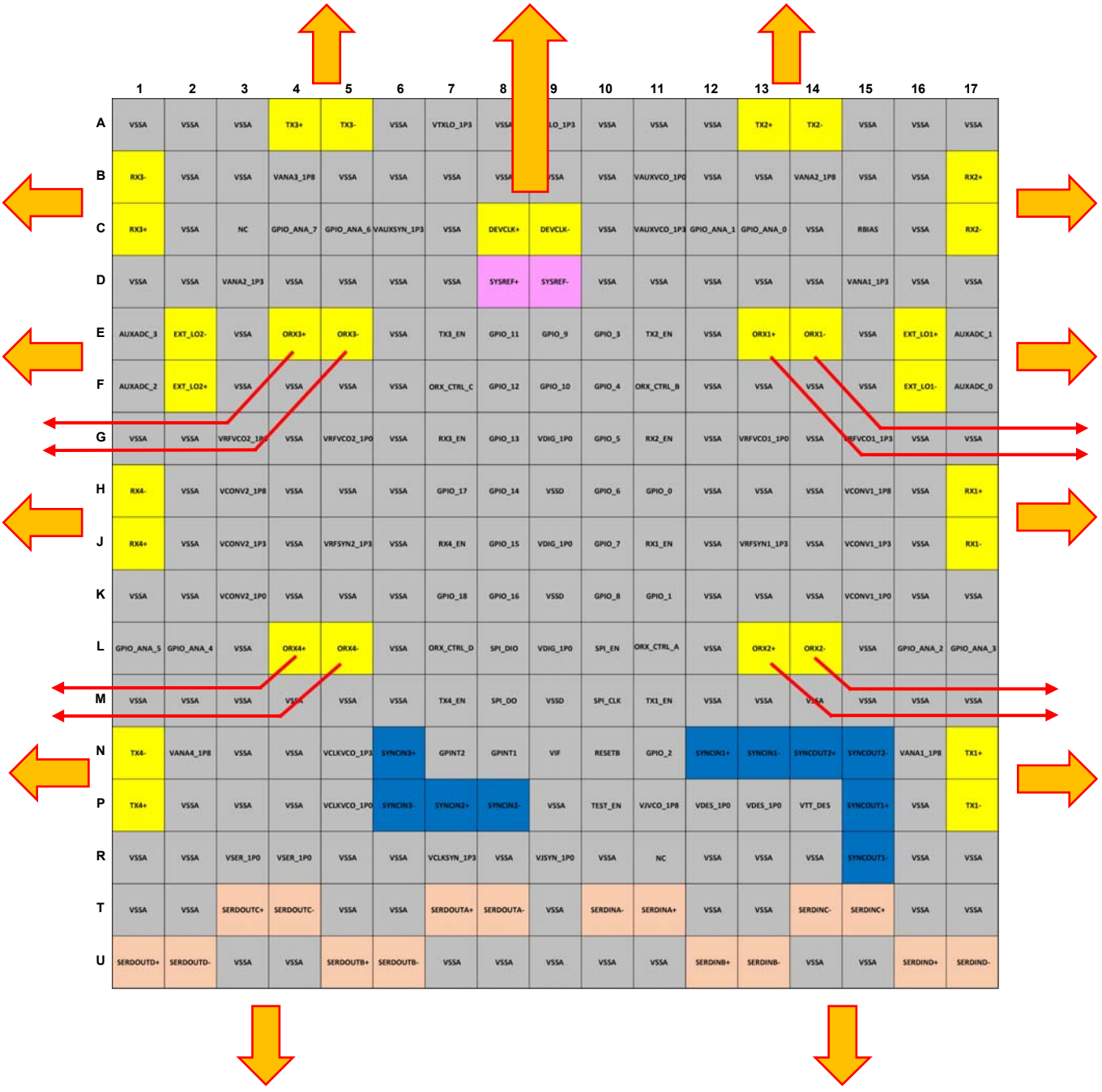


Figure 138. RF IO, DEVCLK, EXT LO, and JESD204B Routing Guidelines

Tx, Rx, and ORx routing (also referred to as trace routing), physical design (trace width/spacing), matching network design, and balun placement significantly impact RF transceiver performance. Make every effort to optimize path design, component selection, and placement to avoid performance degradation. The RF Routing Guidelines section describes proper matching circuit placement and routing in greater detail. Additional related information can be found in the RF Port Interface Overview section.

To achieve desire levels of isolation between RF signal paths, use the considerations and techniques described in the Isolation Techniques section in designs.

For RF Tx outputs, install a 10 µF capacitor near the Tx balun(s) VANAx_1P8 dc feed(s). This capacitor acts as a reservoir for the Tx supply current. The Tx Bias Supply Guidelines section discusses Tx dc supply design in detail.

Connect external clock inputs to DEVCLK+ and DEVCLK− through ac coupling capacitors. Place a 100 Ω termination across the input near Pin C8 and Pin C9, as shown in Figure 139. Shield traces by ground planes above and below with vias staggered along the edges of the differential pair routing. This shielding is important because it protects the reference clock inputs from spurious signals that can transfer to different clock domains within the device. Refer to the Synthesizer Configuration section for more details regarding the clock signals.

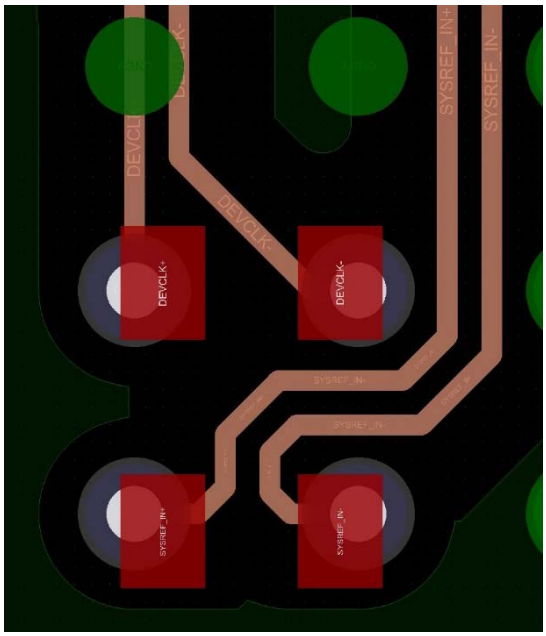


Figure 139. DEVCLK and SYSREF Termination

Route JESD204B high speed digital interface traces at the beginning of the PCB design process with the same priority as the RF signals. The JESD204B/JESD204C Routing Recommendations section outlines launch and routing guidelines for these signals. Provide adequate isolation between interface differential pairs.

If an external LO source is used, connect to the port through ac coupling capacitors. EXT_LO1± (E16, F16) and EXT_LO2± (E2, F2) pins are internally dc biased. An on-chip 100 Ω termination is provided.

Signals with Second Routing Priority

Power supply routing and quality has a direct impact on overall system performance. The Power Management Layout section provides recommendations for how to best route power supplies to minimize loss as well as interference between RF channels. Follow recommendations provided in this section to ensure optimal RF and isolation performance.

Signals with Lowest Routing Priority

Route remaining low frequency digital inputs and outputs, auxiliary ADCs and DACs, and SPI signals. It is important to route all digital signals bounded between rows E and R and Column 6 and Column 15 down and away from sensitive analog signals on PCB signal layers with a solid ground layer shielding other sensitive signals from the potentially noisy digital signals (refer to Figure 138 for the ball diagram). The ADRV9026 CE board uses Layer 9 as a solid ground flood on the entire layer to act as a shield and delineation between analog and digital domains. All RF, analog power, and high speed signaling is routed on Layer 1 through Layer 8 and Layer 16, while digital power and signaling is routed on Layer 10 through Layer 15. Auxiliary ADC and DAC signal traces are routed on layers separated from RF IO and high speed digital, but still on the analog side of the PCB.

RF AND JESD TRANSMISSION LINE LAYOUT

RF Routing Guidelines

The ADRV9026 evaluation boards use both surface coplanar waveguide and surface edge coupled coplanar waveguide transmission lines for Tx, Rx, and ORx RF signals. In general, Analog Devices does not recommend using vias to route RF traces unless a direct route on the same layer as the device is not possible. Keep balanced lines for differential mode signaling used between the device and the RF balun as short as possible. Keep the length of the single ended transmissions lines for RF signals as short as possible. Keeping signal paths as short as possible reduce susceptibility to undesired signal coupling and reduce the effects of parasitic capacitance, inductance, and loss on the transfer function of the transmission line and impedance matching network system. The routing of these signal paths is the most critical factor in optimizing performance and, therefore, must be routed prior to any other signals and maintain the highest priority in the PCB layout process.

All 12 RF ports are impedance matched using PI matching networks, both differential and single ended. Take care in the design of impedance matching networks including balun, matching components, and ac coupling capacitor selection. Additionally, external LO ports and DEVCLK may require impedance matching to ensure optimal performance. Figure 140 depicts the path from device to external connector that is used to route Tx4 on the CE board. Component placement for matching components are highlighted in red. Refer to the RF Port Interface Overview section for more information on RF impedance matching recommendations.

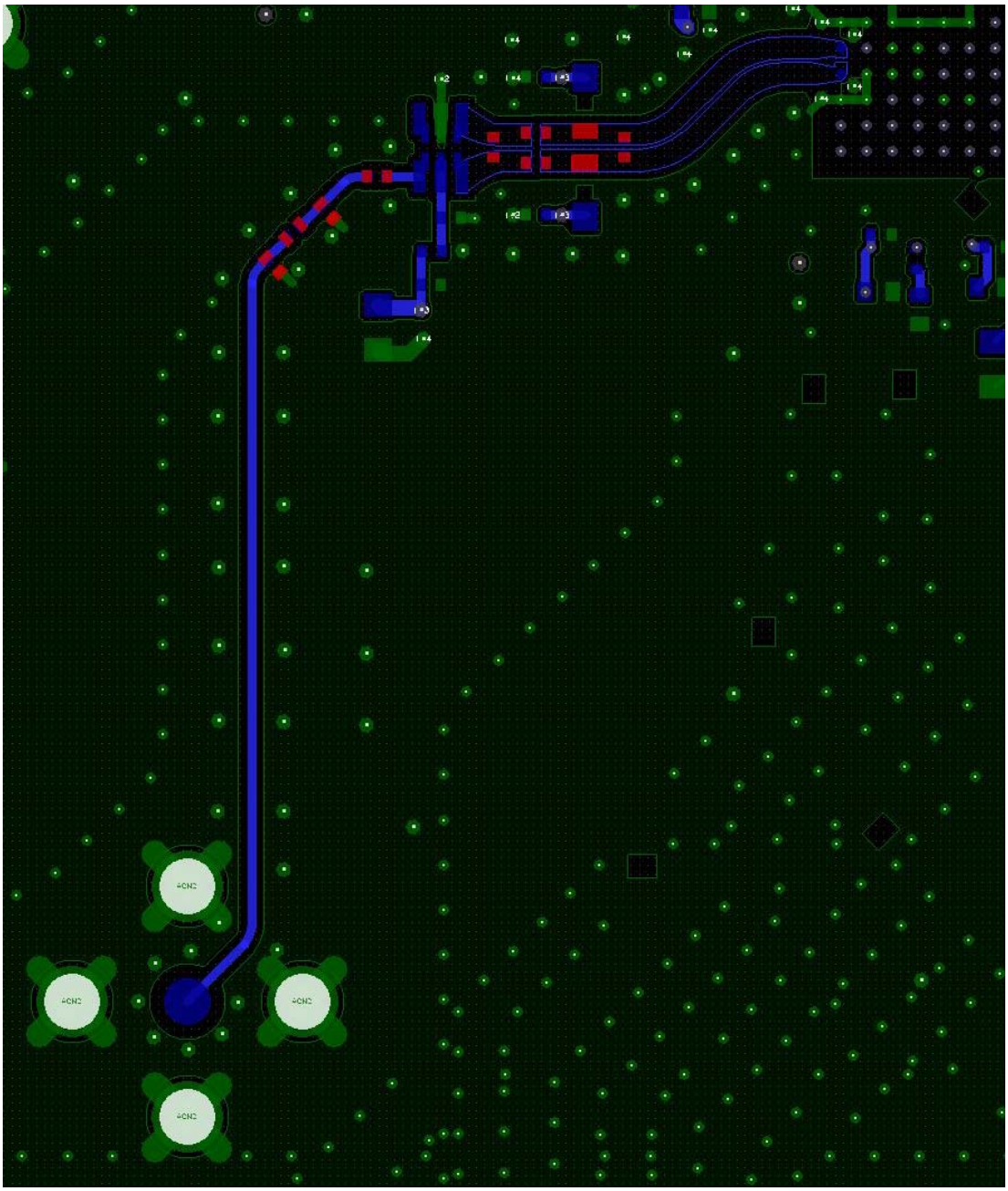


Figure 140. Tx RF Routing and Matching Network

All the RF signals must have a solid ground reference under each path to maintain the desired impedance. Ensure that none of the critical traces run over a discontinuity in the ground reference.

Tx Bias Supply Guidelines

Each transmitter requires approximately 125 mA supplied through an external connection. In the ADRV9026 CE board, bias voltages are supplied at the dc feed of a center tapped balun in the RF signal path as shown in Figure 141.

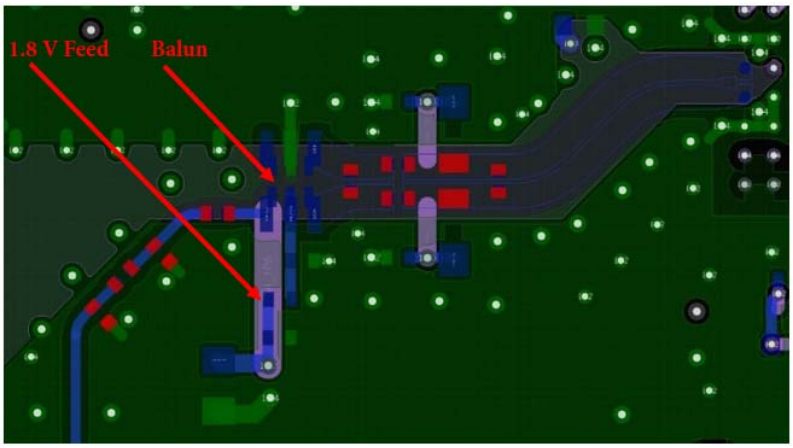


Figure 141. 1.8V TX Bias Routing at Balun

To reduce switching transients due to attenuation setting changes, power the balun dc feed directly from the 1.8 V supply plane. Design the geometry of the plane to isolate each transmitter from the others. Figure 142 shows the 1.8 V supply distribution on the ADRV9026 CE board. The primary 1.8 V distribution is through a plane that transitions to two wide fingers on Layer 5, which run up both sides of the device. The finger width is designed to minimize voltage drop at the tap points. Each transmitter is biased with a finger on layer 3 that taps the main 1.8 V supply. The fingers are designed and routed to present a low impedance at the connection point to the Tx input.

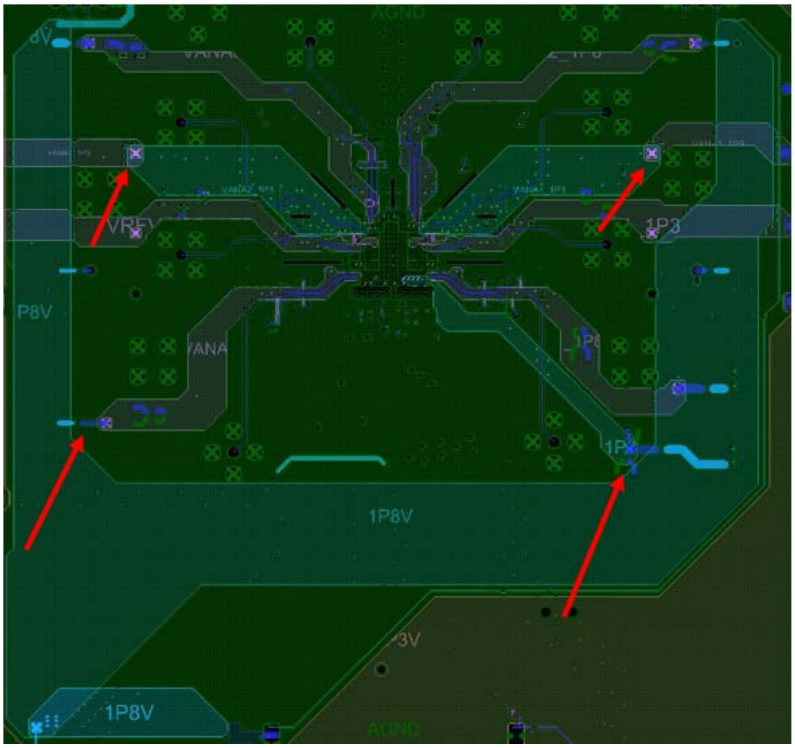


Figure 142. 1.8 V Supply Distribution

As previously mentioned, the ADRV9026 evaluation board couples the supply into the transmitter via a center tapped balun, but it is also provisioned for an external choke feed inductor with an ac decoupling capacitor. This topology helps in improving transmitter-to-transmitter isolation.

When a balun is selected that does not have a dc feed capability, RF chokes must be used to supply current to the transmitters. Chokes are connected from the 1.8 V supply to each Tx output. Note that in this scenario, the Tx balun must be ac-coupled. The RF chokes must also

be decoupled by capacitors from the power feed to ground. Place the ground connections to these capacitors as close as possible to the Tx output pins. Take care to match both chokes and their layout to avoid peaking due to current transients.

JESD204B/JESD204C Routing Recommendations

The ADRV9026 uses a JESD204B/JESD204C high speed serial interface. To ensure performance of this interface, keep the differential traces as short as possible by placing the device as close as possible to the baseband processor and routing the traces as directly as possible between the devices. Using a PCB material with a low dielectric constant and loss tangent is also strongly recommended. For a specific application, loss must be modeled to ensure adequate drive strength is available in both the ADRV9026 and the baseband processor.

Route the differential pairs on a single plane using a solid ground plane as a reference on the layers directly above and/or below the signal layer. Reference planes for the impedance controlled traces must not be segmented or broken along the entire length of a trace.

All JESD lane traces must be impedance controlled, targeting 100 Ω differential. Ensure that the pair is loosely coplanar edge-coupled. The ADRV9026 CE board uses 4 mil wide traces and a separation of approximately 10 mil. This varies depending on the stack up and selected dielectric material. Minimize the pad area for all the connector and passive components to reduce parasitic capacitance effects on the transmission lines, which can negatively impact signal integrity. Via use to route these signals must be minimized as much as possible. Use blind vias wherever possible to eliminate via stub effects and use micro vias to minimize inductance. If using standard vias, use maximum length vias to minimize the stub size. For example, on an 8-layer board, use Layer 7 for the stripline pair, thus reducing the stub length of the via to that of the height of a single layer. For each via pair, a pair of ground vias must be placed nearby to minimize the impedance discontinuity.

For JESD signal traces, the recommendation is to route them on the top side of the board as a 100 Ω differential pair (coplanar edge coupled waveguide). In the case of the ADRV9026 CE board, the JESD signals are routed on inner Layer 2 and Layer 4. To minimize coupling, these signals are placed on an inner layer using a via in pad of the component footprint. ac coupling capacitors (100 nF) are places in series near the FMC connector away from the chip. The JESD interface can operate at frequencies up to 16 GHz.

Figure 143 and Figure 144 show the transition between ball and launch. Surrounding ground references, above and below the signal layer are designed to tune the modal impedances ideal for the high speed signaling and according to the JESD204B standard.

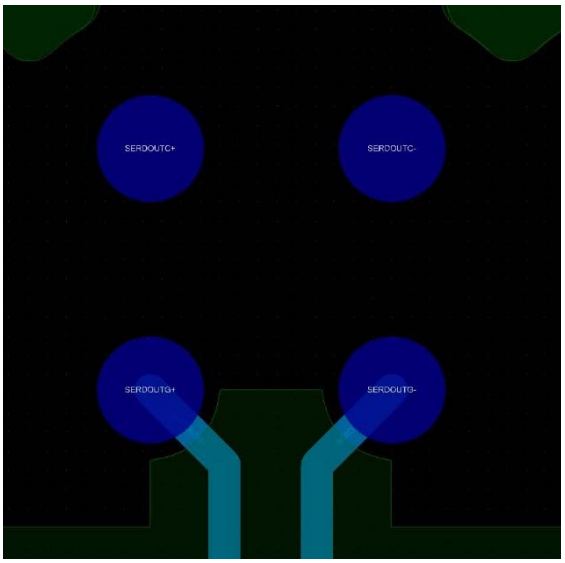


Figure 143. JESD Signal Launch on Layer 2

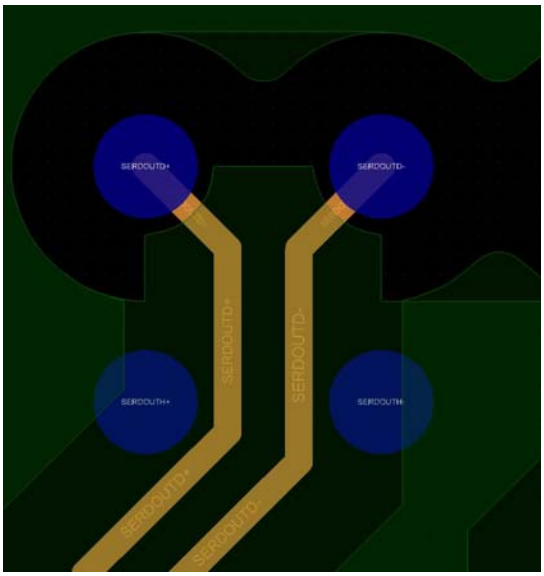


Figure 144. JESD Signal Launch on Layer 4

ISOLATION TECHNIQUES

Given the density of sensitive and critical signals, significant isolation challenges are faced when designing a PCB for the ADRV9026 device. Isolation requirements listed below were followed to accurately evaluate the ADRV9026 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 256 lists the isolation targets for each RF channel-to-channel combination type. To meet these goals with significant margin, isolation structures were designed into the ADRV9026 CE board.

Table 256. Port to Port Isolation Goals

Port	650 MHz to 4 GHz	4 GHz to 6 GHz
Tx to Tx	65 dB	60 dB
Tx to Rx	70 dB	65 dB
Tx to ORx	70 dB	65 dB
Rx to Rx	65 dB	60 dB
Rx to ORx	70 dB	65 dB

Isolation Between RF IO Ports

These are the primary coupling mechanisms between RF IO paths on the evaluation board:

- Magnetic field coupling
- Surface propagation
- Cross domain coupling via ground

To reduce the impact of these coupling mechanisms on the ADRV9026 CE board, several strategies were used. Large slots are opened in the ground plane between RF IO paths. These discontinuities prevent surface propagation. A careful designer may notice various bends in the routing of differential paths. These routes were developed and tuned through iterative electromagnetic simulation to minimize magnetic field coupling between differential paths. These techniques are illustrated in Figure 145.

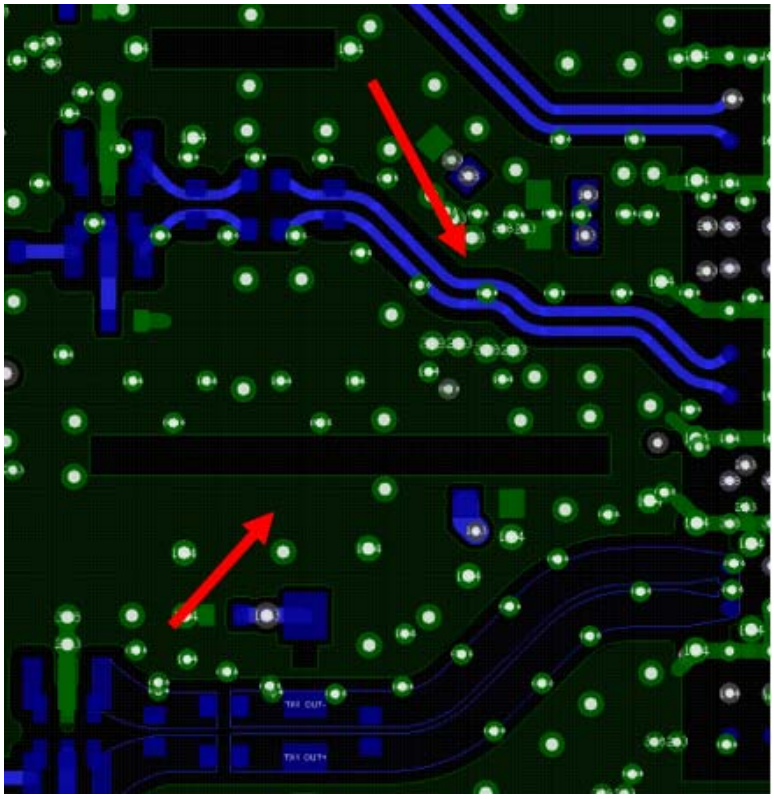


Figure 145. RF IO Isolation Structures

Additional shielding is provided by using connecting VSSA balls under the device to form a shield around RF IO ball pairs. This ground provides a termination for stray electric fields. Figure 146 shows how this is done for Tx1. The same is done for each set of sensitive RF IO ports. Ground vias are used along single ended RF IO traces. Optimal via spacing is 1/10 of a wavelength for the highest signal frequency, but that spacing can vary somewhat due to practical layout considerations.

$$wavelength\,(m) = \frac{300}{frequency\,(MHz) \times \sqrt{\varepsilon_r}}$$

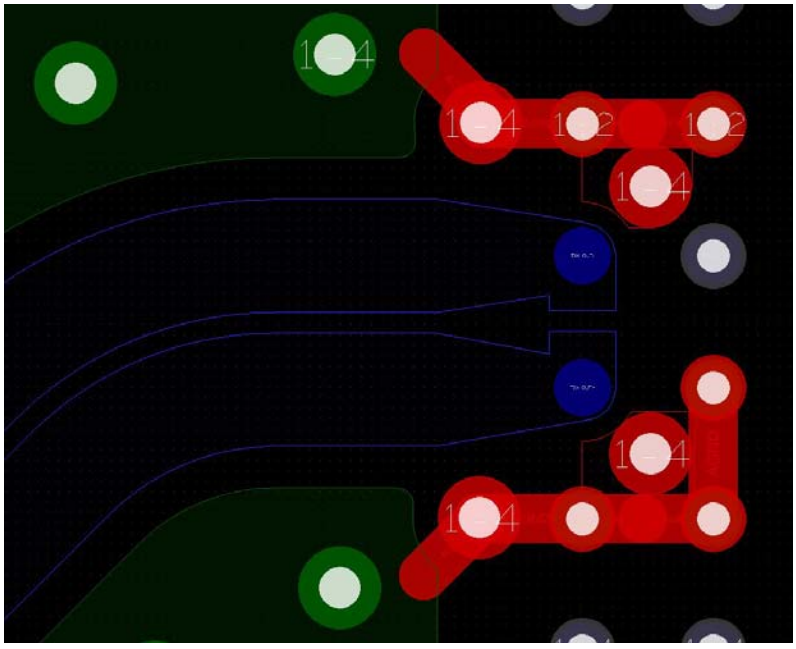


Figure 146. Shielding of TX Launches

RF IO baluns are spaced and aligned to reduce magnetic coupling from the structures in the balun package. Care must also be taken to reduce cross talk over shared grounds between baluns. Another precaution taken involved placing and orienting SMA connectors to minimize connector to connector coupling between ports.

Isolation Between JESD204B Lines

The JESD204B interface uses 16 lane pairs that can operate at speeds up to 16 GHz. Take care when doing PCB layout to make sure those lines are routed following rules described in the JESD204B/JESD204C Routing Recommendations section. In addition, use isolation techniques to prevent crosstalk between the different JESD204B lane pairs. Via fencing is the primary technique used on the ADRV9026 CE board.

Figure 147 illustrates this technique. Ground vias are placed along and between each pair of traces to provide isolation and decrease crosstalk. Spacing between vias, marked as A in Figure 147 follows the rule provided in the equation below. For most accurate spacing of fencing vias, use layout simulation software.

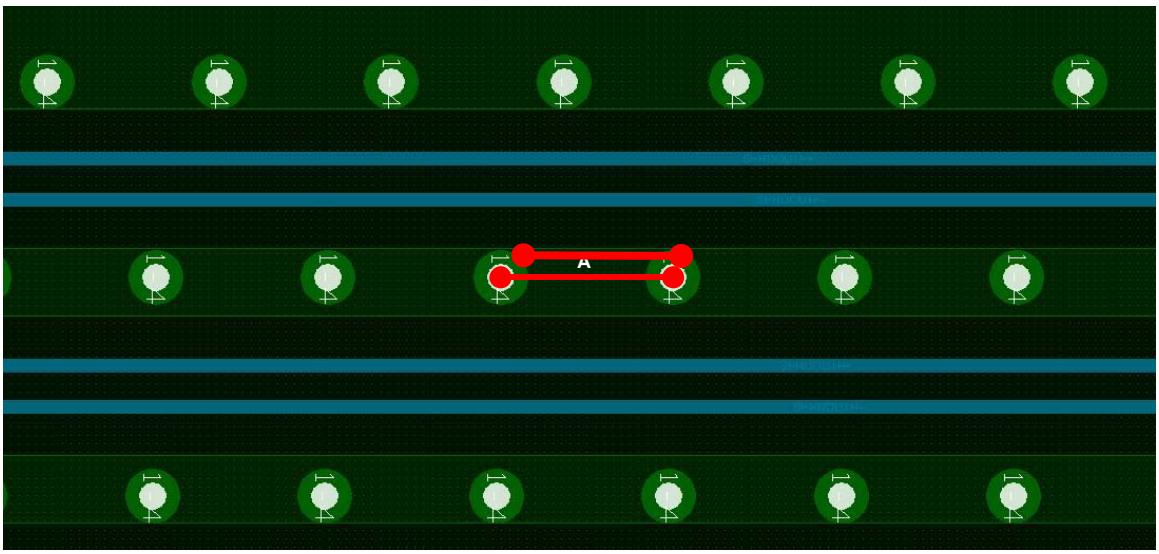


Figure 147. JESD204B Lane Via Fencing

POWER MANAGEMENT LAYOUT DESIGN

Due to the complexity and high level of integration in the ADRV9026, power supply routing is critical to achieve optimum RF performance. The device is designed to minimize power supply coupled noise by implementing several internal linear regulators that isolate circuits from each other when connected to a common power supply rail. This provides an improved level of isolation compared to previous products, but it is only one level of protection. Proper power supply layout can also help isolate individual circuits in the device.

Analog Power Ring Approach

The RF section is designed as two hemispheres with two transmitters, two receivers, and as many as two observation receivers on each side. To reduce coupling between channels and keep each power supply input isolated from others, a star connection approach is used. This approach involves connecting each power supply input to a common power supply bus using an isolated trace designed specifically for the current requirements of the particular input. The ADRV9026 evaluation board uses a power ring approach to provide the power supply bus for the 1.8 V and 1.3 V analog supplies. The 1.8 V supply is routed as an inner ring to provide a more direct connection to the 1.8 V transmitter output supplies and the 1.3 V supply is routed in a similar fashion as an outer ring that can be star-connected to each 1.3 V supply on the device. Figure 148 shows this layout approach on the ADRV9026 CE board. The inner purple “u-shape” is the 1.8 V supply and the outer pink shape is the 1.3 V supply. Note that neither shape forms a complete ring. This was done to better control the current path for each supply and avoid current loops and coupling between the two hemispheres of the board. Also, there is a thin strip of ground plane that is routed between the two supply rings, maintaining some separation and prevention of direct coupling on the same layer.

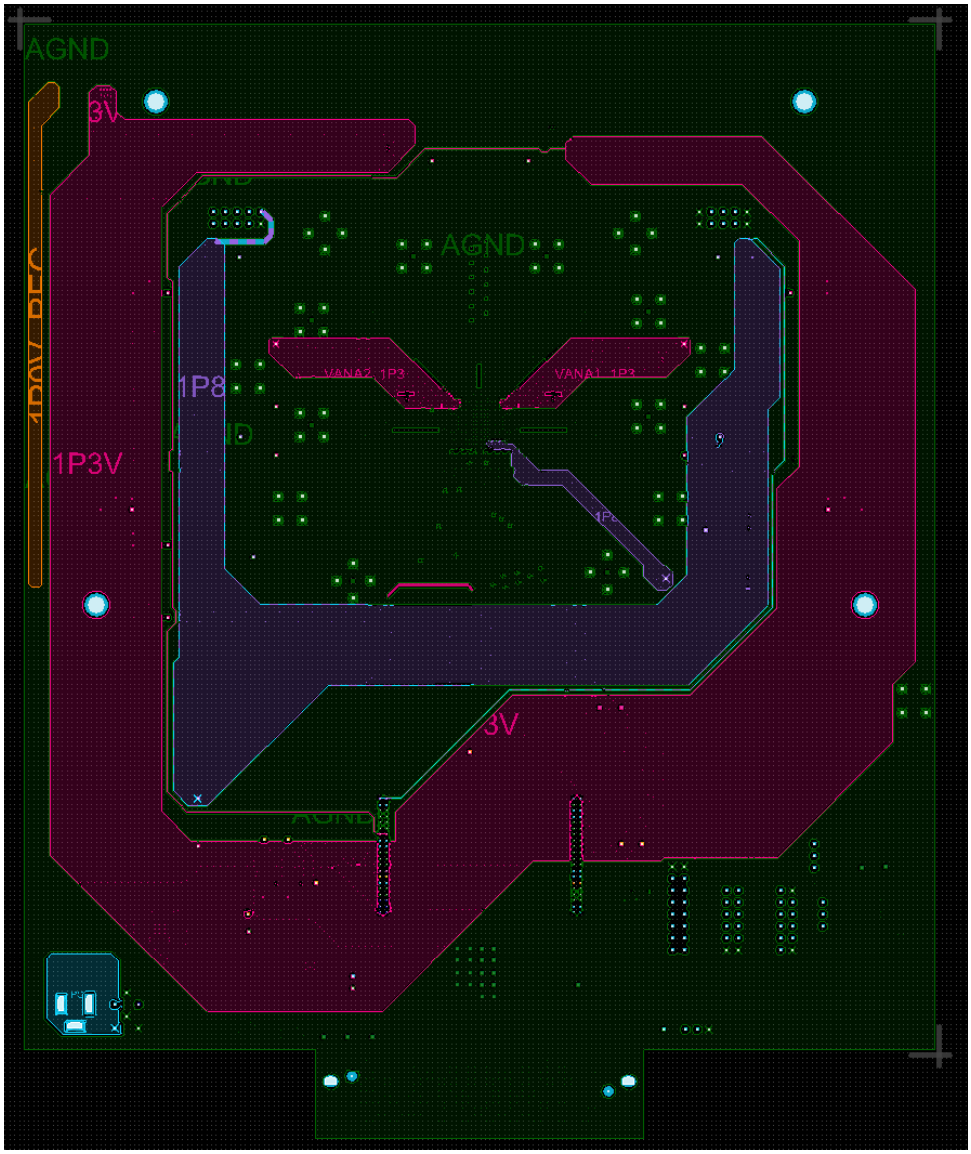


Figure 148. Analog Power Ring Layout Approach

Analog Power Star Connections

The analog power ring approach provides ample locations for the individual star connections to be made. This approach enables the designer to control the current paths for each supply as well as design individual traces that better control the effect of voltage drops on other circuits when large load current changes occur. Each individual power supply input is evaluated for its maximum current consumption value, and the star connection trace is then designed to minimize the voltage drop for that particular supply input while still providing isolation from the other inputs. Figure 149 and Figure 150 illustrate how these star connections are made to the individual supply balls of the device. Some of the connections are made directly to the corresponding supply ring, some are made through a ferrite bead or similar filter device. Note that the thickness and layer of each trace was determined to minimize voltage drops and maximize isolation between aggressor and victim inputs. The layers with the thicker metal in the stackup drawing are used for the inputs with the highest current consumption values.

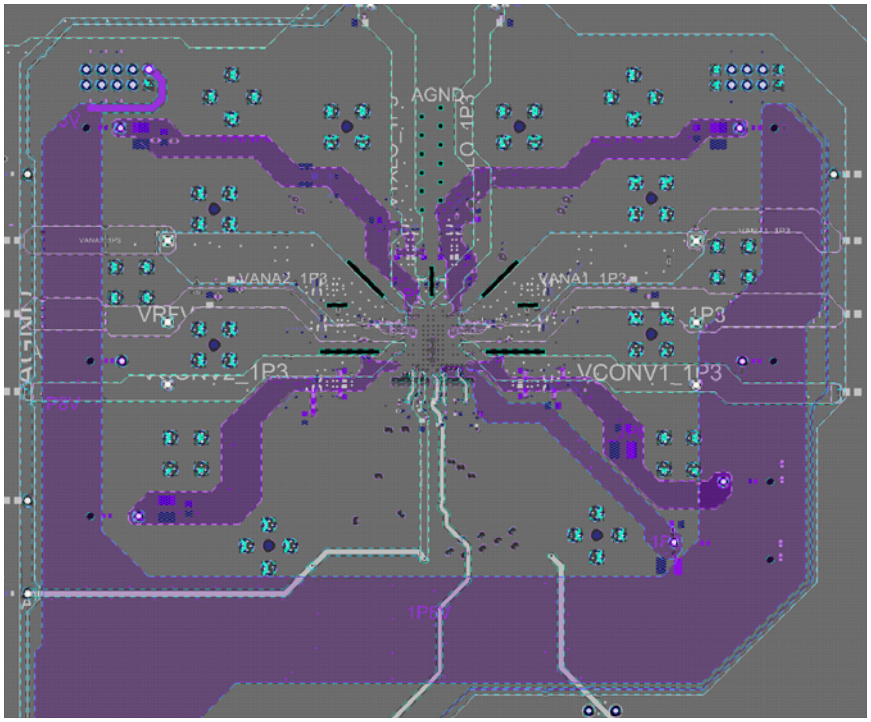


Figure 149. 1.8 V Supply Routing Using Star Connections

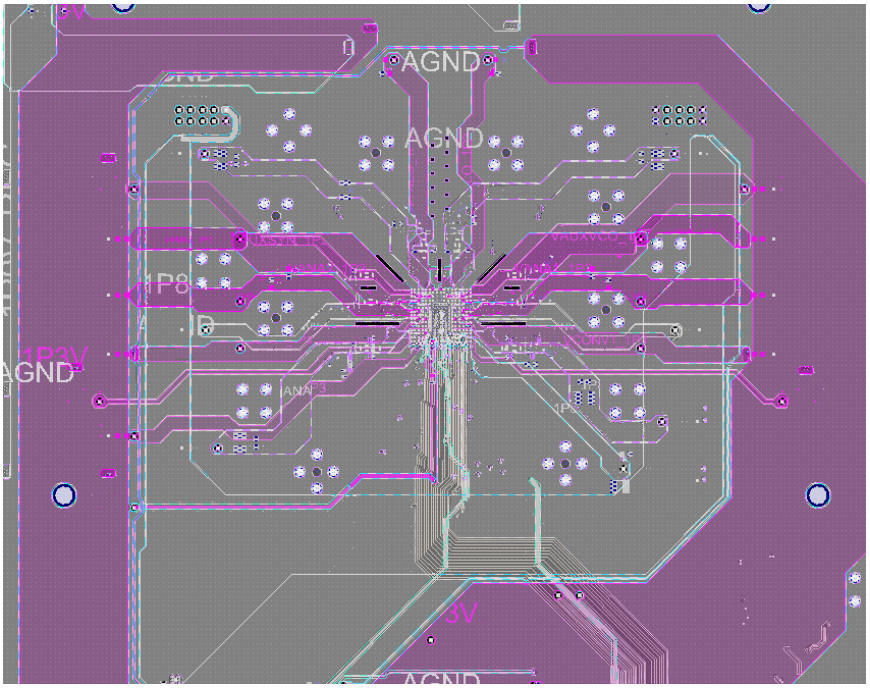


Figure 150. 1.3 V Supply Routing Using Star Connections

Digital Power Routing

The digital 1.0 V supply is the noisiest supply in the system, so it is important to keep this supply shielded from the other supplies. It is also the highest current supply, so the thickness of the traces needs to be adequate to carry the load current to the device without experiencing significant voltage drops. There are three digital power input pins to the device, so the routing into the device is also critical. Each input that connects to an input pin must match the others in length and thickness so there is no additional voltage drop in one connection compared to the others. Figure 151 illustrates the approach used on the CE board to supply this current. A digital power channel is routed from the power supply to the device and the entire area is flooded with copper to provide a low resistance supply trace. This channel is shielded on all sides so that it is isolated from other signals. Figure 152 shows a zoomed-in view of the connection to the device. Note that all three connections are made using two traces to reduce the trace resistance. Each connection is equal in total copper volume to the others, so their voltage drops are equal when the device is active.

Figure 151. Digital Supply Routing

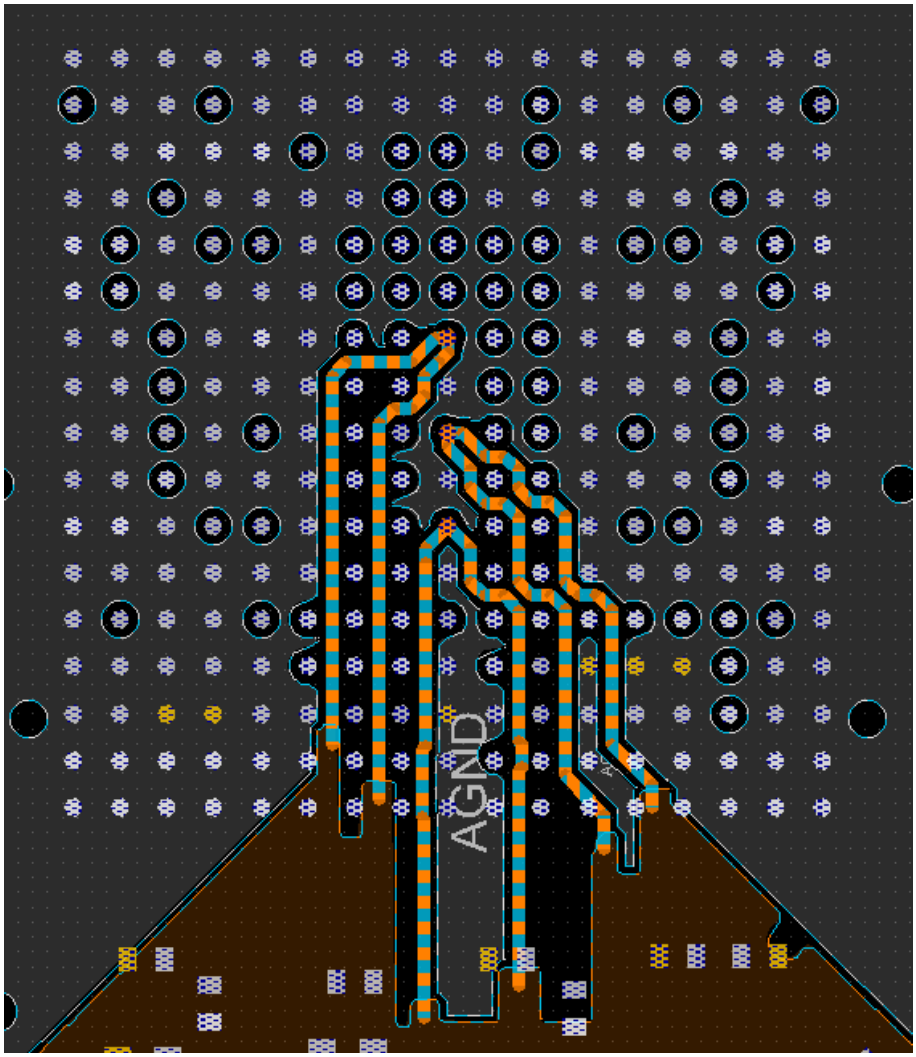


Figure 152. Digital Supply Connection to Three VDIG Input Pins of the Device

JESD 1.0 V Supply Inputs

After careful evaluation, it was determined that the 1.0 V supply needed for the JESD interface can be supplied directly from the 1.0 V digital supply without any interference or noise problems. The CE boards have these supplies routed separately from the common 1.0 V supply shared by VDIG_1P0 as traces using a similar star connection approach that was used by the analog 1.3 V and 1.8 V supplies. Note that the serializer and deserializer supply inputs carry the majority of the current, so these traces are made by creating filled areas as wide as possible to minimize voltage drop. The VTT_DES and VJSYN_1P0 traces carry vary little current and are, therefore, routed using standard traces. Figure 153 illustrates how these traces are routed to the device. Note that the VTT_DES and VJSYN_1P0 traces were routed on a different PCB layer than the VSER_1P0 and VDES_1P0 supplies.

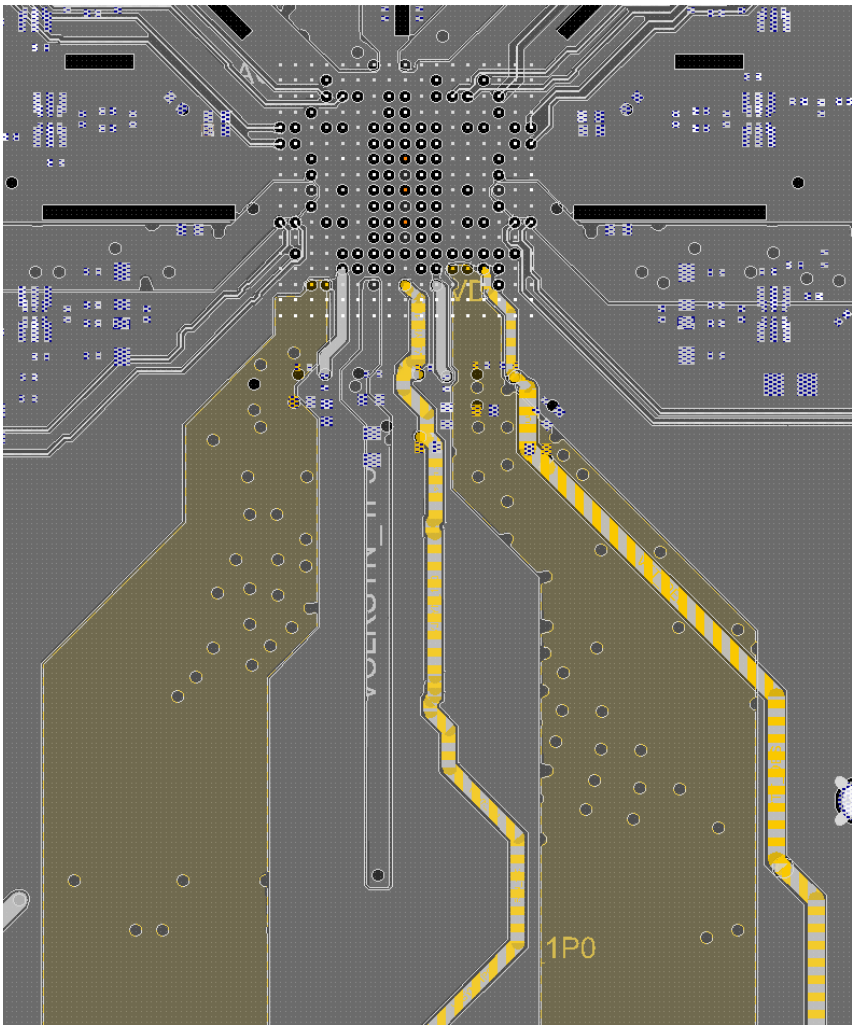


Figure 153. SERDES Power Supply Input Routing

Interface Supply Input

The interface supply is a low current input that provides the supply reference for the SPI serial interface. It can be routed as a signal trace with adequate thickness to minimize voltage drop when the device is active. Route this trace in the digital area like the VDIG_1P0 supply and keep it isolated from other signals to ensure it is not corrupted by other active digital signals or by the JESD interface lanes.

Ground Returns

Another critical routing consideration is how to control the mixing of ground currents to avoid noise coupling between different power domains. One way to keep domains separated is to provide different ground return planes for each supply domain. This approach can complicate a dense PCB layout like what is required for the ADRV9026. Another option is to connect all ground to the same plane system and use cutouts and channeling like those used in the RF sections to provide better channel to channel isolation. Creating such ground channels can provide the benefit of steering ground currents in a desired path without the complexity of trying to keep ground planes isolated from each other. The specifics of such designs are highly dependent on the PCB layout and the level of isolation is desired.

Input Bypass Component Placement

There are subtle component placement techniques for placing power supply bypass components that can have a substantial impact on radio performance. Follow these guidelines when placing components on power supply inputs:

- Each power supply pin requires 0.1 µF bypass capacitor near the pin at a minimum. For inputs that require a large current step, a 10 µF capacitor in parallel is recommended. Place the ground side of the bypass capacitor(s) so that ground currents flow away from other power pins and their bypass capacitors.
- Route power supply input traces to the bypass capacitor and the connect capacitor(s) as close to the supply pin as possible through a via to the component side of the PCB. If possible, it is recommended that the via be located inside the power supply pin pad to minimize trace inductance.
- Some power supplies require a ferrite bead in series with the supply line to prevent RF noise from coupling between different inputs, while others can do without the extra protection. It is recommended that each line be connected with a series component – either a ferrite bead or a 0 Ω place holder. Ensure that the device is sized properly to handle the current load for the particular power supply input of concern.
- Figure 154 and Figure 155 illustrate an example of how the power supply routing from the common power ring to a bypass capacitor and into the ADRV9026 device is implemented. Note that the bypass capacitor is connected directly to the vias leading from the bottom of the PCB to the ball pads on the top of the PCB.

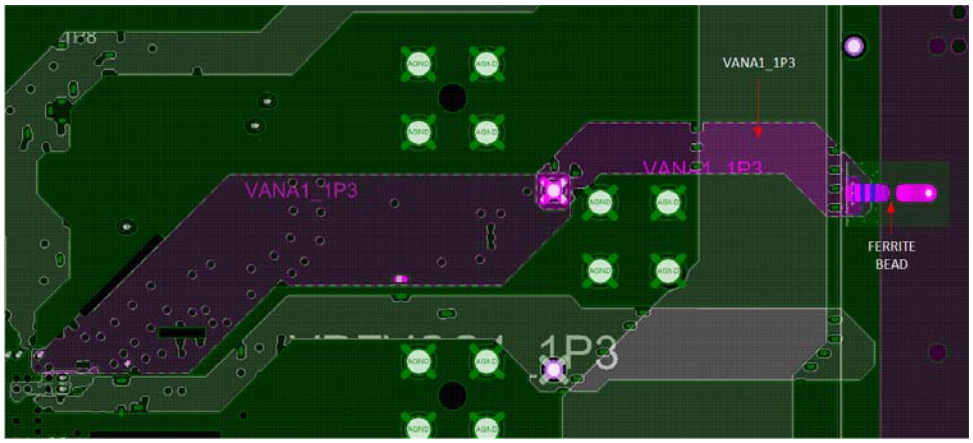


Figure 154. Power Supply Routing Example with Ferrite Bead at the Input (VANA1_1P3)

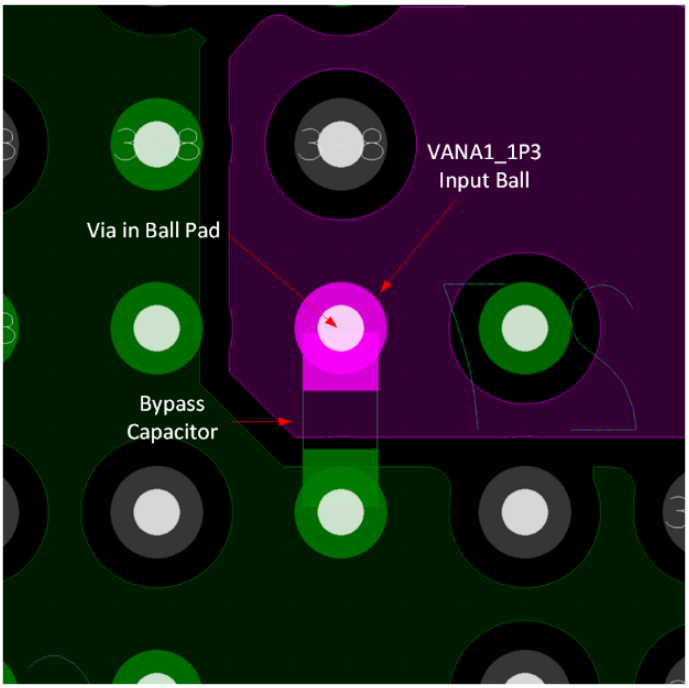


Figure 155. Power Connection to Supply Ball with Bypass Capacitor Between Vias

ANALOG SIGNAL ROUTING CONSIDERATIONS

Other analog signals in and out of the device such as the auxiliary ADCs and DACs do not require critical routing considerations. Use standard routing techniques for these signals to keep them shielded from interference or noise that may affect their desired levels.

DIGITAL SIGNAL ROUTING CONSIDERATIONS

The digital signal routing (for example, SPI, enable controls, and GPIO) is the least sensitive area, but it is nevertheless very important to isolate from other signals to avoid digital noise coupling into other circuits. In the evaluation board these signals are routed from the bottom of the board up through the same channel created for the VDIG power supply on Layer 10, Layer 11, and Layer 12. This provides the benefit of using the same ground return area as the VDIG supply, which keeps the return currents from intermixing with currents from the analog and RF functions of the device. Most of these signals are static or infrequently changing state, so once signals are routed out of the device, they can be fanned-out to other parts of the PCB without concern of interfering with radio functions. Figure 156 illustrates how the signals are routed out of the device following the same path as the VDIG supply (brown fill area). Note that there are designated layers on the customer evaluation PCB for digital routing and these layers are isolated by ground layers from other sensitive signals such as the JESD lanes and the RF inputs and outputs. It is strongly recommended to keep any traces that are non-static, such as the SPI or SPI2 buses, isolated from the sensitive analog/RF/JESD signals by ensuring there is ample ground between the traces and that there is also no overlap of signals from layer to layer.

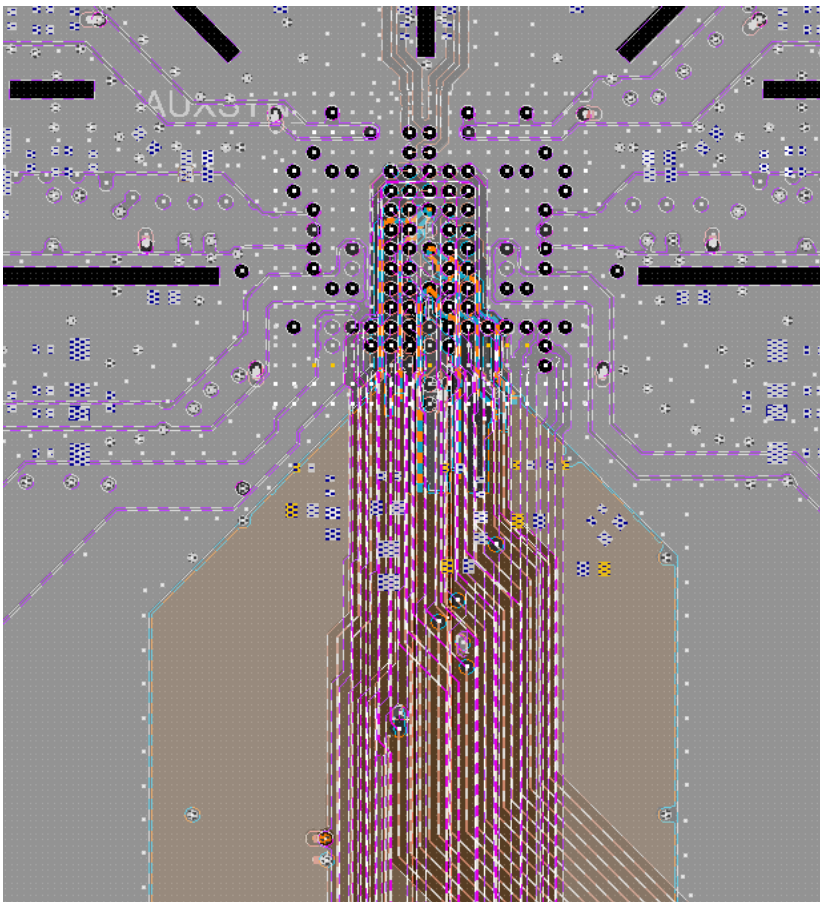


Figure 156. Digital Routing Out of ADRV9026

UNUSED PIN INSTRUCTIONS

In some applications, the user may decide not to use all available inputs or outputs. In these cases, take care to follow the recommendations listed in Table 257 for unused pins.

Table 257. Recommendations for Unused Pins

Pin No.	Type	Mnemonic	When pins are not used:
A4, A5, A13, A14, P1, N1, N17, P17	O	TX3+, TX3−, TX2+, TX2−, TX4+, TX4−, TX1+, TX1−	Do not connect.
E4, E5, E13, E14, L4, L5, L13, L14	I	ORX3+, ORX3−, ORX1+, ORX1−, ORX4+, ORX4−, ORX2+, ORX2−	Connect to VSSA.
C1, B1, B17, C17, J1, H1, H17, J17	O	RX3+, RX3−, RX2+, RX2−, RX4+, RX4−, RX1+, RX1−	Connect to VSSA.
F2, E2, E16, F16	I/O	EXT_LO2+, EXT_LO2−, EXT_LO1+, EXT_LO1−	Do not connect.
C4, C5, L1, L2, L17, L16, C12, C13	I/O	GPIO_ANA_7 to GPIO_ANA_0	Connect to VSSA with a 10 kΩ resistor or configure as outputs, drive low, and leave disconnected.
E1, E17, F1, F17	I	AUXADC_3, AUXADC_1, AUXADC_2, AUXADC_0	Do not connect.
E7, E11, M7, M11	I	TX3_EN, TX2_EN, TX4_EN, TX1_EN	Connect to VSSA.
G7, G11, J7, J11	I	RX3_EN, RX2_EN, RX4_EN, RX1_EN	Connect to VSSA.
F7, F11, L7, L11	I	ORX_CTRL_C, ORX_CTRL_B, ORX_CTRL_D, ORX_CTRL_A	Connect to VSSA directly or with a 10 kΩ pull-down resistor.
H11, K11, N11, E10, F10, G10, H10, J10, K10, E9, F9, E8, F8, G8, H8, J8, K8, H7, K7	I/O	GPIO_0 to GPIO_18	Connect to VSSA with a 10 kΩ resistor or configure as outputs, drive low, and leave disconnected.
N7, N8	O	GPINT2, GPINT1	Do not connect.
M8	O	SPI_DO	Do not connect.
P10	I	TEST_EN	Connect to VSSA.
N6, P6, N12, N13, P7, P8	I	SYNCIN3+, SYNCIN3-, SYNCIN1+, SYNCIN1-, SYNCIN2+, SYNCIN2-	Connect to VSSA.
N14, N15, P15, R15	O	SYNCOUT2+, SYNCOUT2-, SYNCOUT1+, SYNCOUT1−	Do not connect.
U1, U2, T3, T4, U5, U6, T7, T8	O	SERDOUTD+, SERDOUTD− SERDOUTC+, SERDOUTC−, SERDOUTB+, SERDOUTB−, SERDOUTA+, SERDOUTA−,	Do not connect.
U16, U17, T15, T14, U12, U13, T11, T10,	I	SERDIND+, SERDIND−, SERDINC+, SERDINC-, SERDINB+, SERDINB−, SERDINA+, SERDINA−	Do not connect.

TRANSCEIVER EVALUATION SOFTWARE (TES) OPERATION

The ADRV9026 demonstration system enables customers to evaluate the device without having to develop custom software or hardware. The system comprises a radio daughtercard, an ADS9 motherboard, a microSD card with operating system, a power supply for the ADS9, a 12 V power supply that connects to a wall outlet, and a C#-based evaluation software application. The evaluation system uses Ethernet interface to communicate with the PC.

INITIAL SETUP

The ADRVTRX TES is the graphical user interface (GUI) used to communicate with the evaluation platform. It can run with or without evaluation hardware connected. When TES runs without the hardware connected, it can be fully configured for a particular operating mode. If the evaluation hardware is connected, the desired operating parameters can be setup with TES and then the software can program the evaluation hardware. Once the device is configured, the evaluation software can be used to transmit waveforms generated from the internal NCO block or using custom waveform files as well as observe signals received on one of the receiver or observation input ports. An initialization sequence in form of an IronPython script can be generated and executed using TES if customized scripts are desired.

HARDWARE KIT

The ADRV9026 demonstration system kit contains:

- The CE board in form of a daughter card with FMC connector
- One (1) 12 V wall connector power supply cable
- One (1) SD card containing image of Linux operating system with required evaluation software. SD card type is 16 GB size, Type 10

The ADS9 demonstration system kit contains:

- The ADS9 motherboard with FMC connector
- One (1) 12 V, 1.5 A power supply for powering the board

REQUIREMENTS

The hardware and software require the following:

- The ADS9 demonstration system kit
- The ADRV9026 demonstration system kit
- The operating system on the controlling PC must be Windows 7 (×86 and ×64) or Windows 10 (×86 and ×64)
- The PC must have a free Ethernet port with the following constraints:
  - If the Ethernet port is occupied by another LAN connection, a USB to Ethernet adapter can be used
  - The PC must be able to access over this dedicated Ethernet connection via the following ports:
    - Port 22: SSH protocol
    - Port 55556: access to the evaluation software on the ADS9 platform
  - TES: contact a local Analog Devices representative to obtain access to this software.
  - The user must have administrative privileges. To run software automatic updates, the PC must have access to the internet. If internet access is restricted, a manual software update can be performed.

HARDWARE SETUP

Before setup, the ADS9 platform requires the user to insert the SD card included with the evaluation kit into the J6 slot of the ADS9 (MicroZED) platform. The evaluation hardware setup is shown in Figure 157 and Figure 158.

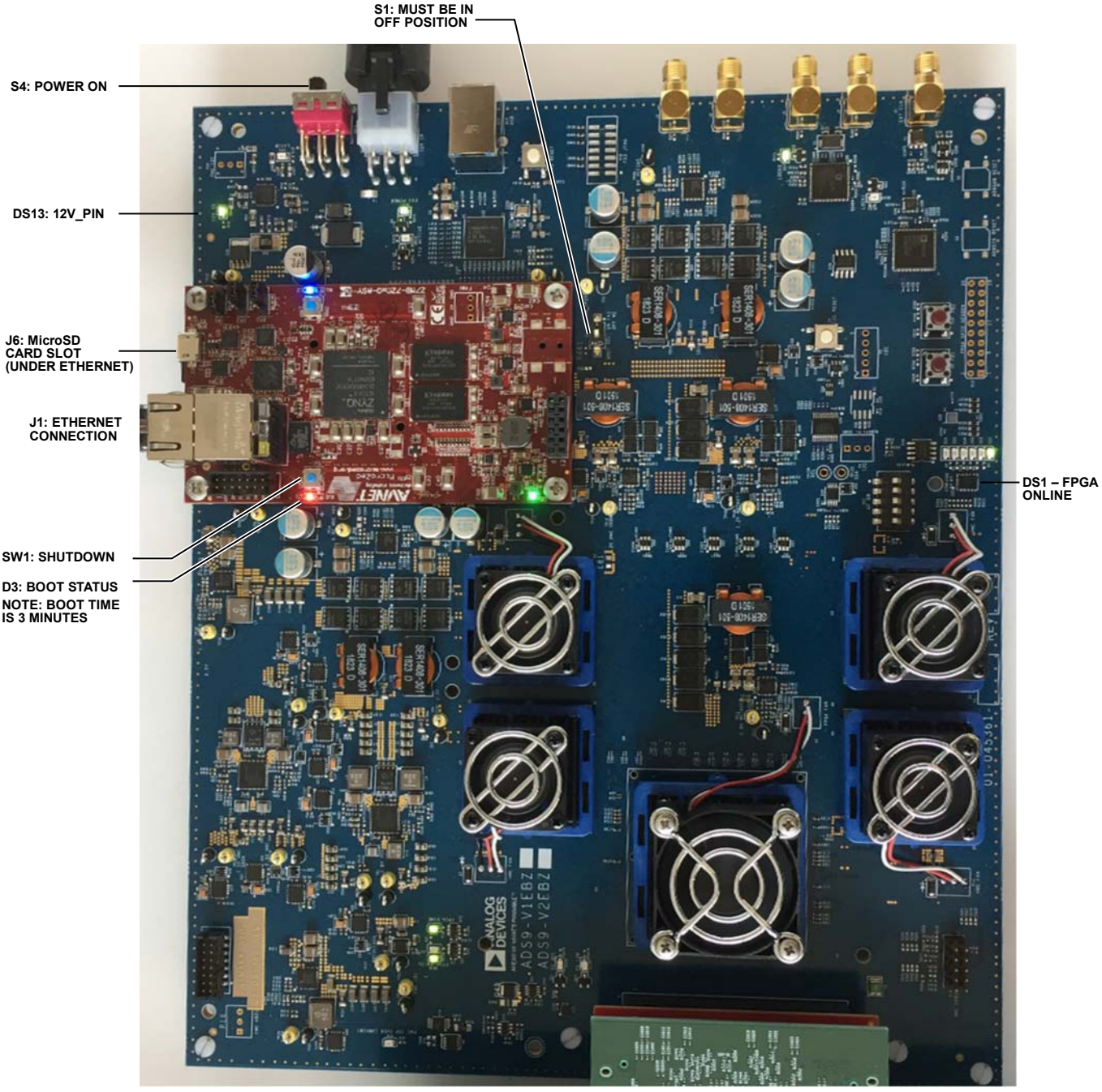


Figure 157. Analog Devices ADS9 Mother Board Configured to Work with ADRV9026 Evaluation Boards

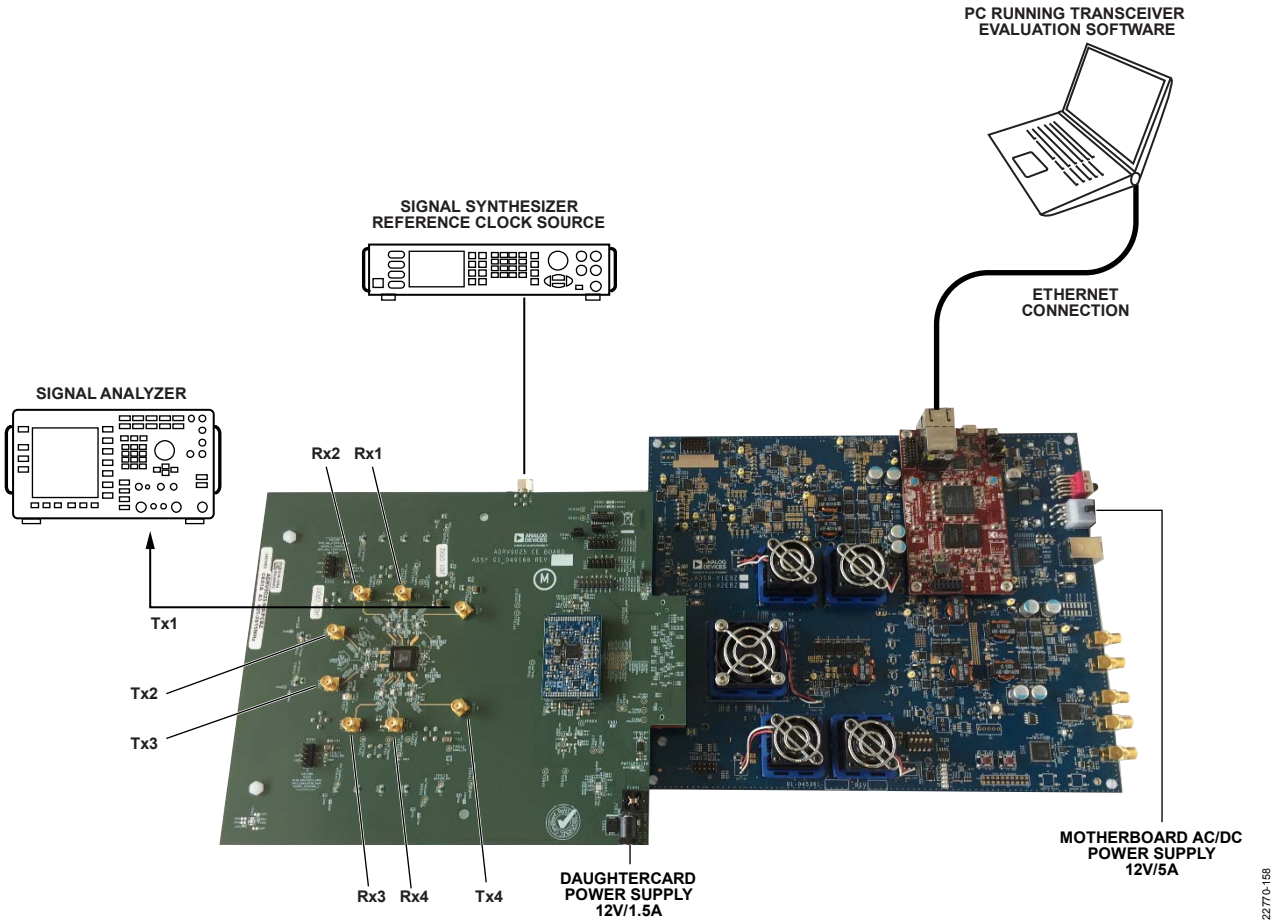


Figure 158. CE Board and ADS9 Motherboard with Connections Required for Tx Testing

To set up the evaluation board for testing, follow steps listed below:

1. Connect the ADRV9026 evaluation board and the ADS9 evaluation platform together as shown in Figure 158. Use the HPC FMC connector (P1001/P2). Ensure the connectors are properly aligned.
2. Insert the SD card that came with the evaluation kit into ADS9 microSD card slot (J6).
3. On the ADRV9026 evaluation card, provide a reference clock source (122.88 MHz is the default, or frequency match the setting selected on the AD9528 configuration tab), at a 7 dBm power level to the J613 connector. (This signal drives the reference clock into the AD9528 clock generation chip on the board. The REFA/REFA_N pins of AD9528 generate the DEV_CLK for the device and REF_CLK for the FPGA on the ADS9 platform).
4. Connect a 12 V, 1.5 A power supply to the ADS9 evaluation platform at the P1 header.
5. Connect the ADS9 evaluation platform to the PC with an Ethernet cable (connect to P3). There is no driver installation required.

In the case when the Ethernet port is already occupied by another connection, use an USB to Ethernet adapter.

On an Ethernet connection dedicated to the ADS9 platform, the user must manually set the following:

- IPv4 address: 192.168.1.2
- IPv4 subnet mask: 255.255.255.0

Refer to Figure 159 for more details. Make sure that the following ports are not blocked by firewall software on their PC:

- Port 22: SSH protocol
- Port 55556: access to the evaluation software on ADS9 platform

Note that the ADS9 IP address is set by default to 192.168.1.10.

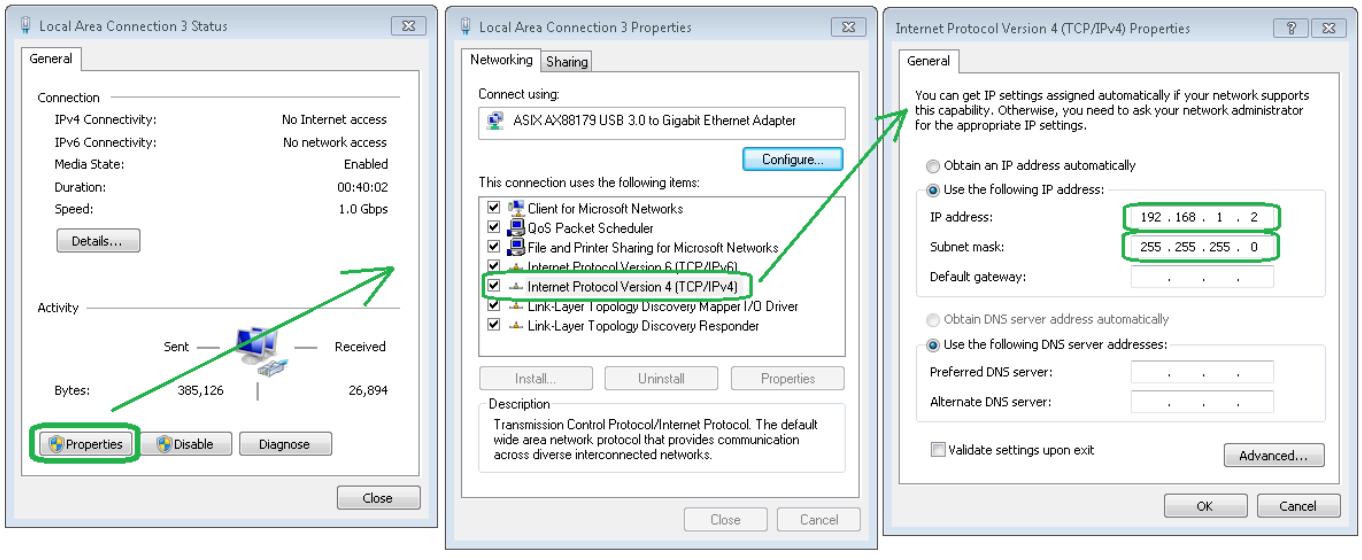


Figure 159. IP Settings for Ethernet Port Dedicated for ADS9 Platform

HARDWARE OPERATION

1. Turn on the evaluation system by switching the ADS9 motherboard power switch (S4) to the on position. If hardware is connected correctly, the green LED (DS13) on the ADS9 motherboard turns on.
2. The ADS9 motherboard uses a Linux operating system. It takes approximately 3 minutes before the system is ready for operation and can accept commands from PC software. Boot status can be observed on ADS9 LED (D3, on the MicroZED daughtercard). This LED illuminates red for approximately 3 minutes after power on. When it goes off, this indicates that the board is booted properly. When D3 transitions from red to off, the system is ready for normal operation and awaits connection with the PC over Ethernet (which must be established using TES).
3. Connect the reference clock signal (122.88 MHz CW tone, 7 dBm maximum) to J613 on the underside of the CE board.
4. Before applying power to the CE board, ensure that each of the four Tx output ports (J501 to J504) are properly terminated.
5. After the ADS9 system has properly booted, LED DS801 on the evaluation board illuminates. At this point, power from the 12 V wall adapter must be connected to the CE board. When power is applied, DS802 illuminates on the CE board.
6. For transmitter testing, connect a spectrum analyzer to any Tx output on the evaluation board. Use a shielded RG-58, 50 Ω coaxial cable (1 m or shorter) to connect the spectrum analyzer. Terminate all Tx paths, either into spectrum analyzers or into 50 Ω if unused.
7. The CE board must be powered off before the motherboard by unplugging the wall adapter. When power is removed from the CE board, click on disconnect in the TES window and then press and hold SW1 on the ADS9 evaluation board (MicroZED daugher card) until LED D3 illuminates. When LED D3 goes off, it is safe to turn off the ADS9 power using Switch S4.

TES INSTALLATION

Customers must contact an Analog Devices representative to obtain access to TES. After the initial software download, copy the software to the target system and unzip the files (if not already unzipped). The downloaded zip container has an executable file called **ADRV9025 Transceiver Evaluation Software.exe**.

Administrator privileges are required to install TES. After running an executable file, a standard installation process follows. Parts of the installation build are Microsoft .NET Framework 4.5 (which is mandatory for the software to operate) and IronPython 2.7.4 (which is optional and recommended). Figure 160 shows the recommended configuration. Note that the Microsoft .NET Framework and the IronPython 2.7.4 installations are not necessary to select once they have been installed. If you are updating the version of the TES, these boxes can be left unchecked to save installation time.

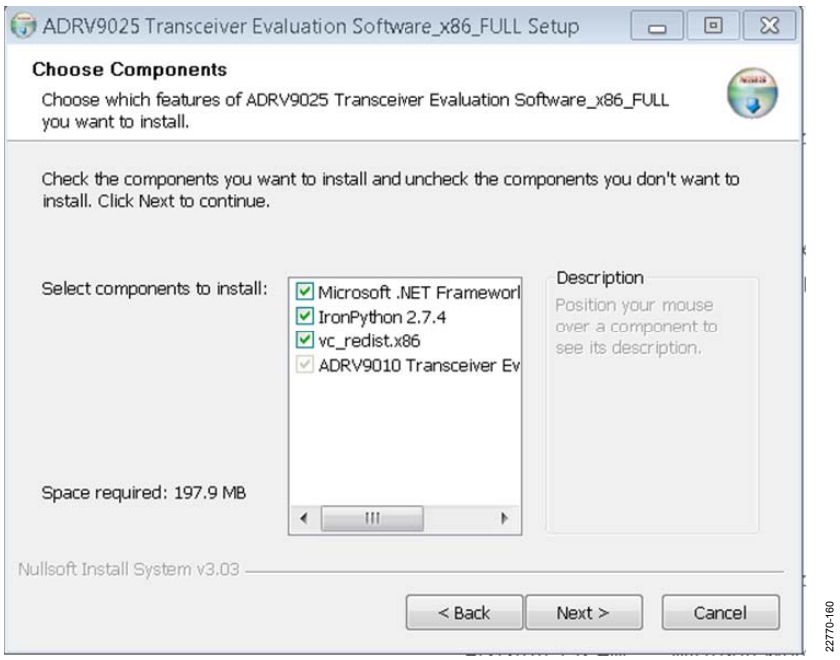


Figure 160. Software Installation Components

The last step of the instalation process is to select shortcut configuration as shown in Figure 161. The user can select a shortcut to be placed in the Windows Start Menu and/or on the Windows desktop.

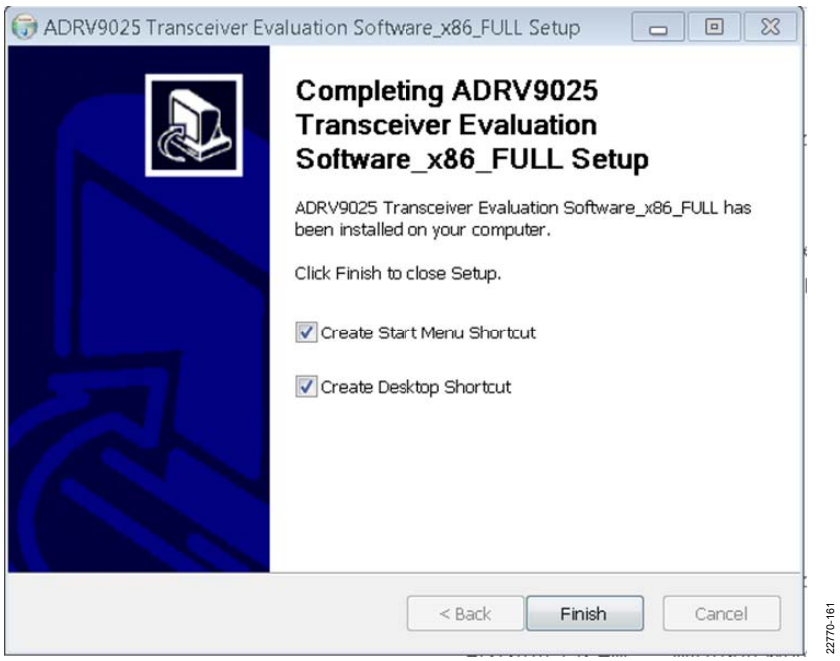


Figure 161. Transceiver Evaluation Software Shortcut Configuration

STARTING THE TRANSCEIVER EVALUATION SOFTWARE

Depending on the user selection during the installation process (see Figure 161), users can start the customer software by clicking on **Start** > **All Programs** > **Analog Devices** > **ADRV9025 Transceiver Evaluation Software_x86_FULL** > **ADRV9025 Transceiver Evaluation Software** or by clicking on the desktop shortcut called **ADRV9025 Transceiver Evaluation Software**. Figure 162 shows the opening page of the TES after it is activated.

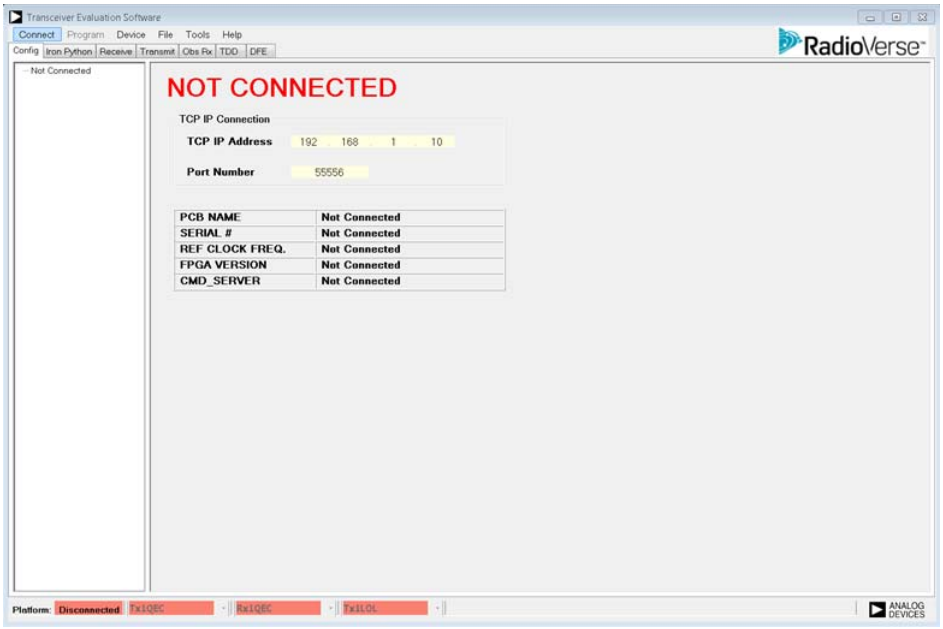


Figure 162. Transceiver Evaluation Software Interface

Demo Mode

Figure 162 shows the opening page of the TES. In the case when evaluation hardware is not connected, the user can still use the software in demo mode by following these steps:

1. Click on **Connect** (top left corner).
2. The software moves into demo mode in which a superset of all transceiver family features is displayed.

NORMAL OPERATION

When hardware is connected to a PC and the user wants to start using the complete evaluation system, TES establishes a connection with the ADS9 system via Ethernet after clicking the **Connect** option in the drop down menu. When proper connection is established, the user can click on the **DaughterCard** position in device tree on the left. After selecting **DaughterCard**, information about revisions of different setup blocks appears in the main window. The bottom part of that window shows the TCP IP Address set to 192.168.1.10 and Port Number set to 55556. Contact the Analog Devices Applications Engineering team if the ADRV9026 Evaluation System needs to operate over a remote connection and a different IP address for the ADS9 platform is desired. Figure 163 shows an example of correct connection between a PC and a ADS9 system with a daughter card connected to it.

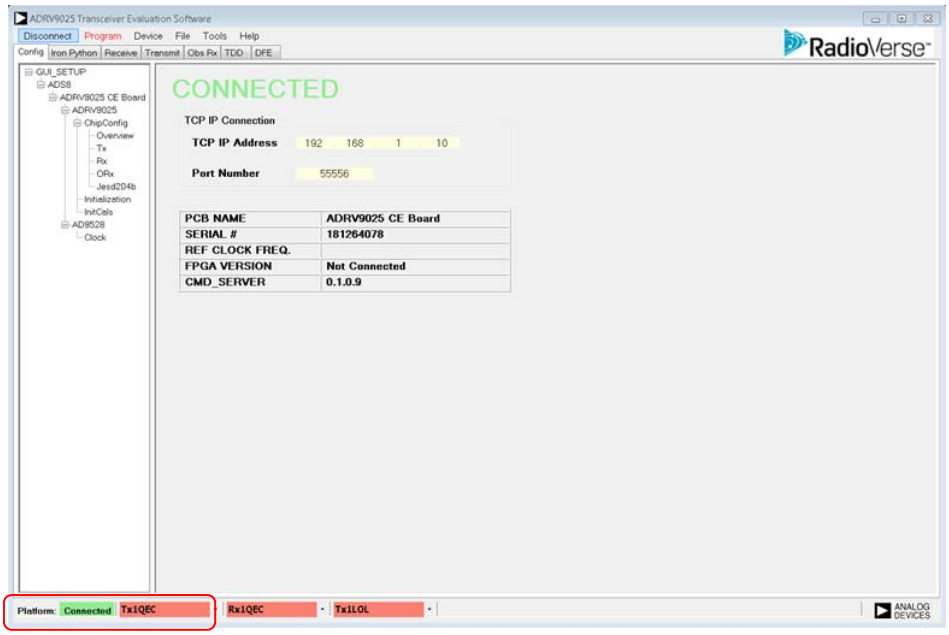


Figure 163. Setup Revision Information

Configuring the Device

Contained within the **Config** tab are sub tabs that contain setup options for the device. The first one displayed is the **Overview** tab. Figure 164 shows the initial screen for the device Tx. In this page the user can select the following:

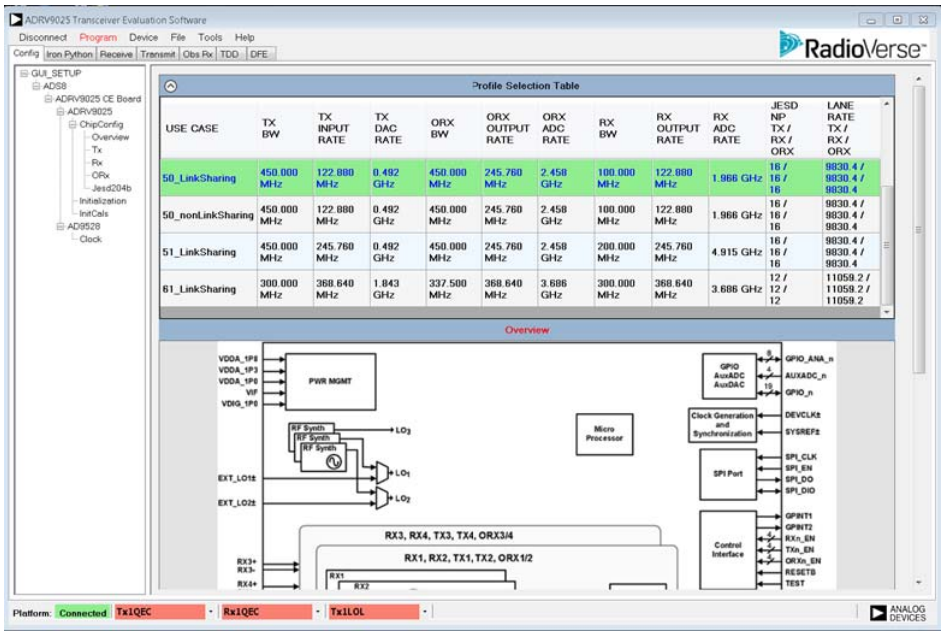
- Device to be programmed
- Select profiles

Profile Options

The TES contains the following profile options:

- ADRV9025Init_StdUseCase50_nonLinkSharing
- ADRV9025Init_StdUseCase50_LinkSharing
- ADRV9025Init_StdUseCase51_LinkSharing
- ADRV9025Init_StdUseCase61_ LinkSharing

These profiles configure the device for different Tx, Rx, and ORx bandwidths, sample rates, and clock rates. They also set different JESD configurations and lane rates. By default, the platform boots to JESD204B mode. Note the available use cases may vary based on the version of the software being run.



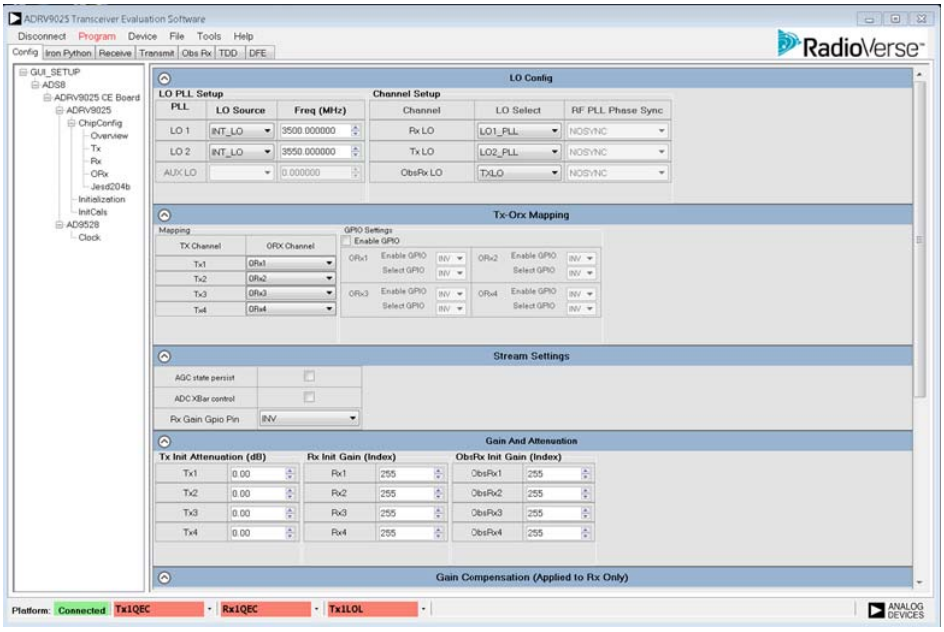
22770-164

Figure 164. Main Overview Tab

Additional 204C use cases are also available. The platform must be switched to 204C mode then the available 204C profiles are displayed. To switch the platform select **Device** > **FPGA switch JESD** > **Jesd 204C**. At this point the platform reboots (which takes approximately 3 minutes). Upon reconnecting, the 204C profiles are available.

Initialization

The **Initialization** tab provides access to the settings that are used to configure the device at startup. This page allows the user to set the LO frequencies used, initial Tx attenuation settings, initial Rx gain index settings, and the initial gain index for ORx1 (the only ORx channel available at this stage in development). These and other settings that are provided in future revisions are shown in Figure 165.



22770-165

Figure 165. Initialization Configuration Tab

InitCals

The **InitCals** tab is used to set which calibrations take place at initialization. The default settings are shown in Figure 166. These can be enabled/disabled by clicking on the box next to the calibration. A check mark indicates the calibration is run at startup.

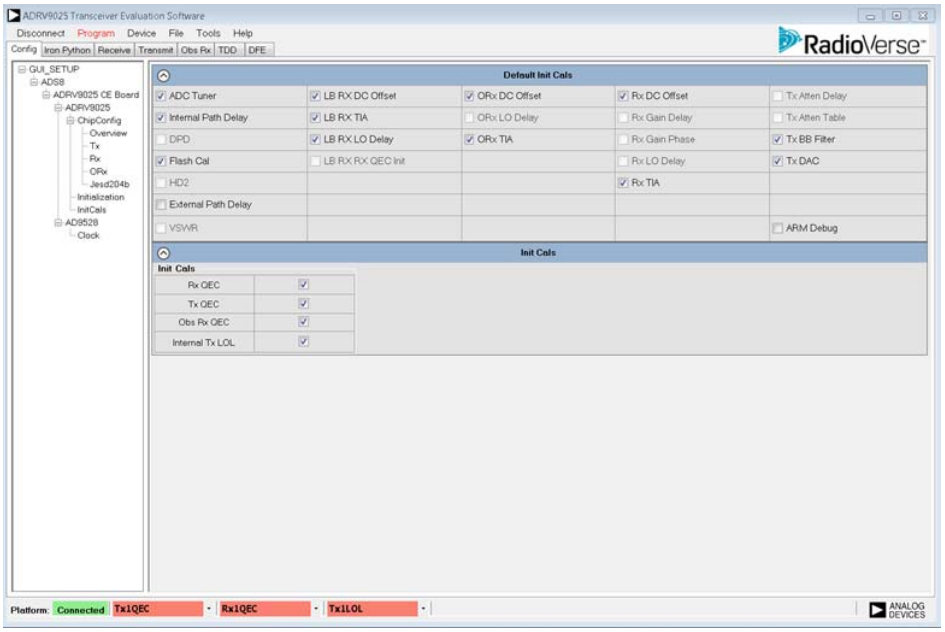


Figure 166. InitCals Tab with Default Settings

Tx Configuration

The Tx tab is primarily informative and is based on the profile selection in the **Overview** tab (Figure 164). In this tab, the user can check clock rates at each filter node as well as filter characteristics and pass-band flatness. Quick zooming capability allows zooming of the pass-band response as well as restoring to the full scale plot. Figure 167 shows an example of the **Tx** tab with the resulting composite filter response for the chosen profile.

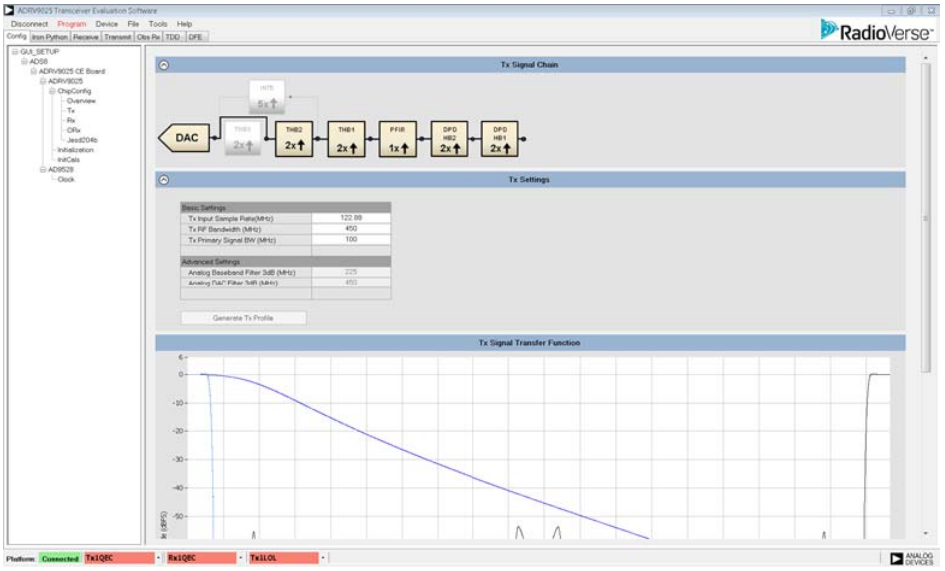


Figure 167. Tx Summary Tab

Rx Configuration

The Rx tab is primarily informative and is based on the profile selection in the **Overview** tab (see Figure 164). In this tab, the user can check clock rates at each filter node as well as filter characteristics and pass-band flatness. Quick zooming capability allows zooming of the pass-band response as well as restoring to the full scale plot. Figure 168 shows an example of the Rx tab with the resulting composite filter response for the chosen profile.

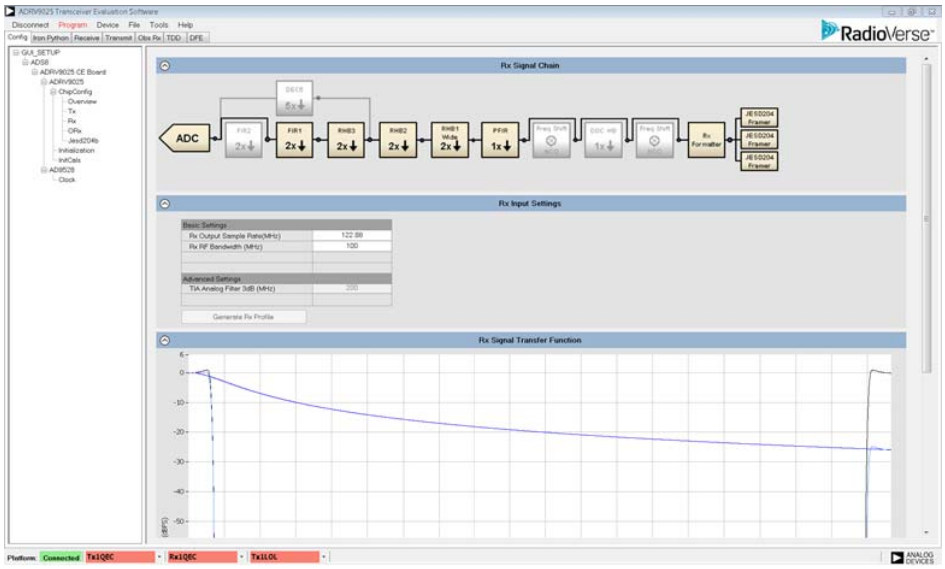


Figure 168. Rx Summary Tab

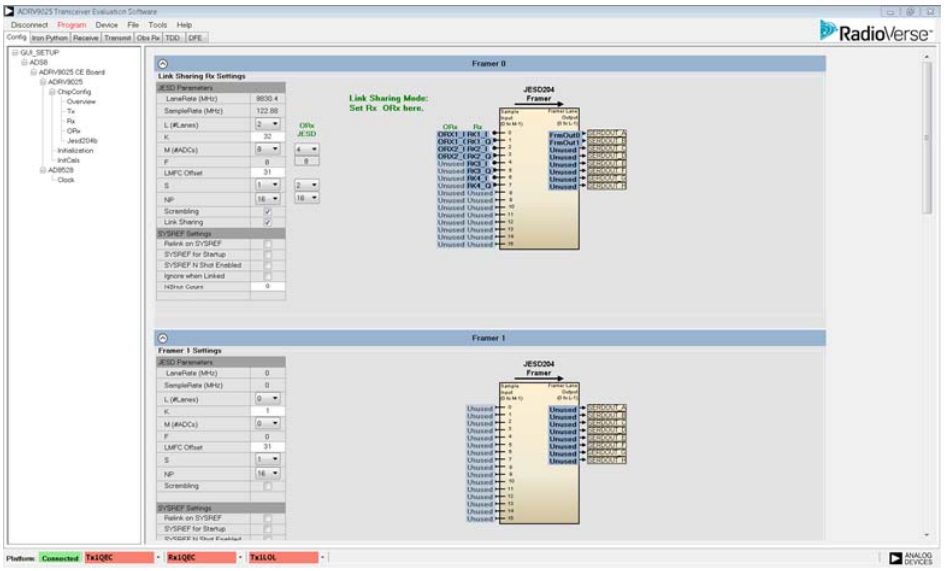
ORx Configuration

The ORx tab is primarily informative and is based on the profile selection in the Overview tab (Figure 164). In this tab, the user can check clock rates at each filter node as well as filter characteristics and pass-band flatness. Quick zooming capability allows zooming of the pass-band response as well as restoring to the full scale plot. Figure 169 shows an example of the ORx tab with the resulting composite filter response for the chosen profile.

Figure 169. ORx Summary Tab

JESD Configuration

The JESD tab is primarily informative and is based on the profile selection in the **Overview** tab (see Figure 164). In this tab, the user can check the Tx Deframer settings and the Rx and ORx Framer settings. Figure 170 shows an example of the **JESD** tab with the settings for Use Case 13.



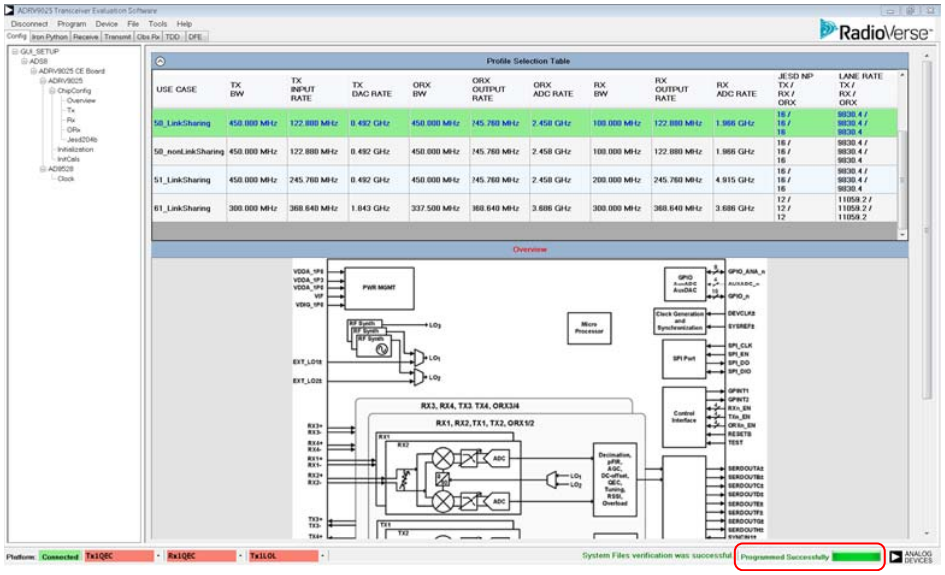
22770-170

Figure 170. JESD Summary Tab

Programming the Evaluation System

After all tabs are configured, the user must press the Program button. This kicks off initialization programming. TES sends a series of API commands that are executed by a dedicated Linux application on the ADS9 platform.

When programming has completed, the system is ready to operate. There is a progress bar at the bottom of the window. Figure 171 displays the window with the progress bar and message after the device has been programmed.



22770-171

Figure 171. Device Programmed

Initialization Script

TES allows the user to create a script with all API initialization calls in the form of IronPython functions. When the user clicks the **Tools** > **Save Python Script** option, the script can be given a file name and stored in a location of the choosing of the user for future use. TES generates the script in the form of a python (.py) file. That file can then be executed using the **IronPython Script** tab shown in Figure 176. There is an option to save a MATLAB based initialization script for the chosen setup in the GUI.

TRANSMITTER OPERATION

Selecting the **Transmit** tab opens a page as shown in Figure 172. The upper plot displays the FFT of the digital input data and the lower plot shows its time domain waveform. When multiple Tx outputs are enabled, the user can select desired data to be displayed in the Spectrum plot using the checkboxes below the plot. In the time domain plot, the user can select Tx1, Tx2, Tx3, Tx4, or any combination of the data input channels, with I and/or Q data displayed.

Once the **Transmit** tab is open, the user can enter the RF Tx center frequency in MHz for Tx LO1 (used for Tx operation), change attenuation level, and transmit CW tones.

Figure 172. Transmit Data Tab

Transmitter Data Options

The TES provides the following options for inputting transmitter data:

- A single tone from the internal NCO can be generated on each channel by the evaluation system using the **Tone Parameters** menu shown in Figure 173. This window is accessed by pressing the **TONES** button near the upper left of the **Transmit** page. In that window, the user can enable the tone (**Number of Tones** = 0 to 3) to be transmitted on the selected Tx output.
- The user can also chose to input a waveform file instead of using the internal NCO by selecting the **LoadFile** box and entering the path to the waveform file.

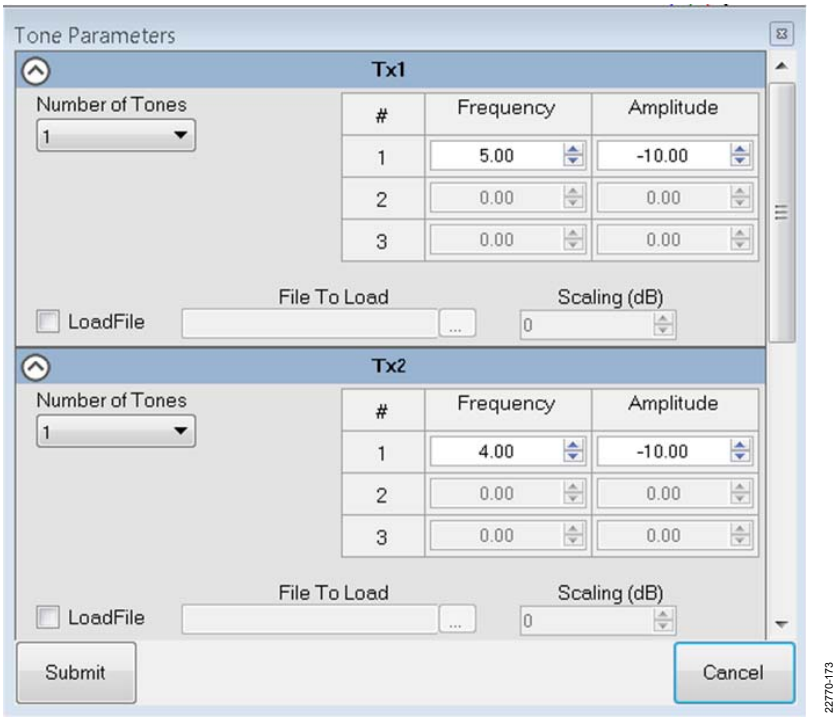


*Figure 173. Tx **Tone Parameters** Setup Menu*

Pressing the play symbol moves the device to the transmit state and starts a process where the NCO generated CW data is enabled.

The **Tx2 Attenuation (dB)** input allows the user to control analog attenuation in the Tx2 channel. It provides 0.05 dB of attenuation control accuracy. The **Tx3 Attenuation (dB)** and **Tx4 Attenuation (dB)** perform the same operation on the Tx3 and Tx4 channels.

RECEIVER OPERATION

Rx Signal Chain

The **Receive** tab opens a window as shown in Figure 174. The upper plot displays the FFT of the received input data and the lower plot shows its time domain waveform. When multiple Rx inputs are enabled, the user can select the desired data to be displayed in the Spectrum plot using the checkboxes below the plot. In the Time Domain plot, the user can select Rx1, Rx2, Rx3, Rx4, or any combination of the input channels, with I and/or Q data displayed.

Once the **Receive** tab is open, the user can enter the RF LO frequencies for LO1 and LO2 PLLs, select which LO is used by Rx1 and Rx2, select which LO is used by Rx3 and Rx4, select the Rx Trigger type, enter the sample time for the data, and select gain levels and tracking calibrations for each channel. Pressing the play symbols enables the selected receivers and displays their waveform data.

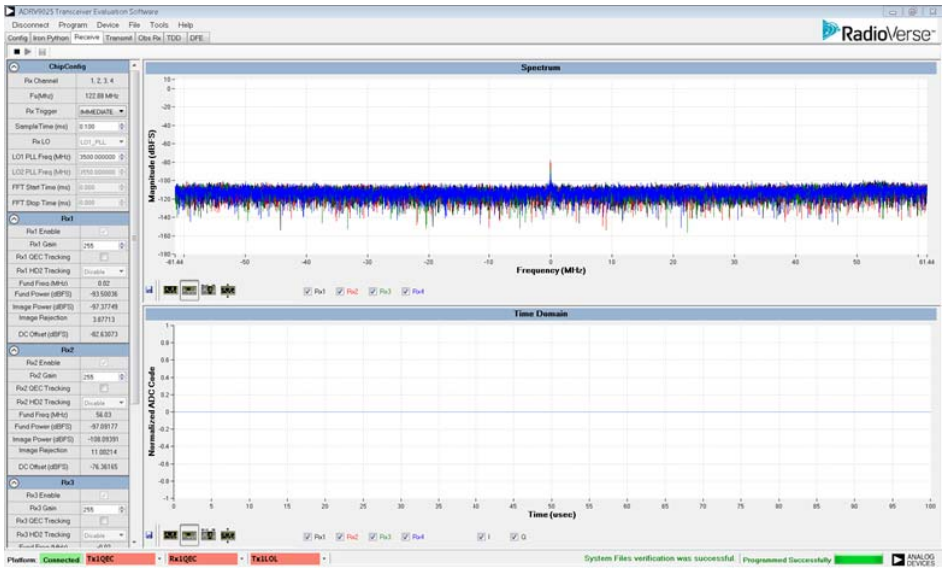


Figure 174. Receive Data Tab

Observation Rx Signal Chain

The **Obs Rx** tab opens a window as shown in Figure 175. The upper plot displays the FFT of the received input data and the lower plot shows its time domain waveform. When multiple ORx inputs are enabled, the user can select the desired data to be displayed in the Spectrum plot using the checkboxes below the plot. In the Time Domain plot, the user can select Obs1, Obs2, Obs3, Obs4, or any combination of the input channels, with I and/or Q data displayed.

Once the **Obs Rx** tab is open, the user can enter the RF LO frequencies for LO1, LO2, and Aux LO PLLs, select which LO is used by ObsRx1 and ObsRx2, select which LO is used by ObsRx3 and ObsRx4, select the Obs Rx Trigger type, enter the sample time for the data, and select gain levels and QEC for each channel. Pressing the play symbols enables the selected receivers and displays their waveform data.

Note that only Obs Rx1 is enabled for use at this stage in product development. All four channels are available in future software revisions.

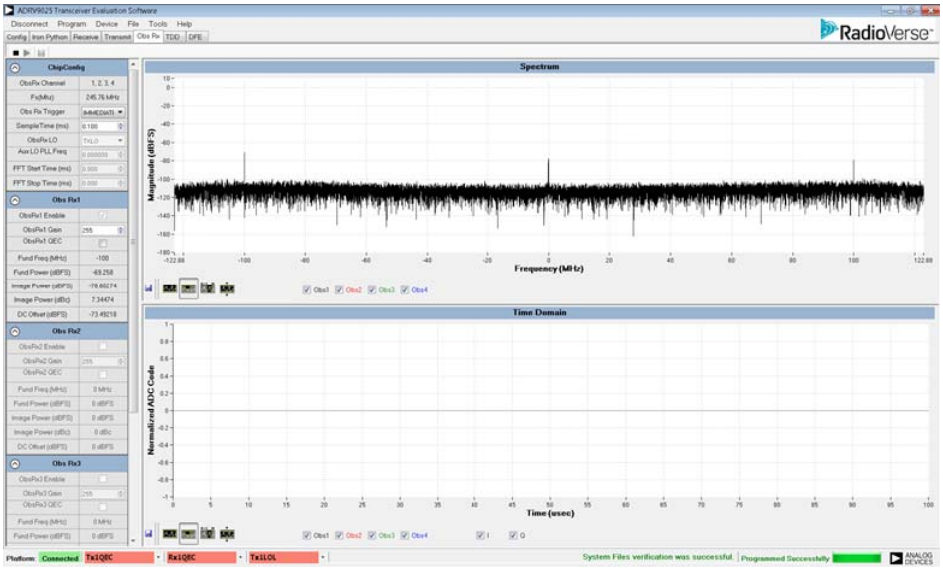


*Figure 175. **Obs Rx** Data Tab*

SCRIPTING

The **Iron Python** tab allows the user to use IronPython language to write a unique sequence of events and then execute them using the evaluation system. Scripts generated using this tab can be loaded, modified if needed, and run on the evaluation system. Figure 176 shows the **Iron Python** tab after executing the **File** > **New** script function. The top part of the window contains IronPython commands while the bottom part of the window displays the script output. Scripts are run by selecting **Build** > **Run**. Scripts that provide useful functions that the user may want to use in the future can be saved by selecting **File** > **Save** and entering the path and file name for saving the script.

After the user configures the part to the desired profile, a script can be generated with all API initialization calls in the form of IronPython functions. Selecting **Tools** > **Create Script** > **Python** accomplishes this task. This generates a script with the initialization sequence and open a Dialogue box to save this file. Basic script with no initializaion sequence can be generated by selecting **File** > **New** option.

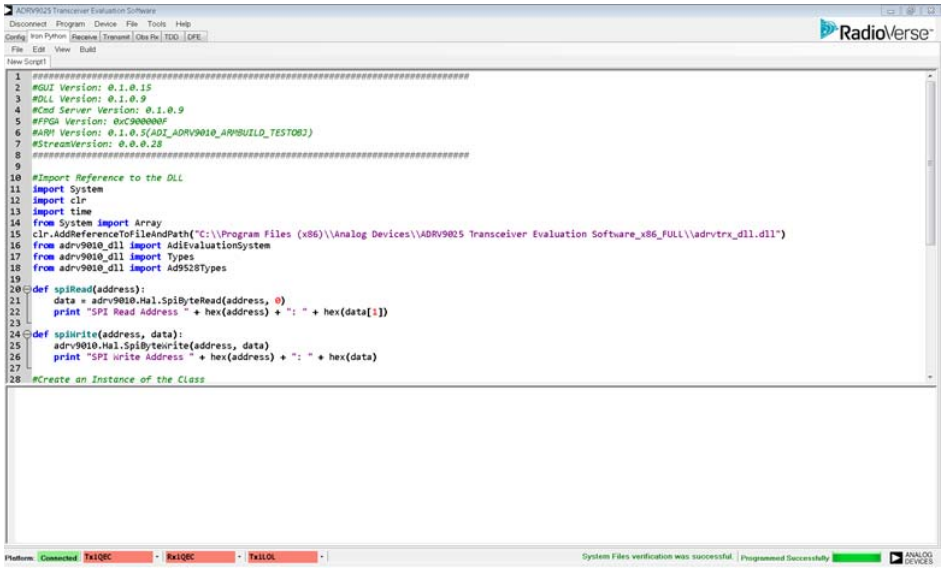


Figure 176. Iron Python Scripting Window

IronPython Script Example

The following example sets the RF LO frequency of LO1 and LO2 and reads back the configured values.

```
################################################################################
#GUI Version: 0.1.0.19
#DLL Version: 0.1.0.11
#Cmd Server Version: 0.1.0.11
#FPGA Version: 0xC900000F
#Arm Version: 0.1.0.5(ADI_ADRV9025_ARMBUILD_TESTOBJ)
#StreamVersion: 0.0.0.28
################################################################################

#Import Reference to the DLL
import System
import clr
import time
from System import Array
clr.AddReferenceToFileAndPath("C:\\Program Files (x86)\\Analog Devices\\ADRV9025 Transceiver
Evaluation Software_x86_FULL\\adrvtrx_dll.dll")
from adrv9025_dll import AdiEvaluationSystem
from adrv9025_dll import Types
from adrv9025_dll import Ad9528Types

#Create an Instance of the Class
Link = AdiEvaluationSystem.Instance

if (Link.IsConnected):
    fpga9025 = Link.FpgaGet()
    adrv9025 = Link.ADRV9025Get(1)

    print "Setting PLL LO1 and LO2"
    adrv9025.RadioCtrl.PllFrequencySet(Types.adi_adrv9025_PllName_e.ADI_ADRV9025_LO1_PLL,
3500000000)
    adrv9025.RadioCtrl.PllFrequencySet(Types.adi_adrv9025_PllName_e.ADI_ADRV9025_LO2_PLL,
3550000000)

    print "Readback PLL"
    lo1 = adrv9025.RadioCtrl.PllFrequencyGet(Types.adi_adrv9025_PllName_e.ADI_ADRV9025_LO1_PLL,
0)
    print "LO1 set to :" + str(lo1[1])
    lo2 = adrv9025.RadioCtrl.PllFrequencyGet(Types.adi_adrv9025_PllName_e.ADI_ADRV9025_LO2_PLL,
0)
    print "LO2 set to :" + str(lo2[1])

else:
    print "Not Connected"

print "Finished Setting RF PLL"
```

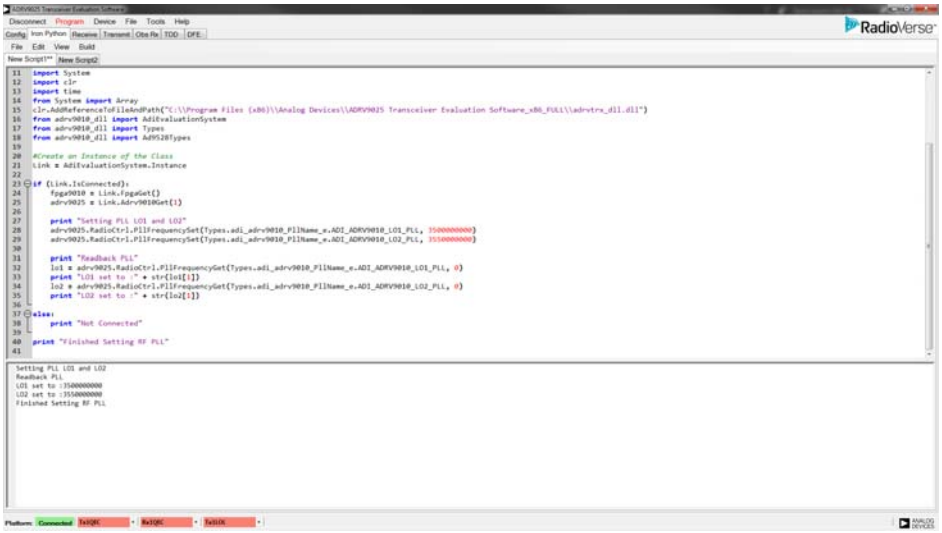


Figure 177.

When using the Iron Python window, the user can execute any API command that is available in the loaded software build.

C CODE GENERATION

It is possible to generate C initialization structure from the GUI. To generate this code, select **Tools** > **Create Script** > **Init c files**.

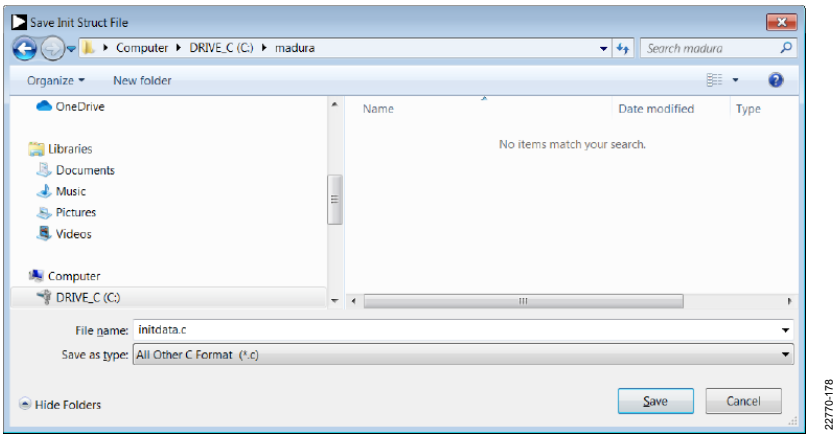


Figure 178.

On selecting this option, GUI open a dialogue box for selecting a location and file name to store this code. preferred file name **initdata.c**. You can choose to store resource files at the same location or another location by selecting **Yes** or **No** on the prompt.

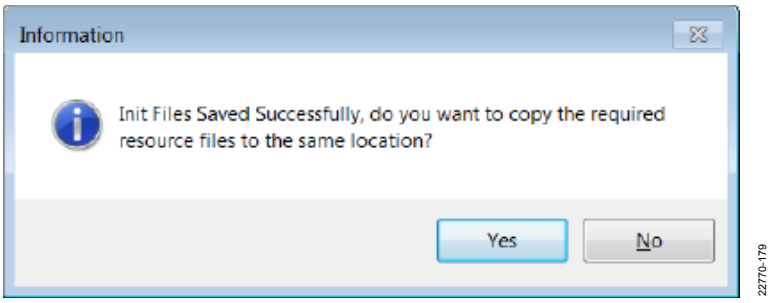


Figure 179.

To use this code, follow these steps:

1. Copy the **c_src** folder from **Adi.ADRV9025.Api\public\src** to the location **/home/analog/adrv9025_c_example/** on platform. (create directory **adrv9025_c_example** if not present).
2. Copy the generated resources folder to the platform at the same location **/home/analog/adrv9025_c_example/**.
3. Copy the generated files, **initdata.h**, **initdata.c**, and **main.c** to **/home/analog/adrv9025_c_example/c_src/app/example/**.
4. Go the the example directory using the terminal and run following command command to enter the directory (see Figure 180).

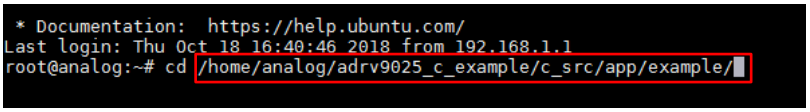


Figure 180.

5. Run Comand **make** at this location. This compiles the code. If no errors, this generates an executable with name **main** in **/home/analog/adrv9025_c_example/c_src/app/example/.**

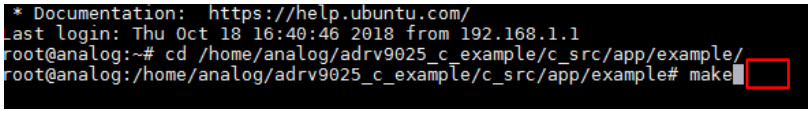


Figure 181.

6. Run command **./main** in the same folder to run the initialization sequence.

I²C refers to a communications protocol originally developed by Philips Semiconductors (now NXP Semiconductors).