ANALOG DEVICES

ADRV9026/ADRV9029 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

ADRV9026/ADRV9029 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., ADRV902x family of software defined radio transceivers. This family consists of the ADRV9026 integrated quad RF transceiver and the ADRV9029 integrated quad RF transceiver with digital predistortion (DPD) and crest factor reduction (CFR) capability. The content of the user guide covers all functions that are common to both devices and some that are unique to the ADRV9029 device. Throughout the user guide, the term transceiver is used with functions that are common to both devices. Functions that are unique to the ADRV9029 device use ADRV9029 in the description. This user guide must be used in conjunction with the product data sheets to incorporate all necessary specifications and descriptions when designing these devices into new equipment.

TABLE OF CONTENTS

REVISION HISTORY

6/2022—Revision 0: Initial Version

SYSTEM OVERVIEW

The ADRV9026 and ADRV9029 are 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 transceiver 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 transceiver 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 user guide is designed to encompass description of all functions available in the these transceivers. Note that some variants may be developed for specific design targets that do not encompass all available functions, therefore, refer to the data sheets for the specific transceiver to determine which features are included. To avoid confusion, the term transceiver is used throughout this user guide to refer to any variant that employs a specific function. When a function that applies to a specific device is described, the device part number is used to delineate which transceiver is being described.

These transceivers are 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 24.33 Gbps (see data sheets for specifications). 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 datapaths to the internal ARM processor for use in calibration and signal enhancement algorithms.

Multiple fully integrated PLLs are included in the transceiver 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.

The power supply for each block is distributed across four different voltage supplies, three analog voltage supplies and one digital voltage supply. 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 respective data sheets for a high level view of the functions in each transceiver. Descriptions of each block with setup and control details are provided in subsequent sections of this document.

SYSTEM ARCHITECTURE DESCRIPTION

Analog Devices developed a proprietary application programming interface (API) software for the these transceiver devices. However, this section outlines the overall architecture, folder structure, and methods for using API software on the customer platform, but this section does not explain the API library functions. Detailed information regarding the API functions is in the **doxygen** file 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 ADRV9025 is the generic transceiver reference number for the ADRV902x family; all API functions use the ADRV9025 number to delineate the product from other transceiver products. With respect to this user guide, ADRV9025 is interchangeable with ADRV9026 and ADRV9029.

SOFTWARE ARCHITECTURE

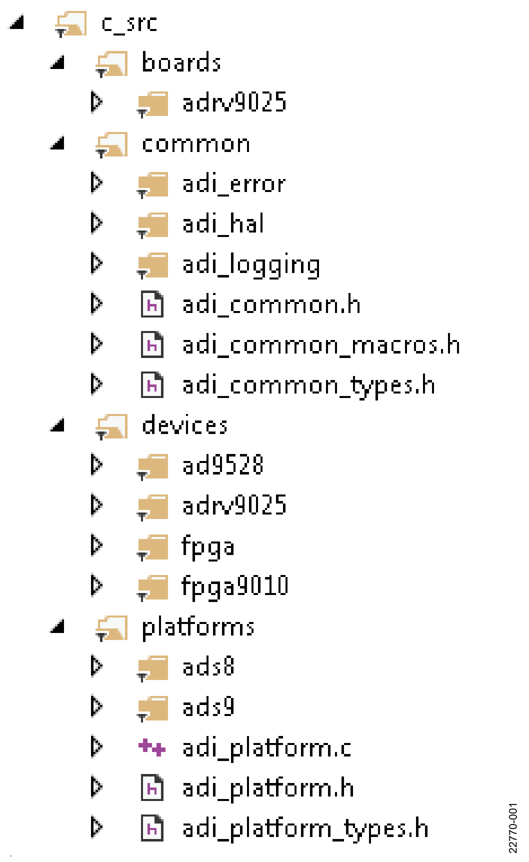
Figure 2 illustrates the software architecture for the system evaluation platform.

This architecture can be broadly divided into the following three main layers:

- Hardware abstraction 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. Analog Devices understands that each 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 can 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.



```
c_src
    boards
        adrv9025
    common
        adi_error
        adi_hal
        adi_logging
        adi_common.h
        adi_common_macros.h
        adi_common_types.h
    devices
        ad9528
        adrv9025
        fpga
        fpga9010
    platforms
        ads8
        ads9
        adi_platform.c
        adi_platform.h
        adi_platform_types.h
```

Figure 1. API Folder Structure

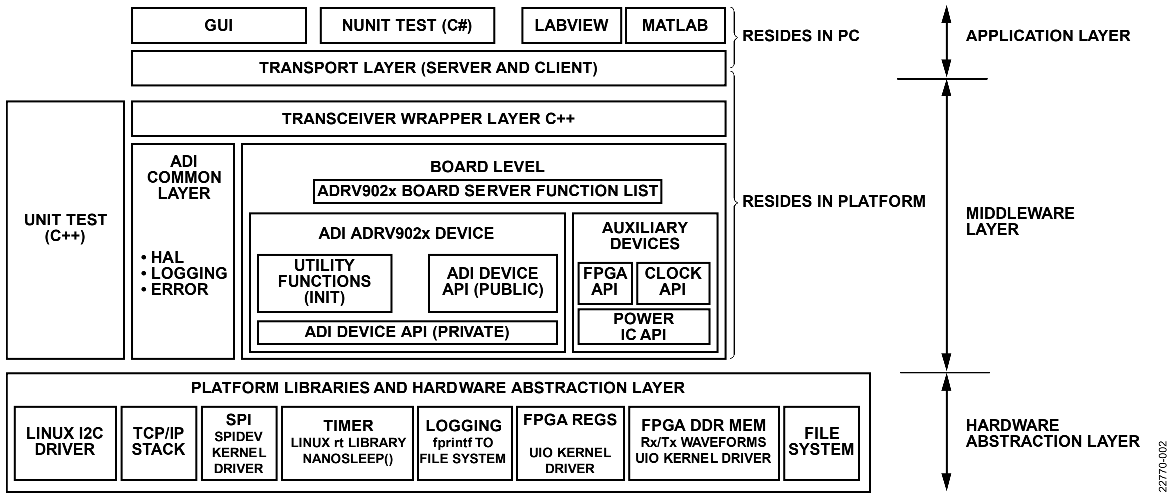


Figure 2. ADRV9025 API Software Architecture (Analog Devices Evaluation Platform)

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 used by the evaluation system. 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)

The /c_src/doc 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 the C computer programming language, 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 must access the target platform hardware. The implementation of this interface is platform dependent and must 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.

The following is a snippet 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. 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. For more details of the $\overline{\text{RESET}}$ pin, refer to the target platform schematic and transceiver data sheet that can be found in the folder at the following link: ADRV9026 Datasheet and Product Info.

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. The open hardware function, adi_hal_HwOpen, calls adi_hal_LogWrite to set the desired logging operation. Available logging levels are given by the functions 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	Bit field 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, such as SPI chip select and reset signal. The HAL function prototypes first parameter, void* devHalCfg, provides the platform layer functions with device specific settings, such as SPI chip select and log file names. 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 that 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 acts as a state container for the relevant hardware information for a particular device. Note that within the platform layer (adi_platform.h), devHalInfo is the same as devHalCfg.

The API user must define this state container as per system HAL implementation requirements. The user may implement any structure to pass any hardware configuration information that the hardware requires between the application layer and platform layer. This state container may be used to transfer device reference information in multithreaded and multitransceiver 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 describes 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 an SPI interface, accessed via the HAL. The references to the SPI driver must be updated by the user in the HAL.
- Compilation and Programming: When the files required for software integration are available, the device API can be compiled, and the transceiver specific platform files programmed into the transceiver.

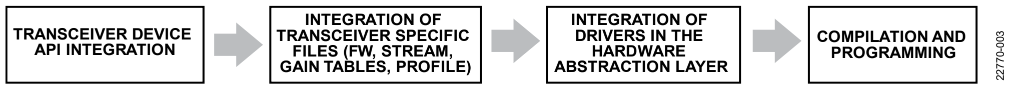
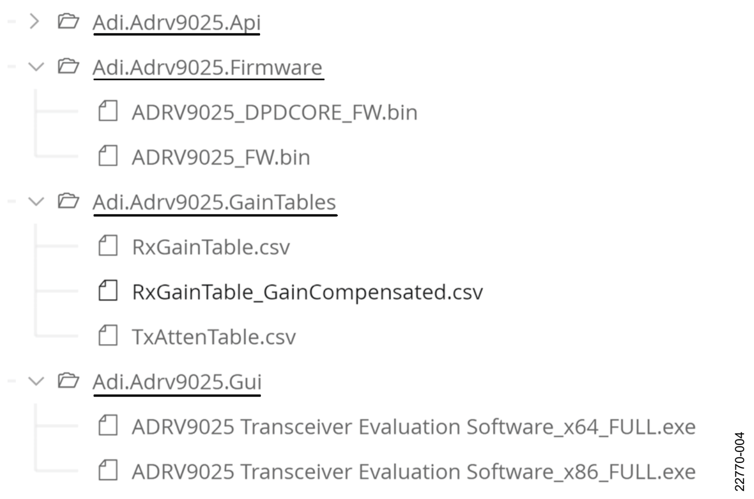

TRANSCEIVER DEVICE API INTEGRATION
INTEGRATION OF TRANSCEIVER SPECIFIC FILES (FW, STREAM, GAIN TABLES, PROFILE)
INTEGRATION OF DRIVERS IN THE HARDWARE ABSTRACTION LAYER
COMPILATION AND PROGRAMMING
22770-003

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 the following four 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.



```
> Adi.Adrv9025.Api
v Adi.Adrv9025.Firmware
    ADRV9025_DPDCORE_FW.bin
    ADRV9025_FW.bin
v Adi.Adrv9025.GainTables
    RxGainTable.csv
    RxGainTable_GainCompensated.csv
    TxAttenTable.csv
v Adi.Adrv9025.Gui
    ADRV9025 Transceiver Evaluation Software_x64_FULL.exe
    ADRV9025 Transceiver Evaluation Software_x86_FULL.exe
```

Figure 4. Software Package Folder Structure

API SOFTWARE ARCHITECTURE

The API architecture is implemented as three main layers, as shown in Figure 5. This section describes how to use the API in a custom embedded 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.

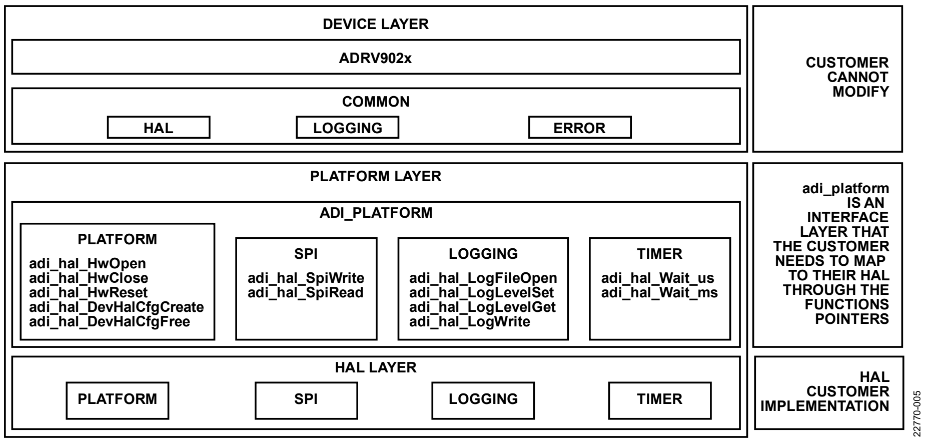


Figure 5. Software Integration

IMPLEMENTING HARDWARE ABSTRACTION INTERFACE

Users that 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 must access the target platform hardware. The Analog Devices HAL is defined by adi_platform.h. The implementation of this interface is platform dependent and is implemented by the developer in a platform specific subfolder. The prototypes of the required functions defined in adi_platform.h may not be modified, because 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 must 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 devices in a single system, the application layer code must 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 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 devices. The devHalInfo structure defined by the developer identifies which physical 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 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 device.

The devStateInfo member is of the adi_adrv9025_Info_t structure type in the C programming 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 a 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 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 that the API places typedef definitions in files with _types.h suffixes, such as ADRV9025_types.h. These _types.h files are included within their corresponding .h files and do not need to be manually included in the application layer code.

Note that the ADRV9025_user.h contains 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 receiver gain table, transmitter 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 automatic gain control (AGC) functionality

API Error Handling and Debug

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

- Warning actions are those that do not have an impact at the time of executing the device API, but can cause performance issues or logging problems. The value of these 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 these actions are negative.

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

- errSource: current source of error detected, indicating the source file where the error occurred.
- 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 must be taken due to a possible error in the API function call. The error structure contains further information 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/I^{2}C/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 the following different blocks:

- Parameter
  - Parameter either passed to function or member of structure
  - This action can be assigned to set a feature/module/interface when it is not configured correctly
- Feature (parts of a module or device)
  - GPIO control for transmitter attenuation
  - General purpose interrupt
  - ARM initial calibrations
  - ARM tracking calibrations
  - ARM control
  - AGC control
  - Power amplifier protection

- Module (individual blocks that are contained in the device that are to contain features)
  - ARM
  - Caching/merging/streaming
- Interface:
  - Device interface
  - SPI/I^{2}C/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.

The following procedure is the suggested application layer action:

1. Set PA and other RF front-end components in powered down 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 rerun 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 returned an error in any interface. Further information can be extracted by reading the error structure, which contains extended information about the error.

The following issues 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.

The following are 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. When 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 software passes the 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.

The following recommended sequence is to implement the suggested application layer actions:

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.

The following recommended sequence is to implement the suggested application layer actions:

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, perform a reconfiguration of the same feature.

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 in any of the following modules:

- ARM processor module that requires a complete reset and reload of the ARM firmware. This type of action may be required if the communication interface to the 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 power amplifier and other RF front-end components in powered down 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.

The following is the recommended sequence to implement the suggested application layer actions:

1. Put system hardware in safe state.
   a. Set the power amplifier and other RF front-end components in powered down state.
   b. Hard reset transceiver 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 the /c_src/devices/* folder other than changing the adi_platform.c and 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 multithreaded 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 enter wait/sleep state 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 timeout period the status of the transceiver is polled. The frequency of 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 after 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 transmitter to observation receiver external signal routing	1000000	100000
GETTXLOLSTATUS_	Status of transmitter local oscillator leakage external tracking cal	1000000	100000
GETTXQECSTATUS_	Status of transmitter QEC tracking cal	1000000	100000
GETRXQECSTATUS_	Status of receiver QEC tracking cal	1000000	100000
GETORXQECSTATUS_	Status of observation receiver QEC tracking cal	1000000	100000
GETRXHD2STATUS_	Status of receiver 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 interface 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 transceiver. 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 user guide and in the documentation included with the software package. This section contains descriptions and parameters that explain the specifics of this interface.

SPI BUS SIGNALS

The SPI bus consists of the four signals described in this section.

$\overline{\text{CS}}$

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

Forcing $\overline{\text{CS}}$ 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 $\overline{\text{CS}}$ deasserting are 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 $\overline{\text{CS}}$ 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 transceiver.

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 transceiver is the SPI_DIO pin. SDO is the data output from the transceiver 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 transceiver. SDIO carries the control field from the baseband processor to the transceiver 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 transceiver 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 $\overline{\text{CS}}$ input is high. The transceiver 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 scalable source routing (SSR) protocols. The control field width for this transceiver is limited to 16 bits, and multibyte IO operation is allowed. This device cannot be used to control other devices on the bus, it only operates as a 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 transceiver. The control word provides the serial port controller with information regarding the data field transfer cycle, which is Phase 2 of the communication cycle. The Phase 1 control field defines whether the upcoming data transfer is read or write. It also defines the register address being accessed.

Phase 1 Instruction Format

The 16-bit control field contains the information in Table 7.

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.

Phase 2 data transfer is performed in 8 bit words. Both single byte and multibyte transfers can be configured using the API, as described in the SPI Configuration Using API Function section.

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 transceiver with different SPI controller configurations by configuring member values of the adi_adrv9025_SpiSettings_t data structure. The adi_adrv9025_SpiSettings_t data structure parameters are listed in Table 8. Any value that is not listed in Table 8 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 that this parameter is 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 that this parameter is not implemented in the Analog Devices platform layer.	
autoIncAddrUp	0x00	Autoincrement. Functionality intended to be used with SPI streaming. Sets address autoincrement -> next addr = addr − 4. Note that this parameter is not implemented in the Analog Devices platform layer.	0x01
	0x01	Autodecrement. Functionality intended to be used with SPI streaming. Sets address autodecrement -> next addr = addr + 4. Note that this parameter is 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 that 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.	

Structure Member	Value	Function	Default
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, $\overline{\text{CS}}$ goes active low, the SCLK signal activates, and the address is transferred from the baseband processor to the transceiver. This mode is always used in direct communication between the baseband processor and the transceiver.

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 transceiver transmits the next 8 bits LSB to MSB.

In MSB mode, the first bit transmitted is the R/$\overline{\text{W}}$ 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 transceiver 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 D …) until the $\overline{\text{CS}}$ 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 transceiver. 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 Figure 6, the value 0x55 is written to Register 0x00A. In Figure 7, Register 0x00A is read and the value returned by the transceiver 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.

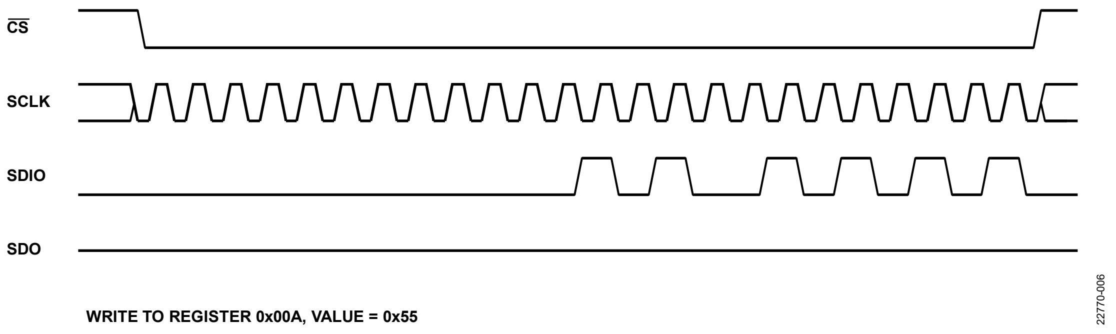


Figure 6. Nominal Timing Diagram, SPI Write Operation

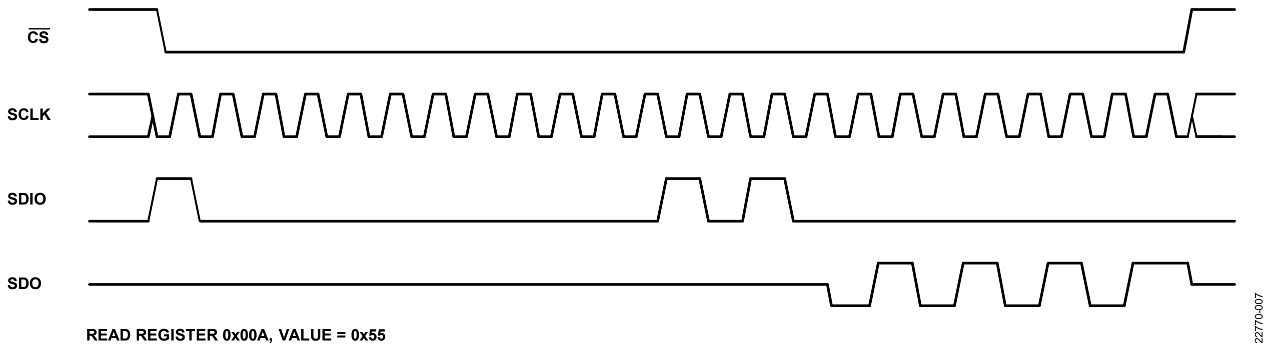


Figure 7. Nominal Timing Diagram, SPI Read Operation

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 timing parameters marked. Note that this is a single read operation. Therefore, the bus ready parameter after the data is driven from the transceiver (t_{HZS}) is not shown in Figure 8. Note that the byte to byte delay time (t_{INT}) is also not shown in Figure 8 because Figure 8 only shows a single byte write operation.

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 transceiver drives the last data bit
t_{INT}			400 ns	Byte to byte delay time during any single read or write operation

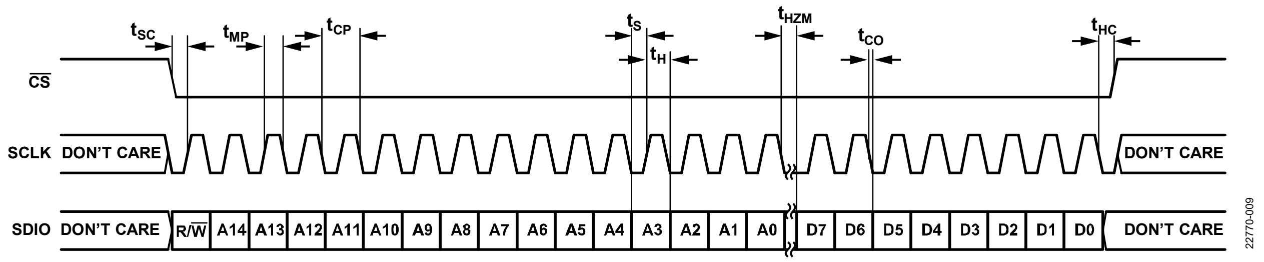


Figure 8. 3-Wire SPI Timing with Parameter Labels

SYSTEM INITIALIZATION

This section provides information about the initialization process for the transceiver utilizing the API developed by Analog Devices. Each subsection describes 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. Details about API integration and the hardware abstraction interface 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 transceiver configuration, calibration, and control. The user is required to insert the 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 and JESD204C interface, data path control, and various system checks governed by the application.

The initialization process consists 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).

The following steps are the pre-multichip synchronization (MCS) initialization sequence:

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 transmitter attenuation tables
    h. Load stream binary
    i. Load ARM binary
    j. Write initialization structure/receiver/transmitter profile info into ARM memory
    k. ARM run = 1
    l. Wait for ARM boot to complete
    m. Verify ARM checksum
2. ARM configuration
    a. Receiver/transmitter channel configuration (all half-band filter enables, clock dividers)
    b. Clock PLL and SERDES PLL configuration
    c. JESD204B and JESD204C configuration
    d. ARM switches to clock PLL output after PLL locked

The following steps are the post MCS initialization sequence:

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. Run ARM init calibrations
3. Enable tracking cals
    a. Enable radio control pin mode or not
    b. Setup any GPIO for ARM/streams

The system is now ready.

SERIALIZER/DESERIALIZER (SERDES) INTERFACE

The transceiver employs a SERDES high speed serial interface based on the JESD204B and JESD204C standards to transfer ADC and DAC samples between the transceiver and a baseband processor. The transceiver can support high speed serial lane rates up to 24.33 Gbps. An external clock distribution solution provides a device clock and SYSREF to both the transceiver and the baseband processor. The SYSREF signal ensures deterministic latency between the transceiver and the baseband processor.

Note that the initialization sequence of the device is critical to guarantee deterministic latency. Specifically, the ARM init calibrations must be run before the SERDES links are established, as described in the Initialization Sequence section. It is also imperative to check the FIFO depth after the link has been established. Major blocks in the interface include clock distribution, SERDES framer, and SERDES deframer.

JESD204B AND JESD204C STANDARD

The JESD204B and JESD204C 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 one direction of data of the octets and descrambles the other direction of data of the octets, spreading the spectral peaks to reduce EMI. The data link layer handles link synchronization, setup, and maintenance. The data link layer also 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.

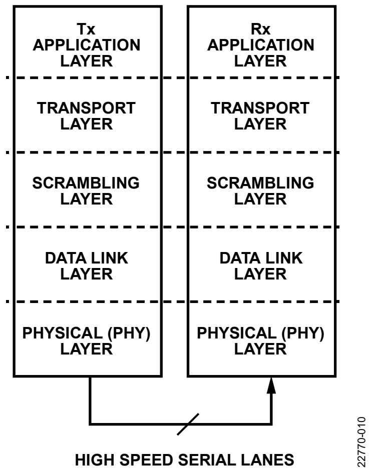


Figure 9. Key Layers of the JESD204B and JESD204C Standard

Figure 10 and Figure 11 illustrate how the JESD204B and JESD204C transmit and receive protocols are implemented.

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

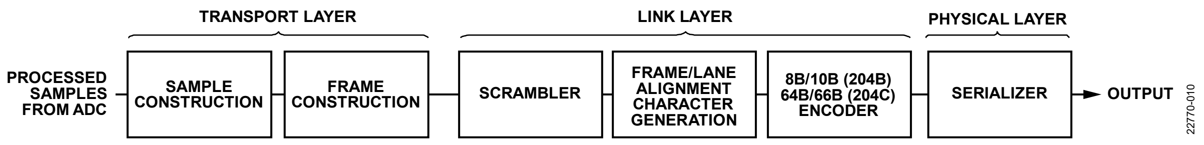


Figure 10 JESD204B and JESD204C Framer (JTX)

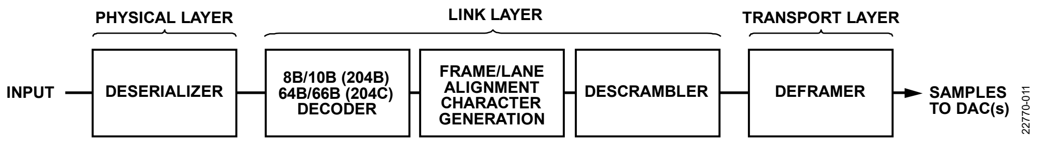


Figure 11 JESD204B and JESD204C Deframer (JRX)

DIFFERENCES BETWEEN JESD204B AND JESD204C

The initial revision of the interface provided supports both single and multiple lanes per converter 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 the same as the JESD204B link layer.

In the 8-bit/10-bit link layer, the data is organized into multiframes where in the 64-bit/66-bit link layer data is organized into multiblocks of 32 blocks where each block contains 8 octets. In the 8-bit/10-bit link layer, phase synchronization is done by the local multiframe clock (LMFC) where 64-bit/66-bit uses the local extended multiblock clock (LEMC). In the 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 LMFC or 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 the 64-bit/66-bit link layer, the alignment between multiple converter devices is done by the alignment of the LEMC to an external signal SYSREF/multireference 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 release buffer delay (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 synchronization header stream, which transmits the information parallel to the user data. This information is encoded using the synchronization header portion of the 66-bit word block. One synchronization header per block is decoded to a single bit, and 32 of these bits across a multiblock makes a 32-bit synchronization word. The synchronization 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, JESD204B uses the SYNC interface for synchronization and error reporting. The 64-bit/66-bit encoding synchronization 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 transceiver allows the SERDES to be driven 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 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).

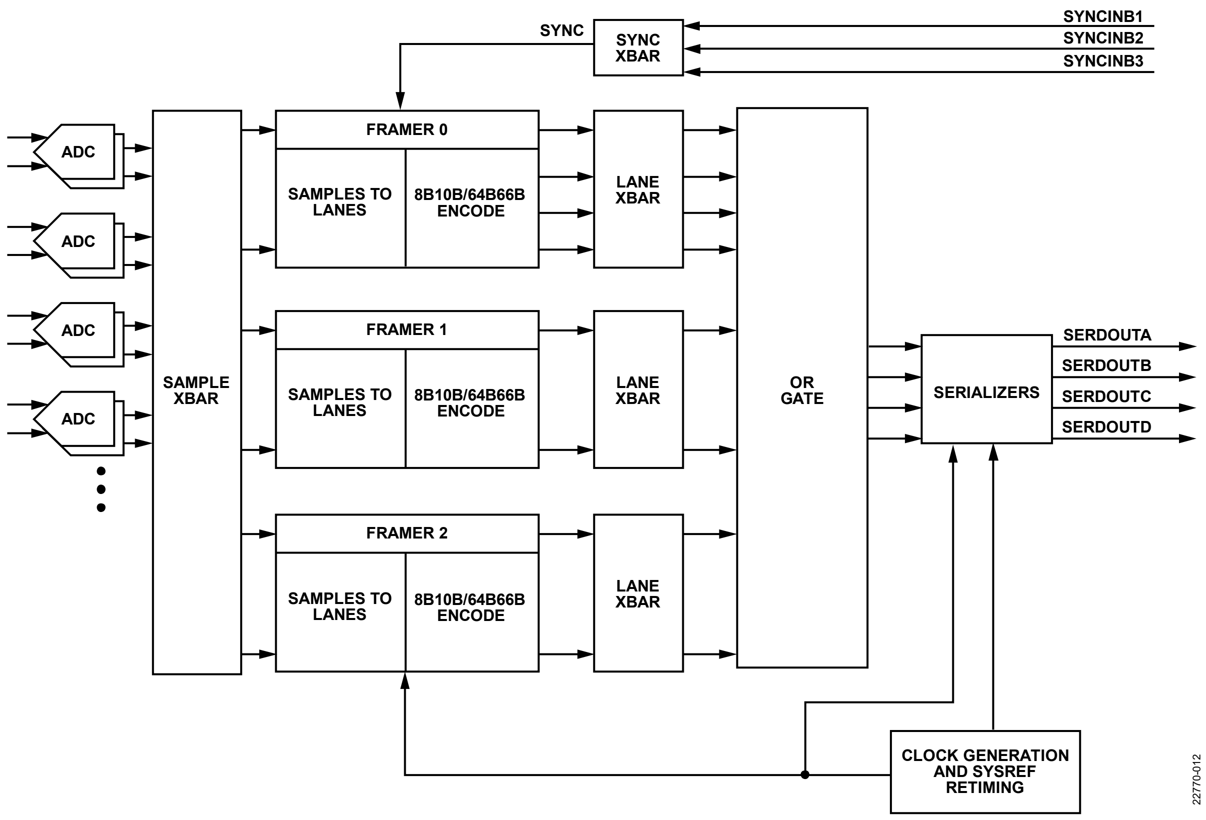
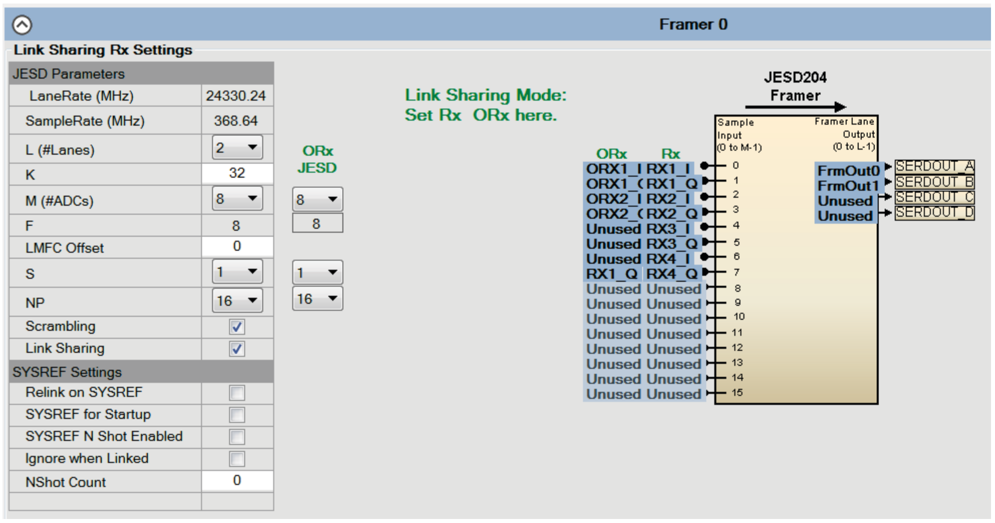


Figure 12. High Level JESD204B/JESD204C Interface Block Diagram (Receiver Only)

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 and observation receiver data streams, either separately or utilizing link sharing (receiver and observation receiver data time multiplexed on the same lane according to the receiver and transmitter frame timing) for up to four lanes. To assist in debugging, the framers contain 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 transceiver 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 Framer 0. Framer 1 and Framer 2 are not needed and are unused. This profile is a 25G 204C profile.



Framer 0

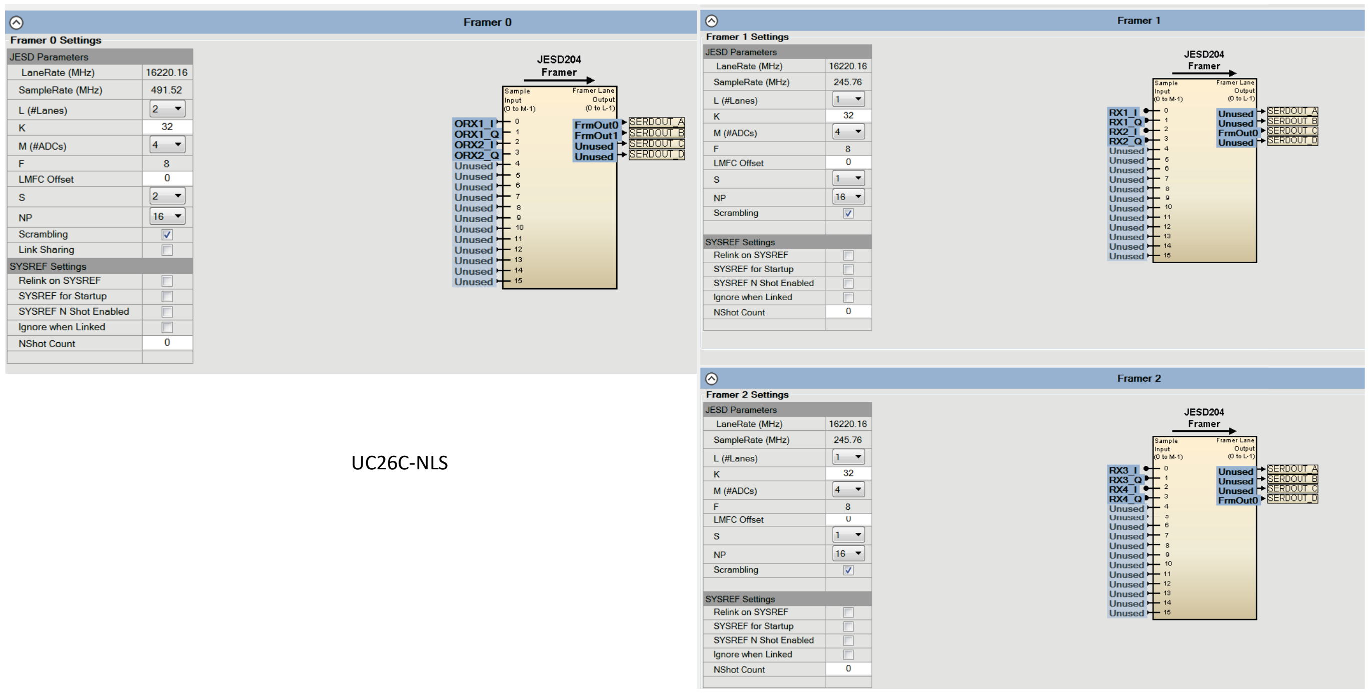
Link Sharing Rx Settings

JESD Parameters		ORx JESD
LaneRate (MHz)	24330.24	
SampleRate (MHz)	368.64	
L (#Lanes)	2	
K	32	
M (#ADCs)	8	8
F	8	8
LMFC Offset	0	
S	1	1
NP	16	16
Scrambling	✓	
Link Sharing	✓	
SYSREF Settings		
Relink on SYSREF	☐	
SYSREF for Startup	☐	
SYSREF N Shot Enabled	☐	
Ignore when Linked	☐	
NShot Count	0	

UC83C-LS

22770-013

Figure 13. Example Framer Configuration for UC83C-LS



Framer 0 Settings

JESD Parameters	
LaneRate (MHz)	16220.16
SampleRate (MHz)	491.52
L (#Lanes)	2
K	32
M (#ADCs)	4
F	8
LMFC Offset	0
S	2
NP	16
Scrambling	✓
Link Sharing	☐
SYSREF Settings	
Relink on SYSREF	☐
SYSREF for Startup	☐
SYSREF N Shot Enabled	☐
Ignore when Linked	☐
NShot Count	0

Framer 1 Settings

JESD Parameters	
LaneRate (MHz)	16220.16
SampleRate (MHz)	245.76
L (#Lanes)	1
K	32
M (#ADCs)	4
F	8
LMFC Offset	0
S	1
NP	16
Scrambling	✓
SYSREF Settings	
Relink on SYSREF	☐
SYSREF for Startup	☐
SYSREF N Shot Enabled	☐
Ignore when Linked	☐
NShot Count	0

Framer 2 Settings

JESD Parameters	
LaneRate (MHz)	16220.16
SampleRate (MHz)	245.76
L (#Lanes)	1
K	32
M (#ADCs)	4
F	8
LMFC Offset	0
S	1
NP	16
Scrambling	✓
SYSREF Settings	
Relink on SYSREF	☐
SYSREF for Startup	☐
SYSREF N Shot Enabled	☐
Ignore when Linked	☐
NShot Count	0

UC26C-NLS

22770-014

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 observation receiver must have the data in a specific format (IIQQ). Framer 0 has more flexibility than the other two framers. For this case Framer 0 is used to format the observation receiver data as needed, and the other two framers are used to route the receiver data on the lanes. This is a 16G JESD204C profile.

The transport and link layers for JESD204B/JESD204C are performed in the framers. This transceiver 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 or 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×). 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 8B/10B or 64B/66B 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. After 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 preemphasis of the physical signal to help combat bit errors due to various PCB trace lengths.

Supported Framer Link Parameters

This transceiver 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, Framer 1 and Framer 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, ($20 \leq F \times K \leq 256$), $F \times K$ 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 bit rate 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	$1.00 \times V_{TT}$
1	$0.85 \times V_{TT}$
2	$0.75 \times V_{TT}$
3	$0.50 \times V_{TT}$

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 preemphasis is used because it is simpler to realize bit delays with flip flops than trying to implement analog delays at the receiver.

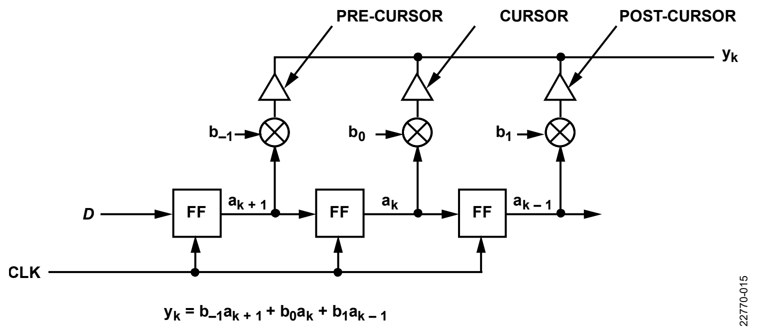


Figure 15. Serializer Emphasis Implementation

This serializer preemphasis circuit allows boosting the amplitude anytime the serial bit changes state. If no bit transition occurs, the amplitude is left unchanged. Preemphasis helps open the eye for longer PCB traces or when the parasitic loading of connectors has a noticeable effect. In most cases, to find the best setting, a simulation or measurement of the eye diagram with a high-speed scope at the receiver is recommended, or as mentioned above an internal eye monitor after the equalizer is the optimum solution. The serializer preemphasis is controlled by setting 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 of 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, and S = 1 as an example.

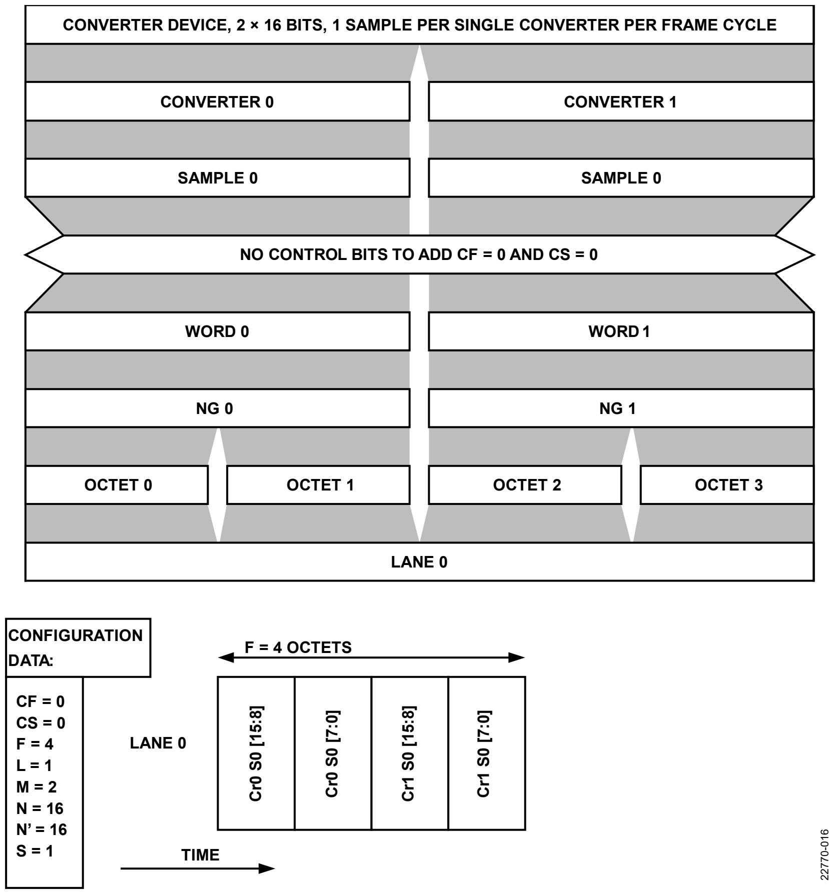


Figure 16. Framer Data Packing for M = 2, L = 1, and S = 1

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 transceiver to output serial data even when the link cannot be established. With this mode enabled, the serializer amplitude and preemphasis 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.

The following list is the typical PRBS generator usage sequence:

1. Initialize the device as outlined in the Link Initialization and Debugging section
2. Run the adi_adrv9025_FramerTestDataSet(…) with the required framer, test data source set to desired PRBS order, and injection point set 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 an adequate number of samples to be transmitted, and then check the PRBS error count of the baseband processor.
5. Adjust framer amplitude and preemphasis settings and/or deframer equalization settings and repeat Step 3 and Step 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 and 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 be 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 debugging 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 to 15	JESD204B/JESD204C configuration Bank ID—extension to device ID
deviceId	0 to 255	JESD204B/JESD204C configuration Device ID—link identification number
lane0Id	0 to 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, scrambling is disabled
		If scramble = 1, scrambling is enabled
externalSysref	0, 1	External SYSREF enabled
		If externalSysref = 0, use internal SYSREF
		If externalSysref = 1, 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, $\overline{\text{SYNCIN0}}$ is connected to the framer
		If syncinbSelect = 1, $\overline{\text{SYNCIN1}}$ is connected to the framer
		If syncinbSelect = 2, $\overline{\text{SYNCIN2}}$ 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, bit repeat mode is selected
		If oversample = 1, 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 polarity not inverted, 1 = $\overline{\text{SYNC}}$ LVDS polarity inverted
newSysrefOnRelink	0, 1	Set the flag to determine if SYSREF is set on relink, where, if >0 = set, 0 = not set
sysrefForStartup	0, 1	1 = framer: require a SYSREF before code group synchronization (CGS) is output from serializer, 0: Allow CGS to output before SYSREF occurs (recommended on framer to allow deframer clock data recovery (CDR) to lock and equalization 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 the JESD204B and JESD204C 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 PRBS23 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/local extended multiblock clock (LEMF), 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 synchronization state machine still does not allow the external SYSREF to reach the framer until the other stages of multichip synchronization have completed. As long as the external SYSREF is correctly configured before performing MCS, this function may not be needed by the baseband processor, 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. adi_adrv9025_FramerSysrefCtrlSet(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure
framerSelMask	Select framer to enable/disable SYSREF input for (valid for 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. General API Function 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

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 has 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 receiver JESD204B/JESD204C link(s) must be configured and running to use this function

Dependencies

device->devHalInfo

Parameters

Table 21. adi_adrv9025_FramerStatusGet(…) Parameters

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

See Table 19.

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 the receiver framer PRBS generation. This is a debug function 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 may be called any time after device initialization.

Dependencies

device->devHalInfo

Parameters

Table 22. adi_adrv9025_FramerTestDataSet(…) Parameters

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

Return Values

See Table 19.

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 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 23. adi_adrv9025_FramerTestDataInjectError(…) Parameters

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

Return Values

See Table 19.

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 must 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 24. adi_adrv9025_FramerLinkStateSet(…) Parameters

Parameter	Description
*device	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

See Table 19.

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 transceiver 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 Deframer 0 and utilizes four lanes at 16G to support four transmitters at maximum bandwidth.

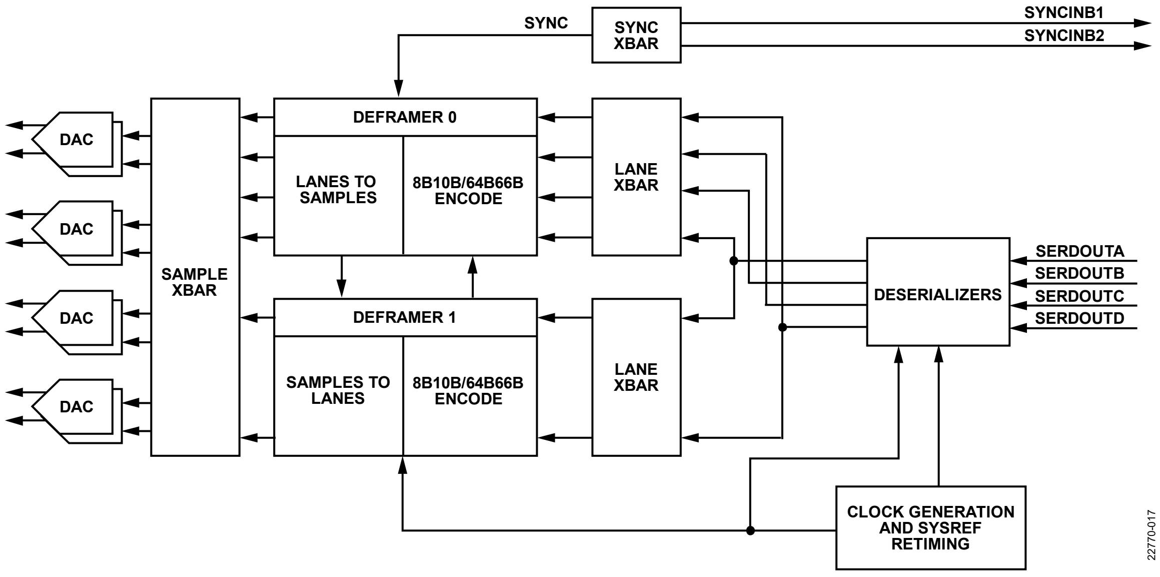
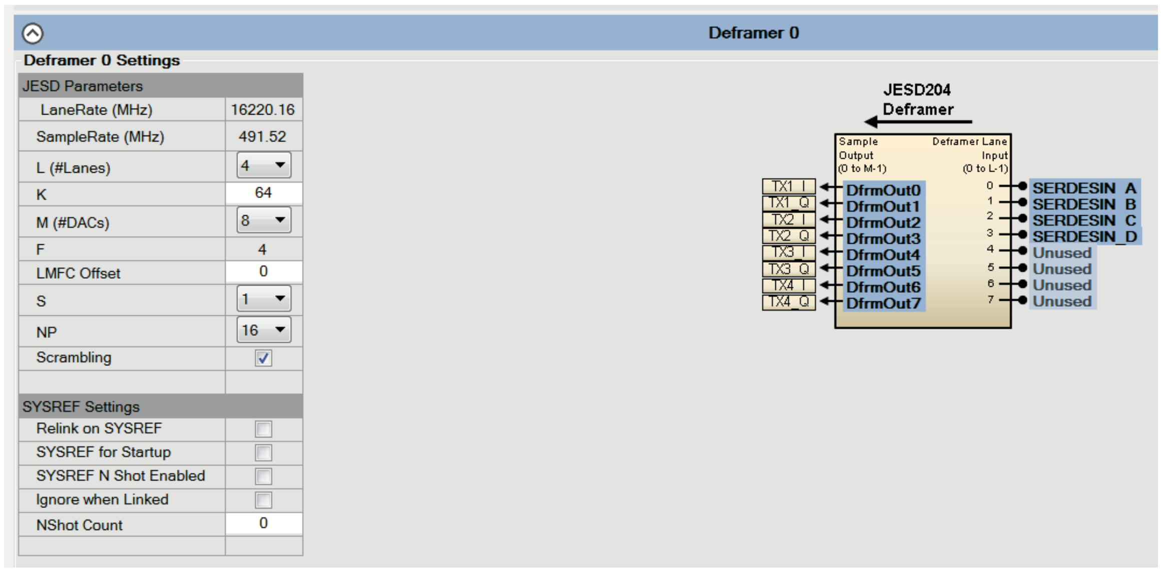


Figure 17. High Level JESD204B/JESD204C Interface Block Diagram (Transmitter Only)



Deframer 0 Settings

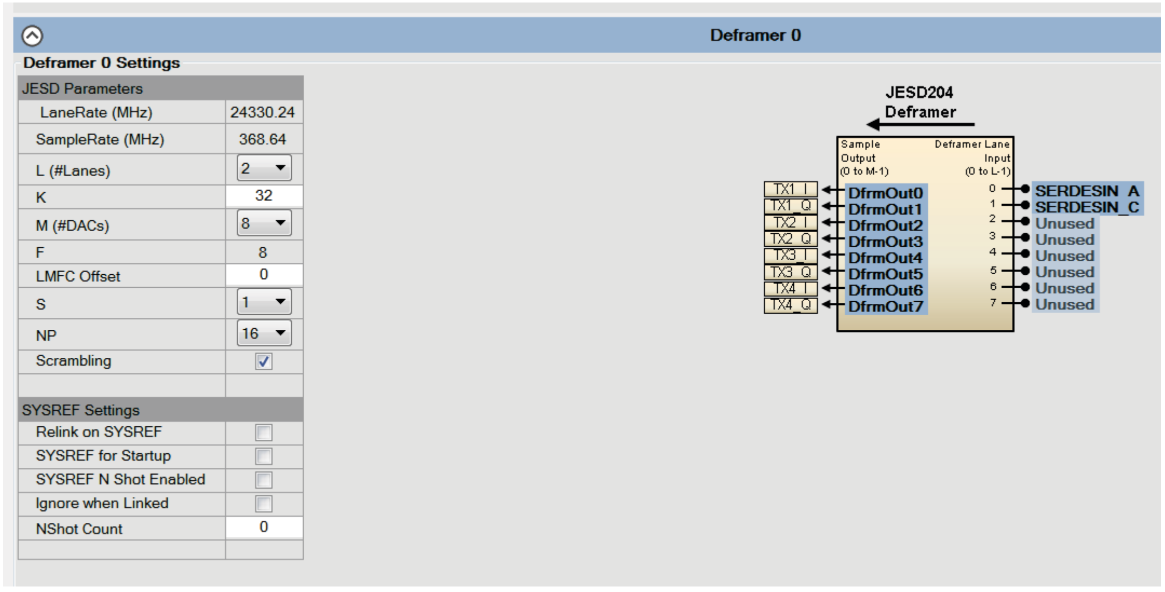
JESD Parameters	
LaneRate (MHz)	16220.16
SampleRate (MHz)	491.52
L (#Lanes)	4
K	64
M (#DACs)	8
F	4
LMFC Offset	0
S	1
NP	16
Scrambling	✓
SYSREF Settings	
Relink on SYSREF	☐
SYSREF for Startup	☐
SYSREF N Shot Enabled	☐
Ignore when Linked	☐
NShot Count	0

22770-018

UC26C-NLS

Figure 18. Example Deframer Configuration for UC26C-NLS

Figure 19 shows the configuration for UC83C-LS that uses Deframer 0. Only two lanes are needed to realize the maximum chip RF bandwidth (450 MHz) across all four transmitters. 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.



Deframer 0 Settings

JESD Parameters	
LaneRate (MHz)	24330.24
SampleRate (MHz)	368.64
L (#Lanes)	2
K	32
M (#DACs)	8
F	8
LMFC Offset	0
S	1
NP	16
Scrambling	✓
SYSREF Settings	
Relink on SYSREF	☐
SYSREF for Startup	☐
SYSREF N Shot Enabled	☐
Ignore when Linked	☐
NShot Count	0

22770-019

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 25. 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 ≤ F × K ≤ 256), F × K must be a multiple of 4, K ≤ 32
E	JESD204C only: number of multiblocks in an extended multiblock.

For a particular converter sample rate, not all combinations listed in Table 25 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 nonadaptive, programmable equalizer. This helps in compensating for signal integrity distortions for each channel due to PCB trace length and impedance. Table 26 summarizes the amount of insertion loss each equalizer setting can overcome. Equalizer boost settings can range from 0 (maximum boost) to 3 (default).

Table 26. Deserializer EQ Boost Correction

EQ Boost Settings	Boost (dB)
0	0
1	−3
2	−6
3	−12

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

The adi_adrv9025_DesCfg_t data structure contains the information required to properly configure the deserializer. Details of each member can be found in the 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 transceiver uses passive equalizer architecture that deemphasizes 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 27.

Table 27. Deserializer Equalizer Data Members

Structure Member	Description
desInvertLanePolarity	Deserializer lane polarity inversion select. Bit 0 = invert polarity of Lane 0, Bit 1 = invert polarity of Lane 1.
desEqBoostSetting	It sets how much high frequency attenuation the user is trying to compensate.
desEqGainSetting	Gain is setting the number of stages of limiting amplifier. This compensates for the amount of EqBoost added.

Structure Member	Description
desEqFeedbackSetting	This is the amount of feedback set for each gain stage. The gain stage works like a basic operational amplifier, 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) and can cause peaking in the total channel response. It is not recommended to tune this data member while compensating for insertion losses.

When operating in JESD204C mode, the equalization is done with a continuous time linear equalizer (CTLE) 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. The following is a list of the functions of the deframer:

- Monitor the health of the JESD204B/JESD204C link
- Control the JESD204B/JESD204C interrupt signal (can output on a GP_INTx pin on the device general purpose interrupt pin) to signal baseband processor when certain JESD204B/JESD204C error conditions arise.
- Remove character replacement (valid for only JESD204B).
- Perform 8B10B/64B66B decoding.
- 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, and S = 1 as an example.

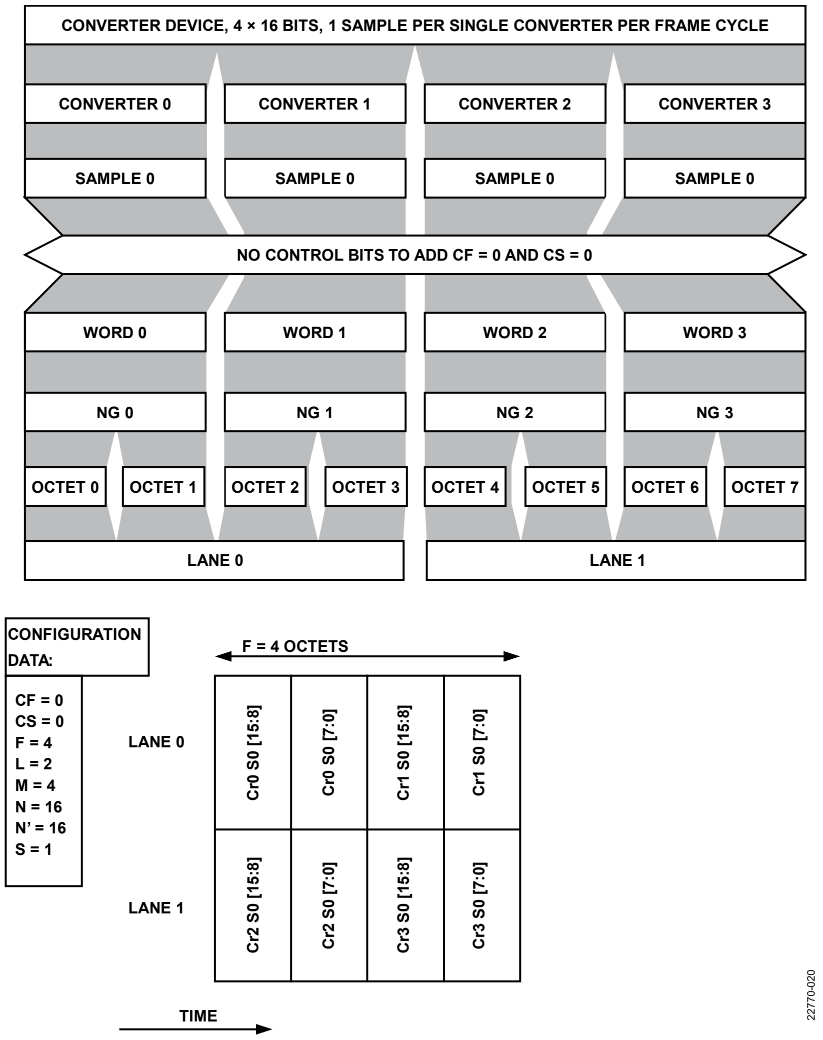


Figure 20. JESD204B Deframer Configuration (M = 4, L = 2)

Other Useful Deframer IP Features

PRBS Checker

The deframer has a built in PRBS checker. The PRBS checker can self synchronize and check for PRBS errors on a PRBS7, PRBS15, or PRBS31 sequence. Because 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 baseband processor 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 Deserializer Configuration 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 28. 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 to 15	JESD204B/JESD204C Configuration Bank ID (extension to device ID)
deviceId	0 to 255	JESD204B/JESD204C Configuration Device ID (link identification number)
lane0Id	0 to 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 to 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 to 255	JESD204C E parameter
Scramble (JESD204B Only)	0, 1	Scrambling enabled If scramble = 0, scrambling is disabled If scramble = 1, scrambling is enabled
externalSysref	0, 1	External SYSREF enabled If externalSysref = 0, use internal SYSREF If externalSysref = 1, 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 multiframe 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 reestablish the link If newSysrefOnRelink = 0, no SYSREF is required If newSysrefOnRelink = 1, SYSREF is required
enableManualLaneXbar	0, 1	$\overline{\text{SYNC}}$ selection: selects which $\overline{\text{SYNCOUT}}$ output is driven by the deframer If syncbOutSelect = 0, the deframer drives $\overline{\text{SYNCOUT0}}$ If syncbOutSelect = 1, 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 to determine if SYSREF is set 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 equalization 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 the JESD204B and JESD204C 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 structure member can be found in the 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 29. 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 preemphasis 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 30.

Table 30. 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 31.

Table 31. 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 32.

Table 32. 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, a SYSREF rising edge is required. The external SYSREF signal at the input pins on the device can be gated off internally. Therefore, 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 synchronization state machine still does not allow the external SYSREF to reach the deframer until the other stages of multichip synchronization have completed. As long as the external SYSREF is correctly configured before performing MCS, this function may not be needed by the baseband processor because 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 33. adi_adrv9025_DeframerSysrefCtrlSet(…) Parameters

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

See Table 19.

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 must 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. Therefore, the device CDR is locked.

Precondition

This function can be called any time after device initialization.

Dependencies

device->devHalInfo

Parameters

Table 34. adi_adrv9025_DfrmLinkStateSet(…) Parameters

Parameter	Description
*device	Pointer to the device settings data structure
deframerSelMask	Desired deframer to reset. Valid states are 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

See Table 19.

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 35.

Table 35. Deframer Status Parameters

Deframer Status Bit	Bit Name	Description
7	Valid Checksum	= 1 if the checksum calculated by the device matched the checksum 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 there is a 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 there is a framing error during ILAS.
4	FS Lost	This bit captures the internal status of the frame symbol event of the deframer. Value = 1 if there is a 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	$\overline{\text{SYNC}}$ level	Current level of $\overline{\text{SYNC}}$ signal internal to deframer (= 1 means link is up).

Precondition

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

Dependencies

device->devHalInfo

Parameters

Table 36. adi_adrv9025_DeframerStatusGet(…) Parameters

Parameter	Description
*device	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

See Table 19.

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 transmitter JESD204B/JESD204C lanes.

If the checkerLocation parameter is ADI_ADRV9025_PRBSCHECK_LANEDATA, the PRBS is checked at the output of the deserializer. If the checkerLocation parameter 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 can be called any time after device initialization.

Dependencies

device->devHalInfo

Parameters

Table 37. adi_adrv9025_DfrmPrbsCheckerStateSet(…) Parameters

Parameter	Description
*device	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, or ADI_ADRV9025_PRBS31)
checkerLocation	Check at deserializer (deframer input) or sample (deframer output)

Return Values

See Table 19.

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 by resetting the PRBS error counters for all lanes. It is recommended to clear the error counters after enabling the deframer PRBS checker.

Precondition

The transmitter JESD204B/JESD204C link(s) must be configured to use this function.

Dependencies

device->devHalInfo

Parameters

Table 38. adi_adrv9025_DfrmPrbsCountReset(…) Parameter

Parameter	Description
*device	A pointer to the device settings structure

Return Values

See Table 19.

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 DAC 0 in turn.

Precondition

The transmitter JESD204B/JESD204C link(s) must be configured to use this function.

Dependencies

device->devHalInfo

Parameters

Table 39. adi_adrv9025_DfrmPrbsErrCountGet(…) Parameters

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

Return Values

See Table 19.

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 structure 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 40. 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 preemphasis 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

The following list contains the recommendations for implementing a JESD204B and JESD204C interface in hardware:

- 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 approximately 500 mV p-p to 700 mV p-p if insertion loss is approximately 5 dB at room temperature.
- 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 bring up procedure follows the steps outlined for both JESD204B and JESD204C configurations in the JESD204B and JESD204C subsections.

JESD204B

For the deframer side in JESD204B mode, 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 the ILAS phase.
5. If SYNC stays asserted, this indicates that the interface is stuck in the CGS phase. If the link parameters match, check the signal integrity (refer to the Sample Iron Python Code for PRBS Testing section).

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

JESD204C

For the deframer side in JESD204C mode, 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 because it is now protocol based (no SYNC signal needed).
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 procedure. First the framer is enabled and then the baseband processor deframer synchronizes to the signal.

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

FIRST TIME SYSTEM BRING UP—CHECKING LINK INTEGRITY

The following is a list of suggested actions when checking the link integrity during first time system bring up:

1. For ease of debug during bring up, it is recommended to start with a single lane on both sides and with the minimum possible link speed.
2. Check that the parameters are configured the same at both ends of the 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 transceiver as outlined in the Link Initialization and Debugging section. Enable the PRBS generator on the baseband processor with the desired PRBS sequence.
4. Confirm that the lanes baseband processor is transmitting PRBS on are the actually configured in the transceiver. Start with the PRBS errors. Ensure baseband processor and the transceiver are both using the same PRBS signal and the transceiver expects the same PRBS 7 from baseband processor.
5. Call the API adi_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 the 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 that only the first four positions are valid and the last four positions are always 0. To create errors as a test, change the 0x1 in the line immediately below the 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 transceiver. Start with the PRBS errors. Ensure that the baseband processor and the transceiver are both using the same PRBS signal and the transceiver expects the same PRBS 7 from baseband processor.

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

1. Power down the system and check 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 that differential impedance of the traces is 100 Ω.
2. Power up the system and check the following:
   a. If there is a buffer/translator in the SYNC path, make sure it is functioning properly.
   b. Check that the SYNC source is properly configured to produce compliant logic levels.
3. Check SYNC signaling using the following actions:
   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 JESD204B receiver is not decoding the samples properly. Verify /K/ characters are being sent, verify receive configuration settings, and verify the SYNC source. Consider overdriving the SYNC signal and attempt to force link into ILAS mode to isolate link receiver vs. transmitter issues.

   b. If SYNC is static and logic high, verify that 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 a 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 that the /K/ characters are okay and the basic function of the CDR is 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 Link Initialization and Debugging for troubleshooting tips.
4. Check serial data using the following actions:
   a. Verify that the transmitter data rate and the receiver expected rate are the same.
   b. Measure lanes with a high impedance probe (a differential probe, if possible). If characters appear incorrect, ensure 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 a high impedance probe. If /K/ characters are correct, the transmitter side of the link is working properly. If /K/ characters are not correct, the transmitter device or the board lane signals 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 M and L configuration parameters values match between the baseband processor and the transceiver. Otherwise, the data rates may not match. For example, for 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 that 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 that ILAS multiframes are transmitting properly and verify the link parameters on the transmitter device, the receiver device, and those parameters transmitted in the ILAS second multiframe are all valid.
   b. Calculate expected ILAS length t_{FRAME}, $t_{MULTIFRAME}$, and $4 \times t_{MULTIFRAME}$ and verify that ILAS is attempted for approximately four multiframes.
2. Verify that all lanes are functioning properly. Ensure that there are no multilane/multilink conflicts.

If the interface enters data phase but occasionally the link resets (returns to CGS and ILAS before returning to data phase), look for the following issues and make adjustments to the link parameter to remove the condition:

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

STATIC PHASE OFFSET (SPO) 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 JESD204C mode the channel is estimated during an initialization calibration that configures the CTLE 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 an SPO 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. The link is considered good overall operating conditions. The SPO test requires PRBS transmission in the FPGA and setup of the PRBS pattern checker in the transceiver device.

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

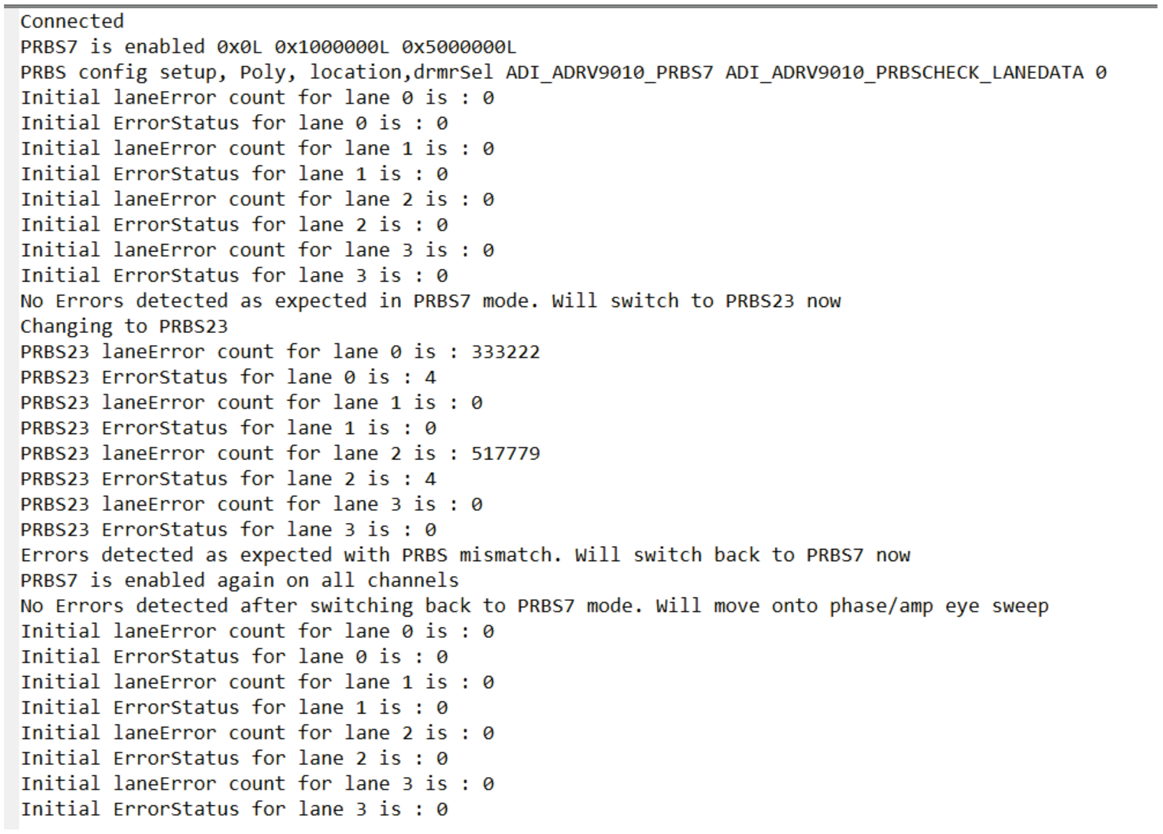
SPO Test Example Python Script

The SPO test code can be run in the GUI and works for both JESD204B and 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
```

22770-021

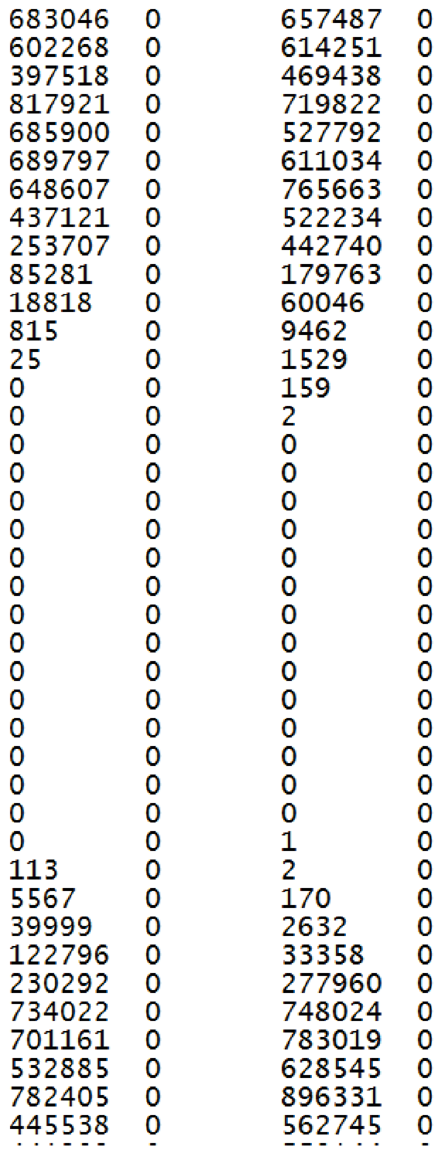
Figure 21. SPO Test Measurement Result

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 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.

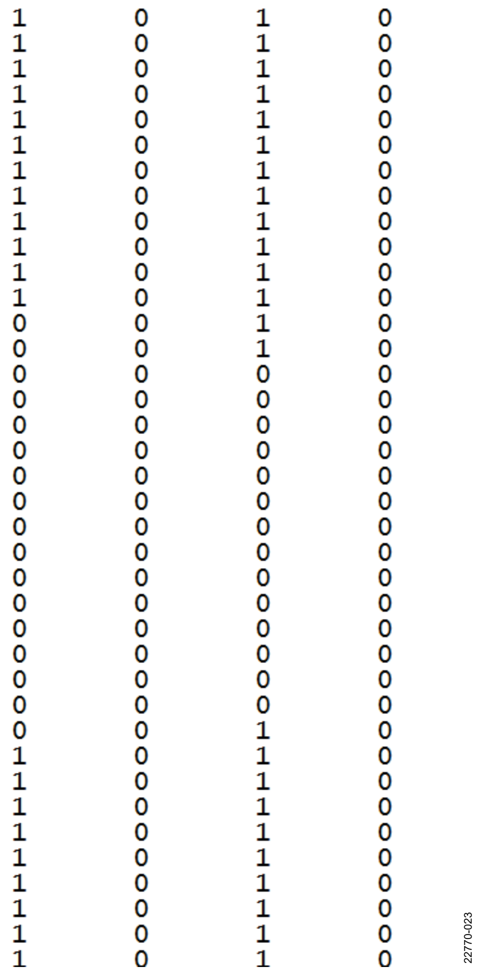
In the eyedata_lane.txt file, errors are represented by 1 and the eye indicated by 0. Similarly, the 0s occur toward the center of the waveform, indicating an acceptable eye width. Figure 22 and Figure 23 show excerpts from the center of the files.



```
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       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
```

22770-022

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



```
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
0    0    1    0
0    0    1    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    0    0
0    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
1    0    1    0
```

Figure 23. eyedata_lane.txt Showing Center of the Eye

CHECKING JESD204C LINK STATUS

JESD204C link status can be examined by using SPI commands to read the following address locations:

- 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, it is only necessary to check the first two registers in each deframer, not all four.

Table 41. Deframer Register Bit Function Assignments

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

Table 42. Deframer Register State Options

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 AND LEMC OFFSET FOR A DEFRAMER

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

Deterministic Latency in JESD204B Mode

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

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

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

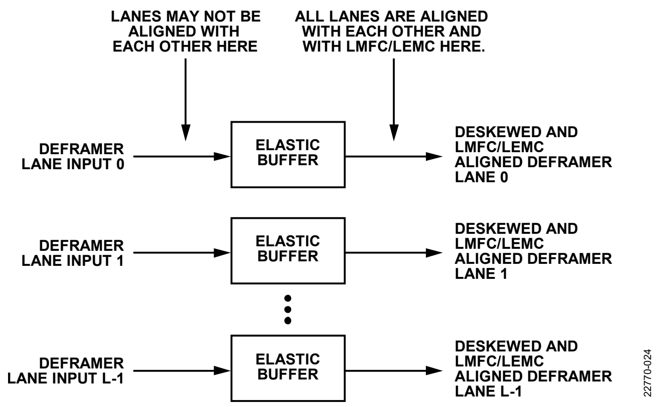

LANES MAY NOT BE ALIGNED WITH EACH OTHER HERE
ALL LANES ARE ALIGNED WITH EACH OTHER AND WITH LMFC/LEMC HERE.
DEFRAMER LANE INPUT 0
ELASTIC BUFFER
DESKEWED AND LMFC/LEMC ALIGNED DEFRAMER LANE 0
DEFRAMER LANE INPUT 1
ELASTIC BUFFER
DESKEWED AND LMFC/LEMC ALIGNED DEFRAMER LANE 1
DEFRAMER LANE INPUT L-1
ELASTIC BUFFER
DESKEWED AND LMFC/LEMC ALIGNED DEFRAMER LANE L-1
22770-024

Figure 24. Elastic Buffers in the Deframers

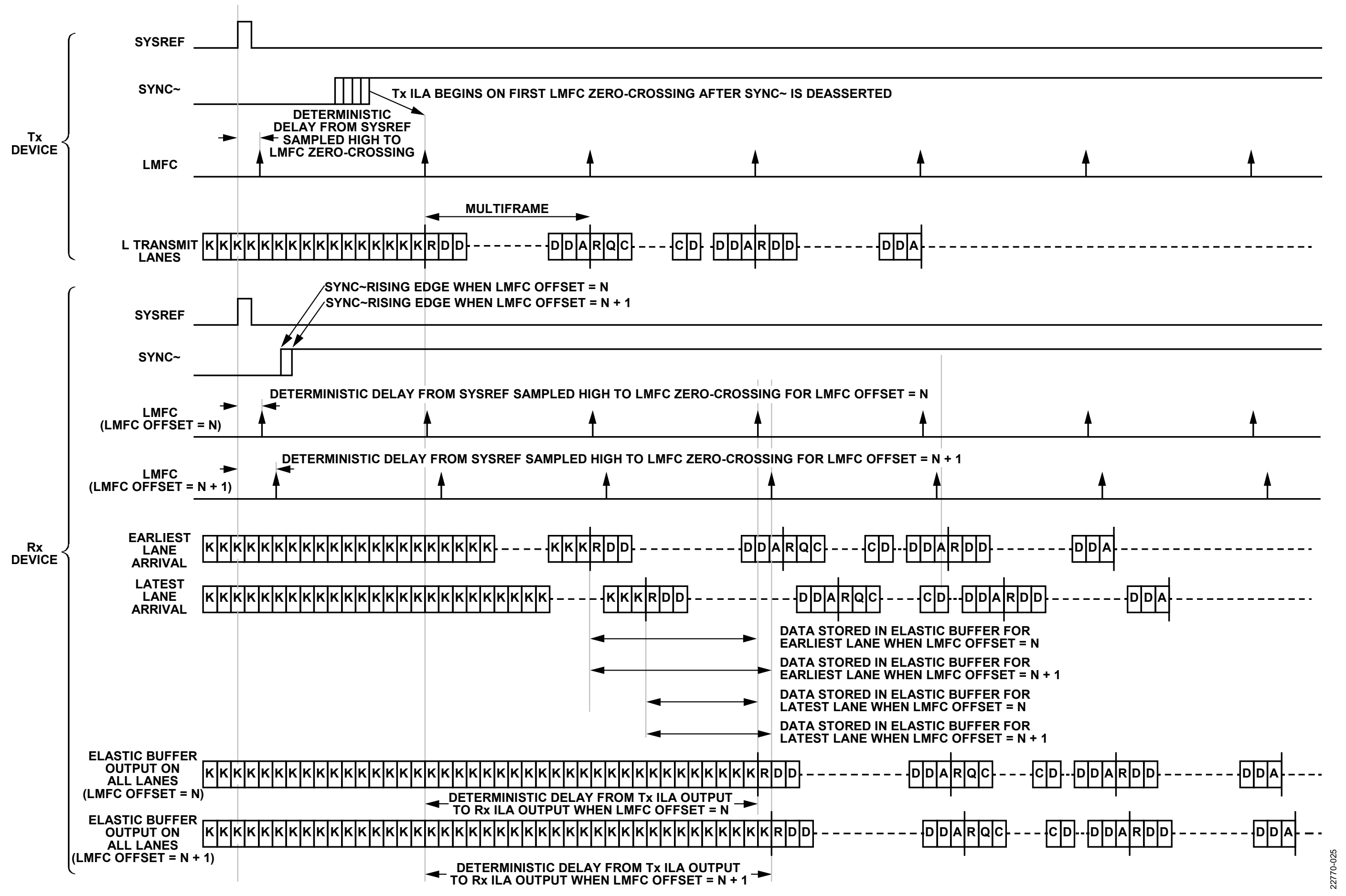

Tx DEVICE
SYSREF
SYNC~
Tx ILA BEGINS ON FIRST LMFC ZERO-CROSSING AFTER SYNC~ IS DEASSERTED
DETERMINISTIC DELAY FROM SYSREF SAMPLED HIGH TO LMFC ZERO-CROSSING
LMFC
MULTIFRAME
L TRANSMIT LANES
Rx DEVICE
SYNC~RISING EDGE WHEN LMFC OFFSET = N
SYNC~RISING EDGE WHEN LMFC OFFSET = N + 1
DETERMINISTIC DELAY FROM SYSREF SAMPLED HIGH TO LMFC ZERO-CROSSING FOR LMFC OFFSET = N
LMFC (LMFC OFFSET = N)
DETERMINISTIC DELAY FROM SYSREF SAMPLED HIGH TO LMFC ZERO-CROSSING FOR LMFC OFFSET = N + 1
LMFC (LMFC OFFSET = N + 1)
EARLIEST LANE ARRIVAL
LATEST LANE ARRIVAL
DATA STORED IN ELASTIC BUFFER FOR EARLIEST LANE WHEN LMFC OFFSET = N
DATA STORED IN ELASTIC BUFFER FOR EARLIEST LANE WHEN LMFC OFFSET = N + 1
DATA STORED IN ELASTIC BUFFER FOR LATEST LANE WHEN LMFC OFFSET = N
DATA STORED IN ELASTIC BUFFER FOR LATEST LANE WHEN LMFC OFFSET = N + 1
ELASTIC BUFFER OUTPUT ON ALL LANES (LMFC OFFSET = N)
DETERMINISTIC DELAY FROM Tx ILA OUTPUT TO Rx ILA OUTPUT WHEN LMFC OFFSET = N
ELASTIC BUFFER OUTPUT ON ALL LANES (LMFC OFFSET = N + 1)
DETERMINISTIC DELAY FROM Tx ILA OUTPUT TO Rx ILA OUTPUT WHEN LMFC OFFSET = N + 1
22770-025

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 because of 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 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 shown in Figure 24 and Figure 26.

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.

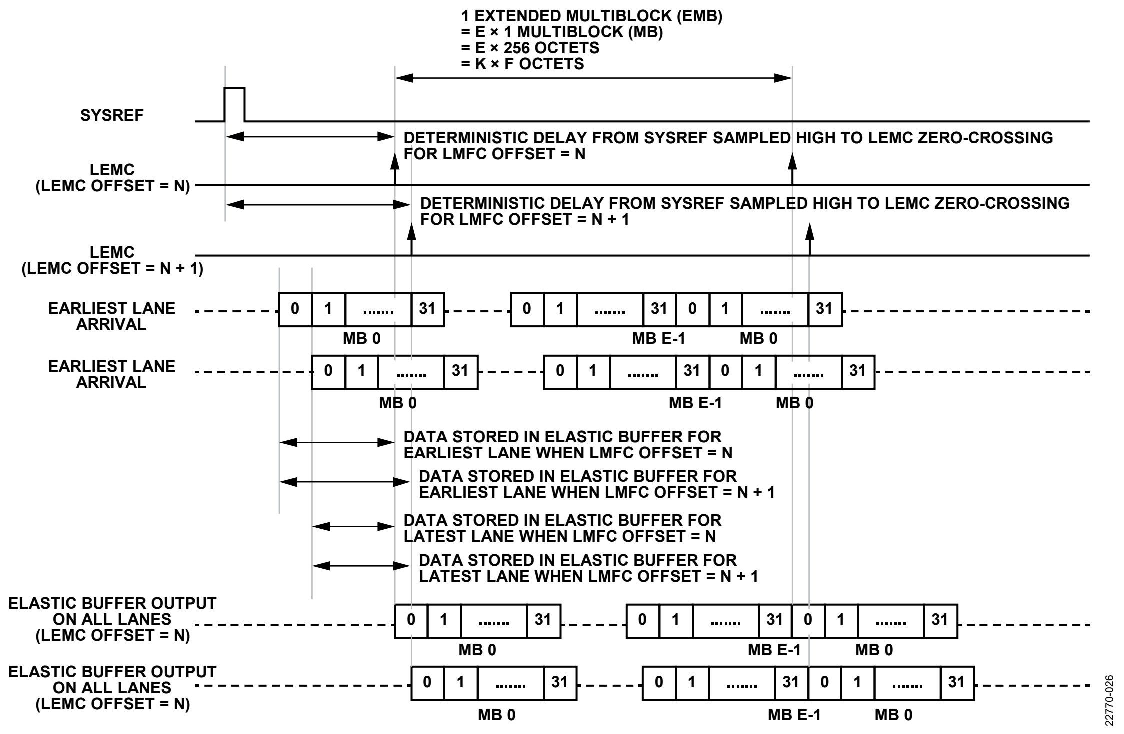

1 EXTENDED MULTIBLOCK (EMB)
= E × 1 MULTIBLOCK (MB)
= E × 256 OCTETS
= K × F OCTETS
SYSREF
DETERMINISTIC DELAY FROM SYSREF SAMPLED HIGH TO LEMC ZERO-CROSSING FOR LMFC OFFSET = N
LEMC (LEMC OFFSET = N)
DETERMINISTIC DELAY FROM SYSREF SAMPLED HIGH TO LEMC ZERO-CROSSING FOR LMFC OFFSET = N + 1
LEMC (LEMC OFFSET = N + 1)
EARLIEST LANE ARRIVAL
0
1
....
31
MB 0
MB E-1
EARLIEST LANE ARRIVAL
DATA STORED IN ELASTIC BUFFER FOR EARLIEST LANE WHEN LMFC OFFSET = N
DATA STORED IN ELASTIC BUFFER FOR EARLIEST LANE WHEN LMFC OFFSET = N + 1
DATA STORED IN ELASTIC BUFFER FOR LATEST LANE WHEN LMFC OFFSET = N
DATA STORED IN ELASTIC BUFFER FOR LATEST LANE WHEN LMFC OFFSET = N + 1
ELASTIC BUFFER OUTPUT ON ALL LANES (LEMC OFFSET = N)
ELASTIC BUFFER OUTPUT ON ALL LANES (LEMC OFFSET = N)
22770-026

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

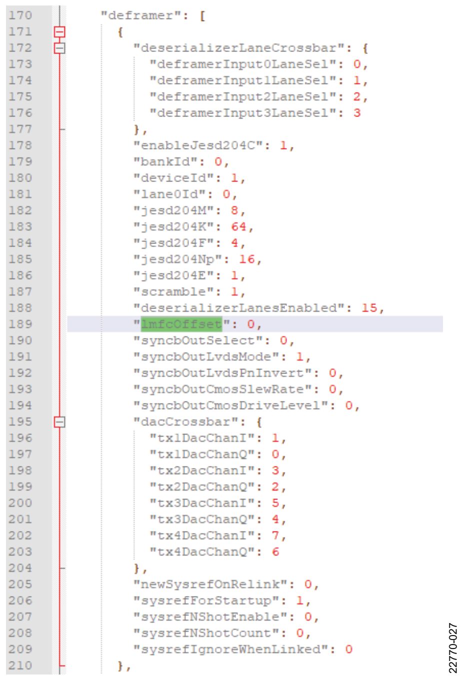
Programming the LMFC Offset for a Deframer

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

- Modify the profile file being used
- Use the adi_adrv9025_DfrmCfg data structure
- Write directly to the relevant SPI registers

Setting the LMFC/LEMC Offset in the Profile File

There is a lmfcOffset field for each of the two deframers in the profile file. This field corresponds to the LMFC offset in JESD204B mode, and corresponds to the LEMC offset in JESD204C mode. The lmfcOffset field 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).



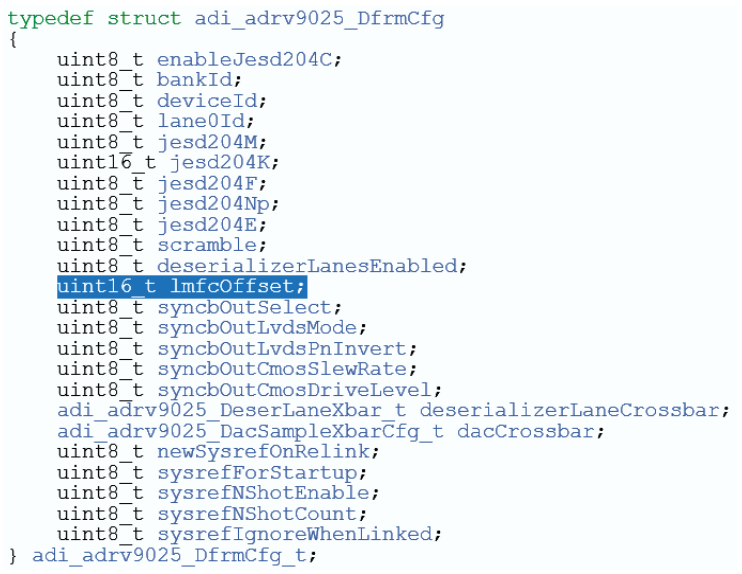
```
"deframer": [
  {
    "deserializerLaneCrossbar": {
      "deframerInput0LaneSel": 0,
      "deframerInput1LaneSel": 1,
      "deframerInput2LaneSel": 2,
      "deframerInput3LaneSel": 3
    },
    "enableJesd204C": 1,
    "bankId": 0,
    "deviceId": 1,
    "lane0Id": 0,
    "jesd204M": 8,
    "jesd204K": 64,
    "jesd204F": 4,
    "jesd204Np": 16,
    "jesd204E": 1,
    "scramble": 1,
    "deserializerLanesEnabled": 15,
    "lmfcOffset": 0,
    "syncbOutSelect": 0,
    "syncbOutLvdsMode": 1,
    "syncbOutLvdsPnInvert": 0,
    "syncbOutCmosSlewRate": 0,
    "syncbOutCmosDriveLevel": 0,
    "dacCrossbar": {
      "tx1DacChanI": 1,
      "tx1DacChanQ": 0,
      "tx2DacChanI": 3,
      "tx2DacChanQ": 2,
      "tx3DacChanI": 5,
      "tx3DacChanQ": 4,
      "tx4DacChanI": 7,
      "tx4DacChanQ": 6
    },
    "newSysrefOnRelink": 0,
    "sysrefForStartup": 1,
    "sysrefNShotEnable": 0,
    "sysrefNShotCount": 0,
    "sysrefIgnoreWhenLinked": 0
  },
```

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 initialization calibrations when programming the transceiver. Not running the init calibrations makes 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 of 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 transceiver. Not running the init cals makes 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;
```

Figure 28. LMFC Offset Field in adi_adrv9025_DfrmCfg Data Structure

Setting the LMFC/LEMC Offset Through SPI Registers Controls

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

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 JESD204B and JESD204C framer and JESD204B and JESD204C 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.

It is also 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).

Reading Back the Buffer Depths for Each Deframer Lanes

It is possible to read back the depths of the elastic buffers for each deframer lane in the Deframer 0 and Deframer 1 SPI registers of the device. The corresponding registers for Deframer 0 and Deframer 1 are:

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, Bits[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 of 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, because 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 predetermined 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 the elastic buffers if the buffer protection is active in the Deframer 0 and Deframer 1 SPI registers.

Table 43. 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 are too close with the chosen LMFC/LEMC offset setting. A predetermined buffer depth is used. Deterministic latency is lost.

Table 44. 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 are 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 Deframer 0 and Deframer 1 SPI register bits.

Table 45. 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 46. 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

Figure 29 shows an example that corresponds to the elastic FIFO buffer depths for Lane 0 and Lane 1 vs. the LMFC offset setting measured for Deframer 0 on an ADI customer evaluation 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 change with the LMFC offset.

During the measurement, the link between the JESD204B framer and JESD204B deframer of the transceiver is reestablished 10 times (with application of a new SYSREF pulse each time) for each LMFC offset value and each time the buffer depth is 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 JESD204B link establishments that the elastic buffers are to correct.

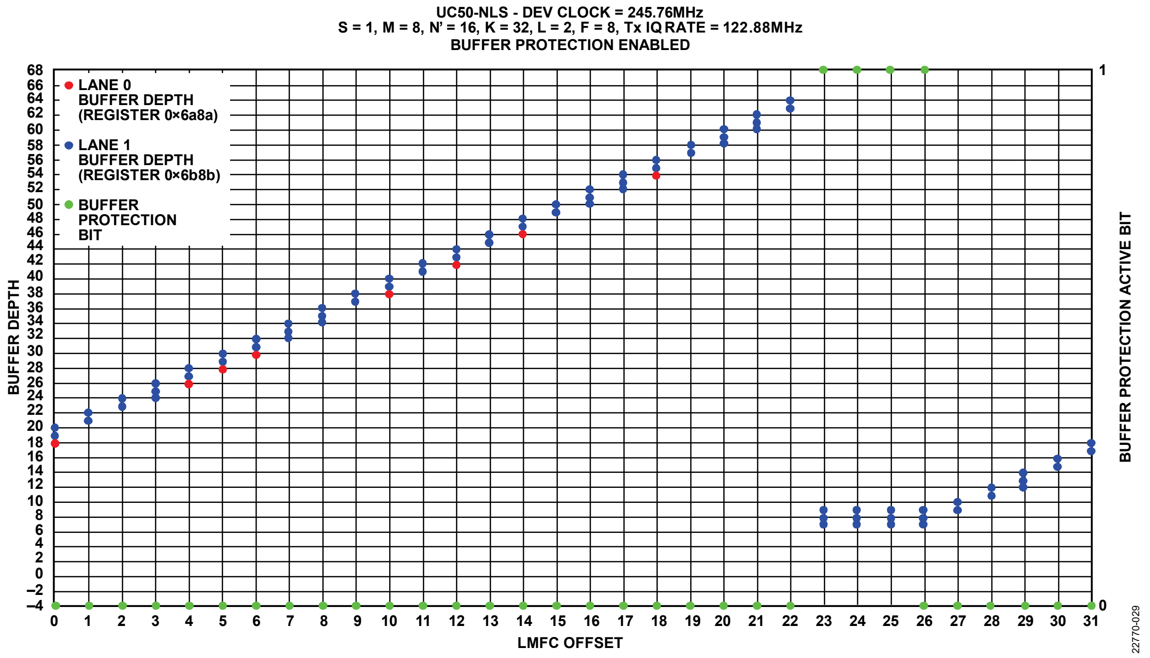


Figure 29. Buffer Depths for Lane 0 and Lane 1 vs. LMFC Offset on the Customer Evaluation Board with the ADRV9025Init_StdUseCase50_nonLinkSharing Profile and Buffer Protection Enabled

Selecting the Optimal LMFC/LEMC Offset for a System

The buffer depths are expected to be slightly different after power cycling the system or from one link establishment to another due to the variance in parameters such as synchronization delays and physical lane skews. Buffer depths are also expected to slightly change from system to system due to process, voltage, and temperature (PVT) variations.

Therefore, it is recommended to select an LMFC/LEMC offset value resulting in optimal buffer depths to account for those variations and maintain deterministic latency on all boards for a given system. 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 depth 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 link establishments (with application of a new SYSREF pulse each time) provides a good indication of the buffer depths spread for each LMFC offset value. Select an LMFC offset value that results in buffer depths as close as possible to the center of the linear part of the buffer depth vs. the LMFC offset plot the user creates for all the used lanes.

Figure 29 shows this process using the customer evaluation board programmed with the ADRV9025Init_StdUseCase50_nonLinkSharing profile, with automatic buffer protection enabled. In that example, an LMFC offset value of 9 is optimal 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 that results in 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 with large buffer depths (that is, near a value of (K × F)/4) because, for some combinations of JESD204B 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 JESD204B Mode with Buffer Protection Disabled

When buffer protection is disabled, it is recommended to select an LMFC offset value that has 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 the 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 JESD204B link establishments (with application of a new SYSREF pulse each time) provides an accurate indication of the buffer depth spread for each LMFC offset value.

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

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

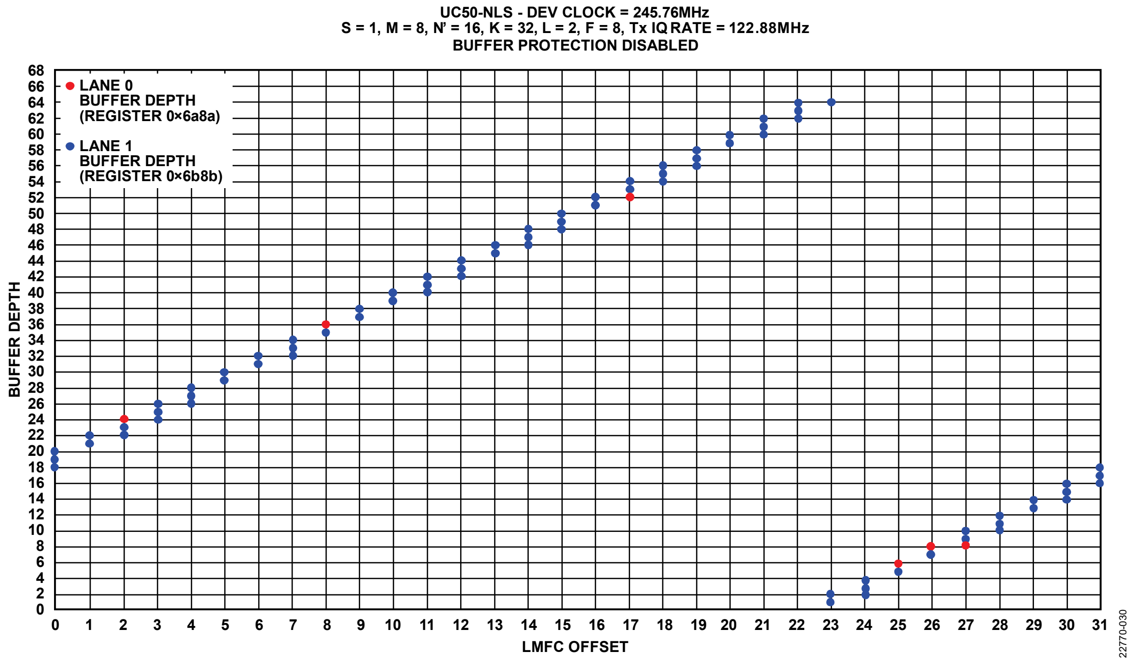


Figure 30. Buffer Depths Read Back for Lane 0 and Lane 1 vs. LMFC Offset on the Customer Evaluation Board with ADRV9025Init_StdUseCase50_nonLinkSharing Profile and Buffer Protection Disabled

In this example, an LMFC offset value of 6 or 7 is an optimal 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 that results in buffer depths as small as possible but still 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 that results in a large buffer depth (that is, near a value of (K × F)/4) because, for some combinations of JESD204B 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 JESD204C Mode When E ≤ 2 with Buffer Protection Enabled

In JESD204C mode, when E ≤ 2, it is also recommended to select an LEMC offset that results in buffer depth 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 JESD204C link establishments (with application of a new SYSREF pulse each time) provides an optimal indication of the buffer depths spread for each LEMC offset.

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

In this example, LEMC offset values between 36 and 40 are optimal 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 that results in 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 that result in large buffer depths (near a value of E × 32) because, for some combinations of JESD204C parameters, it can lead to the write and read pointers being too close and, therefore, can result in data corruption.

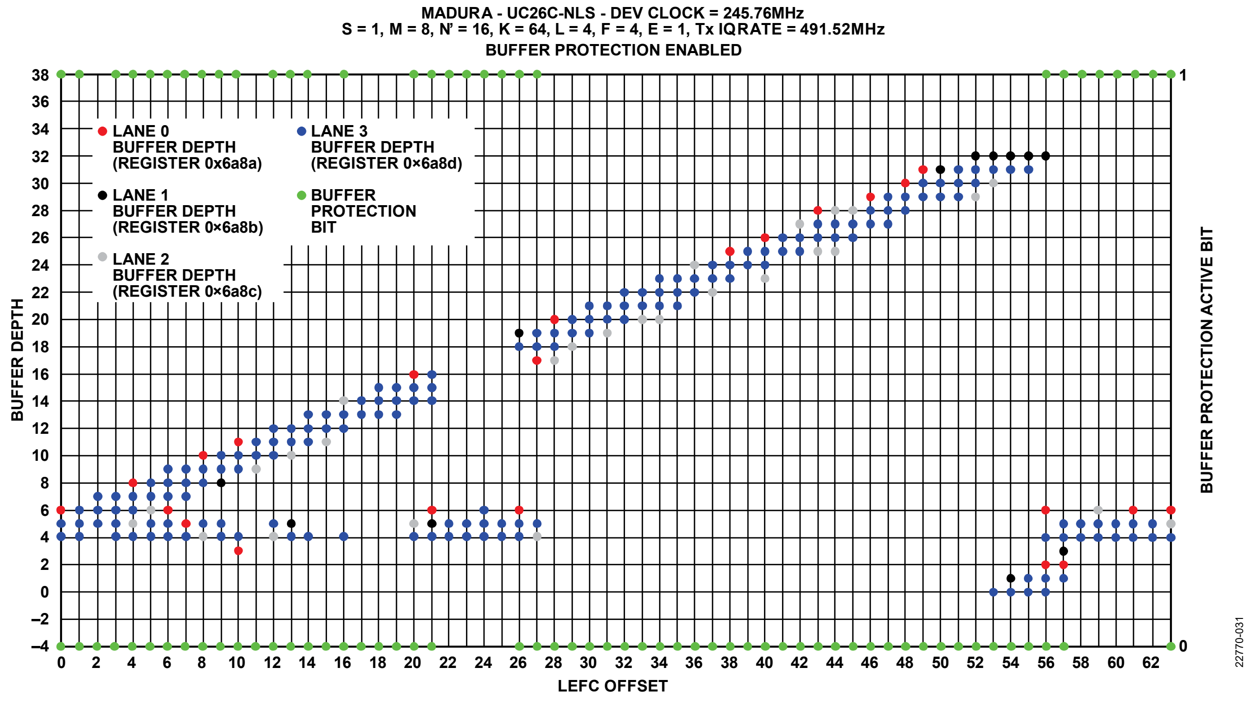


Figure 31. Buffer Depths for Lane 0, Lane 1, Lane 2, and Lane 3 vs. LEMC Offset on the Customer Evaluation Board with the ADRV9025Init_StdUseCase26C_nonLinkSharing Profile and Buffer Protection Enabled

Selecting the Optimal LEMC Offset for a System in JESD204C Mode When E ≤ 2 with Buffer Protection Disabled

When buffer protection is disabled, it is recommended to select an LEMC offset value that results in 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 JESD204C link establishments (with application of a new SYSREF pulse each time) provides an accurate indication of the buffer depth spread for each LEMC offset value. Select an LEMC offset value that results in buffer depths as close as possible to (E × 32)/2 for all lanes.

Figure 32 shows this process using the same customer evaluation board with the ADRV9025Init_StdUseCase26C_nonLinkSharing profile and automatic buffer protection disabled.

In this example, an LEMC offset value between 21 and 24 is an optimal 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 that results in buffer depths as small as possible but still 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 JESD204C parameters, it can lead to the write and read pointers being too close and, therefore, can result in data corruption.

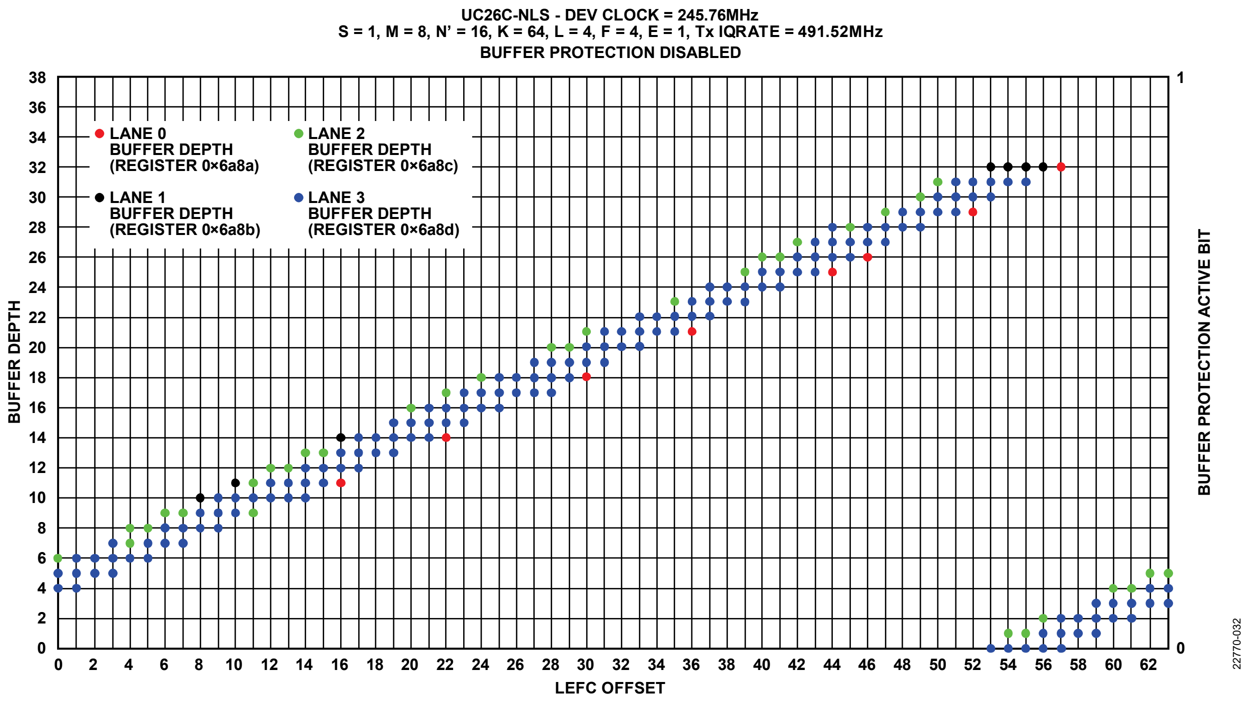


Figure 32. Buffer Depths for Lane 0, Lane 1, Lane 2, and Lane 3 vs. LEMC Offset on the Customer Evaluation Board with the ADRV9025Init_StdUseCase26C_nonLinkSharing Profile and Buffer Protection Disabled

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

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

In JESD204C mode, when E > 2, it is recommended to select an LEMC offset that results in buffer depth values as close as possible to the center of the linear part of the buffer depths vs. LEMC offset plot in Figure 33 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 JESD204C link establishments (with application of a new SYSREF pulse each time) provides an accurate indication of the buffer depths spread for each LEMC offset.

Select an LEMC offset value that results in 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 a customer evaluation board programmed with the ADRV9025Init_StdUseCase14C_LinkSharing profile, with automatic buffer protection enabled.

In this example, LEMC offset values between 87 and 89 are optimal choices because they result in a buffer depth around 41 for each lane, which is in the middle of the linear part of the Figure 33 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 JESD204C parameters, it can lead to the write and read pointers being too close and, therefore, can result in data corruption.

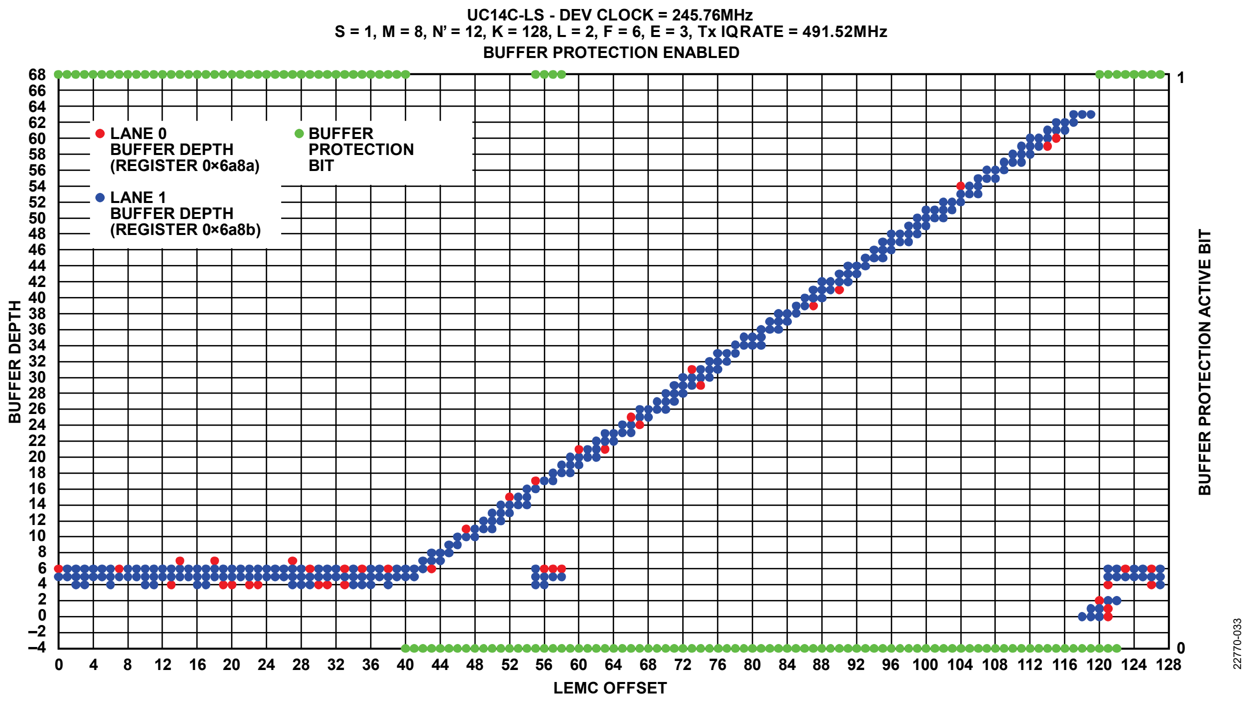


Figure 33. Buffer Depths for Lane 0 and Lane 1 vs. LEMC Offset on the Customer Evaluation Board with the ADRV9025Init_StdUseCase14C_LinkSharing Profile and Buffer Protection Enabled

SYNTHESIZER CONFIGURATION

OVERVIEW

The transceiver 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, digital control block, and either a 1 or 4 core voltage-controlled oscillator (VCO). The auxiliary PLL and clock PLL 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 LO generator, auxiliary LO generator, and clock generator 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 transceiver. 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 illustrates the common PLL block used in the transceiver.

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 transceiver and, therefore, must be a very clean clock source with respect to noise. Connect the external clock inputs to the DEVCLK+ pin (C8) and DEVCLK– pin (C9) 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 is not less than 50 mV or greater than 1 V.

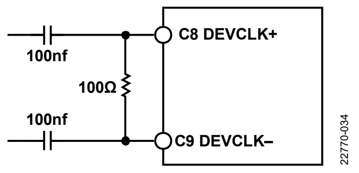


Figure 34. Reference Clock Input Connections

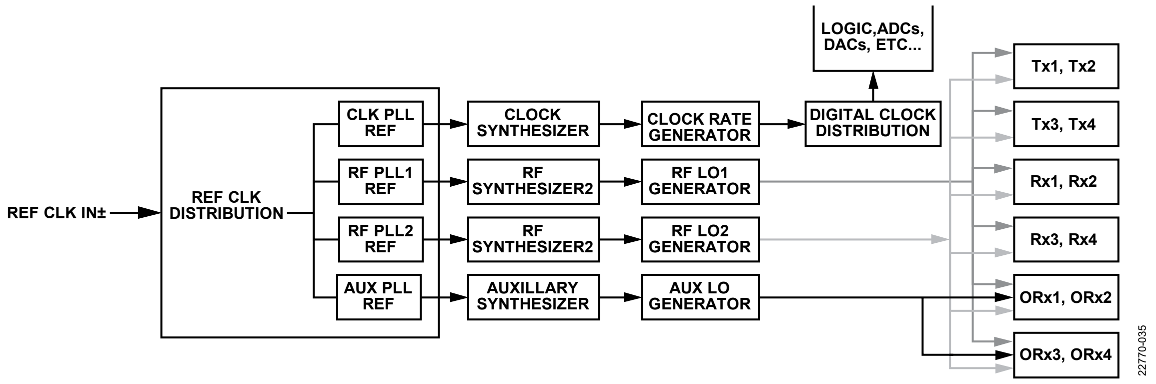


Figure 35. Synthesizer Interconnection, Clock, and LO Distribution Block Diagram

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 ADRV902x family data sheets 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 root square sum (RSS) added to the phase noise specified in the data sheets.

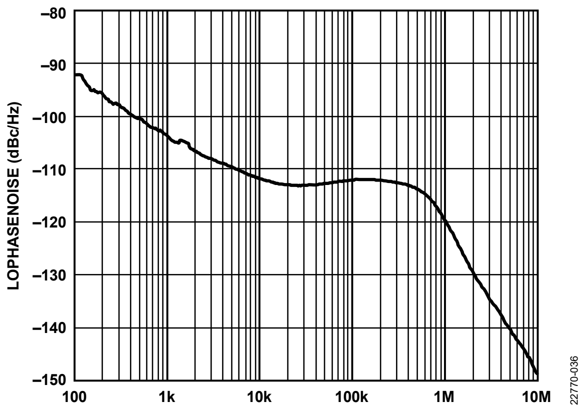


Figure 36. LO Phase Noise vs. Frequency Offset, F_{LO} =2600 MHz, Loop Bandwidth = 500 kHz, Phase Margin = 60°, DEVCLK Supplied by a Wenzel VCXO

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

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

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

where
N is the multiplier applied to the reference clock frequency (f_{REF}) to generate the desired LO frequency
$H(s)$ is the PLL loop transfer function

Noise power from the reference experiences a multiplication factor equal to the 20 × $\log_{10}$(N) term. Outside the loop bandwidth, the multiplied reference noise is attenuated by the loop filter. This means the reference phase noise is typically only a contributor for close-in offsets less than the loop bandwidth. The loop bandwidth and phase margin are provided in the caption of the phase noise figures provided in the product 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 an f_{LO} of 2600 MHz and an f_{REF} of 245.76 MHz, the gain is 20.5 dB. The total noise can be calculated with the reference clock noise, the RF LO phase noise, and this transfer function.

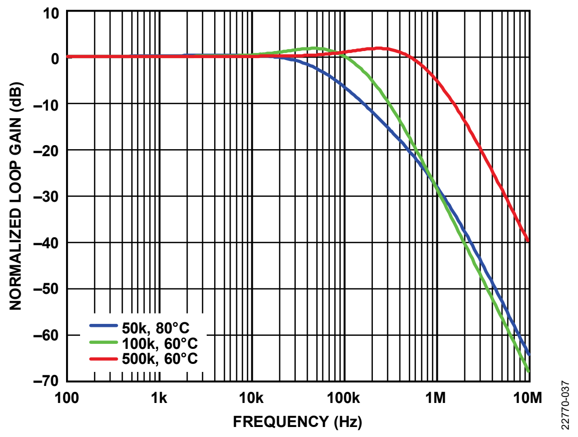


Figure 37. Normalized PLL Closed Loop Response

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 the result of multiplying the phase noise of the reference clock by the PLL closed loop transfer function 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 shown in Figure 40. Note that the data sheet reference information 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. The phase noise of the AD9528 is included in Figure 40 for reference. The AD9528 clock device is used on the evaluation board for all clock generation. Therefore, measured phase noise is dominated by the AD9528.

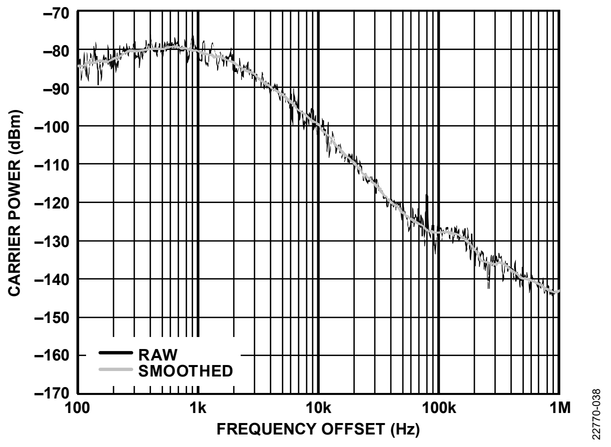


Figure 38. Phase Noise Plot for a Noisy 245.76 MHz Reference Clock

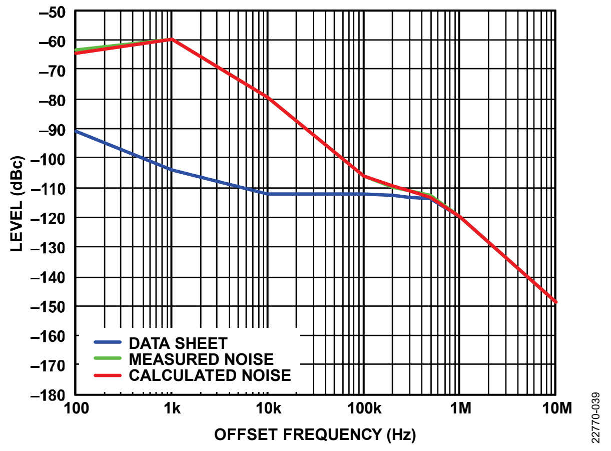


Figure 39. Measured and Calculated Phase Noise vs. Offset Frequency

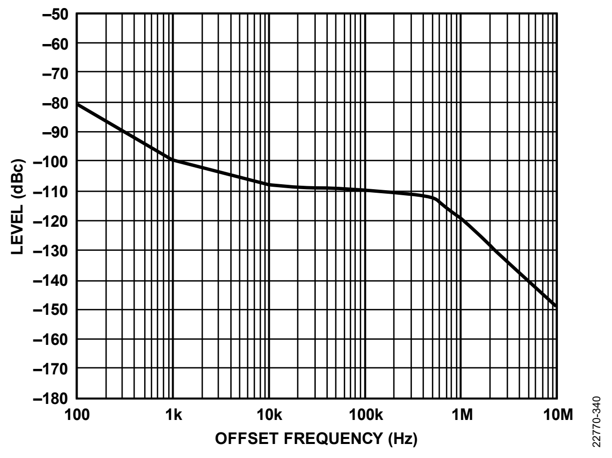


Figure 40. AD9528 Phase Noise Using a High Phase Noise Reference Clock

CLOCK SYNTHESIZER

The clock synthesizer is used to generate all the clocking signals necessary to run the transceiver. 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 JESD204B and JESD204C interface rates typically require that the synthesizer operates in integer mode. Profiles that are included in the Transceiver Evaluation Software 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, receiver ADC sample and interface clocks, observation receiver ADC sample and interface clocks, and the transmitter DAC sample and interface clocks.

RF SYNTHESIZER

The transceiver contains two RF PLLs. Each RF PLL uses the PLL block common to all synthesizers in the transceiver and employs 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 transceiver, the reference for RF PLL 1 and RF PLL 2 are sourced from the reference generation block of the transceiver. 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 47.

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 MHz to 6400 MHz. The LO divider boundaries are given in Table 48. Note that it is recommend rerunning the initialization calibrations when crossing a divide by 2 boundary or when changing the LO frequency by ±100 MHz or more from the frequency at which the initialization calibrations were performed.

Table 47. RF Synthesizer Divider Ranges

	LO Frequency Limits (MHz)									
	Lower Limit	Upper Limit	Lower Limit	Upper Limit	Lower Limit	Upper Limit	Lower Limit	Upper Limit	Lower Limit	Upper Limit
Divide By	32		16		8		4		2	
Auxiliary PLL	203.125	406.25	406.25	812.5	812.5	1625	1625	3250	3250	6500
RF PLL1 and RF PLL2	200	400	400	800	800	1600	1600	3200	3200	6400

Table 48. 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	

	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**
	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 transceiver to provide flexibility in LO assignment for the two RF LO sources. The switching network is diagrammed in Figure 41. Note that it is not recommended to set RFLO1 = RFLO2, which can cause unwanted coupling between the two PLLs. To set RFLO1 = RFLO2, set either RFPLL1 or RFPLL2 to the desired frequency and mux that PLL to both the 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 transceiver. This synthesizer utilizes a single core VCO. The reference frequency for the auxiliary 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 receiver signal chain as an offset LO. Calibrations are executed during the initialization sequence at startup. There is no signal present at the receiver or observation receiver 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 auxiliary LO signal is also available as an LO source for the observation receiver mixers.

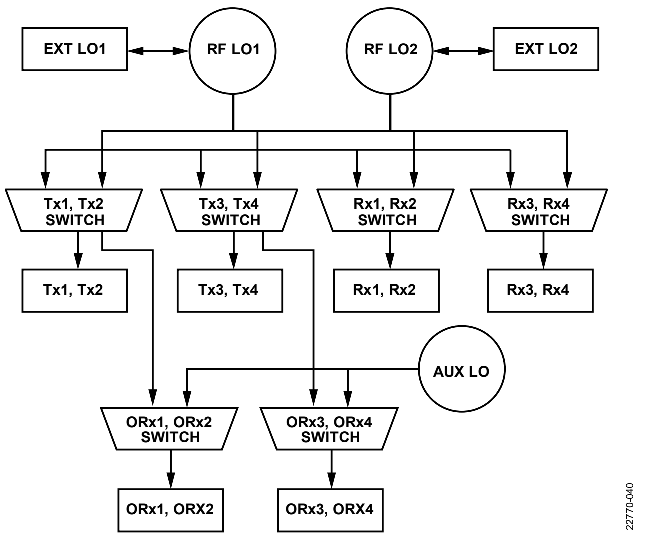


Figure 41. LO Switching Network

SETTING THE LO FREQUENCIES

There are two commands that the user can execute to select the LO frequency in the transceiver. One command is used when the user does not have special phase requirements between the transmitter LO and the auxiliary LO. The other command 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 auxiliary LO at a user defined offset from the transmitter LO that is typically 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.

Both commands can be run any time after device initialization, and neither command has any prerequisite commands or requirements. The structures and enumerators for these API commands are detailed in Table 49 through Table 53.

Table 49. adi_adrv9025_PllConfig_t Structure

Data Type	Parameter	Range	Description
PllName_e	pllName	Table 50	Name of the PLL the user wants to control.
PllModeSel_e	pllModeSel	Table 51	The user can select between slow locking or fast locking mode.
PllAuxLoResyncSel_e	pllAuxLoResyncSel	Table 52	The user can select between resyncing and not resyncing the auxiliary LO to the transmitter LO after a frequency change.
PllAuxLoOffsetProgSel	pllAuxLoOffsetProgSel	Table 53	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 that the customer wants to set in Hz.

Table 50. adi_adrv9025_PllName_e Enumerator

Enumerator	Enumerator Values	Description
PllName_e	ADI_ADRV9025_LO1_PLL	Selects LO1 PLL for transmitter/receiver/observation receiver.
	ADI_ADRV9025_LO2_PLL	Selects LO2 PLL for transmitter/receiver/observation receiver.
	ADI_ADRV9025_AUX_PLL	Selects auxiliary PLL for observation receiver.

Table 51. adi_adrv9025_Pll_ModeSel_e Enumerator

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

Table 52. adi_adrv9025_pllAuxLoResyncSel_e Enumerator

Enumerator	Enumerator Values	Description
pllAuxLoResyncSel_e	ADI_ADRV9025_PLL_AUX_LO_RESYNC_ENABLE	Resyncs the auxiliary LO to the transmitter LO after a frequency change.
	ADI_ADRV9025_PLL_AUX_LO_RESYNC_DISABLE	Does not resync the auxiliary LO to the transmitter LO after a frequency change.

Table 53. adi_adrv9025_pllAuxLoOffsetProgSel_e Enumerator

Enumerator	Enumerator Values	Description
pllAuxLoOffsetProgSel_e	ADI_ADRV9025_PLL_AUX_LO_OFFSET_PROG_ENABLE	Programs the auxiliary LO to be +(bandwidth/2 + 5) MHz from the transmitter LO after every frequency change.
	ADI_ADRV9025_PLL_AUX_LO_OFFSET_PROG_DISABLE	Does not set the auxiliary 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

Device initialization is the only precondition.

Parameters

Table 54. adi_adrv9025_PllFrequencySet(…)

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

Device initialization is the only precondition.

Parameters

Table 55. adi_adrv9025_PllFrequencyGet(…) Parameters

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 must be put on certain PLLs to meet system requirements. adi_adrv9025_PllFrequencySet_v2(…) is equivalent to adi_adrv9025_PllFrequencySet(…) with the parameters in Table 56 set in the adi_adrv9025_PllConfig_t structure.

Table 56. adi_adrv9025_PllConfig_t Structure Parameters

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

Device initialization is the only precondition.

Parameters

Table 57. adi_adrv9025_PllFrequencySet_v2(…) 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

Device initialization is the only precondition.

Parameters

Table 58. adi_adrv9025_PllLoopFilterSet(…) Parameters

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

Device initialization is the only precondition.

Parameters

Table 59. adi_adrv9025_PllLoopFilterGet(…) Parameters

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.

RF PLL PHASE SYNCHRONIZATION

This function allows 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 RF PLL to the same point. Therefore, the phase between each device is aligned at startup so that phasing between transceivers is repeatable and fixed. At startup, the standard JESD204B multichip synchronization (MCS) mechanism 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 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 synchronization 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. Therefore, at every sample, the observed LO phase is derotated by the digitally desired phase. Derotating 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 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 42 is a block diagram of the phase synchronization system.

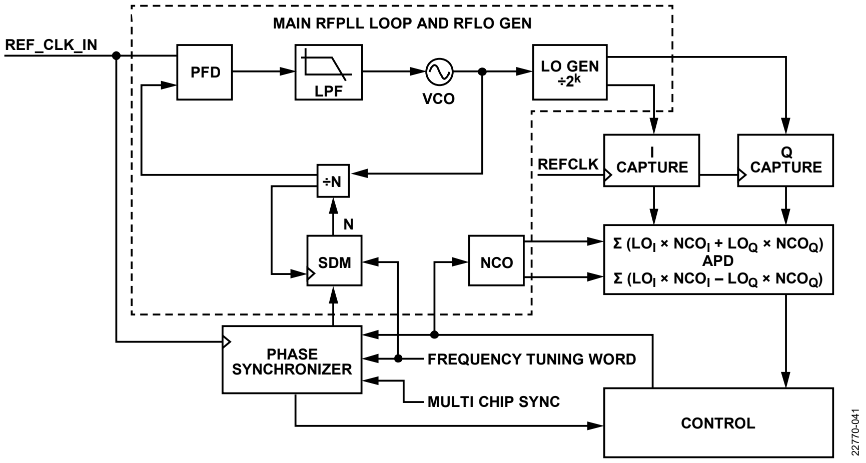


Figure 42. LO Phase Synchronization Functional Diagram

System Level Considerations

Overall phase synchronization is determined by a number of factors, including the printed circuit board (PCB) 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 shown in Figure 43. 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 following list is the goals of this transceiver mechanism:

- 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.

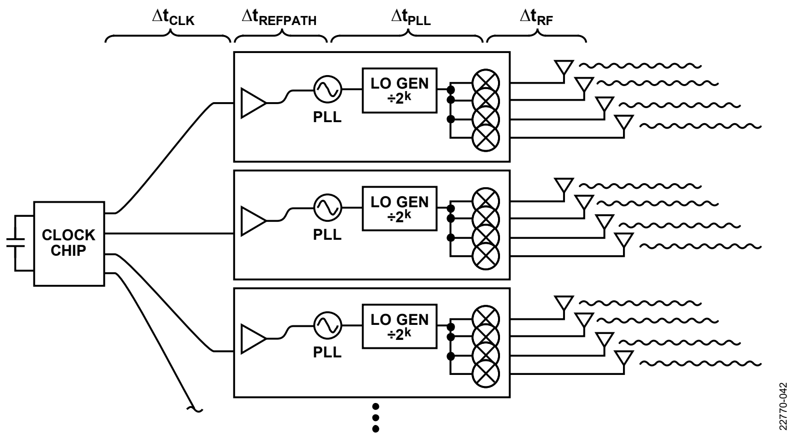


Figure 43. High Level Contributions to System Phase Per Antenna

Enabling the RF PLL Phase Synchronization Function Using the API

To enable the LO phase synchronization function, peform the following steps:

1. Set the phase synchronization bit in the initialization data structure.
2. Perform MCS to set the JESD204B and JESD204C deterministic latency using SYSREF pulses as normal. LO phase synchronization uses existing signaling and SYSREF to accomplish LO phase synchronization. The following structure definition describes the parameters needed for this process.

```
adi_adrv9025_Init_t deviceInitStruct =
{
    { // clocks
        245760,  // deviceClock_kHz
        9830400,  // clkPllVcoFreq_kHz
        9830400,  // serdesPllVcoFreq_kHz
        0,  // ldoSelect
        0,  // extLoFreq1_kHz
        0,  // extLoFreq2_kHz
        ADI_ADRV9025_INTLO_NOOUTPUT,  // rfPll1LoMode
        ADI_ADRV9025_INTLO_NOOUTPUT,  // rfPll2LoMode
        0,  // rfPll1LoOutDivider
        0,  // rfPll2LoOutDivider
        ADI_ADRV9025_RFPLLMCS_INIT_AND_CONTTRACK,  // rfPllPhaseSyncMode
```

Possible enumerator values are shown in the following code:

```
    /**
     * \brief Enumerated list of RFPLL phase synchronization modes
     *
     * RFPLL Phase sync requires extra time to sync each time the RFPLL frequency
     * is changed. If RFPLL phase sync is not required, it may be desired to
     * disable the feature to allow the RFPLL to lock faster.
     *
     * Depending on the desired accuracy of the RFPLL phase sync, several options
     * are provided.
     */
    typedef enum adi_adrv9025_RfPllMcs
    {
        ADI_ADRV9025_RFPLLMCS_NOSYNC                = 0,    /*!< Disable RFPLL phase
    synchronization */
        ADI_ADRV9025_RFPLLMCS_INIT_AND_SYNC         = 1,    /*!< Enable RFPLL phase sync init
    only */
        ADI_ADRV9025_RFPLLMCS_INIT_AND_CONTTRACK    = 2     /*!< Enable RFPLL phase sync init and
    track continuously */
    } adi_adrv9025_RfPllMcs_e;
```

RF PLL Phase Synchronization Demo Setup

A vector network analyzer is used to measure the phase difference between two transmitter outputs, LO1 and LO2, with the same frequency assigned to each transmitter output. The test setup is shown in Figure 44. It is important to use the same clock reference for all the equipment in the setup. In this diagram, all equipment is locked to the same 10 MHz reference.

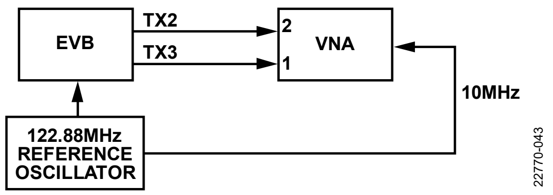


Figure 44. RFPLL Phase Sync Test Setup

Users can edit specific use case files (also referred to as JSON files or profiles) to set the RF PLL phase synchronization using the parameters shown here under rfPllPhaseSyncMode Options and Clock Options. When using the evaluation board system, the Transceiver Evaluation Software must be restarted for these changes to take effect.

rfPllPhaseSyncMode Options (parameter in use case block below is **bold**):

0: Disable RFPLL phase synchronization

1: Enable RFPLL phase sync init only

2: Enable RFPLL phase sync init and track continuously

Clock Options:

```
"deviceClock_kHz": 245760,
"clkPllVcoFreq_kHz": 9830400,
"serdesPllVcoFreq_kHz": 9830400,
"ldoSelect": 0,
"extLoFreq1_kHz": 0,
"extLoFreq2_kHz": 0,
"rfPll1LoMode": 0,
"rfPll2LoMode": 0,
"rfPll1LoOutDivider": 0,
"rfPll2LoOutDivider": 0,
"rfPllPhaseSyncMode": 2,
```

Figure 45 shows five power cycles of the same device with the phase synchronization function beginning in the disabled state. At each power up, the phase difference between the two LOs is a random value. This diagram also shows initialization and tracking results, which brings initial random phase to a repeatable value.

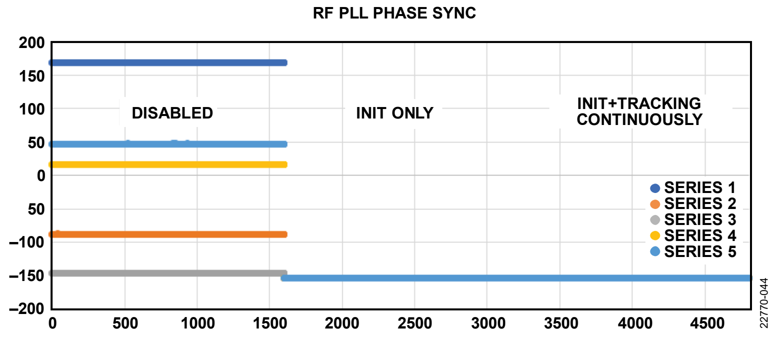


Figure 45. RF PLL Phase Sync Transitions from Disabled through Inititialization and into Tracking Mode (5 Independent Power Up Sequences Shown)

Figure 46 shows a close up view of the transition from initialization to continuosly tracking.

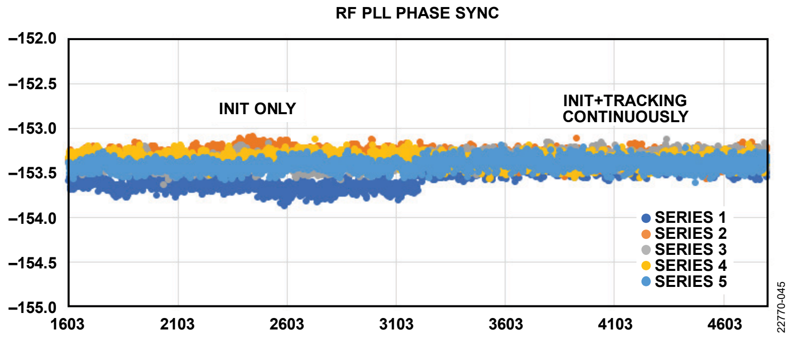


Figure 46. RF PLL Phase Synchronization Initialization to Tracking Results

ARM PROCESSOR AND DEVICE CALIBRATIONS

The transceiver 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 another 160 kB of data memory to utilize. The ARM is tasked with configuring the transceiver for the selected use case, performing initial calibrations of the signal paths, and maintaining device performance over time through tracking calibrations.

ARM STATE MACHINE OVERVIEW

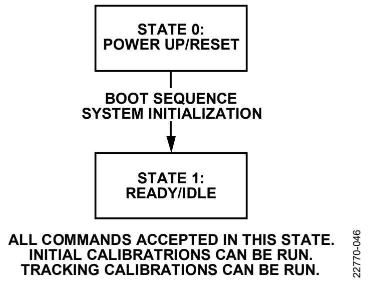

STATE 0:
POWER UP/RESET
BOOT SEQUENCE
SYSTEM INITIALIZATION
STATE 1:
READY/IDLE
ALL COMMANDS ACCEPTED IN THIS STATE.
INITIAL CALIBRATRIONS CAN BE RUN.
TRACKING CALIBRATIONS CAN BE RUN.
22770-046

Figure 47. ARM State Machine

When the arm core is powered up, the ARM moves into its power-up/reset state, shown as State 0 in Figure 47. The ARM firmware image is loaded at this point. When the ARM image has been loaded, the ARM is enabled and begins its boot sequence.

After the arm has been booted, it enters its ready/idle state, shown as State 1 in Figure 47. In this state, it can receive configuration settings or commands (instructions), such as performing the initial calibrations or enabling tracking calibrations.

SYSTEM INITIALIZATION

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

Pre-MCS initialization initializes the device up to the multichip synchronization procedure. The pre-MCS initalization is split into two commands that the application layer function calls. These commands are adi_adrv9025_PreMcsInit_v2(…) and adi_adrv9025_PreMcsInit_NonBroadCast(…). The adi_adrv9025_PreMcsInit_v2(…) command 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(…) command verifies that the ARM is programmed properly by verifying the ARM checksum and that the ARM is in the ready/idle state.

The multichip synchronization (MCS) step uses SYSREF pulses to synchronize internal clocks within the transceiver, which is required for deterministic latency.

Post-MCS initalization 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);
```

An important system 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. This function 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. The 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 (transmitter/receiver/observation receiver 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 JESD204B and JESD204C 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 and compares the computed checksums during booth to precomputed checksums. For example, comparing the ARM firmware checksum vs. the ARM checksum that is calculated after 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 to be not 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 JESD204B and JESD204C link. When complete, the tracking calibrations can be enabled. The RF data paths can then be enabled using either the SPI or pin modes.

Note that there is no absolute requirement to follow this sequence. The initialization calibrations and tracking calibrations do not have to be run 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 transceiver to achieve its performance specifications. These calibrations are split into two categories: initial calibrations, which are run either before the transceiver is operational or after LO frequency change, and tracking calibrations, which are used to maintain performance during runtime.

A number of transmitter calibrations use an observation path to observe the signal at the output of the transmitter. For the most part, these calibrations use an internal loopback path from transmitter to observation receiver. The exception is the external LOL initialization and tracking algorithms that require the use of an external path connection between the transmitter output and an observation input.

A requirement for this device is that the observation receiver channel used to calibrate a transmitter channel must be on the same side of the chip as that transmitter channel. Table 60 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 60. 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 48 shows an example of four feedback paths, each transmitter going back to an observation receiver, obeying the principle of each transmitter being fed back to an observation receiver on the same side of the device. It is also possible to have both Tx1 and Tx2 going back to a single observation receiver input (either ORx1 or ORx2) through a switch. Similarly, Tx3 and Tx4 can go back to a single observation receiver input (either ORx3 or ORx4).

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

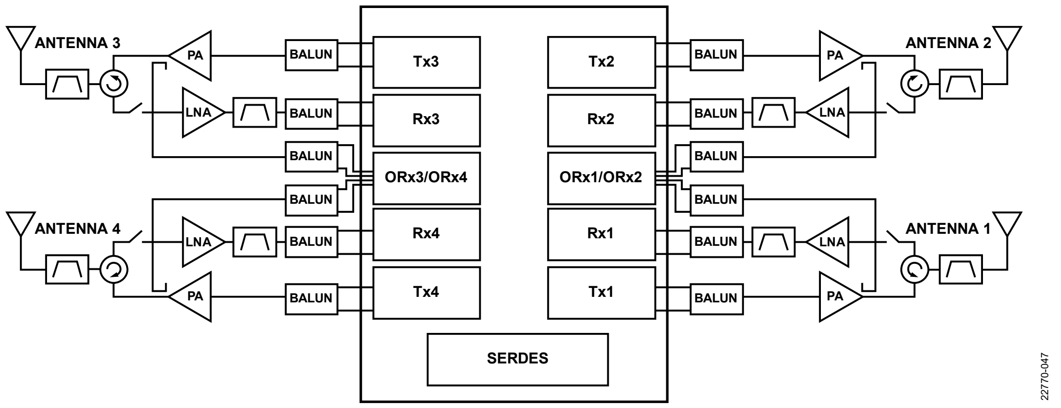


Figure 48. External Feedback for Transmitter Tracking Calibrations

INITIAL CALIBRATIONS

The ARM processor in the transceiver is tasked with scheduling/performing initial calibrations to optimize the performance of the signal paths prior to device operation. These calibrations are run as part of the utility API function adi_adrv9025_PostMcsInit( ). To correctly perform the initial calibrations, this utility function must 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 API 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 adi_adrv9025_InitCalsRun( ).

Table 61 shows the bit assignments of the calibration mask. Note that Table 61 provides a full list of initialization calibrations for the device. Some initial calibrations are not available for certain transceivers and applications.

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 62. The mask is universally applied to all calibrations selected in the current call of adi_adrv9025_initCalsRun( ), regardless of the paths for which the calibrations are being run. For example, if 0xF is chosen as a mask and both receiver and transmitter calibrations are selected in the calMask, when the ARM runs a receiver calibration it then does so on all four receiver channels. Likewise, when the ARM runs a transmitter calibration, the calibration is run on all four transmitter channels.

Table 61. calMask Bit Assignments

Bits	Corresponding Enumerator	Calibration	Description
D0	ADI_ADRV9025_TX_BB_FILTER	Transmitter baseband filter calibration	This is used to tune the corner frequency of the transmitter 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	Receiver TIA filter calibration	This is used to tune the corner frequency of the receiver TIA filter.
D3	ADI_ADRV9025_ORX_TIA	Observation receiver TIA filter calibration	This is used to tune the corner frequency of the observation receiver TIA filter.
D4	ADI_ADRV9025_LBRX_TIA	Loopback receiver TIA filter calibration	This is used to tune the corner frequency of the loopback receiver TIA filter.
D5	ADI_ADRV9025_RX_DC_OFFSET	Receiver dc offset calibration	This is used to correct for dc offset within the receiver chain.

Bits	Corresponding Enumerator	Calibration	Description
D6	ADI_ADRV9025_ORX_DC_OFFSET	Observation receiver dc offset calibration	This is used to correct for dc offset within the observation receiver chain.
D7	ADI_ADRV9025_LBRX_DC_OFFSET	Loopback receiver dc offset calibration	This is used to correct for dc offset within the loopback receiver 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 transmitter to internal loopback path delay, which is required for the TxQEC initial calibration and tracking.
D10	ADI_ADRV9025_TX_LO_LEAKAGE_ INTERNAL	Transmitter LO leakage initial calibration	This performs an initial LO leakage calibration for the transmitter path. It utilizes the transmitter path and the internal loopback path (see Figure 51).
D11	ADI_ADRV9025_TX_LO_LEAKAGE_EXTERNAL	Transmitter LO leakage external initial calibration	This performs an initial external LO leakage calibration for the transmitter path. It utilizes the transmitter path, a required external loopback path, and the observation receiver path (see Figure 52). The external loop must be enabled such that the transmitter output is observable by the observation receiver.
D12	ADI_ADRV9025_TX_QEC_INIT	Transmitter QEC initial calibration	This performs an initial QEC calibration for the transmitter path. It utilizes the transmitter path and an internal loopback path (see Figure 51).
D13	ADI_ADRV9025_LOOPBACK_RX_LO_DELAY	Loopback receiver 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 receiver QEC initial calibration	This performs an initial QEC calibration for the receiver path.
D15	ADI_ADRV9025_RX_LO_DELAY	Receiver LO delay calibration	This is used to perform an LO delay calibration for the receiver path.
D16	ADI_ADRV9025_RX_QEC_INIT	Receiver QEC initial calibration	This performs an initial QEC calibration for the receiver path.
D17	ADI_ADRV9025_ORX_LO_DELAY	Observation receiver LO delay calibration	This is used to perform an LO delay calibration for the observation receiver path.
D18	ADI_ADRV9025_ORX_QEC_INIT	Observation receiver QEC initial calibration	This performs an initial QEC calibration for the observation receiver path.
D19	ADI_ADRV9025_TX_DAC	Transmitter DAC initial calibration	This performs a calibration of the transmitter DAC.
D20	Reserved		
D21	ADI_ADRV9025_EXTERNAL_PATH_DELAY	External transmitter to observation receiver path delay initial calibration	This acquires an estimation of the transmitter to observation receiver 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 receiver path (typically required only in GSM applications).
D24	ADI_ADRV9025_TX_ATTENUATION_DELAY	Transmitter attenuation delay calibration	This is used to calculate the path delay between the transmitter analog and digital attenuation blocks. This delay is then used to delay the onset of transmitter analog attenuation when the transmitter attenuation changes. This synchronizes the attenuation change at the transmitter output.
D25	ADI_ADRV9025_TX_ATTEN_TABLE	Transmitter attenuation table linearization calibration	This is used to correct for phase changes between different attenuation indices in the transmitter attenuation table.

Bits	Corresponding Enumerator	Calibration	Description
D26	ADI_ADRV9025_RX_GAIN_DELAY	Receiver gain delay calibration	This is used to calculate the path delay between the receiver analog and digital attenuation blocks. This delay is then used to delay the onset of receiver analog attenuation when the receiver gain index is changed. This synchronizes the gain change in the baseband data. This calibration does not check the status of the DDC filter, so if the NCO is enabled it may cause the calibration to fail with no warning if the calibration tone is placed outside the pass band. The NCO must not be used when doing this calibration.
D27	ADI_ADRV9025_RX_GAIN_PHASE	Receiver gain phase calibration	This is used to correct for phase changes between different gain indices in the receiver gain table.
D28	Reserved		
D29	ADI_ADRV9025_CFR_INIT	CFR initialization calibration	This performs an initialization calibration for the transceiver CFR hardware (ADRV9029 only).
D30	ADI_ADRV9025_SERDES_INIT	SERDES initialization cal	This performs an initialization calibration for the JESD204C data interface.
D31	Reserved		

Table 62. 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 in Table 61. 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 is an ARM error when 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 command:

```
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 is 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 63.

Table 63. 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 after power up. Each element of the array corresponds to calibrations performed on each channel.
calsLastRun[4]	A 4-element array indicating 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

Figure 49 through Figure 52 show how the transceiver is configured for notable calibrations with external system requirements, such as the QEC and LOL calibrations. In all diagrams, gray lines and blocks are not active in the calibration. Lines showing the path of the LOs are shown in color to distinguish them from the signal paths. A brief explanation of the calibration is provided. Note that as the ARM performs each of the calibrations, it is tasked with configuring the device as per Figure 49. For example, with respect to enabling/disabling paths. 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 receiver input properly terminated for a receiver QEC initialization calibration.

Receiver QEC Initial Calibration

The receiver QEC initialization calibration algorithm is utilized to improve the receiver path QEC performance. The receiver 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 calibration PLL (CAL PLL), which is the auxiliary PLL. When the receiver QEC initialization calibration runs, the ARM configures the receiver to the maximum gain index (255).

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

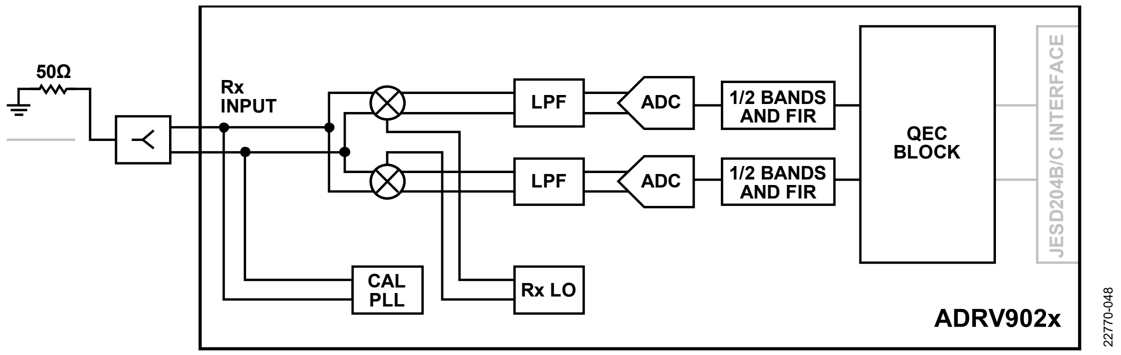


Figure 49. Receiver QEC Initial Calibration System Configuration

Observation Receiver QEC Initial Calibration

The observation receiver 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 Figure 50, the transmitter LO is the LO source for the observation receiver channel and the auxiliary PLL acts as the calibration PLL.

It is a system requirement that for optimum performance, it is recommended to set the internal observation receiver attenuation to 10 dB for TXLO ≤ 2.8 GHz or 14 dB to 16 dB for TXLO > 2.8 GHz.

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

Receiver/Observation Receiver TIA Initial Calibration

The receiver/observation receiver TIA calibration is used to calibrate the corner frequency of the analog baseband TIA filter in the receiver/observation receiver signal path. The signal path used for this calibration is the same as the receiver QEC initialization calibration shown in Figure 49. 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.

It is a system requirement to isolate the input port from incoming signals or the calibration may fail to complete.

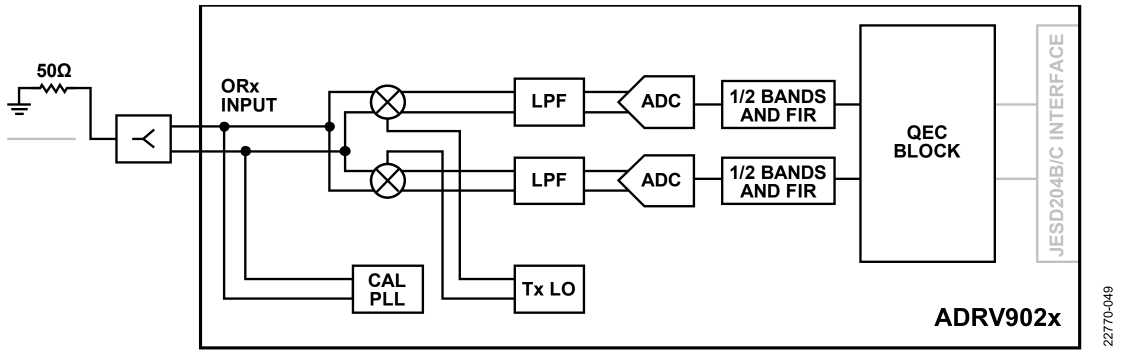


Figure 50. Observation Receiver QEC Initial Calibration System Configuration

Internal Transmitter LO Leakage and Transmitter QEC Initial Calibrations

The transmitter LO leakage and transmitter QEC initial calibrations utilize the internal loopback path and the baseband section of the observation receiver path to calculate its initial correction factors. During these calibrations, test signals (tones and wideband signals) are output. These appear at the transmitter output, so it is important that the PA connected to the transceiver 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 transmitter attenuation setting, the corresponding QEC and LOL correction values are applied to the transmitter channel by the ARM. The transceiver configuration for this calibration is shown in Figure 51.

It is a system requirement that the PA in the transmitter 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 transmitter output is 50 Ω.

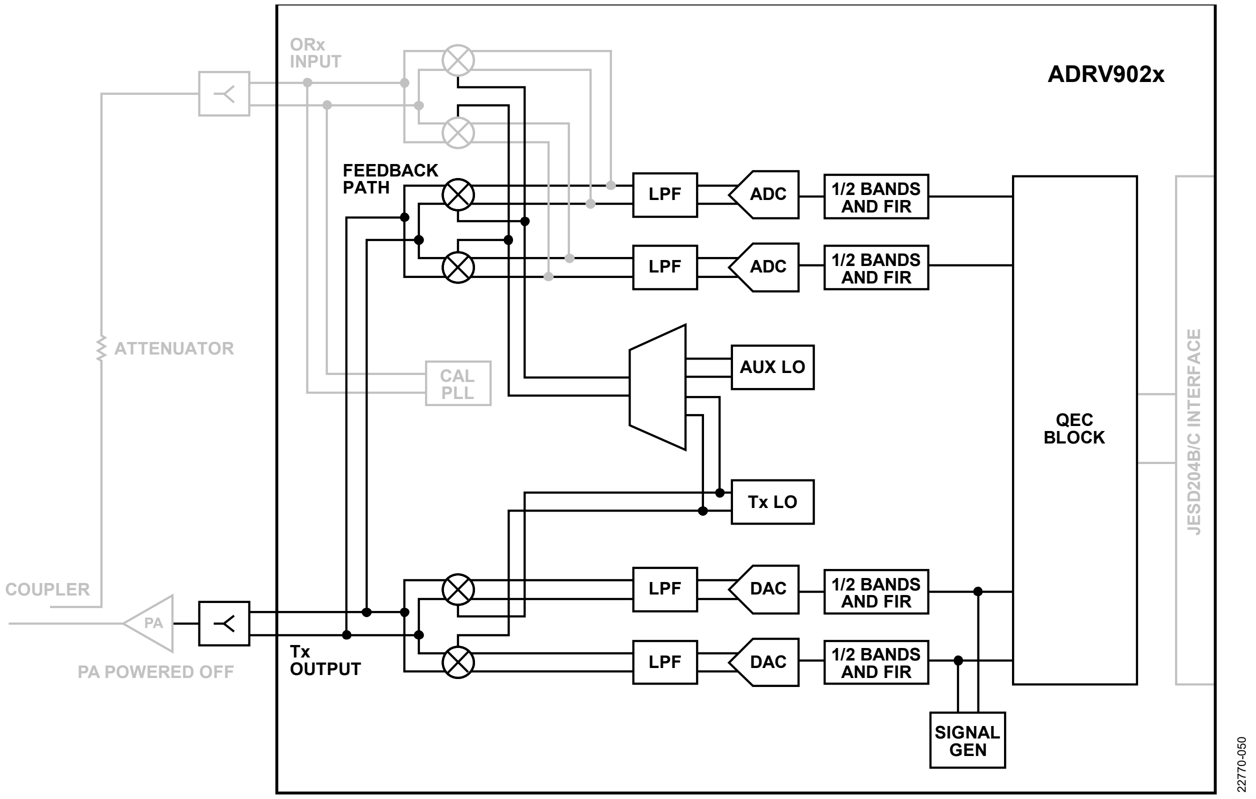


Figure 51. Device Path Configuration for Transmitter LOL and QEC Initial Calibrations

External Transmitter 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 transmitter outputs and the observation receiver inputs. The purpose of this calibration is to obtain a good estimate (gain/phase) of the external loop channel conditions prior to operation. The transceiver configuration is shown in Figure 52. The calibration utilizes a pseudorandom 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 observation receiver input. This is to prevent transmitter data from saturating the observation receiver input. This is also necessary from the perspective of digital predistortion (DPD) operation.

Note that if the observation receiver 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 transmitter to observation receiver mapping does not match the programmed transmitter to observation receiver mapping.

It is a system requirement that the output of the transmitter channel to be calibrated must be routed to the utilized observation receiver path to properly observe the calibration signal. If there is a PA in the path, it must be enabled during this calibration. The transmitter to observation receiver mapping must be configured (via API or GPIO) prior to the calibration to indicate which transmitter is routed back to which observation receiver (see the Transmitter to Observation Receiver Feedback section).

Choose the combined external coupler plus attenuation to provide a peak input power close to P_{HIGH} (as specified in the device's datasheet) at the observation receiver input pin so 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 observation receiver attenuation to 10 dB for TXLO ≤ 2.8 GHz or 14 dB to 16 dB for TXLO > 2.8 GHz.

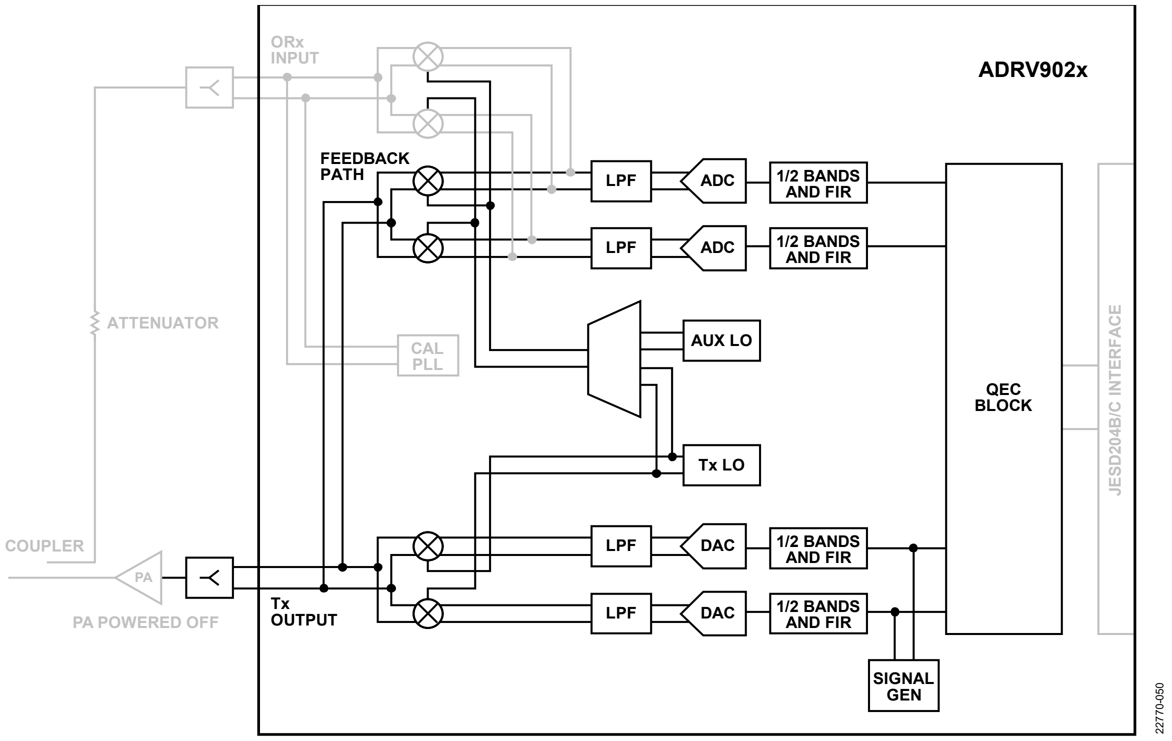


Figure 52. External LOL System Configuration (Grayed Out Circuitry Not in Use)

Receiver Gain Delay Initial Calibration

The receiver datapath features an analog and a digital gain/attenuation element. If the analog and digital gain changed simultaneously, 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 nonzero data path latency between the analog and digital gain/attenuation elements.

The receiver gain delay calibration measures the latency between the analog and digital gain/attenuation elements to delay the onset of digital gain. This ensures that when the analog and digital gain change, 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 automatic gain control (AGC) timing due to this calibration.

Receiver Gain Phase

The receiver 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 auxiliary PLL is used to transmit a tone at the receiver input and measure the phase difference. The phase shift is introduced by a digital phase shifting element.

Transmitter 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 transmitter 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 calibration 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.

Transmitter Attenuation Delay

Similar to the receiver, the transmitter datapath features an analog and digital gain/attenuation element. The transmitter attenuation delay calibration helps to ensure that when a change in attenuation occurs in both analog and digital, the transmitter 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 receiver gain delay calibration, which delays the onset of digital gain or attenuation changes.

Transmitter to Observation Receiver Feedback

For the external transmitter LO leakage initial calibration to complete, the ARM must be advised of the current transmitter to observation receiver feedback paths through the external circuitry. Specify this path at initialization, through the adi_adrv9025_PostMcsInit_t structure that is passed to adi_adrv9025_PostMcsInit( ). In this structure, there are four variables that indicate which transmitter is being fed back to each observation receiver. These variables are shown in Table 64.

Table 64. Definition of adi_adrv9025_TxToOrxMappingConfig_t

Observation Receiver Maps	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 that in the case of multiple transmitter channels being fed back to a single observation receiver, a multiple pass is required for the external transmitter 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 transmitter paths that have a feedback path to an observation receiver. 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 Auxiliary LO Settings During Initialization Calibrations

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

1. Set the transmitter PLL frequency to the desired frequency.
    a. If the user uses adi_adrv9025_PllFrequencySet(…), the auxiliary PLL is configured to the default offset frequency when the transmitter PLL is programmed.
    b. If the user uses adi_adrv9025_PllFrequencySet_v2(…), the auxiliary 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 auxiliary PLL frequency to the desired application frequency.

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

Summary of Initial Calibration Requirements

Table 65 summarizes initial calibration requirements and other related details.

Table 65. Recommended Initial Calibrations

Initial Calibration	Recommended Values
Receiver QEC	ADI_ADRV9025_MAP_NONE_ORX1 ADI_ADRV9025_MAP_TX1_ORX1 ADI_ADRV9025_MAP_TX2_ORX1
Receiver TIA	ADI_ADRV9025_MAP_NONE_ORX2 ADI_ADRV9025_MAP_TX1_ORX2 ADI_ADRV9025_MAP_TX2_ORX2
Observation Receiver 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. The ARM processor achieves this optimization by performing calibrations at regular intervals. These calibrations are termed tracking calibrations, and they 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 66 shows the bit assignments of the enable mask (presently only receiver/observation receiver QEC calibrations are available)). enableDiasbleFlag is an enable or disable parameter (valid enumerators are shown in Table 67). Based on the enumerator chosen for enableDiasbleFlag, the selected tracking calibrations in enableMask are enabled or disabled.

Table 66. Tracking Calibrations Enable Mask Bit Assignments

Calibration 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 67. adi_adrv9025_TrackingCalEnableDisable_e Definition

Enumerator	Description
ADI_ADRV9025_TRACKING_CAL_DISABLE	When used, the selected tracking calibrations in enableMask are disabled upon the call to adi_adrv9025_TrackingCalsEnableSet.
ADI_ADRV9025_TRACKING_CAL_ENABLE	When used, the selected tracking calibrations in enableMask are 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 transceiver is configured for each calibration, and a brief explanation of the calibration is provided. In all configuration 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 observation receiver input as per the following set of 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 calibration description 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 in a way so that observations do not have to be continuous. The ARM algorithms are optimized to process batches of 100 μs, but smaller batches are acceptable.

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

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

Receiver QEC Tracking Calibration

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

It is a system requirement that the receiver channels must be enabled. For example, in TDD mode, receiver QEC tracking only runs during receiver periods. If only one channel is enabled, the receiver QEC only runs on this channel. Note that in FDD modes, receiver enable is high at all times. Receiver enable refers to the enable of any of Rx1 to Rx4.

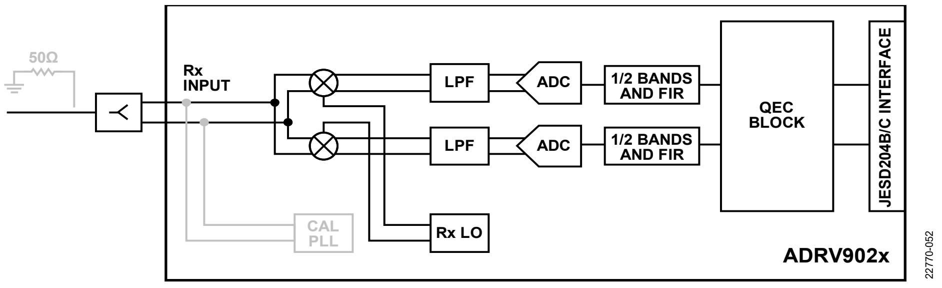


Figure 53. Receiver QEC Tracking

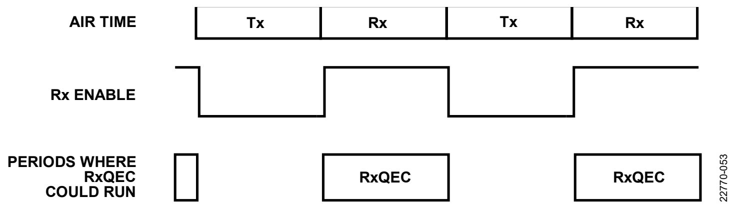


Figure 54. Timing Diagram Showing When Receiver QEC can Run in TDD Mode

Observation Receiver QEC Tracking Calibration

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

It is a system requirement that observation receiver channels must be enabled. For example, in TDD mode, observation receiver QEC tracking only runs during observation receiver periods. If only one channel is enabled, the observation receiver QEC only runs on this channel.

Do not change the observation receiver gain index while the tracking calibration runs. If the observation receiver gain index changes, rerun the observation receiver QEC initial calibration.

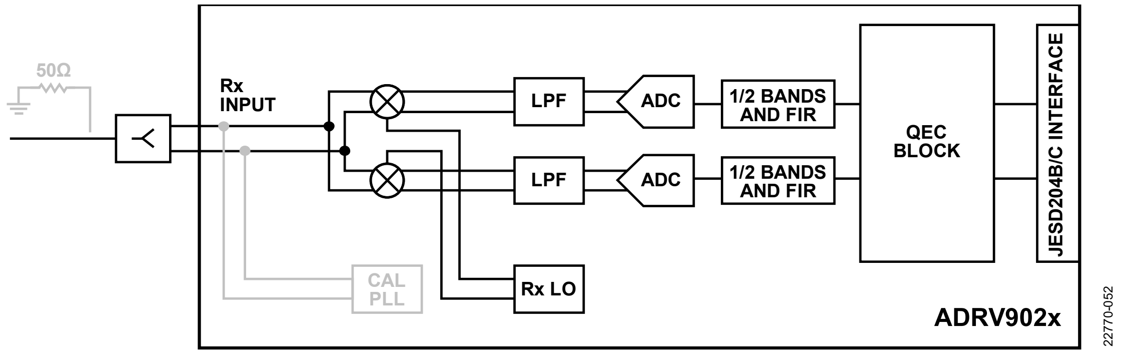


Figure 55. Observation Receiver QEC Tracking

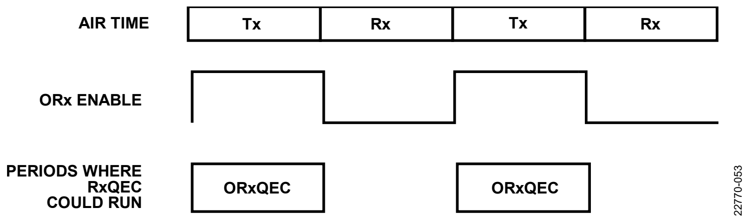


Figure 56. Timing Diagram Showing when Observation Receiver QEC can Run in TDD Mode (Observation Receiver Enable Refers to the Internal Enable Control of ORx1 to ORx4)

Transmitter QEC Tracking Calibration

The transmitter 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 transmit QEC tracking must be interleaved with normal other captures that utilize the observation receiver path. This tracking determines optimal coefficients for the current gain setting, updating the table stored during the transmitter QEC initialization to ensure this table has the best values for the current operating conditions. Figure 57 shows the transceiver configuration for transmitter QEC tracking calibration.

It is a system requirement that the transmitter channel(s) must be enabled. To run, the observation receiver path must be available for the ARM to use (observation receiver enable low). That means the required observation receiver path cannot be required by the user for other (or voltage standing wave radio (VSWR)) captures.

Note that in FDD modes, transmitter enable is high at all times. Transmitter enable refers to the enable of any Tx1 to Tx4. Observation receiver enable refers to the internal enable signal for the selected observation receiver channel.

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

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

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

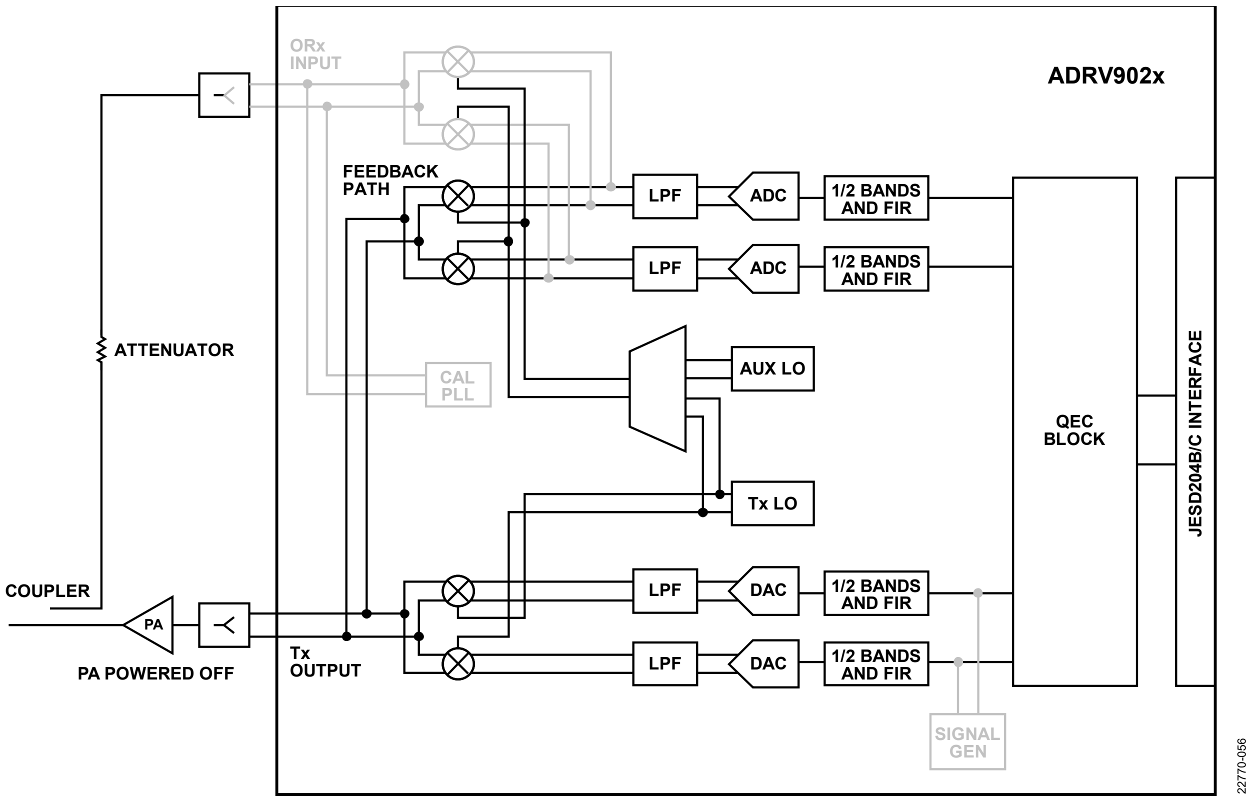


Figure 57. Transmitter QEC Tracking Calibration Configuration

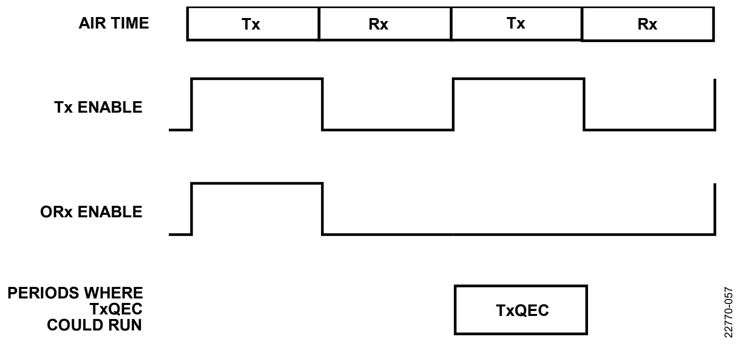


Figure 58. Timing Diagram Showing When Transmitter QEC can Run in TDD Mode

Transmitter LOL Tracking Calibration

The transmitter LO leakage tracking calibration uses an external path between the transmitter output and observation receiver 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 auxiliary LO is used in the observation receiver path to offset the transmitter LO leakage from the observation receiver LO. Figure 59 shows the transceiver configuration for the transmitter LO leakage tracking calibration with the transmitter output looped back to the observation receiver input (an observation receiver on the same side of the chip as the transmitter being calibrated).

Note that if the observation receiver 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.

It is a system requirement that the transmitter channel(s) must be enabled. The observation receiver path must be available for the ARM to use (that is, not required by the user for DPD (or VSWR) captures). The observation receiver path must be connected to the appropriate transmitter to be calibrated, and the ARM must be advised which transmitter output has a connection to which observation receiver.

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 the observation receiver gain index changes. The following are two options to achieve a proper channel estimate, but it is highly recommended to follow the first option:

1. Run external transmitter LO leakage initialization calibration. Ensure that mapping is set up properly, PA is enabled, and all tracking calibrations are disabled.
2. If not running external transmitter LO leakage initialization calibration, follow this procedure:
   a. Run transmitter LOL tracking calibration. If external transmitter LOL initial calibration is skipped, change LO frequency or observation receiver attenuation and disable transmitter LOL tracking (if it is running).
   b. If transmitter 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 transmitter LOL tracking.
   f. Call TrackingCalTxLolStatusGet() again and note the value of iterCount
   g. If the iterCount value has increased by at least 1, enable transmitter data transmission

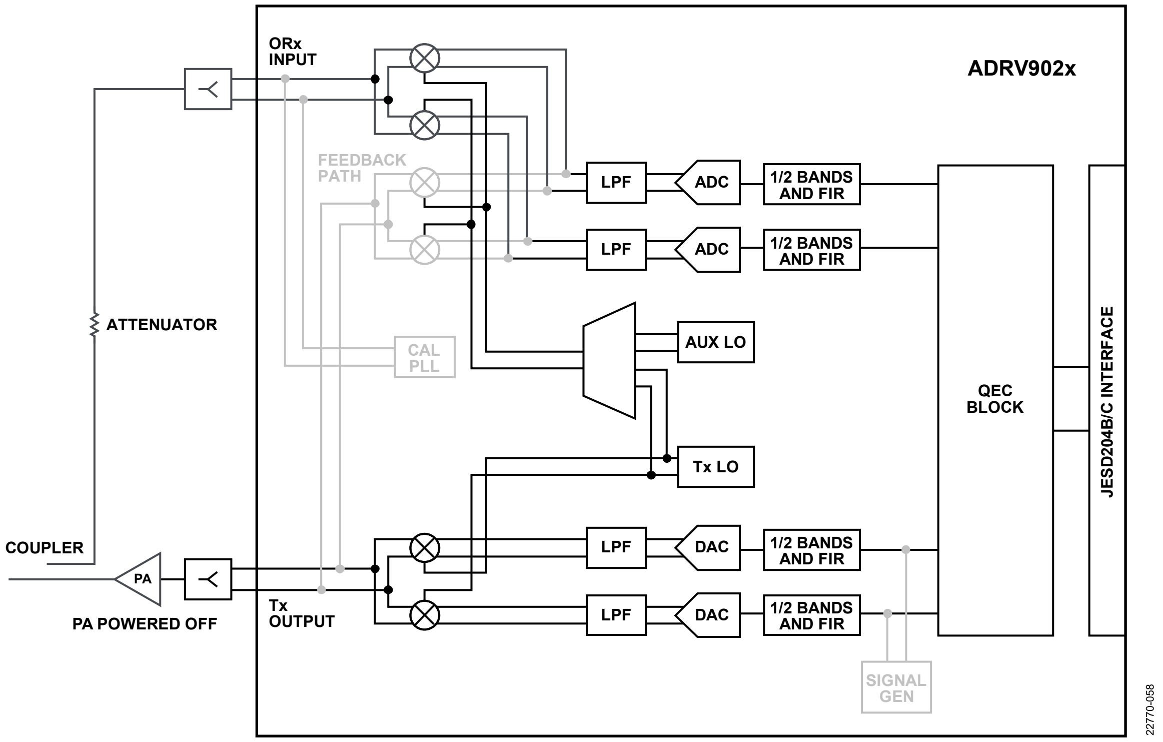


Figure 59. Transmitter LOL Tracking Configuration

CALIBRATION GUIDELINES AFTER PLL FREQUENCY CHANGES

Some applications require changing the PLL frequency for transmitter, receiver, or observation receiver signal paths after the transceiver has started normal operation and tracking calibrations have improved performance. Some tracking calibrations require rerunning initial calibrations after the PLL frequency change to relearn the new channel conditions. It is important that certain procedures are followed to maintain proper operation of the tracking calibrations.

The LO frequency changes fall into one of two types. Type 1 is the 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.

Type 2 is the 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 transceiver into command control mode via adi_adrv9025_RadioCtrlCfgSet(…), then call adi_adrv9025_RxTxEnaleSet(…) to disable all channels.
3. Rerun the following initial calibrations. Ensure to follow system considerations as described in System Considerations for Initial Calibrations. 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 the following is true:
   a. ADI_ADRV9025_INTERNAL_PATH_DELAY (if transmitter QEC tracking is used) 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 transmitter to observation receiver mappings used in the application. If the previous step is not executed, it is mandatory to call the adi_adrv9025_ExtTxLolChannelReset(…) command for each transmitter channel. It must be called one transmitter channel at a time. Then a special procedure must be followed to relearn the channel estimate described in the Transmitter LOL Tracking Calibration section.
   b. Enable relevant tracking calibrations.
   c. Transition back to pin control mode, if necessary.

Type 2 Frequency Change Procedure

If the LO frequency change falls into Type 2 as 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

To achieve best performance, the transceiver features autonomous internal calibrations that are performed during device initialization. The calibrations are run in the post-MCS part of device initialization. The majority of the calibrations are run with a single API call after the calibration structure is set. These are the internal calibrations that utilize internal loopback paths. Those that utilize external paths (such as the external transmitter LOL calibration) are run separately afterward.

All of the calibrations are overseen and scheduled by the ARM processor, therefore the user does not have to be concerned about what order the calibrations are run. The sequence is defined in a way so 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 receiver, transmitter, and observation receiver paths. The ARM clock is derived from the clock PLL.

In the Figure 60, 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 other calibrations require observation of either internally generated calibration tones or pseudorandom noise to calculate the required coefficients that are used to define the characteristics of the channel. However, other calibrations, for example the transmitter QEC calibration, use an algorithm to determine the correction factors that can be influenced by the actual load conditions to which the transmitter is connected. For these reasons, the amount of time each of the calibrations require to complete may vary slightly.

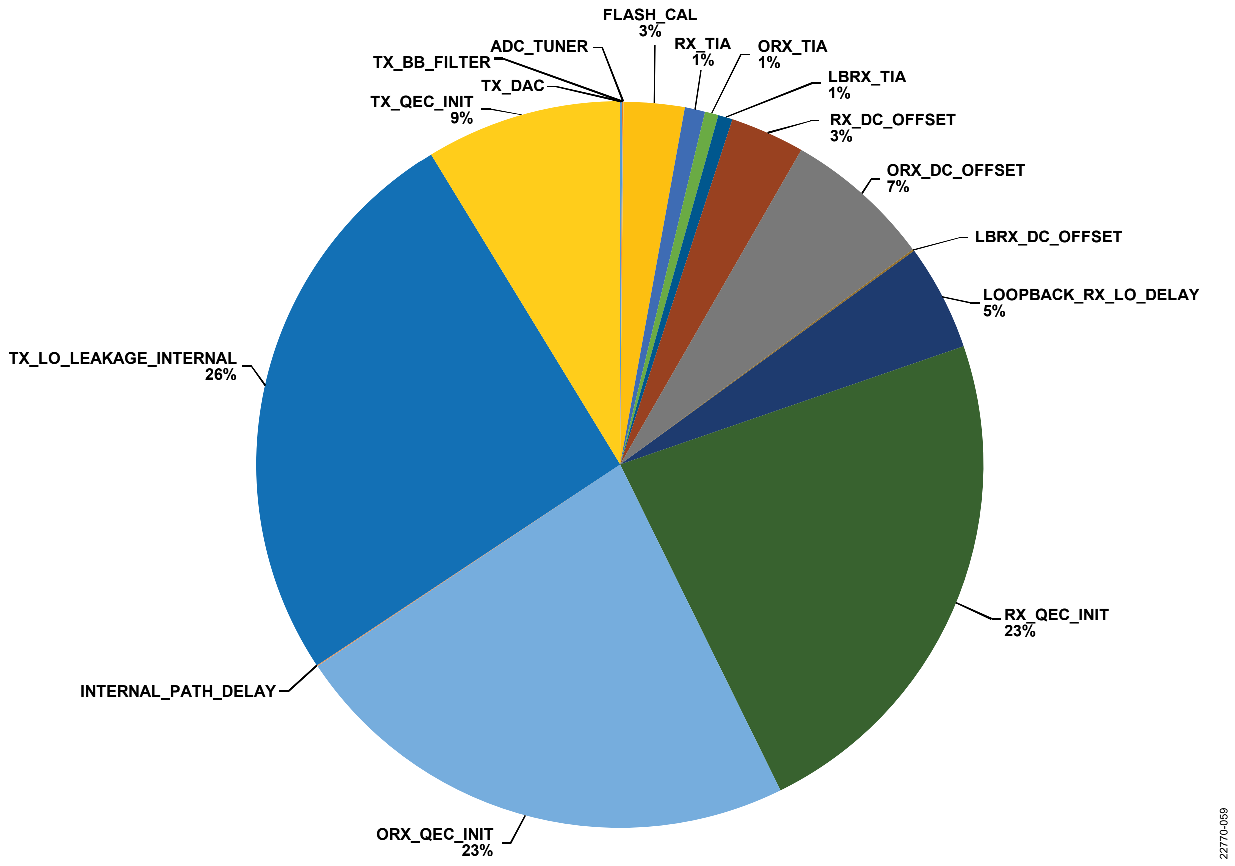


Figure 60. Relative Time Distribution of Initialization Calibrations

Table 68 through Table 97 are measured calibration times of the transceiver 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. Table 68 through Table 97 are listed in pairs. The first of each pair lists the relevant bandwidths and sample rates. The second table of each pair lists the calibration timing results in milliseconds for 1, 2, 3, and 4 enabled receiver or transmitter channels. In the case of observation receiver 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 channels enabled. Other observation receiver calibrations show differences from 1 to 4 channels because the paths from each of the transmitters are calibrated individually.

Table 68. ADRV9025Init_StdUseCase13_nonLinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 69. ADRV9025Init_StdUseCase13_nonLinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4545**	**8302**	**11616**	**14932**

Table 70. ADRV9025Init_StdUseCase14_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 71. ADRV9025Init_StdUseCase14_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4226**	**7658**	**10680**	**13654**

Table 72. ADRV9025Init_StdUseCase14C_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 73. ADRV9025Init_StdUseCase14C_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4226**	**7657**	**10674**	**13649**

Table 74. ADRV9025Init_StdUseCase23C_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 75. ADRV9025Init_StdUseCase23C_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4645**	**8429**	**11767**	**15108**

Table 76. ADRV9025Init_StdUseCase26C_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 77. ADRV9025Init_StdUseCase26C_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4210**	**7657**	**10660**	**13654**

Table 78. ADRV9025Init_StdUseCase26C_nonLinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 79. ADRV9025Init_StdUseCase26C_nonLinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4211**	**7658**	**10663**	**13649**

Table 80. ADRV9025Init_StdUseCase50_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 81. ADRV9025Init_StdUseCase50_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
TX_LO_LEAKAGE_INTERNAL	876	1749	2621	3494
TX_QEC_INIT	299	588	899	1186
Total Calibration Time	**4251**	**7721**	**10771**	**13766**

Table 82. ADRV9025Init_StdUseCase50_nonLinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 83. ADRV9025Init_StdUseCase50_nonLinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4219**	**7663**	**10651**	**13634**

Table 84. ADRV9025Init_StdUseCase51_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 85. ADRV9025Init_StdUseCase51_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
Total Calibration Time	**4203**	**7656**	**10682**	**13637**

Table 86. ADRV9025Init_StdUseCase51_nonLinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 87. ADRV9025Init_StdUseCase51_nonLinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4210**	**7659**	**10686**	**13664**

Table 88. ADRV9025Init_StdUseCase54_nonLinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 89. ADRV9025Init_StdUseCase54_nonLinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**5443**	**8742**	**12917**	**16370**

Table 90. ADRV9025Init_StdUseCase55_nonLinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 91. ADRV9025Init_StdUseCase55_nonLinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**5186**	**8315**	**12274**	**15552**

Table 92. ADRV9025Init_StdUseCase61_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 93. ADRV9025Init_StdUseCase61_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4701**	**8489**	**11889**	**15236**

Table 94. ADRV9025Init_StdUseCase82C_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 95. ADRV9025Init_StdUseCase82C_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4224**	**7654**	**10676**	**13652**

Table 96. ADRV9025Init_StdUseCase83C_LinkSharing

Use Case	Transmitter Bandwidth	Transmitter Input Rate	Transmitter DAC Rate	Observation Receiver Bandwidth	Observation Receiver Output Rate	Observation Receiver ADC Rate	Receiver Bandwidth	Receiver Output Rate	Receiver 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 97. ADRV9025Init_StdUseCase83C_LinkSharing Calibration Durations

Calibration	1 Channel (ms)	2 Channels (ms)	3 Channels (ms)	4 Channels (ms)
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	**4328**	**7781**	**10782**	**13826**

INITIALIZATION CALIBRATIONS TO BE RUN AFTER DEVICE INITIALIZATION

The transceiver requires a few additional initialization calibrations to be run after the standard set because they require external signal routing. An external transmitter LOL initialization calibration is available where the observation point is moved from inside the device to the selected observation receiver input. In this case, the transmitter channel 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, crest factor reduction (CFR) calibrations are also run separately and sequentially. Refer to Table 61 for the appropriate calibration mask.

Table 98 addresses the typical times for these initialization calibrations. Note that the CFR initialization calibration is mostly coefficient setting and, therefore, completes quickly.

Table 98. Post Initialization Transmitter Calibrations

Initialization 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
Crest Factor Reduction (ADRV9029 only)	<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 receiver QEC, observation receiver QEC, and on some devices, receiver HD2 tracking calibrations. These calibrations, when enabled, constantly observe the receiver (or observation receiver) spectrum and update the correction parameters while 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 transmitter LOL, transmitter QEC, and some versions of the device also include closed-loop gain control (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.

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

The CLGC tracking calibration runs on a 1 second schedule with similar batch sizes. In the case of JESD204C, an additional tracking calibration is run to maintain the link parameters on a 60 second schedule.

ARM MEMORY DUMP

The contents of the ARM firmware memory and ARM data memory can be captured for debugging purposes by using the adi_adrv9025_ArmMemDump(…) API function.

adi_adrv9025_ArmMemDump(…)

```
adi_adrv9025_ArmMemDump(adi_adrv9025_Device_t *device, const char *binaryFilename)
```

Description

This utility function reads the ARM memory and writes the binary byte array directly to a binary file. The first 224 kB correspond to the program memory. The following 160 kB correspond to the data memory. The binaryFilename file is opened before reading the ARM memory to verify that the file has valid write access. A file IO exception is thrown if the file does not have valid write access.

Precondition

Device initialization is the only precondition.

Parameters

Table 99. adi_adrv9025_ArmMemDump(…) Parameters

Parameter	Description
*device	Pointer to device structure.
*binaryFilename	File opened by the API to store ARM memory contents. Total size must be 384 kB.

Whenever it is necessary to debug an issue, an ARM memory dump can be captured, and the resulting binaryFilename file can be sent to Analog Devices for analysis. To correctly capture the ARM memory content, the adi_adrv9025_ArmMemDump(…) API function must set the ARM processor into an exception state. After calling the adi_adrv9025_ArmMemDump(…) API function, the device must be reinitialized to put the ARM back into its normal operating condition.

STREAM PROCESSOR AND SYSTEM CONTROL

A stream processor is a processor within the transceiver tasked with performing a series of configuration tasks based on some event. After a request from the user, the stream processor performs a series of predefined actions that 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 transceiver 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 JESD204B and JESD204C interface 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. Use this stream file when initializing the transceiver with the profile in question.

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

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

Eleven separate stream processors exist in the transceiver, which are each responsible for the execution of some dedicated functionality within the device. These stream processors 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 transmitter and four receiver data paths, and two for the observation receiver data paths. Note that even though there are four distinct RF front ends for the observation receiver, the transceiver only supports two digital data paths, one shared between Observation Receiver 1 and Observation Receiver 2 and another shared between Observation Receiver 3 and Observation Receiver 4. These observation receiver data paths are also shared with the internal transmitter channel loopback paths to facilitate data collection during the various transmitter calibrations. The existence of individual slice stream processors for each data path enables true real-time parallel operation of all unique transmitter and receiver data paths. The observation receiver data paths still must be managed based on the various system operation use cases detailed in this section.

Because 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 transmitter slice stream processors can only access the transmitter digital sub maps.

Core Stream Processor

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

SYSTEM CONTROL

The signal paths within the transceiver 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 or off signal chains, pin control is recommended. The device defaults to API mode upon power up.

API Control

adi_adrv9025_RxTxEnableSet(…)

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

Description

This API controls and configures the transmitter and receiver data paths.

Parameters

Table 100. adi_adrv9025_RxTxEnableSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
rxChannelMask	The desired receiver/observation receiver signal chain to power up. See Table 101 for the list of enumerators.
txChannelMask	The desired transmitter signal chain to power up. See Table 102 for the list of enumerators.

The enumerators 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 observation receiver channel is enabled continuously and not returned to ADI_ADRV9025_RXOFF for any time, the transmitter tracking calibrations are able to function.

Table 101. adi_adrv9025_RxChannels_e Enumerator Definition

adi_adrv9025_RxChannels_e Enum	Enabled Channels
ADI_ADRV9025_RXOFF	No receiver or observation receiver 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 102. adi_adrv9025_TxChannels_e Enumerator Definition

adi_adrv9025_TxChannels_e Enum	Enabled Channels
ADI_ADRV9025_TXOFF	No transmitter 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 transmitters enabled

Pin Control

The individual channels can also be controlled using a series of enable pins. In pin control mode, the receiver and transmitter signal chains are controlled using 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 observation receiver paths can be controlled in various modes, as indicated in Table 103.

Table 103. Observation Receiver Select Mechanisms

Observation Receiver Pin Mode	Observation Receiver 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 observation receiver is enabled, and when low, all observation receiver paths are disabled. Figure 61 shows 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 observation receiver is enabled, and when low, all observation receiver paths are disabled. The ORX_CTRL_B pin is used to select the observation receiver path, allowing the user to choose between two different observation receiver 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 shown in Figure 62. Note where ORx2 on and ORx3 on are shown in Figure 62. Any of the other observation receivers 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. Observation receiver 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 shown in Figure 63.		
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 observation receiver paths. These paths are predetermined through the API (over SPI). This mode is shown in Figure 64.		
Dual Channel 4-Pin Mode	In this mode, ORX_CTRL_A and ORX_CTRL_C are the enable/disable controls while ORX_CTRL_B and ORX_CTRL_D select which channel is to be enabled, allowing the user to choose between four different observation receiver paths. This mode is shown in Figure 65.		

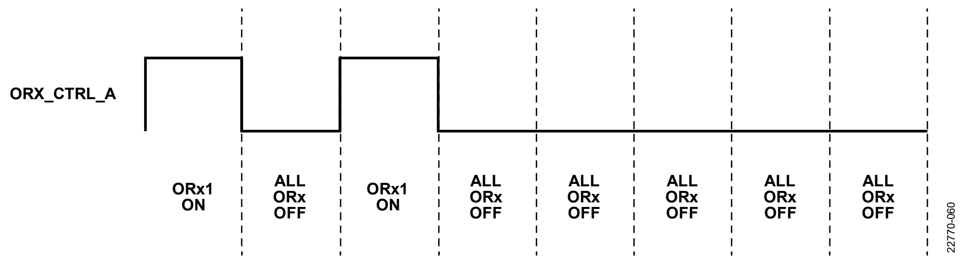


Figure 61. Single-Channel 1-Pin Mode

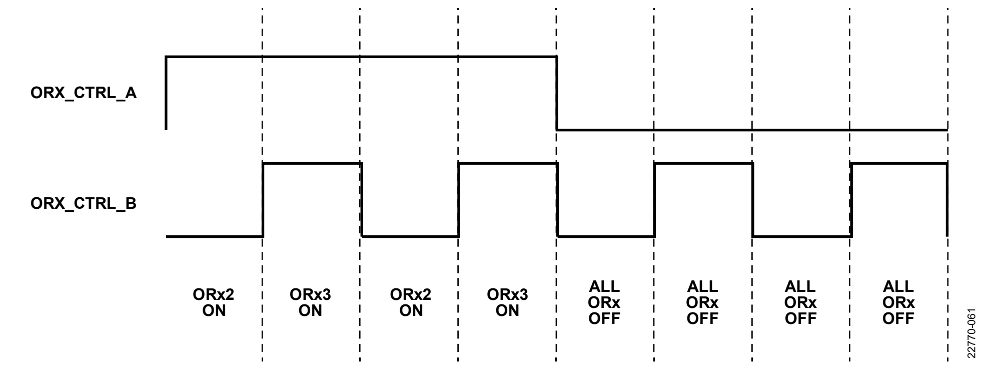


Figure 62. Single-Channel 2-Pin Mode

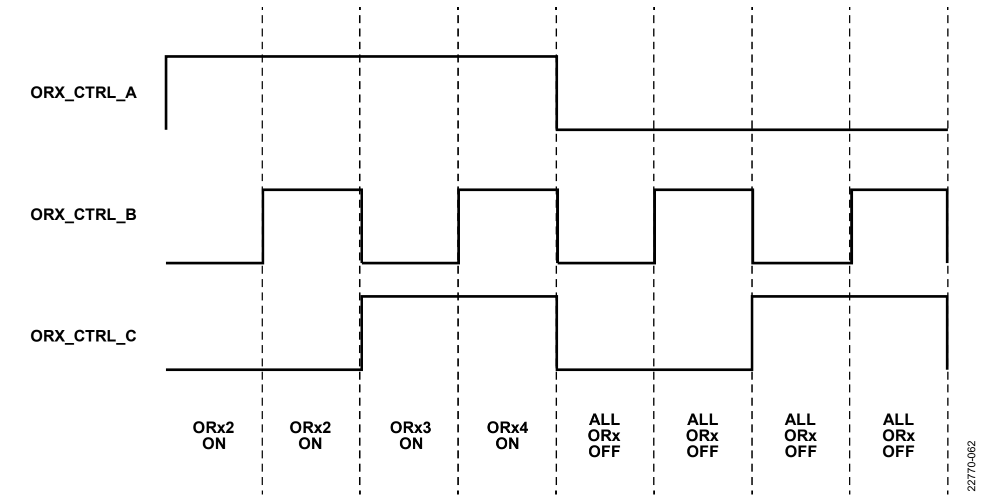


Figure 63. Single-Channel 3-Pin Mode

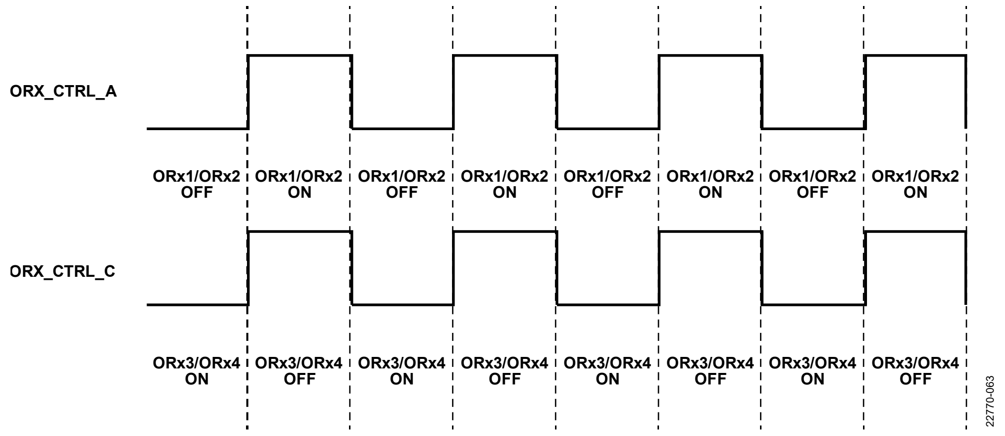


Figure 64. Dual-Channel 2-Pin Mode

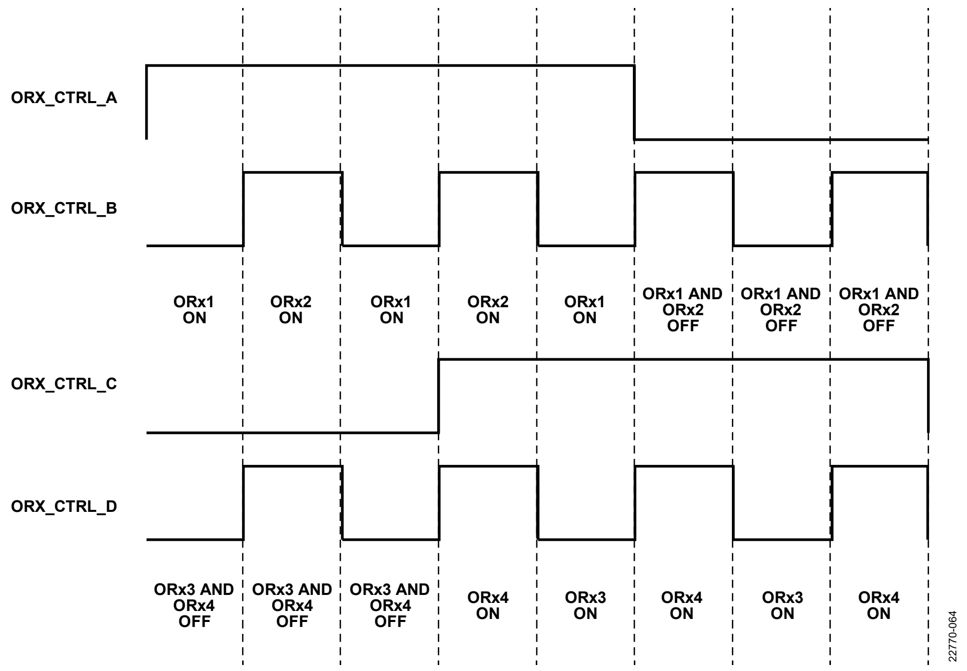


Figure 65. Dual-Channel 4-Pin Mode

The user can set the channel control mode (API/pin) with the post multichip sequence API function.

adi_adrv9025_PostMcsInit(…)

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

Description

This API sets the channel control mode (API or pin).

Parameters

Table 104. adi_adrv9025_PostMcsInit(…) Parameters

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 transmitter, receiver, and observation receiver channels.

This structure is defined in Table 105 and, depending on how the user configures this structure before the call to adi_adrv9025_PostMcsInit(), the device is configured in either pin or API mode.

Table 105. adi_adrv9025_RadioCtrlModeCfg_t Definition

Member Name	Description
txRadioCtrlModeCfg	Transmitter signal path enable mode configuration. See Table 106 for description.
rxRadioCtrlModeCfg	Receiver signal path enable mode configuration. See Table 107 for description.
orxRadioCtrlModeCfg	Observation receiver signal path enable mode configuration. See Table 108 for description.

Table 106. adi_adrv9025_TxRadioCtrlModeCfg_t Definition

Member 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 transmitter signal path.
	ADI_ADRV9025_TX_EN_PIN_MODE	Setting this mode does not modify the currently set mode to control the transmitter signal path.

Member Name	Value	Description
	ADI_ADRV9025_TX_EN_INVALID_MODE	Setting this mode selects no mode to control the transmitter 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 transmitter channels desired to configure with the selected txEnableMode.

Table 107. adi_adrv9025_RxRadioCtrlModeCfg_t Definition

Member 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 receiver signal path
	ADI_ADRV9025_RX_EN_PIN_MODE	Setting this mode selects the pin mode to control the receiver signal path
	ADI_ADRV9025_RX_EN_INVALID_MODE	Setting this mode does not modify the currently set mode to control the receiver 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 receiver channels you want to configure with the selected rxEnableMode

Table 108. adi_adrv9025_ORxRadioCtrlModeCfg_t Definition

Member 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 observation receiver 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 103
	ADI_ADRV9025_ORX_EN_SINGLE_CH_2PIN_MODE	Setting this mode puts the device in Single Channel 2 pin mode, as described in Table 103
	ADI_ADRV9025_ORX_EN_SINGLE_CH_1PIN_MODE	Setting this mode puts the device in Single Channel 1 pin mode, as described in Table 103
	ADI_ADRV9025_ORX_EN_DUAL_CH_4PIN_MODE	Setting this mode puts the device in Dual Channel 4 pin mode, as described in Table 103
	ADI_ADRV9025_ORX_EN_DUAL_CH_2PIN_MODE	Setting this mode puts the device in Dual Channel 2 pin mode, as described in Table 103
	ADI_ADRV9025_ORX_EN_INVALID_MODE	Setting this mode does not modify the currently set mode to control the observation receiver signal path
orxPinSelectSettlingDelay_armClkCycles	Minimum value: 0, maximum value: 16	Amount of time for the firmware to wait before sampling pins used for observation receiver 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 observation receiver enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX2_FE	Selects ORx2 when in Single Channel 1 pin observation receiver enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX3_FE	Selects ORx3 when in Single Channel 1 pin observation receiver enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_ORX4_FE	Selects ORx4 when in Single Channel 1 pin observation receiver enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_INVALID_ORX_SEL	Does not modify the current mode of the observation receiver when in Single Channel 1 pin observation receiver enable mode

Member 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 observation receiver 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 observation receiver 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 observation receiver 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 observation receiver enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_INVALID_ORX_SEL	Does not modify the current mode of the observation receiver when the ORX_CTRL_B pin is low in Single Channel 2 pin observation receiver 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 observation receiver 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 observation receiver 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 observation receiver 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 observation receiver enable mode
	ADI_ADRV9025_SINGLE_CH_PIN_MODE_INVALID_ORX_SEL	Does not modify the current mode of the observation receiver when the ORX_CTRL_B pin is high in Single Channel 2 pin observation receiver 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 device is in Dual Channel 2 pin mode
	ADI_ADRV9025_DUAL_CH_PIN_MODE_ORX1_ORX4_SEL	Selects ORx1 and ORx4 when the device is in Dual Channel 2 pin mode

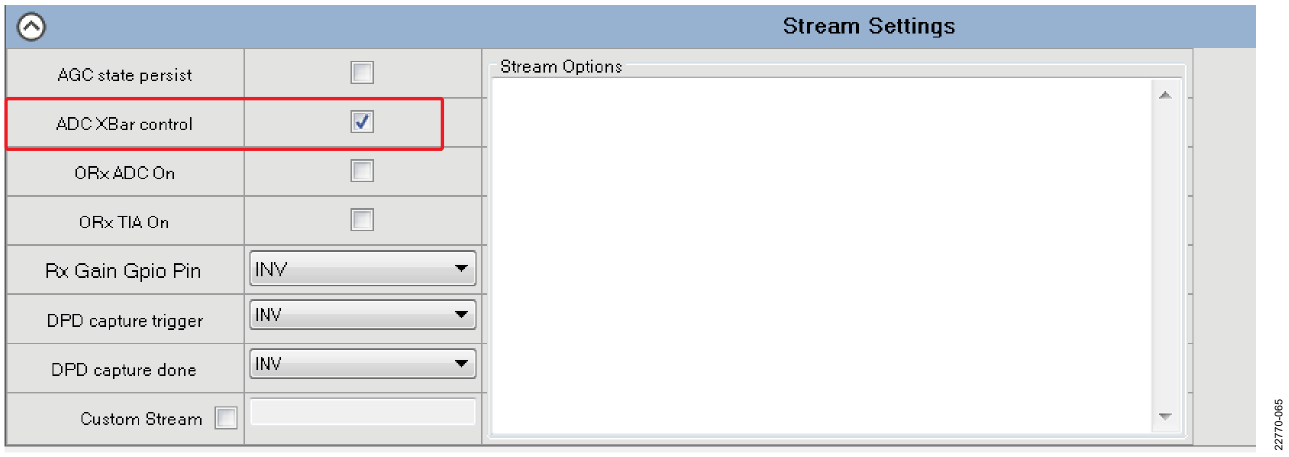
Member Name	Value	Description
	ADI_ADRV9025_DUAL_CH_PIN_MODE_ORX2_ORX3_SEL	Selects ORx2 and ORx3 when the device is in Dual Channel 2 pin mode
	ADI_ADRV9025_DUAL_CH_PIN_MODE_ORX2_ORX4_SEL	Selects ORx2 and ORx4 when the device is in Dual Channel 2 pin mode
	ADI_ADRV9025_DUAL_CH_PIN_MODE_INVALID_ORX_SEL	Does not modify the current mode of the observation receiver when the device is in Dual Channel 2 pin mode

ADC Crossbar Control

There are two control modes for the ADC crossbar (Xbar) switches that feed the JESD204B and JESD204C interface serializers during link sharing mode. In the default mode, the receiver channel is connected to the serializer when the enable pin of the channel is active, and the observation receiver channel is connected to the serializer when the ORX_CTRL pins are driven to select the observation receiver channel. A second mode called ADC crossbar toggling exists that assigns the path control solely to the observation receiver channel control signals.

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

ADC crossbar control 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 66.


Stream Settings

AGC state persist	☐	
ADC XBar control	✓	
ORx ADC On	☐	
ORx TIA On	☐	
Rx Gain Gpio Pin	INV	
DPD capture trigger	INV	
DPD capture done	INV	
Custom Stream ☐		

Figure 66. Stream Settings Window for Selecting ADC Xbar Control Mode

USE CASES

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

4 Transmitter/4 Receiver/2 Observation Receiver Input Use Case

In the 4 transmitter/4 receiver/2 observation receiver use case, the transceiver is configured in a way so that two transmitters feed back into one observation receiver 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 observation receiver is enabled and selected for the observation purposes of the user. ORX_CTRL_A is high at all times, because an observation receiver path is always being used. When ORX_CTRL_A goes low, regardless of the state of ORX_CTRL_B, no observation receiver channel is enabled. ORX_CTRL_B determines which observation receiver 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 can be used in place of ORx3. At least one observation receiver from each side of the device must be used. Therefore, either ORx1 or ORx2 must be used for calibrations on Tx1 and Tx2. The observation receiver from one side of the device cannot be used to calibrate the transmitter 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. Because 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 while 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 that the Tx1 path is routed back to ORx2. Likewise, when ORX2_TX_SEL is high, this indicates that the Tx2 path is routed back to the ORx2 input. This is similar for ORX3_TX_SEL, in a way that when this signal is low, it indicates that the Tx3 path is routed back to the ORx3 input. Likewise, when ORX3_TX_SEL is high, the Tx4 path is routed back to the ORx3 input.

For this use case, internal calibrations can be performed on the inactive observation receiver channel while an external calibration is running on the active channel. In the first time slot of the timing diagram in Figure 68, it is shown that ORx2 is enabled by the user. PA1 and PA3 have been routed back to ORx2 and ORx3, respectively. The transceiver 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 transmitter channel and its corresponding observation receiver 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 68, ORx2 is still enabled for the user with PA2 and PA4 made available to ORx2 and ORx3. The system can perform calculations for PA2 via ORx2 while performing a QEC tracking calibration 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. Also, the same JESD204B and JESD204C link can be used for ORx2 and ORx3 in this scenario because only one observation receiver is used at any given time.

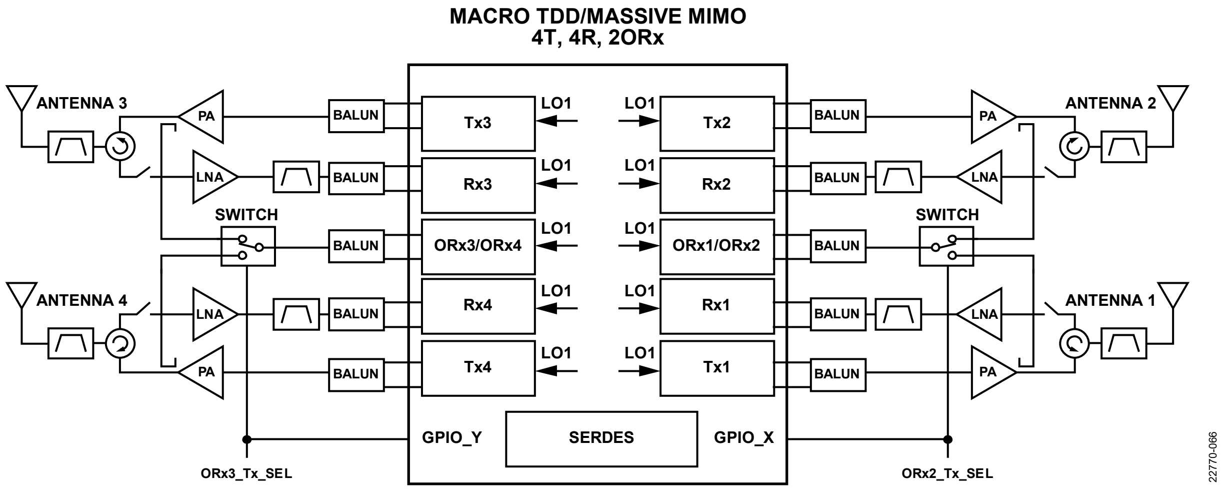


Figure 67. 4 Transmitter/4 Receiver/2 Observation Receiver Configuration

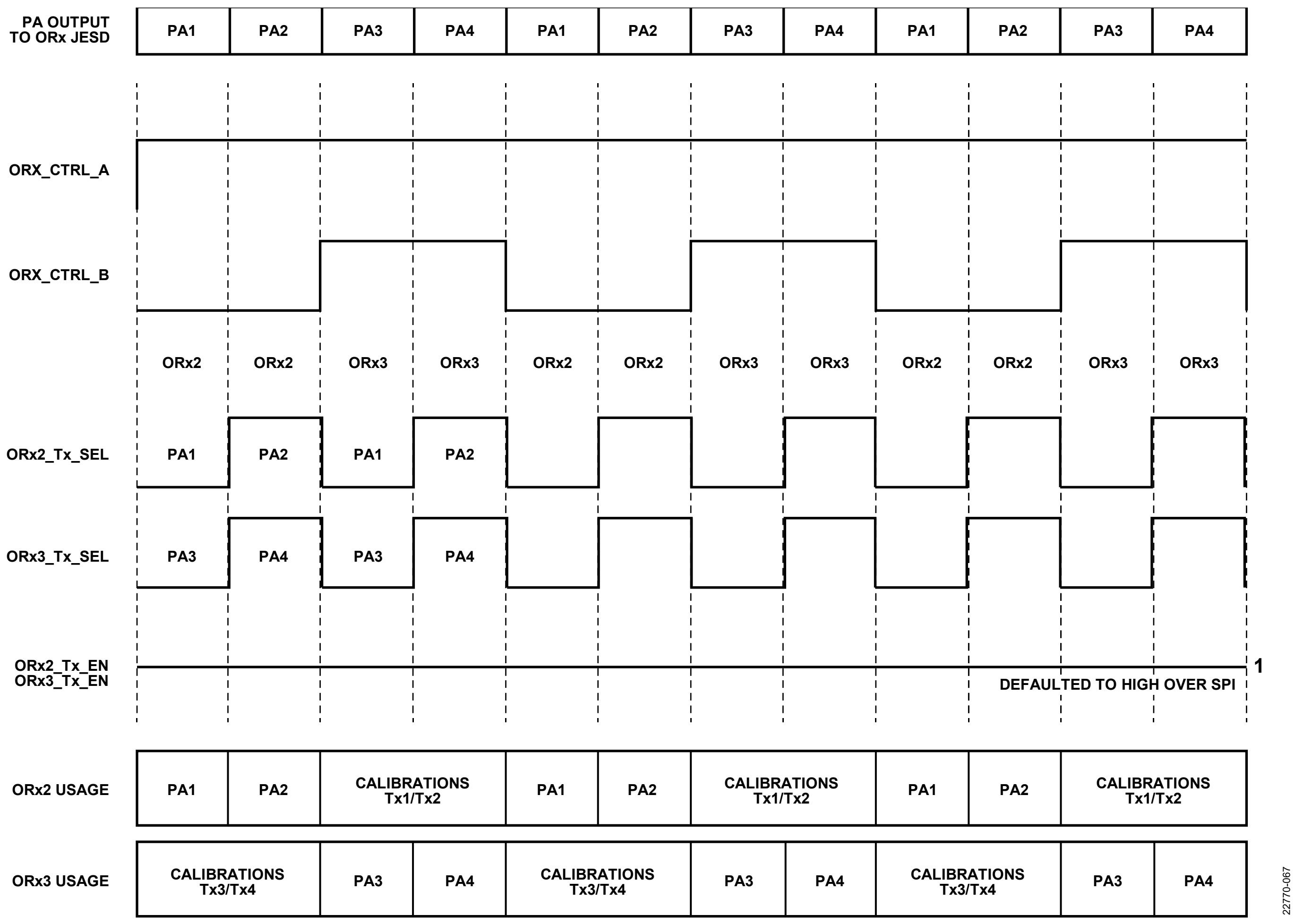

PA OUTPUT TO ORx JESD
PA1 PA2 PA3 PA4 PA1 PA2 PA3 PA4 PA1 PA2 PA3 PA4
ORX_CTRL_A
ORX_CTRL_B
ORx2 ORx2 ORx3 ORx3 ORx2 ORx2 ORx3 ORx3 ORx2 ORx2 ORx3 ORx3
ORx2_Tx_SEL
PA1 PA2 PA1 PA2
ORx3_Tx_SEL
PA3 PA4 PA3 PA4
ORx2_Tx_EN
ORx3_Tx_EN
1
DEFAULTED TO HIGH OVER SPI
ORx2 USAGE
PA1 PA2 CALIBRATIONS Tx1/Tx2 PA1 PA2 CALIBRATIONS Tx1/Tx2 PA1 PA2 CALIBRATIONS Tx1/Tx2
ORx3 USAGE
CALIBRATIONS Tx3/Tx4 PA3 PA4 CALIBRATIONS Tx3/Tx4 PA3 PA4 CALIBRATIONS Tx3/Tx4 PA3 PA4
22770-067

Figure 68. Observation Receiver Enable and Transmitter Select Signals: 4 Transmitter/4 Receiver/2 Observation Receiver Configuration

4 Transmitter/4 Receiver/4 Observation Receiver Input Use Case

In this use case, each transmitter is routed back to its own observation receiver input. The transceiver is configured in Single Channel 3 pin mode for this use case. ORX_CTRL_A is principally high all the time, meaning an observation receiver path is always being used. ORX_CTRL_B and ORX_CTRL_C determine what observation receiver channel is enabled and selected for the observation purposes of the user. Refer to Table 109 for how each observation receiver is selected via the two observation receiver select signals.

Table 109. Observation Receiver Select Logic

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

Because each transmitter is routed back to a separate observation receiver 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 70 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. Therefore, 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 transmitter is to run in a given time slot. Because each transmitter is permanently routed back to its own observation receiver, the external path always exists for external LOL tracking to run.

Because only one observation receiver is used at any given time, the same JESD204B and JESD204C link for ORx1, ORx2, ORx3, and ORx4 can be used in this scenario.

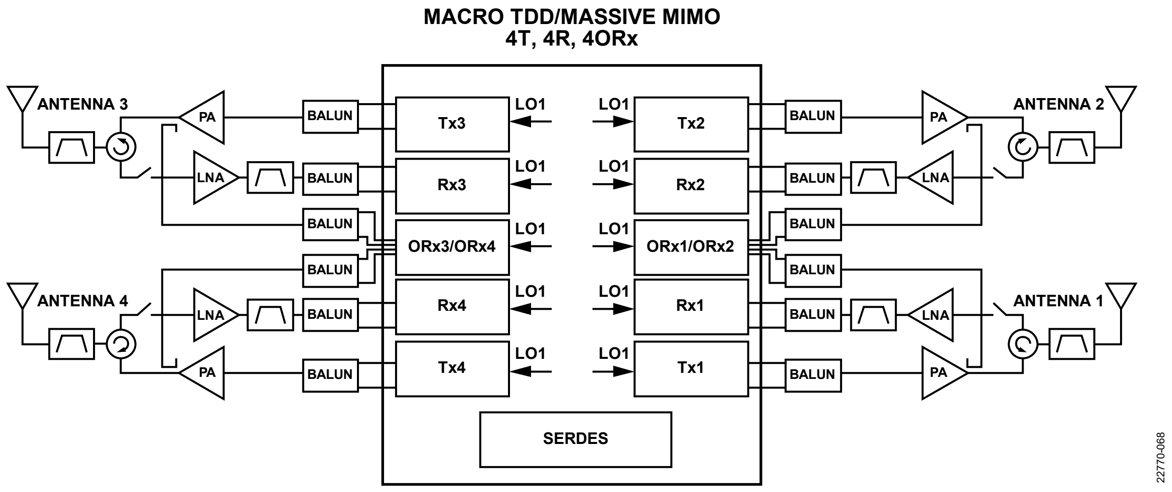


Figure 69. 4 Transmitter/4 Receiver/4 Observation Receiver Configuration

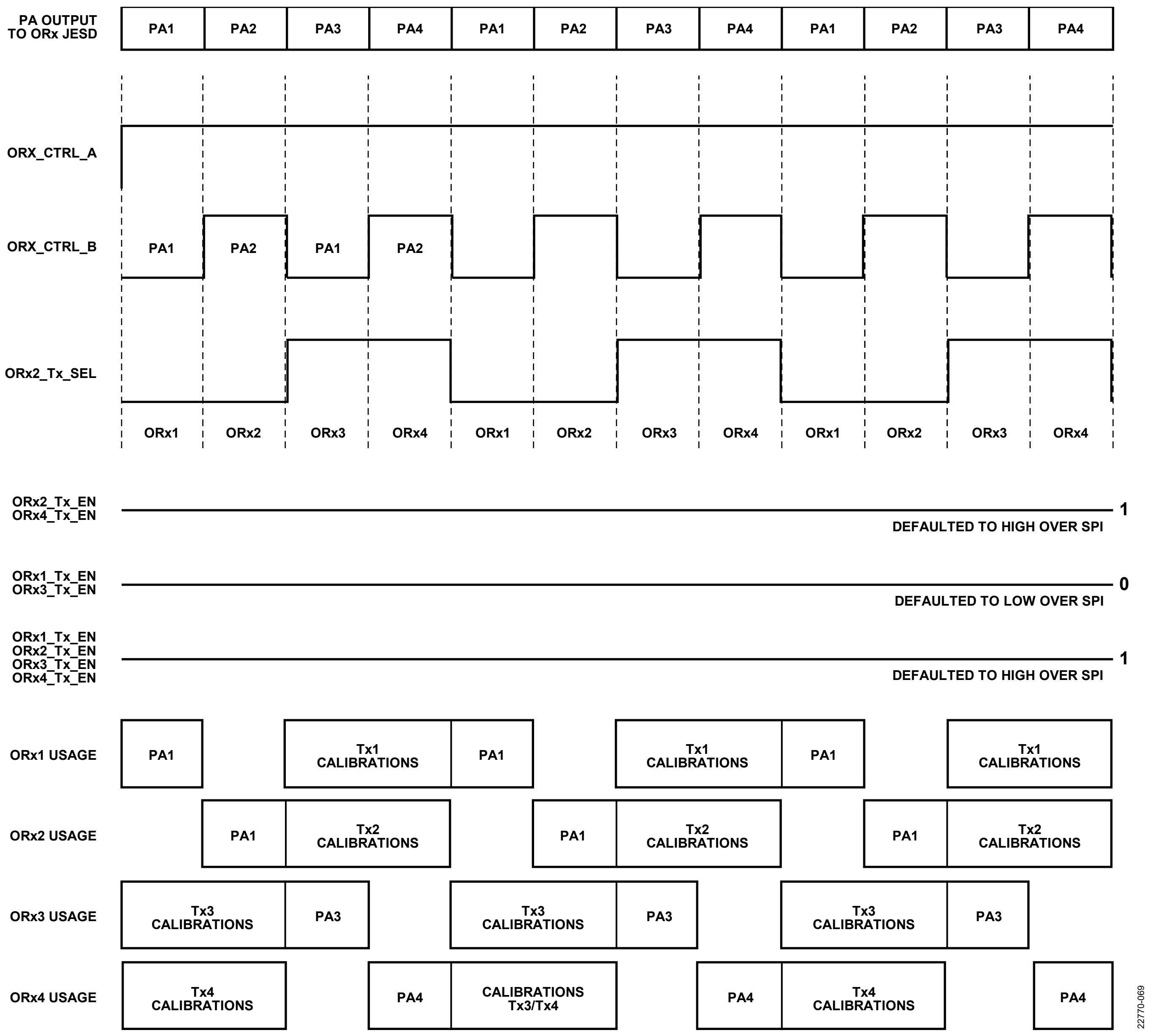


Figure 70. Observation Receiver Enable and Transmitter Select Signals: 4 Transmitter/4 Receiver/4 Observation Receiver Configuration

4 Transmitter/4 Receiver/2 Observation Receiver Input – Single Point of Feedback from 4 Transmitter to Observation Receiver Use Case

This use case shows an example where all the observation receiver paths are shared through one common feedback point. Because there are two sides to the device from a calibration perspective, the user must route Tx1 and Tx2 to either ORx1 or ORx2, respectively. 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. Therefore, if Tx2 is being fed back through this single feedback point, ORx2 is used for transceiver calibrations and 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, use Single Channel 2 pin mode. ORX_CTRL_A is set high all the time because an observation receiver path is always being used. ORX_CTRL_B selects which observation receiver 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 transceiver that there is no external feedback path between this observation receiver input and a transmitter 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 transceiver that there is no external feedback path between this observation receiver input and a transmitter 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 transceiver can perform both calculations on a given PA and calibrations with the other observation receiver input for the same side of the chip. Though the transmitter calibrations must be performed with an observation receiver from the same side of the chip, the PA calculations do not have that constraint. The first time slot in Figure 72 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 Figure 72 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 because 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 JESD204B and JESD204C link can be used for ORx2 and ORx3 in this scenario because only one observation receiver is used at any given time.

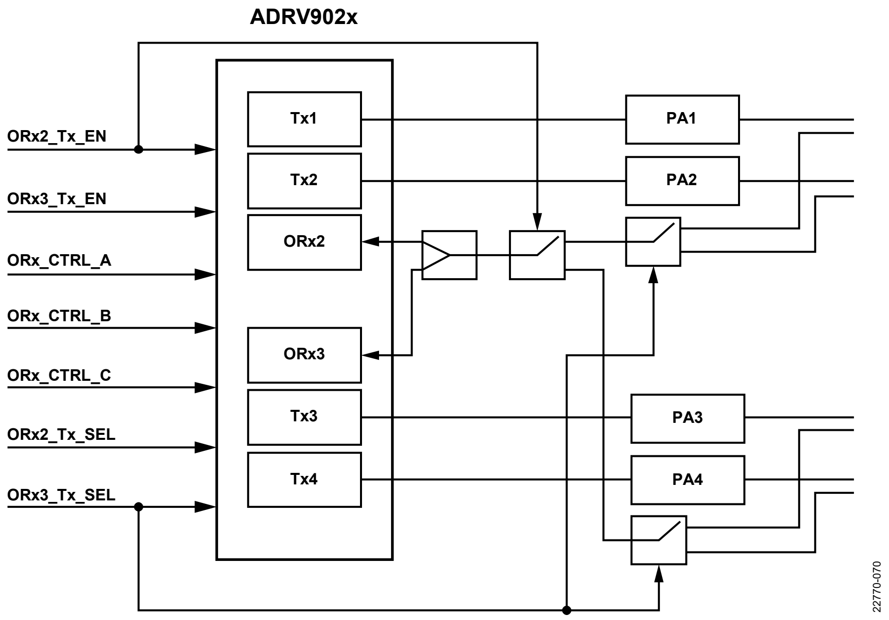


Figure 71. Observation Receiver Channel Routing: 4 Transmitter to 2 Observation Receiver Channels

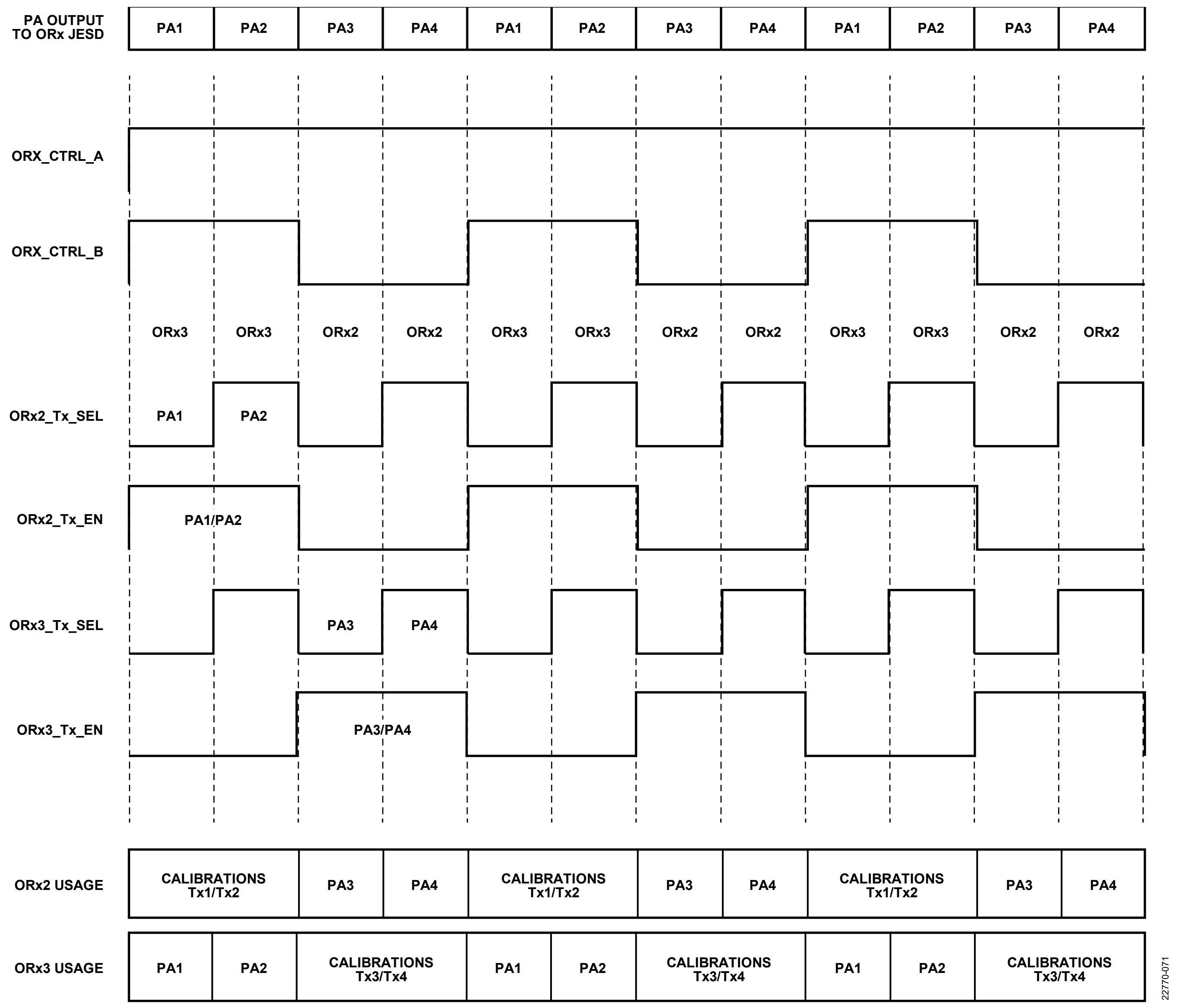


Figure 72. Observation Receiver Enable and Transmitter Select Signals: 4 Transmitter to 2 Observation Receiver Multiplexed Configuration

TRANSMITTER OVERVIEW AND PATH CONTROL

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

There are 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 an SPI2 mode that enables real time operation using a GPIO pin. The choice of attenuation mode is set by the API 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 datapath. The default table provides a range of 0 dB to 41.95 dB of attenuation, with a step size of 0.05 dB, resulting in 840 available attenuation settings.

The transmitter 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, resulting in 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 transmitter datapath can be configured to automatically ramp the attenuation to the maximum level under certain conditions, such as the JESD204B and JESD204C link dropping (rampJesdDfrm) or the transmitter PLL unlocking (disTxDataIfPllUnlock), to prevent spurious transmission in the event of these types of system errors.

Test tones may be generated digitally in the transmitter baseband path. This function is useful for testing/debugging before the JESD204B and JESD204C link has been established. The frequency can be set from −(Transmitter Input Rate)/2 to +(Transmitter Input Rate)/2. The transmitter attenuation is manually overridden when this function is enabled. When test tones are selected as the transmitter input, the analog portion of the transmitter attenuation is set to 0 dB (maximum output power), and the digital portion is set by the API txToneGain.

API COMMANDS

Several API commands are available to adjust the transmitter paths after initialization and during normal operation. The API descriptions in this section 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

This command configures transmitter power control.

Parameters

Table 110. adi_adrv9025_TxAttenCfgSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txAttenCfg[]	An array of structures of type adi_adrv9025_TxAttenCfg_t detailed in Table 111.
attenCfgs	The number of configurations passed in the array.

Table 111. 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 enumerators as listed below. Data type: uint32_t	
	Parameter	**Transmitter Channel**
	ADI_ADRV9025_TXOFF	No transmitter 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 transmitter 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 maximum if the RFPLL unlocks. Data type: adi_adrv9025_TxDataIfUnlock_e.	
	Parameter	**Action**
	ADI_ADRV9025_TXUNLOCK_TX_NOT_DISABLED	Do not alter transmitter attenuation in an unlock event.
	ADI_ADRV9025_TXUNLOCK_TX_RAMP_DOWN_TO_MIN_ATTEN	Ramp transmitter attenuation to maximum in an unlock event.
rampJesdDfrm	Ramp up attenuation when a deframer link unlocks. Note that this field is not being used actively. If the 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 transmitter attenuation mode. Data type: adi_adrv9025_TxAttenMode_e.	
	Parameter	**Mode**
	ADI_ADRV9025_TXATTEN_BYPASS_MODE	Transmitter attenuation mode. Bypass: zero total attenuation.
	ADI_ADRV9025_TXATTEN_SPI_MODE	Transmitter attenuation set by 10-bit index programmed over SPI.
	ADI_ADRV9025_TXATTEN_GPIO_MODE	Transmitter 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 transmitter 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

This command reads transmitter power control configuration one channel at a time.

Parameters

Table 112. adi_adrv9025_TxAttenCfgGet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannel	The transmitter channel to be read back using an enumerator as described in Table 111.
*txAttenCfgs	The pointer to the readback structure of the queried transmitter channel as defined in Table 111.

adi_adrv9025_TxAttenSet

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

Description

This command sets transmitter attenuation when transmitter attenuation mode is set to ADI_ADRV9025_TXATTEN_SPI_MODE.

Parameters

Table 113. adi_adrv9025_TxAttenSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txAttenuation[]	An array of structures of type adi_adrv9025_TxAtten_t detailed in Table 114.
numTxAttenConfigs	The number of configurations passed in the array.

Table 114. 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 enumerators as listed below. Data type: uint32_t.	
	Parameter	**Transmitter Channel**
	ADI_ADRV9025_TXOFF	No transmitter 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

This command reads transmitter attenuation when the transmitter attenuation mode is set to ADI_ADRV9025_TXATTEN_SPI_MODE or ADI_ADRV9025_TXATTEN_GPIO_MODE.

Parameters

Table 115. adi_adrv9025_TxAttenGet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannel	The transmitter channel to be read back using an enumerator as described in Table 114.
*txAttenuation	Pointer to the readback structure of the queried transmitter channel as defined in Table 114.

adi_adrv9025_TxAttenModeSet

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

Description

This command sets the transmitter attenuation mode independent of the initialization structure.

Parameters

Table 116. adi_adrv9025_TxAttenModeSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannel	Transmitter channel upon which the API acts as described in Table 117.
*txAttenMode	Pointer to the desired mode of attenuation using an enum as described in Table 117.

Table 117. 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	**Transmitter Channel**
	ADI_ADRV9025_TXOFF	No transmitter 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 transmitter channels selected.
txAttenMode	Selects the transmitter attenuation mode. Data type: adi_adrv9025_TxAttenMode_e.	
	Parameter	**Mode**
	ADI_ADRV9025_TXATTEN_BYPASS_MODE	Transmitter attenuation mode. Bypass: zero total attenuation.
	ADI_ADRV9025_TXATTEN_SPI_MODE	Transmitter attenuation set by 10-bit index programmed over SPI.
	ADI_ADRV9025_TXATTEN_GPIO_MODE	Transmitter 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

This command generates test tones in the transmitter baseband path.

Parameters

Table 118. adi_adrv9025_TxTestToneSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txNcoTestToneCfg[]	An array of structures of type adi_adrv9025_TxAttenCfg_t as detailed in Table 119.
arraySize	The number of configurations passed in the array.

Table 119. 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	**Transmitter Channel**
	ADI_ADRV9025_TXOFF	No transmitter 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 transmitter 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 transmitter LO leakage (transmitter LOL) specification. As shown in Figure 73, the gain is realized in the DAC output but before the transmitter predistortion (LPF) filters, which is where the majority of the flicker noise observed on the transmitter 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, this function provides an additional 3 dB of separation to the noise and transmitter 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. Therefore, the user is cautioned when transmitting signals with very low PAR. 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 signals with PAR of about 12 dB, the signal chain has enough headroom for minimal performance impact, shown in the data in Table 120, Table 121, and Table 122.

Because the transmitter 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 internal calibrations after the device has been configured.

The transmitter predistortion low pass filters (LPF) are the main contributors of flicker noise to the transmitter signal chain. Because the gain occurs before them, the amount of transmitter LOL emitted from the device is not changed by enabling the 3 dB mode. The transmitter attenuators follow the filters in the signal chain. For this reason, transmitter LOL reduces at the output with each attenuator step, dB for dB. The transmitter LOL measurement for both enabled and disabled modes along with the margin gained when the function is enabled is presented in Figure 73.

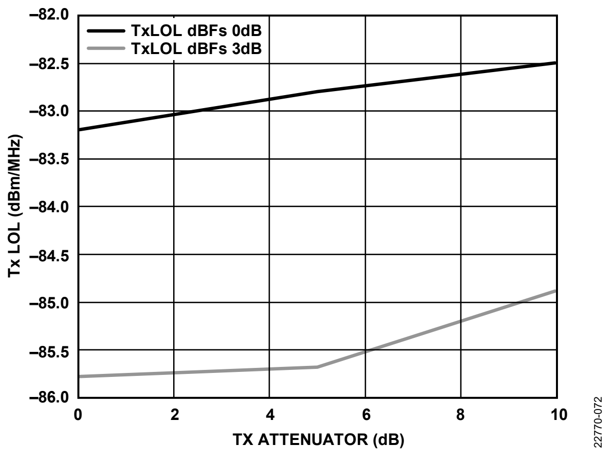


Figure 73. Transmitter LO Leakage with dacFullScale Enabled/Disabled

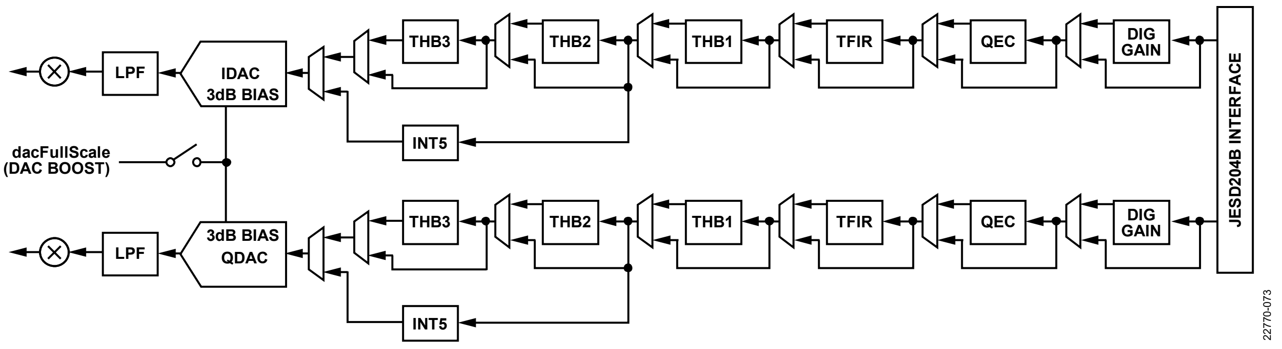


Figure 74. Transmitter Datapath with dacFullScale Function

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

The improvement shown in Table 120 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 the performance levels requirements in the device data sheet, 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 Table 121. In general, the OIP3 is only slightly affected by enabling the boost with the same input tone levels (both tones level= −15 dBFS). Typical performance is approximately 30 dBm, and measurements of the transceiver in both modes are consistent. Impact on linearity is shown in Table 121.

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 122.

It is a system requirement that the desired DAC boost mode must be configured prior to device initialization. The transmitter signal level is increased, which impacts internal calibrations. Therefore, DAC boost mode settings cannot be modified during device operation.

Table 120. dacFullScale Transmitter LOL and Transmitter Output Power Comparison 0 dB Mode and 3 dB Mode

DAC Full Scale Setting	Transmitter Attenuation (dB)	Transmitter LOL (dBm/MHz)	Tone Power (dBm)	0 dBFS in dBm	Transmitter LOL (dBFS)	Improvement (dB) Relative to 0 dB Setting
dacFullScale 0 dB 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 3 dB 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 121. dacFullScale Transmitter 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 122. dacFullScale EVM vs. Mode Selection

	0 dB Mode		3 dB Mode	
Transmitter 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 123 and Table 124. The dacFullScale parameter is also found in the json (profile) file.

Table 123. adi_adrv9025_TxChannelCfg Structure Parameters

Data Fields	Description
adi_adrv9025_TxProfile_t	profile
adi_adrv9025_DacFullScale_e	dacFullScale

Table 124. adi_adrv9025_DacFullScale_e Enumerator 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 transceiver 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 transmitter attenuation ramp to set the transmitter 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 in Table 207.

PA PROTECTION DESCRIPTION

The PA protection circuitry is designed to alert the user that the digital signal power within the transmitter 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 GPINTx 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 the system given the PA damage power level and transmitter RF attenuation.

The following are the two types of thresholds in the PA protection circuit:

- 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 transmitter 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 adi_adrv9025_TxPaProtectionCfgSet(…) API command.

adi_adrv9025_TxPaProtectionCfgSet(…)

```
adi_adrv9025_TxPaProtectionCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_TxPaProtectCfg_t
  txPaProtectCfg[], uint8_t arraySize);
```

Description

This command sets up the PA protection feature.

Parameters

Table 125. adi_adrv9025_TxPaProtectionCfgSet(…) Parameters

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 126.
arraySize	The array length of txPaProtectCfg[].

Table 126. adi_adrv9025_TxPaProtectCfg_t Data Structure Parameters

Parameter Name	Data Type	Parameter Description
txChannel	adi_adrv9025_TxChannels_e	Transmitter channel select based on adi_adrv9025_TxChannel_e. PA protection configuration is applied to channels selected by this parameter

Parameter Name	Data Type	Parameter Description
avgDuration	uint8_t	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): $t_{avgDuration} = \frac{L}{txSampleRate} \times 2^{avgDuration} + 5$
peakDuration	uint8_t	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): $t_{peakDuration} = \frac{1}{txSampleRate} \times 2^{peakDuration} + 5$
powerThreshold	uint16_t	Sets the powerThreshold for average power measurements. If the average power exceeds this threshold, the avgPowerErr signal is asserted. $powerThreshold_{dBFS} = 10log\left(\frac{powerThreshold}{8192}\right)$
peakCount	uint8_t	Sets a limit for the number of peaks detected within a peakDuration. When this limit is exceeded, the PA protection peakPowerErr signal is asserted.
peakThreshold	uint16_t	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. $peakThreshold_{dBFS} = 10log\left(\frac{peakThreshold}{8192}\right)$
avgPowerEnable	uint8_t	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.
peakPowerEnable	uint8_t	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.
inputSel	adi_adrv9025_PaProtectionInputSel_e	Determines the data path location for peak and average power measurement. Options are given by the enumeration described in Table 127.
avgPeakRatioEnable	uint8_t	When set = 1, this enables the average to peak power ratio block. avgPowerEnable and peakPowerEnable must be enabled. When set = 0, average to peak power calculations are not performed.

Table 127 describes the adi_adrv9025_PaProtectionInputSel_e enumeration. These measurement locations are shown in Figure 75.

Table 127. adi_adrv9025_PaProtectionInputSel_e Enumeration Options

Enumeration	Enumeration 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 transmitter QEC actuator output. This selection is only valid when the clear required mode is not set.

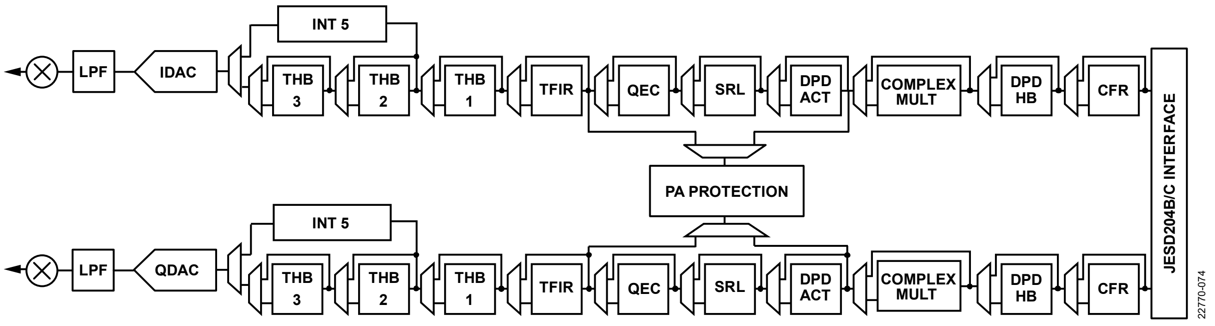


Figure 75. Transmitter Datapath Showing PA Protection Measurement Locations

PA Protection Run Time 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

This command 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 128. adi_adrv9025_TxPaProtectionErrFlagsGet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannel	The transmitter channel mask that selects which transmitter to retrieve error flag information from.
errorFlags	A data structure containing the error flag information for selected transmitter channel.

Table 129. adi_adrv9025_TxPaProtectionErr_t Data Structure Parameters

Data Type	Parameter Name	Parameter Description
uint8_t	peakPowerErr	If value = 1, the peak power error bit is asserted. If value = 0, the peak power error is not asserted. This bit is sticky depending on the configuration applied in adi_adrv9025_TxAttenuationRampUpStickyModeEnable(…).
uint8_t	avgPowerErr	If value = 1, the average power error bit is asserted. If value = 0, the average power error is not asserted. This bit is sticky depending on the 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 when 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(…). After 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 adi_adrv9025_TxPaProtectionErrFlagsReset(…) command. Note that adi_adrv9025_PaPllDfrmEventClear(…) can clear a PA protection error, a PLL unlock interrupt, or a deframer interrupt. The adi_adrv9025_TxPaProtectionErrFlagsReset(…) command is specific to only 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

This command clears PA protection error flags for specified channels.

Parameters

Table 130. adi_adrv9025_TxPaProtectionErrFlagsReset(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannel	The transmitter channel mask that selects which transmitter to clear/reset PA protection errors.
errorFlags	An enumerated data type describing which error flags must be cleared.

Table 131 describes the adi_adrv9025_TxPaProtectErrFlags_e enumeration.

Table 131. adi_adrv9025_TxPaProtectErrFlags_e Enumeration Options

Enumeration	Enumeration 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 datapath. 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

This command reads back the transmitter average IQ sample power.

Parameters

Table 132. adi_adrv9025_TxPaProtectionStatusGet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannel	The transmitter 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 transmitter channel.

The data structure type adi_adrv9025_TxPaProtectStatus_t is described in Table 133.

Table 133. adi_adrv9025_TxPaProtectStatus_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: $P_{AVG} = 10log\left(\frac{avgPower}{2^{16}}\right)$
uint16_t	avgPeakRatio	Measurement describing the average to peak ratio as measured by PA protection. Enable peak and average power measurement for meaningful results. $PAR = 10log\left(\frac{avgPeakRatio}{2^{15}}\right)$
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. $P_{avgErrPow} = 10log\left(\frac{avgErrorPower}{2^{16}}\right)$

adi_adrv9025_PaPllDfrmEventRampDownEnableSet(…)

```
adi_adrv9025_ PaPllDfrmEventRampDownEnableSet(adi_adrv9025_Device_t* device, uint32_t
txChannelMask, uint32_t irqMask, uint8_t enable);
```

Description

This command configures which interrupts can trigger a transmitter attenuation ramp down event.

Parameters

Table 134. adi_adrv9025_PaPllDfrmEventRampDownEnableSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannelMask	The transmitter 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, 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 135.
enable	Bit that controls ramp down for the events selected by irqMask. If set to 0, the function is disabled for all selections.

Table 135. Bit Descriptions 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	Auxiliary PLL Unlock.	
D2	CLK PLL Unlock.	
D1	Deframer 1 Interrupt/IRQ.	
D0	Deframer 0 Interrupt/IRQ.	

Although 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 Transmitter Attenuation Ramp Down

If a transmitter attenuation ramp down interrupt is asserted, there are two modes of interrupt behavior pertaining to when attenuation is restored. The following behavior modes 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 must 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 using the adi_adrv9025_TxAttenuationRampUpStickyModeEnable 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

This command configures transmitter attenuation ramp up sticky mode for the selected transmitter channel.

Parameters

Table 136. adi_adrv9025_TxAttenuationRampUpStickyModeEnable(…) Parameters

Parameter	Description
*device	Pointer to device structure.
channelMask	The transmitter channel mask for selecting which transmitters to configure based on adi_adrv9025_TxChannels_e enumeration.

Parameter	Description
txPllJesdProtClrReqd	Determines if the user is required to manually clear PLL/deframer attenuation ramp down events after assertion. Setting 1 requires the user to clear. Setting 0 does not require the user to clear.
txPaProtectionAvgpowerErrorClearRequired	Determines if the user is required to manually clear PA protection average power error flag after assertion. Setting 1 requires the user to clear. Setting 0 does not require the user to clear.
txPaProtectionPeakpowerErrorClearRequired	Determines if the user is required to manually clear PA protection peak power error flag after assertion. Setting 1 requires the user to clear. Setting 0 does not require the 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 GPINTx 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

This command reads the status of events causing transmitter attenuation ramp down rather than any signal that has asserted GP_INTERRUPT.

Parameters

Table 137. adi_adrv9025_PaPllDfrmEventGet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannelSelect	The transmitter 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 138. If a bit position in this value is set high, the associated interrupt has asserted to cause a transmitter attenuation ramp down.

The command adi_adrv9025_adrv9025_PaPllDfrmEventClear(…) can be used to clear the error.

Table 138. Bit Descriptions 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 Transmitter 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 must 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 adi_adrv9025_PaPllDfrmEventClear 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

This command clears the transmitter attenuation ramp down interrupts caused by the deframer or PLL unlock events.

Parameters

Table 139. adi_adrv9025_PaPllDfrmEventClear(…) Parameters

Parameter	Description
*device	Pointer to device structure.
txChannelSelect	The transmitter 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 138. 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 transceiver has four receivers (Rx1/Rx2/Rx3/Rx4) that 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 datapath. AGC controls the gain of the device based on the information from a number of signal detectors (peak and 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 base station receivers, the received signal is a multicarrier signal in most cases. Perform a gain change only under large overrange or underrange conditions. 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, and peak detect only is provided. This mode can recover receiver gain quickly in addition to the fast attack capability mentioned previously.

This section contains a full description of the gain control functionality available in the transceiver. Some features may not be available depending on the software revision.

RECEIVER DATAPATH

Figure 76 shows the receiver datapath 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 that 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 mode or AGC mode. First, an analog peak detector (APD) exists prior to the ADC. This peak detector is located after the transimpedance amplifier (TIA) filter, so it receives signals first in the analog domain and also has blocker signal visibility, which can overload the ADC. The second peak detector is called the Half-Band 2 (HB2) overload detector because it monitors the data at the HB2 filter in the digital processing section of 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 datapath is user configurable.

This transceiver can also control an external gain element through use of the receiver gain table and the GPIO_ANA pins.

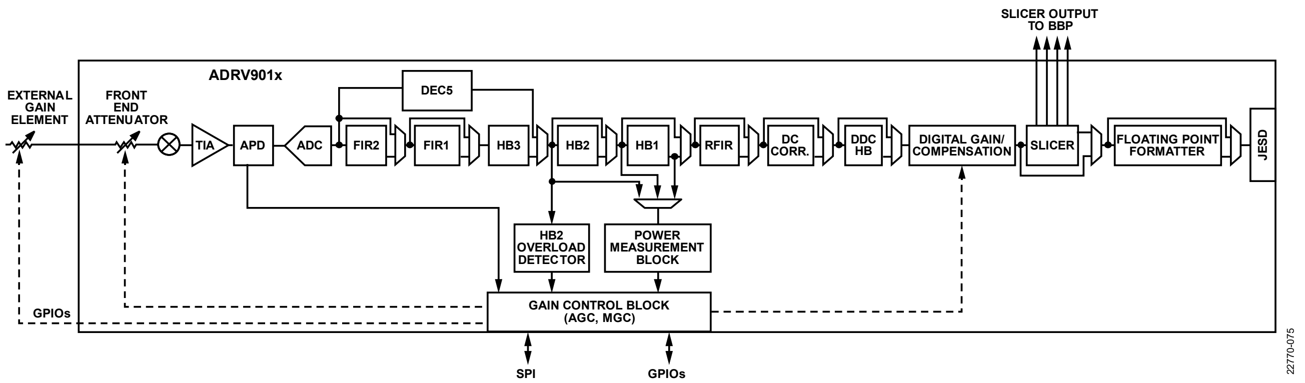


Figure 76. Receiver Data Path and Gain Control Blocks

The gain control block is shown in Figure 77 with multiple inputs providing information. Overload (peak) detectors are shown in red and the power measurement block is shown in blue. The gain control block controls the gain of the signal chain using a gain table.

A gain 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 and MGC) and GPIOs.

Table 140 shows a sample gain table.

Table 140. Sample Rows from the Default Receiver Gain Table

Gain Table Index	Front-End Attenuator, 7 Bits	External Gain Control, 3 Bits	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, but typically the same one is used. The possible range of the gain table is 255 to 0, but 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 receiver. Table 141 shows which analog GPIOs are used for which receiver.

Table 141. 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 the selected gain index, analog GPIO_0 and GPIO_1 are high.

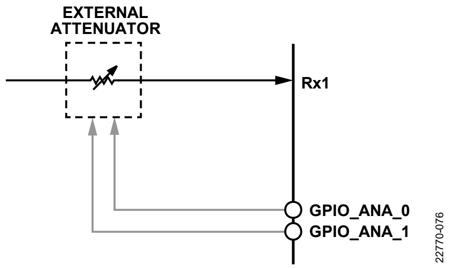


Figure 77. GPIO Control of an External Gain Element to Rx1

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 a 0.7 dB gain, and a value of −14 results in 0.7 dB of attenuation. The combination of TIA and 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 adi_adrv9025_RxGainCtrlModeSet API function.

adi_adrv9025_RxGainCtrlModeSet(…)

```
adi_adrv9025_RxGainCtrlModeSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxAgcMode_t
gainMode[], uint8_t arraySize)
```

Description

This command selects the gain control mode.

Parameters

Table 142. adi_adrv9025_RxGainCtrlModeSet(…) Parameters

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 receiver channel
arraySize	The size of the array

Each adi_adrv9025_RxAgcMode_t instance contains agcMode, an enumerator selecting the chosen gain mode. The possible options are shown in Table 143.

Table 143. Definition of adi_adrv9025_RxAgcMode_t

Enumerator	Gain Mode
ADI_ADRV9025_MGC	Manual gain mode
ADI_ADRV9025_AGCSLOW	Automatic gain control Mode
ADI_ADRV9025_HYBRID	Not currently supported

rxChannelMask

rxChannelMask selects the channels upon which to enable this gain control mode. rxChannelMask is a bit mask with each bit corresponding to a channel, D0 = Rx1, D1 = Rx2, D2 = Rx3, and D3 = Rx4. Setting the rxChannelMask = 15 means that all receivers 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 selecting the gain index. There are two options, API commands and pin control. By default, if MGC is chosen, the device is configured for API commands. The commands described in this section 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

This command selects the gain index in the gain table when in API command mode.

Parameters

Table 144. adi_adrv9025_RxGainSet(…) Parameters

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 the following:

- 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, and 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

This command reads back the gain index in the gain table for the selected channel when in API command mode.

Parameters

Table 145. adi_adrv9025_RxGainGet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
rxChannel	An enumerator as shown in Table 146.

Parameter	Description
*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 146. Definition of adi_adrv9025_RxChannels_e

Receiver	Enumerator
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 and 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 or decrement in gain, as shown in Figure 78. 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).

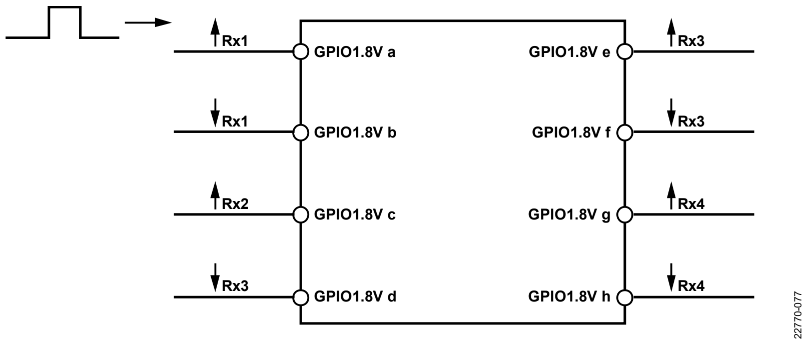


Figure 78. MGC Pin Mode: GPIO1.8V (a through h) Represent Any of GPIO_0 to GPIO_15

adi_adrv9025_RxGainPinCtrlCfgSet(…)

```
adi_adrv9025_RxGainPinCtrlCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxGainPinCfg_t
*rxGainPinCtrlCfg, adi_adrv9025_RxChannels_e rxChannel)
```

Description

This command configures pin control MGC mode.

Parameters

Table 147. adi_adrv9025_RxGainPinCtrlCfgSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
rxChannel	An enumerator indicating which receiver channel to configure, as shown in Table 146.
*rxGainPinCtrlCfg	A configuration structure pointer for the pin control MGC mode containing the members shown in Table 148.

Table 148. 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, 1 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, 1 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 GPIO_0 to GPIO_15 can be used. Acceptable values are ADI_ADRV_9025_GPIO_00 to ADI_ADRV9025_GPIO_15.
adi_adrv9025_GpioPinSel_e rxGainDecPin	GPIO used to decrement gain. Any of GPIO_0 to GPIO_15 can be used. Acceptable values are ADI_ADRV_9025_GPIO_00 to ADI_ADRV9025_GPIO_15.

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 on what causes an overrange condition are provided in the Peak Detect Mode section.

AUTOMATIC GAIN CONTROL

In AGC mode, a built-in state machine automatically controls the gain based on a user defined configuration. The AGC can be configured in one of the following 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 adi_adrv9025_AgcCfg_t AGC configuration structure is used to select the individual modes of the AGC operation, as shown in Table 149.

Table 149. 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 detectors both have a high threshold and a low threshold. These are set with the apdHighThresh, apdLowThresh, hb2HighTresh, and hb2UnderRangeHighThresh parameters. These levels are user programmable, as is the limit for the number of times a threshold must be crossed for an overrange or underrange 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 overrange condition occurs, the AGC reduces the gain (gain attack).

The low thresholds are used as lower limits on signal level. When the signal peaks are not exceeding the lower threshold, this is indicative of a low power signal, and the AGC increases gain (gain recovery). This is termed an underrange. The AGC stable state (where it does not adjust gain) occurs when neither an underrange nor overrange 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 150.

Table 150. Peak Detector Gain Steps

Overload/Underrange	Gain Step
apdHighThresh overrange	Reduce gain by apdGainStepAttack
apdLowThresh underrange	Increase gain by apdGainStepRecovery
hb2HighThresh overrange	Reduce gain by hb2GainStepAttack
hb2UnderRangeHighThresh underrange	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 underrange 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 151 outlines the counter parameters for the individual overload/underrange conditions.

Table 151. 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.

When 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 79 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 again, 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 underrange 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 again.

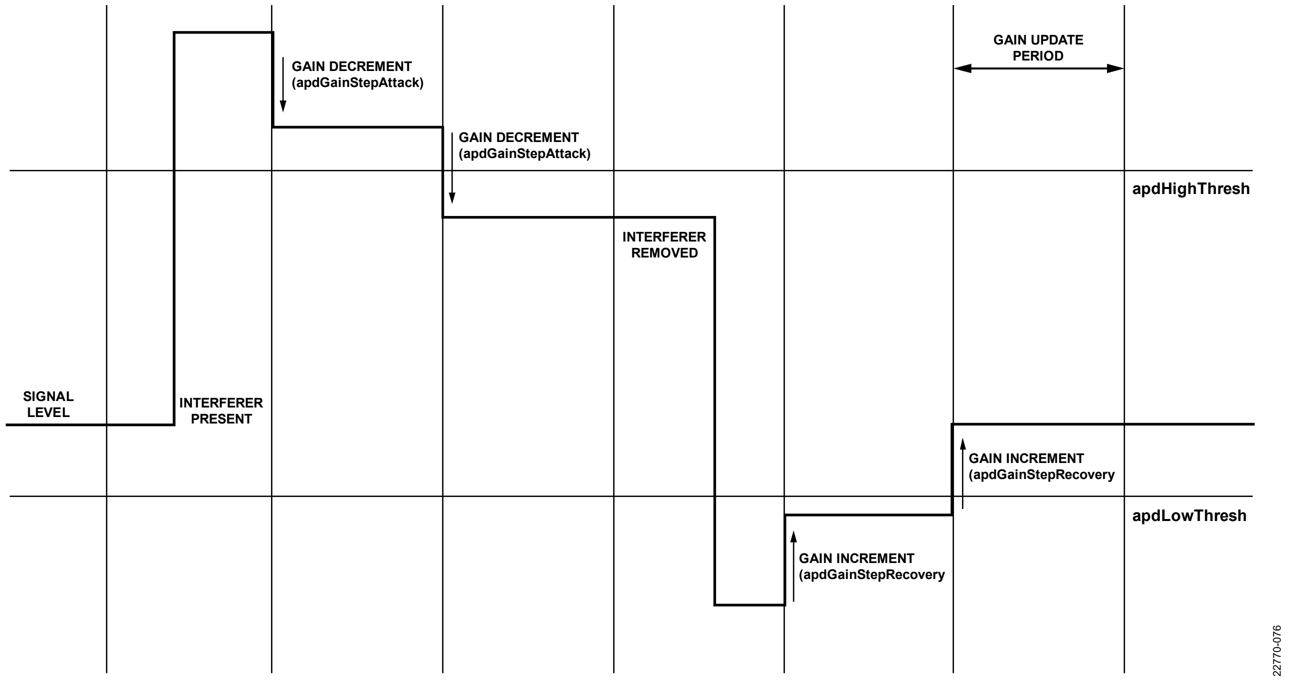

GAIN UPDATE PERIOD
GAIN DECREMENT (apdGainStepAttack)
GAIN DECREMENT (apdGainStepAttack)
apdHighThresh
INTERFERER REMOVED
SIGNAL LEVEL
INTERFERER PRESENT
GAIN INCREMENT (apdGainStepRecovery
apdLowThresh
GAIN INCREMENT (apdGainStepRecovery
22770-076

Figure 79. APD Thresholds and Gain Changes Associated with Underrange and Overrange Conditions

Figure 80 shows the same scenario but from the viewpoint of the HB2 detector considered in isolation.

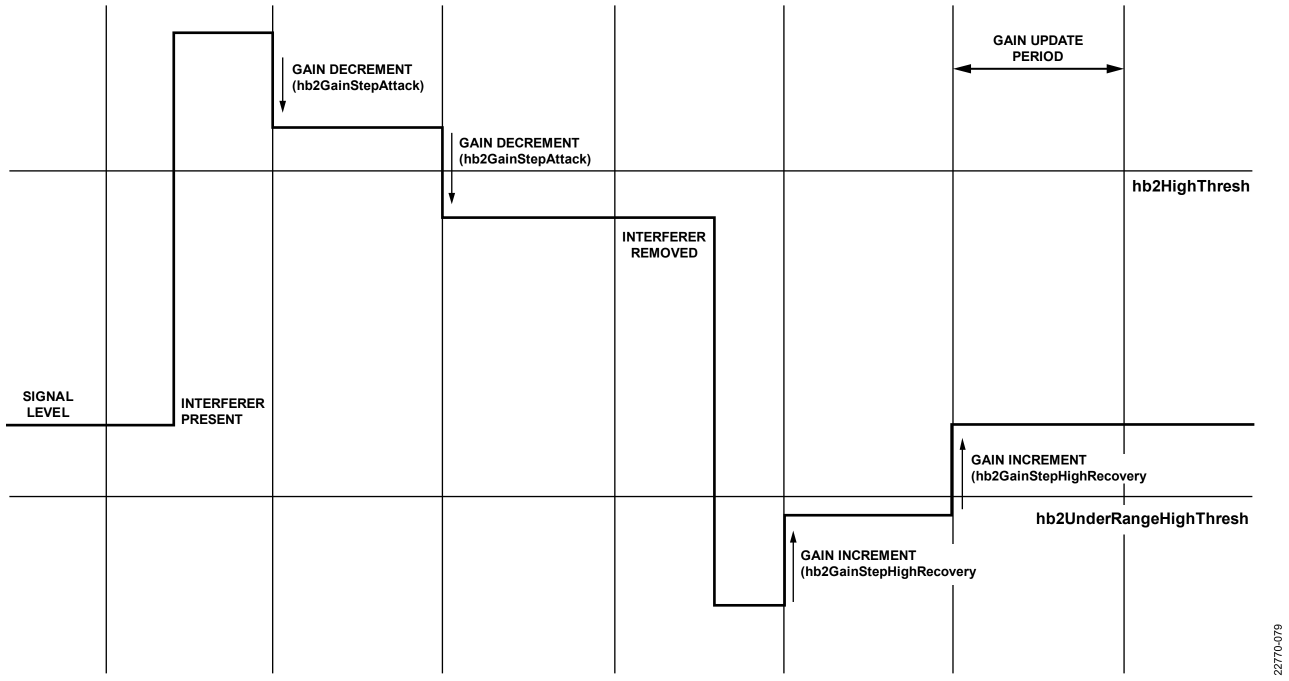

GAIN UPDATE PERIOD
GAIN DECREMENT (hb2GainStepAttack)
GAIN DECREMENT (hb2GainStepAttack)
hb2HighThresh
INTERFERER REMOVED
SIGNAL LEVEL
INTERFERER PRESENT
GAIN INCREMENT (hb2GainStepHighRecovery)
hb2UnderRangeHighThresh
GAIN INCREMENT (hb2GainStepHighRecovery)
22770-079

Figure 80. HB2 Thresholds and Gain Changes Associated with Underrange and Overrange Conditions

It is possible to enable a fast attack mode whereby the AGC is instructed to reduce gain immediately when an overrange condition occurs, instead of waiting until the next expiry of the gain update counter using agcGainChangeIfThreshHigh. This parameter has independent controls for the APD and HB2 detectors. Values from 0 to 3 are valid, as shown in Table 152.

Table 152. 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 81 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, it is best to decrease the gain quickly rather than wait for a suitable moment in the received signal to change the gain.

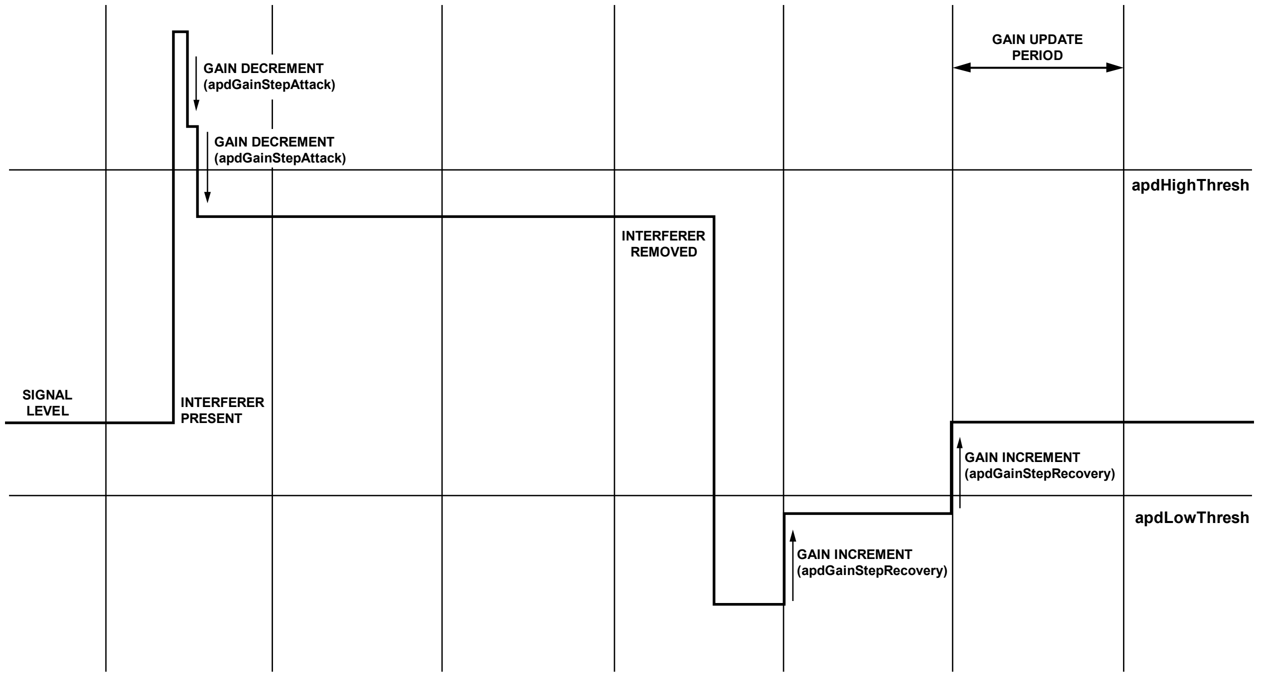


Figure 81. APD Gain Changes with Fast Attack Enabled

Figure 82 shows the same scenario but from the viewpoint of agcChangeGainIfThreshHigh being set for HB2.

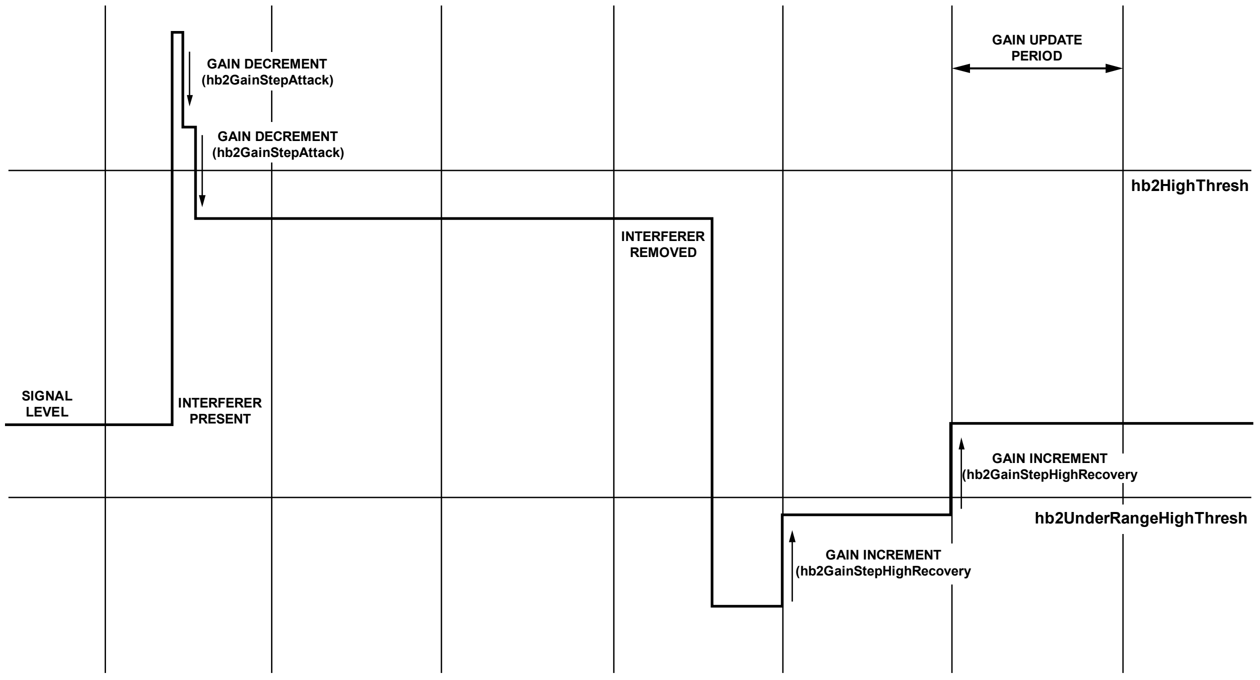


Figure 82. HB2 Gain Changes with Fast Attack Enabled

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 83. This functionality is enabled with the ableFastRecoveryLoop parameter. This fast recovery mode enables the HB2 overload detector. The operation is shown in Figure 84. 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 whereas 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 83 because all gain updates occur at the expiry of the gain update period.

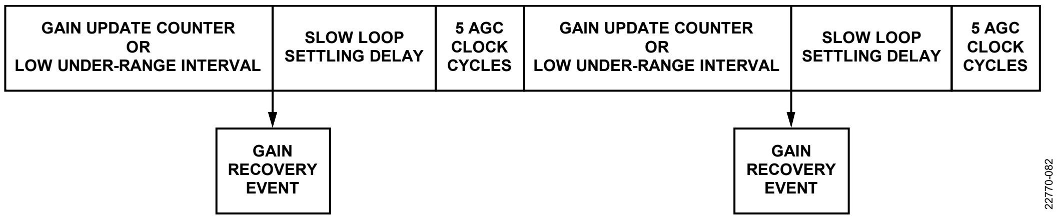


Figure 83. AGC Sequence with HB2 Detector in Fast Recovery Mode

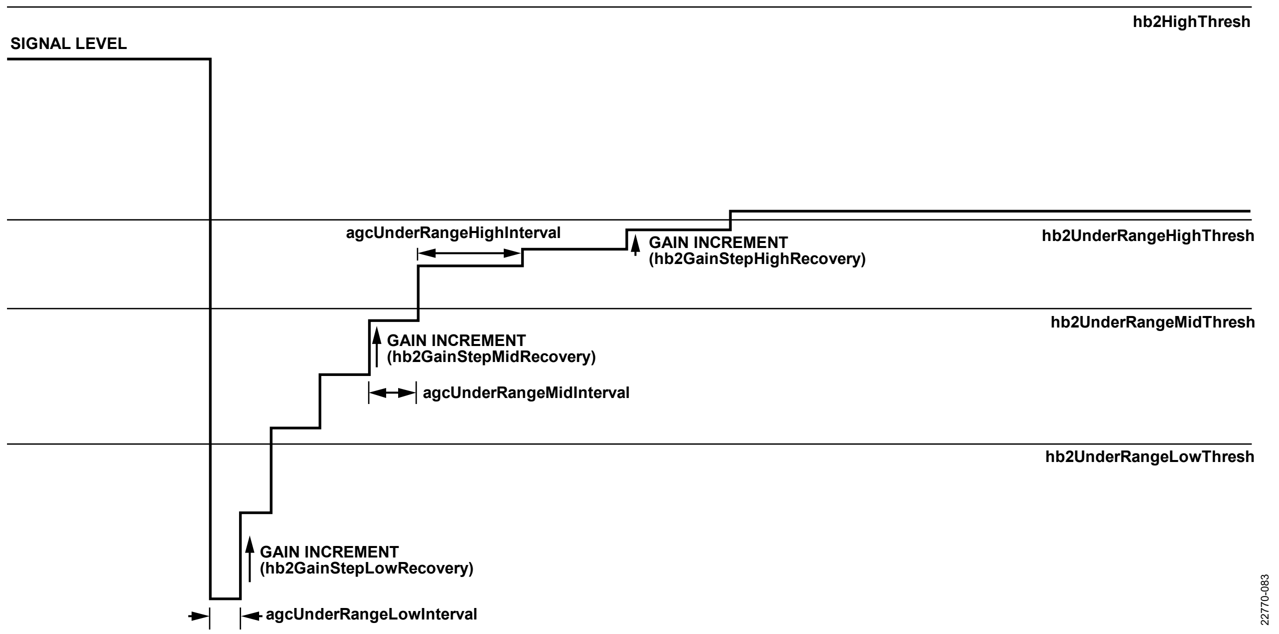


Figure 84. AGC Operation with HB2 Detector in Fast Recovery Mode

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 153 shows the priorities between the detectors when multiple overranges occur.

Table 153. 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 the following:

- hb2UnderRangeLowThresh underrange condition
- hb2UnderRangeMidThresh underrange condition
- hb2UnderRangeHighThresh underrange condition
- apdLowThresh underrange condition

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

The apdLowThresh has priority in terms of preventing recovery. If apdLowThresh reports an overrange condition (sufficient signal peaks have exceeded its threshold in a gain update counter period), 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, a condition where 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 nonzero recovery step. Figure 85 provides a flow diagram of the decisions of the AGC when recovering the gain in peak detect mode.

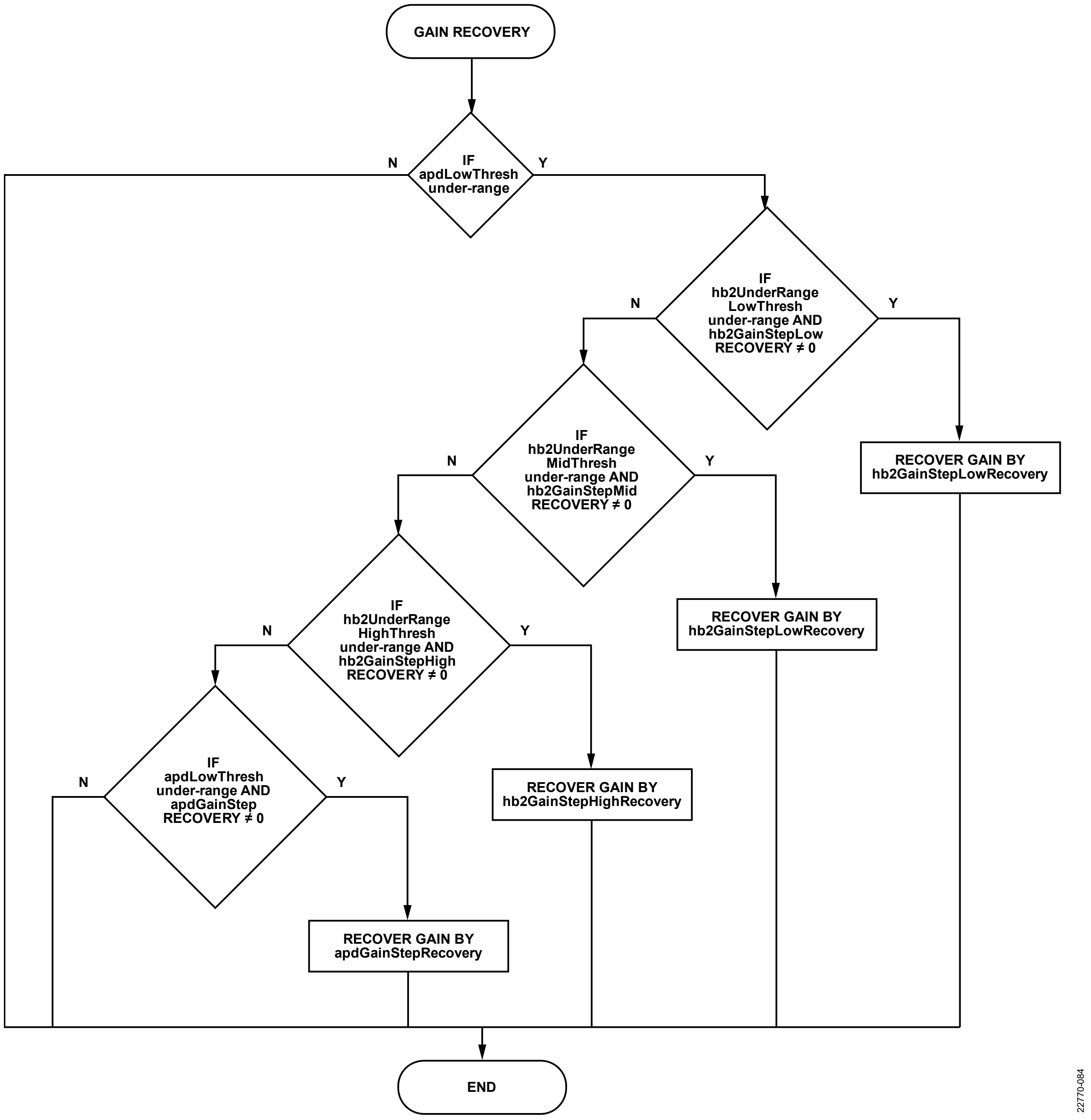

GAIN RECOVERY
IF apdLowThresh under-range
N
Y
IF hb2UnderRange LowThresh under-range AND hb2GainStepLow RECOVERY ≠ 0
RECOVER GAIN BY hb2GainStepLowRecovery
IF hb2UnderRange MidThresh under-range AND hb2GainStepMid RECOVERY ≠ 0
RECOVER GAIN BY hb2GainStepLowRecovery
IF hb2UnderRange HighThresh under-range AND hb2GainStepHigh RECOVERY ≠ 0
RECOVER GAIN BY hb2GainStepHighRecovery
IF apdLowThresh under-range AND apdGainStep RECOVERY ≠ 0
RECOVER GAIN BY apdGainStepRecovery
END
22770-084

Figure 85. 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 receiver chain. In the event of an overrange condition, both the peak detectors and the power detector can instantiate a gain decrement. In the event of an underrange, only the power detector can increment the gain. The power detector changes gain solely at the expiry of the gain update counter. The peak detectors can be set in one of two modes (depending on the setting of agcGainChangeIfThreshHigh) where 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 79 to Figure 84).

The power measurement block provides the rms power of the receiver data at the measurement location. The power measurement block can be configured to monitor the signal in one of three locations, as shown in Figure 76. In power detect mode, the AGC compares the measured signal level to programmable thresholds, which provide a second order control loop, where 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 86 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.

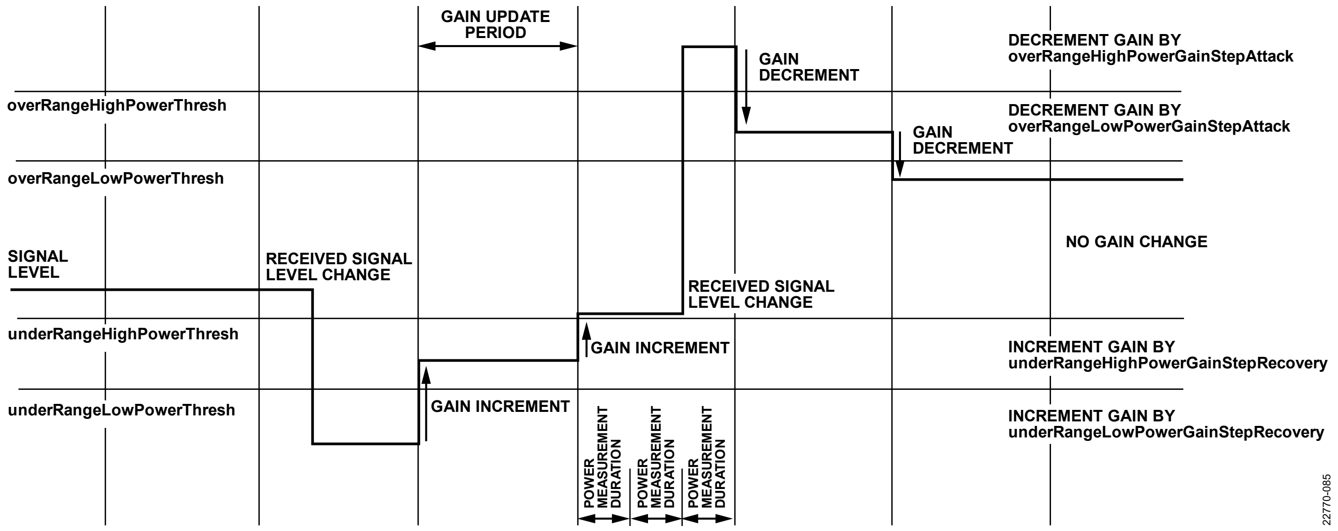

GAIN UPDATE PERIOD
GAIN DECREMENT
DECREMENT GAIN BY overRangeHighPowerGainStepAttack
overRangeHighPowerThresh
DECREMENT GAIN BY overRangeLowPowerGainStepAttack
GAIN DECREMENT
overRangeLowPowerThresh
NO GAIN CHANGE
SIGNAL LEVEL
RECEIVED SIGNAL LEVEL CHANGE
RECEIVED SIGNAL LEVEL CHANGE
underRangeHighPowerThresh
GAIN INCREMENT
INCREMENT GAIN BY underRangeHighPowerGainStepRecovery
underRangeLowPowerThresh
GAIN INCREMENT
INCREMENT GAIN BY underRangeLowPowerGainStepRecovery
POWER MEASUREMENT DURATION
POWER MEASUREMENT DURATION
POWER MEASUREMENT DURATION
22770-085

Figure 86. PMD Thresholds and Gain Changes for Underrange and Overrange Conditions

It is possible for the AGC to get contrasting requests from the power and peak detectors. An example is 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. It is important to note the APD and HB2 lower level overloads. In peak detect mode, the lower level thresholds for these detectors are used to indicate an underrange condition, which caused the AGC to increase the gain. In power detect mode, these detectors are not used for gain recovery, but can be used to control gain recovery by setting the agcLowThreshPreventGain API parameter. 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. The peak detector 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, which 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 receiver chain under such a condition.

AGC CLOCK AND GAIN BLOCK TIMING

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

The AGC state machine contains 3 states, the 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.

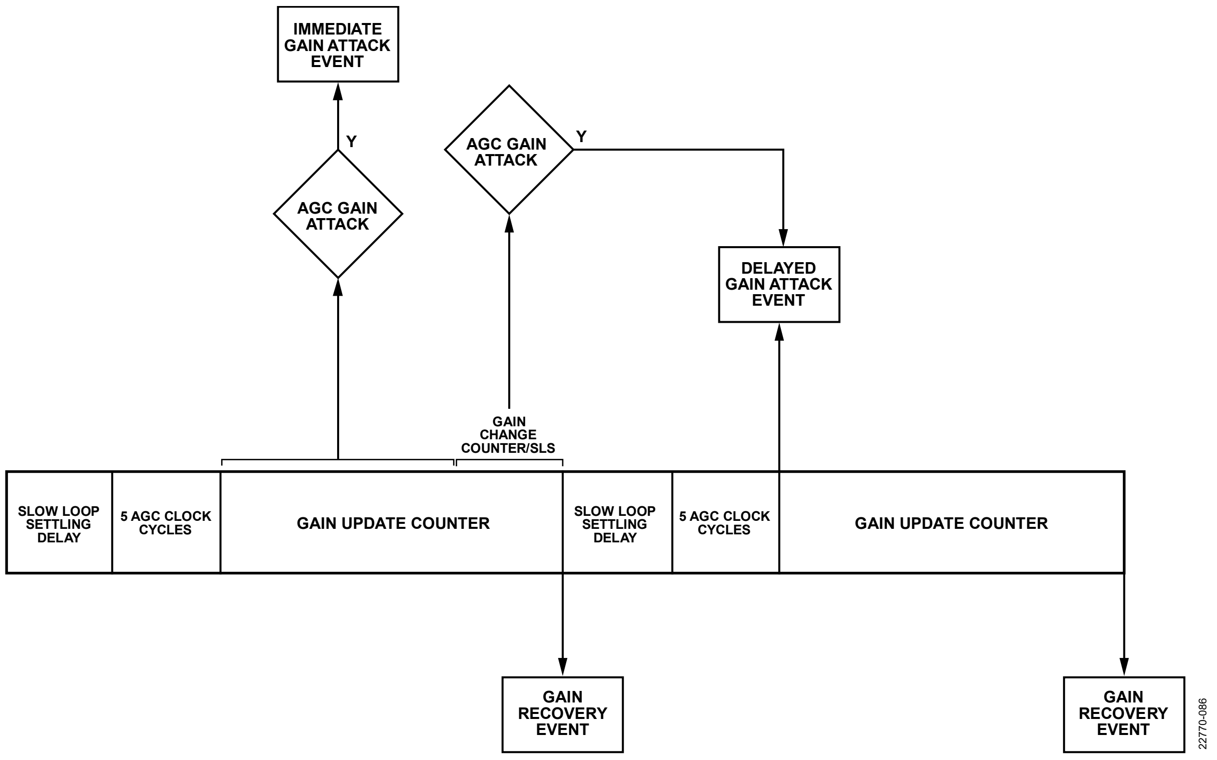


Figure 87. Gain Update Period

Figure 87 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 SLS is typically several orders of magnitude smaller than the gain update counter, this is the most common gain decrement scenario.
- AGC gain attack within the gain update counter, but within an 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 underrange interval, per Figure 83.

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 the 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 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 SLS delay state. Therefore, if a gain attack occurred very close to the end of the gain update counter state, the gain change counter delays the start of the SLS state and shifts the gain recovery event. To prevent this happening and maintain a periodic gain recovery event, gain attacks are prevented from happening towards the end of the gain update counter state, as shown in Figure 87. 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, 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 cause 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 are reset back to zero. When the receiver 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 agcEnableSyncPulseForGainCounter parameter. While this bit is set, the AGC monitors a GPIOx 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 to GPIO_15 can be used for this purpose.

For example, considering a 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.

$$\textit{Gain Update Period (AGC Clocks)} = 491.52\text{ MHz} \times 100\ \mu\text{s} = 49{,}152$$

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.

$$\textit{Gain Update Period (AGC Clocks)} = \textit{agcGainUpdateCounter} \times 2(\textit{agcSlowLoopSettingDelay}) + 5$$

$$\textit{Gain Update Period (AGC Clocks)} = 49{,}139 + 2(4) + 5 = 49{,}152$$

When receiver 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 receiver 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 76). The APD 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, the detector flags that the threshold has been overloaded.

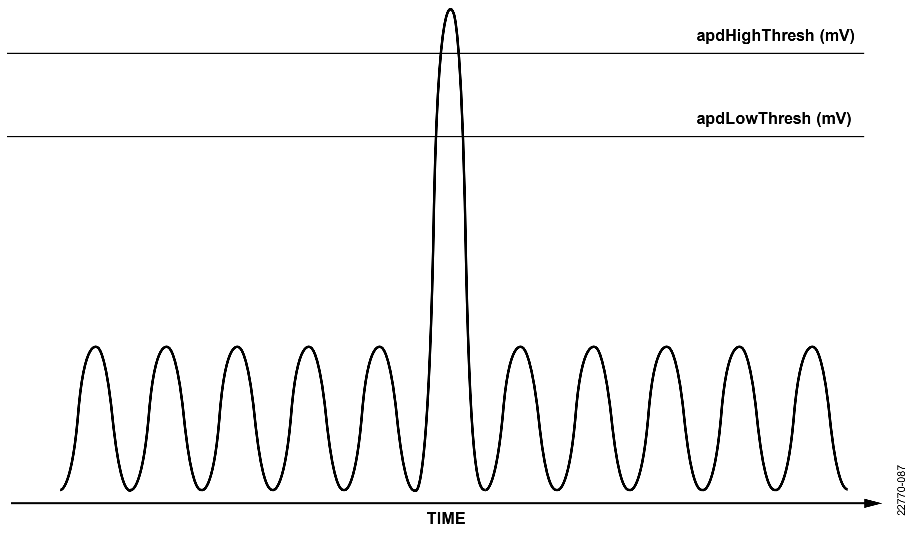


Figure 88. Analog Peak Detector Thresholds

There are two APD thresholds, as shown in Figure 88. 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:

$$apdHighThresh = \text{round}\left(\frac{850 \times 10^{\frac{ADCdBFS}{20}} - 16}{16}\right)$$

$$apdLowThresh = \text{round}\left(\frac{850 \times 10^{\frac{ADCdBFS}{20}} - 16}{16}\right)$$

Note that the APD is an analog circuit located after the TIA filter. The previous equations 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 overrange condition occurs. Both the upper and lower thresholds have a programmable counter in the agcPeak API structure, as indicated in Table 154.

Table 154. APD Programmable Threshold Counters

Threshold	Counter
Upper threshold (apdHighThresh)	apdUpperThreshPeakExceededCnt
Lower threshold (apdLowThresh)	apdLowerThresPeakExceededCnt

As described in the Automatic Gain Control section, 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.

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

Table 155. 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 used for each gain adjustment. The gain table is programmed with the largest gain in the maximum gain index (typically Index 255), with ever decreasing gain for decreasing gain index. Therefore, if the APD gain attack step size is programmed to 6, 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 overrange condition, 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 overrange 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 overranges. 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 2 filter. The HB2 peak detector can, therefore, also be referred to as the decimated data overload detector because it works on decimated data. Like the APD detector, the HB2 peak detector functions by comparing the signal level to programmable thresholds. The HB2 peak detector 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, 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.

After the required number of samples exceeds the threshold in the duration required, 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 89 shows the two level approach. Figure 89 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 are 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 is 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 is 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, an overrange condition is called. Likewise, if the number of overloads of the hb2UnderRangeHighThresh does not exceed hb2LowerThreshPeakExceededCnt, an underrange condition is called.

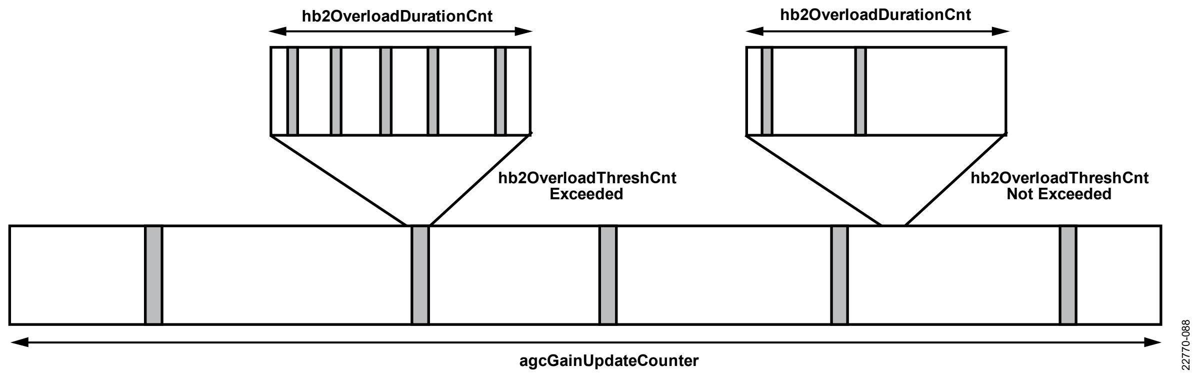


Figure 89. HB2 Detector Two-Level Approach for an Overload Condition

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 156.

Table 156. 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 Peak Detect Mode section, Figure 80, Figure 82, and Figure 84.

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 so that an overrange condition is not flagged until the threshold has been exceeded the amount of times determined by the corresponding equation in a gain update period. Note that these equations only apply if hb2OverladPowerMode = 0. If this parameter is set to 1, the denominator in the exponent of each equation changes from 20 to 10.

Table 157. Gain Steps for HB2 Overrange 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 158. 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. The power measurement block 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 159.

Table 159. 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, and any partial measurements are discarded.

The power measurement block has a dynamic range of 60 dB by default. Power measured in the receiver datapath can be read back with the adi_adrv9025_RxDecPowerGet command.

adi_adrv9025_RxDecPowerGet(…)

```
adi_adrv9025_RxDecPowerGet(adi_adrv9025_Device_t* device, adi_adrv9025_RxChannels_e rxChannel,
uint16_t *rxDecPower_mdBFS)
```

Description

This command readback for receiver power measurement.

Parameters

Table 160. adi_adrv9025_RxDecPowerGet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
rxChannel	An enumerator indicating which receiver channel to configure, as shown in Table 146.
*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 90. 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. When device initialization has proceeded up to the configuration of the JESD204B and JESD204C, 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

This command configures the gain control blocks within the device, such as peak detectors, power detector, and AGC settings.

Parameters

Table 161. adi_adrv9025_AgcCfgSet(…) Parameters

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 AGC Holdover Function section. After agcConfig[] has been configured, the desired gain control mode can be enabled by using the adi_adrv9025_RxGainCtrlModeSet( ) API function.

The final step is to configure any GPIOs as necessary, whether 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.

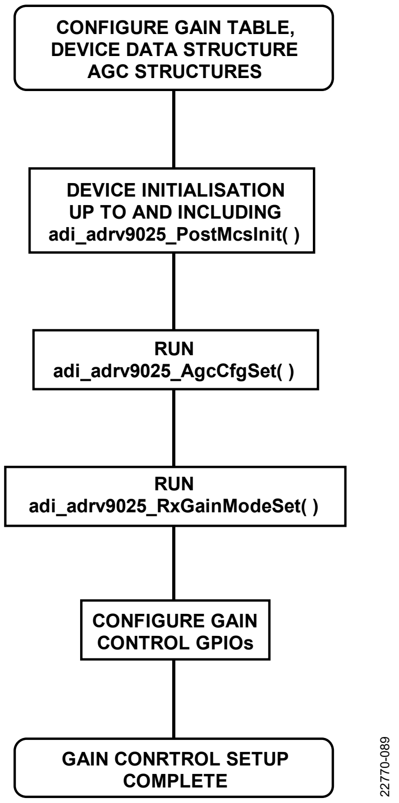
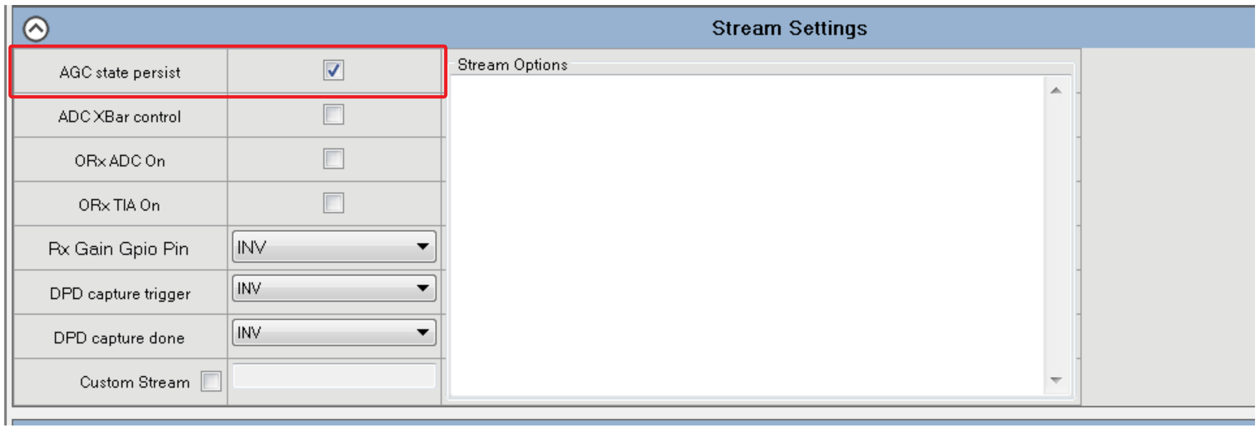

CONFIGURE GAIN TABLE, DEVICE DATA STRUCTURE AGC STRUCTURES
DEVICE INITIALISATION UP TO AND INCLUDING adi_adrv9025_PostMcsInit( )
RUN adi_adrv9025_AgcCfgSet( )
RUN adi_adrv9025_RxGainModeSet( )
CONFIGURE GAIN CONTROL GPIOs
GAIN CONRTROL SETUP COMPLETE
22770-089

Figure 90. Gain Control Programming Flowchart

AGC HOLDOVER FUNCTION

The transceiver AGC uses counters to keep track of any overrange or underrange 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 receiver enable goes low. This reset of the overrange and underrange counters can potentially cause the AGC state machine to never trigger if the gainUpdateCounter is larger than the receiver TDD slot duration. The AGC holdover function has been implemented to avoid this situation by preventing the counters from getting reset when the receiver enable is toggled.

To enable this function, the user must create a stream file using the transceiver evaluation software with the AGC state persist box checked in the stream settings window, as shown in Figure 91. After this box is checked, a stream file can be created with the AGC holdover function enabled to prevent AGC counter resets during TDD operation.



Stream Settings

AGC state persist	✓
ADC XBar control	☐
ORx ADC On	☐
ORx TIA On	☐
Rx Gain Gpio Pin	INV
DPD capture trigger	INV
DPD capture done	INV
Custom Stream ☐	

Stream Options

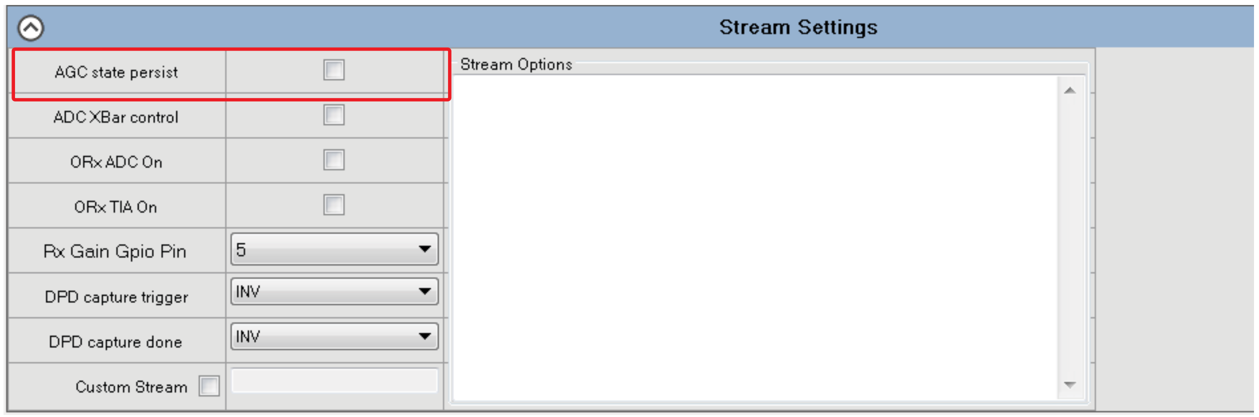
Figure 91. TES Stream Settings Control Window to Enable AGC Holdover

RECEIVER GAIN MODE SWITCHING USING GPIO

This feature allows the use of a GPIOx pin to force receiver 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 receiver signal chain. Such a calibration requires a fixed receiver gain index, which is not possible to guarantee if the device is in AGC mode. The user can change the mode to MGC and then change the receiver 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 GPIOx pin to change the receiver gain index to a fixed predetermined value and move the receiver 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 receiver state. When the GPIO is low, the gain control mode is restored back to AGC to resume normal receiver operation.

To enable receiver GPIO gain mode switching, the user must create a stream file using the TES with the **Rx Gain Gpio Pin** set to the desired GPIO pin, as shown in Figure 92.



Stream Settings

AGC state persist	☐
ADC XBar control	☐
ORx ADC On	☐
ORx TIA On	☐
Rx Gain Gpio Pin	5
DPD capture trigger	INV
DPD capture done	INV
Custom Stream ☐	

Stream Options

Figure 92. TES Stream Settings Control Window to Enable Receiver Gain Mode Switching using GPIO

The user also must use the StreamGpioConfigSet API function to unmask the stream GPIO source 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 receiver gain GPIO tag, as shown in Figure 92.
2. Use StreamGpioConfigSet function (called during postMcsInit) with the correct GPIO pin selected, as shown in the StreamGpioConfigSet Function section.
3. Set the receiver manual gain to the desired value to be used during the calibration.

By following these steps, the user can move the receiver to MGC mode when the GPIO goes high and move the receiver back to AGC mode when the GPIO goes low. Note that this function affects all four channels of the receivers 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 93 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 162, Table 163, and Table 164.

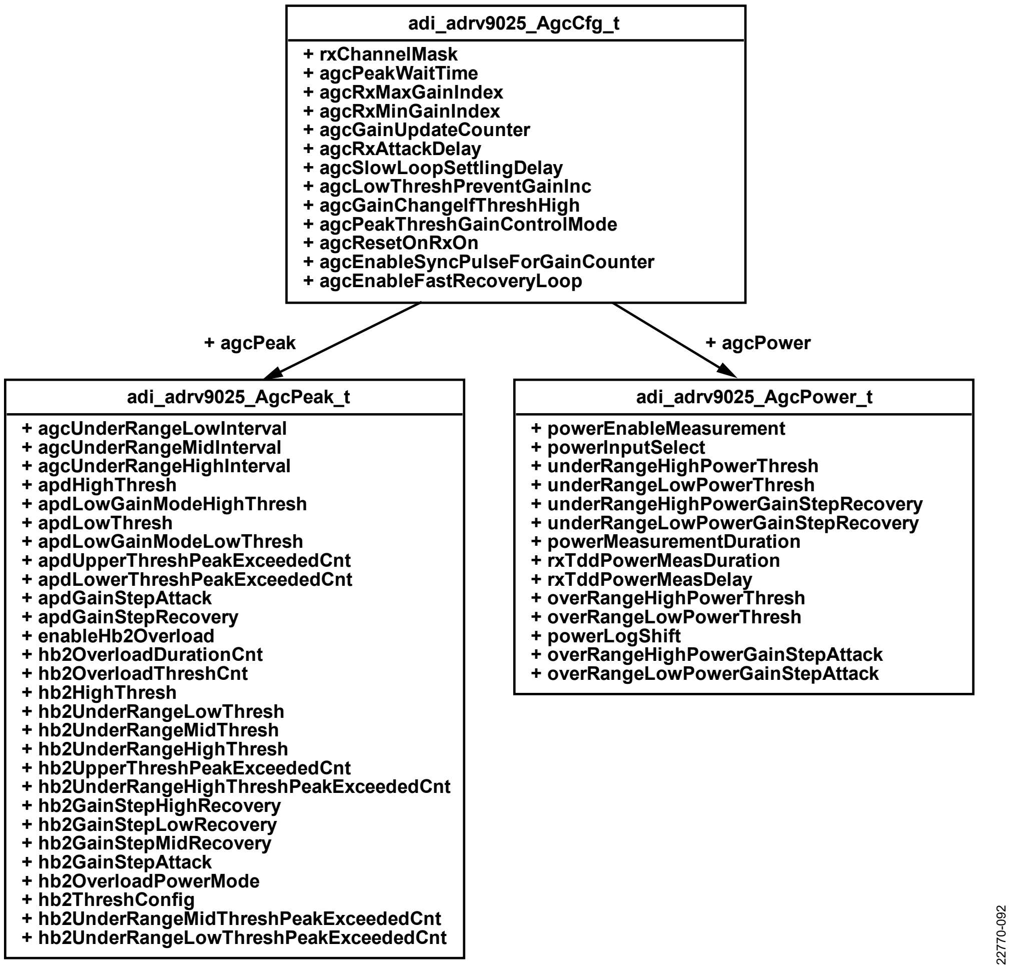

adi_adrv9025_AgcCfg_t
+ rxChannelMask
+ agcPeakWaitTime
+ agcRxMaxGainIndex
+ agcRxMinGainIndex
+ agcGainUpdateCounter
+ agcRxAttackDelay
+ agcSlowLoopSettlingDelay
+ agcLowThreshPreventGainInc
+ agcGainChangeIfThreshHigh
+ agcPeakThreshGainControlMode
+ agcResetOnRxOn
+ agcEnableSyncPulseForGainCounter
+ agcEnableFastRecoveryLoop
+ agcPeak
+ agcPower
adi_adrv9025_AgcPeak_t
+ agcUnderRangeLowInterval
+ agcUnderRangeMidInterval
+ agcUnderRangeHighInterval
+ apdHighThresh
+ apdLowGainModeHighThresh
+ apdLowThresh
+ apdLowGainModeLowThresh
+ apdUpperThreshPeakExceededCnt
+ apdLowerThreshPeakExceededCnt
+ apdGainStepAttack
+ apdGainStepRecovery
+ enableHb2Overload
+ hb2OverloadDurationCnt
+ hb2OverloadThreshCnt
+ hb2HighThresh
+ hb2UnderRangeLowThresh
+ hb2UnderRangeMidThresh
+ hb2UnderRangeHighThresh
+ hb2UpperThreshPeakExceededCnt
+ hb2UnderRangeHighThreshPeakExceededCnt
+ hb2GainStepHighRecovery
+ hb2GainStepLowRecovery
+ hb2GainStepMidRecovery
+ hb2GainStepAttack
+ hb2OverloadPowerMode
+ hb2ThreshConfig
+ hb2UnderRangeMidThreshPeakExceededCnt
+ hb2UnderRangeLowThreshPeakExceededCnt
adi_adrv9025_AgcPower_t
+ powerEnableMeasurement
+ powerInputSelect
+ underRangeHighPowerThresh
+ underRangeLowPowerThresh
+ underRangeHighPowerGainStepRecovery
+ underRangeLowPowerGainStepRecovery
+ powerMeasurementDuration
+ rxTddPowerMeasDuration
+ rxTddPowerMeasDelay
+ overRangeHighPowerThresh
+ overRangeLowPowerThresh
+ powerLogShift
+ overRangeHighPowerGainStepAttack
+ overRangeLowPowerGainStepAttack
22770-092

Figure 93. Member Listing of adi_adrv9025_AgcCfg_t Data Structure

Table 162. 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 receiver 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	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 must be held in reset when the receiver path is enabled.	0	63

Parameter	Description	Min Value	Max Value
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 and power detect AGC operation. 1: If AGC is in peak power detect mode, 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 mode and power detect mode. 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 wait 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 and power AGC mode where both peak detectors and power detectors are utilized.	0	1
agcResetOnRxon	1: AGC state machine is reset when the receiver is disabled. The AGC gain setting is returned to the maximum gain. 0: AGC state machine maintains its state when the receiver is disabled.	0	1
agcEnableSyncPulseForGainCounter	1: Allows synchronization of the 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.	Not applicable	Not applicable
agcPeak	Structure containing all the peak detector settings.	Not applicable	Not applicable

Table 163. 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

Parameter	Description	Min Value	Max Value
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. The value is calculated using the equation: *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. The value is calculated using the equation: *adpdLowThresh* (mV) = (*apdLowThresh* + 1) × 16 mV.	7	apdHighThresh
apdLowGainModeLowThresh	This parameter is not utilized.		
apdUpperThreshPeakExceededCnt	Sets number of peaks to detect greater than apdHighThresh to cause an APD high overrange event. In AGC modes, this results in a gain decrement set by apdGainStepAttack.	0	255
apdLowerThreshPeakExceededCnt	Sets number of peaks to detect greater than apdLowThresh to cause an APD low overload event. In peak detect AGC mode, if an APD low overload event is not occurring, 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 overrange in AGC modes. The step size in dB depends on the gain step resolution of the gain table (default 0.5 dB per index step).	0	31
apdGainStepRecovery	The number of indices that the gain index pointer must be increased in the event of an APD underrange 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. An 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 $2^{(\text{hb2OverloadDurationCnt} + 1)}$.	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
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 underrange 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 underrange 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

Parameter	Description	Min Value	Max Value
hb2UnderRangeHighThresh;	Peak detect mode: threshold used for gain recovery. Peak detect with fast recovery mode: this sets the highest threshold of the HB2 underrange 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 greater than hb2HighThresh (number of times hb2OverloadThreshCnt was exceeded in hb2OverloadDurationCnt) to cause an HB2 high overrrange event. In AGC modes, this results in a gain decrement set by hb2GainStepAttack.	0	255
hb2UnderRangeHighThreshExceededCnt	Sets number of individual overloads greater than hb2UnderRangeHighThresh (number of times hb2OverloadThreshCnt was exceeded in hb2OverloadDurationCnt) to cause an HB2 underrange 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 underrange high threshold underrange 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 underrange low threshold underrange 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 underrange mid threshold underrange event.	0	31
hb2GainStepAttack	The number of indices that the gain index pointer must be decreased in the event of an HB2 high threshold overrange 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. If it is set to 0, the hb2 threshold sample type is signal amplitude. If it is set to 1, the hb2 threshold sample type is signal power.	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 underrange mid 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 hb2GainStepMidRecovery.	0	255
hb2UnderRangeLowThreshExceededCnt	Only applicable in fast recovery mode of peak detect AGC. Sets the number of individual overloads greater than hb2UnderRangeLowThresh (number of times hb2OverloadThreshCnt was exceeded in hb2OverloadDurationCnt) to cause an HB2 underrange low 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 hb2GainStepLowRecovery.	0	255

Table 164. 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

Parameter	Description	Min Value	Max Value
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 the AGC gain update counter.	0	31
rxTddPowerMeasDuration	Following a receiver 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 a receiver 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 receiver 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 transceiver. 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 therefore 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 receiver input port. Any internal front-end attenuation changes made to prevent ADC overloading are transparent to the baseband processor. In this way, the AGC 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 mode or MGC mode. 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 described in the following mode option descriptions. Figure 94 is a block diagram of the gain compensation portion of the receiver chain, showing the locations of the various blocks.

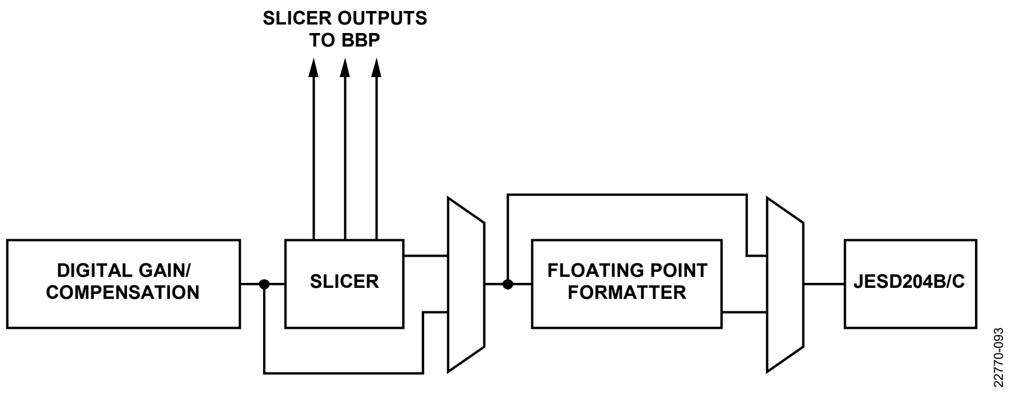


Figure 94. Gain Compensation, Floating Point Formatter, and Slicer Section of the Receiver Datapath

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 or 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, the digital gain block can be utilized to even out the steps for consistency (the default table utilizes the digital gain block to provide consistent 0.5 dB steps).

Neither the slicer nor floating point formatter block is utilized. As no gain compensation is applied, there is no bit width expansion of the digital signal. The signal is provided to the JESD204B and JESD204C 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 transceiver with a gain table that compensates for the analog front-end attenuation applied. As the analog front-end attenuation is increased, an equal amount of digital gain is applied. Considering 16-bit data at the input to the digital compensation block, 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 95 outlines this effect, with yellow boxes indicating the valid (used) bits in each case.

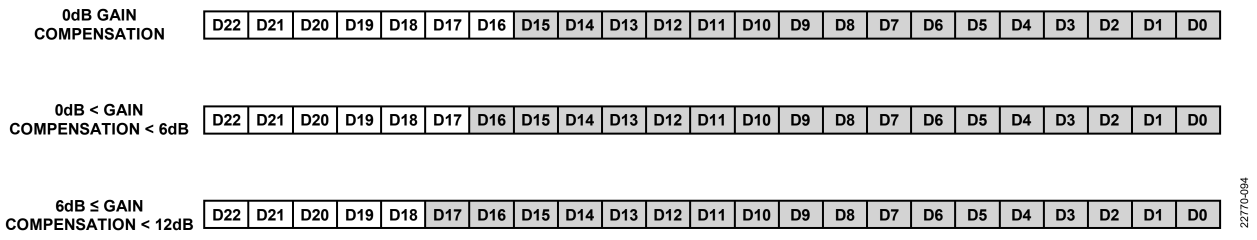

0dB GAIN COMPENSATION
0dB < GAIN COMPENSATION < 6dB
6dB ≤ GAIN COMPENSATION < 12dB
D22 D21 D20 D19 D18 D17 D16 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
22770-094

Figure 95. Bit Width of Received Signal for Increasing Gain Compensation

The slicer is used to attenuate the data after the digital gain block in a way that it can fit into the resolution of the JESD204B and JESD204C datapath. The slicer 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 in 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 JESD204B and JESD204C 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 165. These GPIOs require their pins to be enabled as outputs and configured for slicer output mode (see the GPIO Configuration section).

Table 165. GPIOs Used for Slicer Output Mode

Receiver	GPIOs Utilized (MSB to LSB)
Rx1	GPIO11, GPIO10, GPIO9, GPIO8
Rx2	GPIO_15, GPIO14, GPIO13, GPIO12
Rx3	GPIO7, GPIO6, GPIO5, GPIO4
Rx4	GPIO3, GPIO2, GPIO1, GPIO_0

The following example illustrated in Figure 96 explains the operation of the slicer in detail. In this use case, the JESD204B and JESD204C is configured for 16-bit data resolution. The slicer is configured to 6 dB resolution.

Figure 96 explains the operation. Initially, the analog attenuator is applying no attenuation (0 dB) and, therefore, there is 0 dB digital gain applied to the signal. The slicer is in its default (0000) position. As the attenuation increases (0 dB to 6 dB), a corresponding amount of digital gain is applied to the signal. With any digital gain applied to the signal, the bit width of the signal has increased (the ADC can output 16-bits, and further gain allows a maximum input to go beyond 16-bits). In this case, the signal now has 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 again. 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).

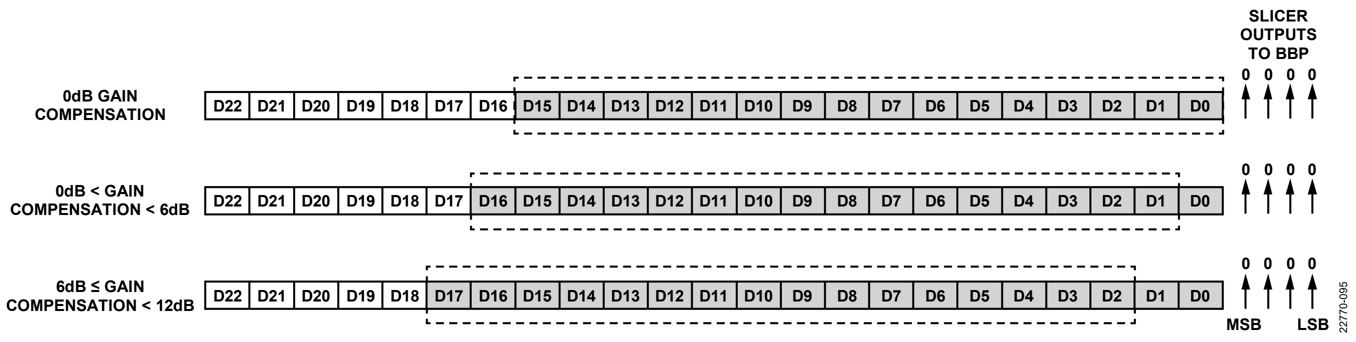

SLICER OUTPUTS TO BBP
0dB GAIN COMPENSATION
0dB < GAIN COMPENSATION < 6dB
6dB ≤ GAIN COMPENSATION < 12dB
D22 D21 D20 D19 D18 D17 D16 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
0 0 0 0
0 0 0 0
0 0 0 0
MSB
LSB
22770-095

Figure 96. 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 166. Equivalent tables can be inferred for the other attenuation options.

Table 166. 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 or 4 slicer bits. If 3 are used, there is another option of inserting a 0 to fill the 4th bit, or to insert a parity bit (by adjusting the intParity API parameter). Table 167 shows the various modes selectable by intEmbeddedBits.

Table 167. 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 97.
ADI_ADRV9025_EMBED_1_SLICERBIT_AT_LSB	Embeds 1 slicer bit on both I and Q at the LSB position. See Figure 98.
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 99. Because this is a 3-bit mode, an extra bit is inserted denoted as P in Figure 99. 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 100. Given this is a 3-bit mode, an extra bit is inserted denoted as P in Figure 100. 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 101.
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 102.

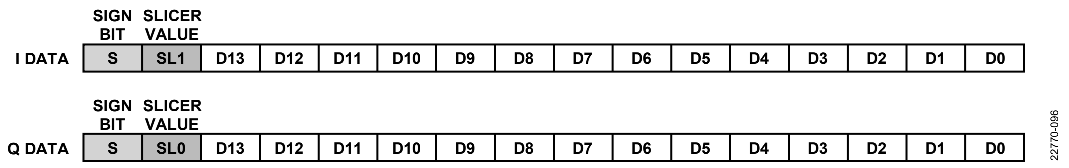


Figure 97. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_1_SLICERBIT_AT_MSB)

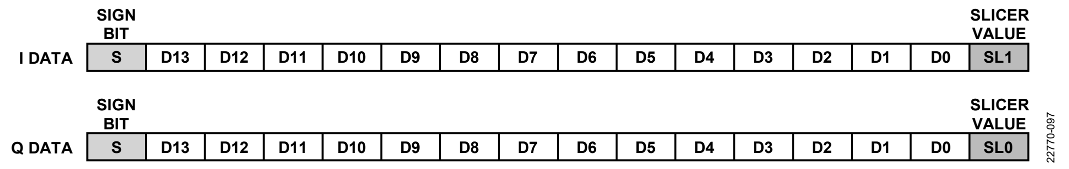


Figure 98. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_1_SLICERBIT_AT_LSB)

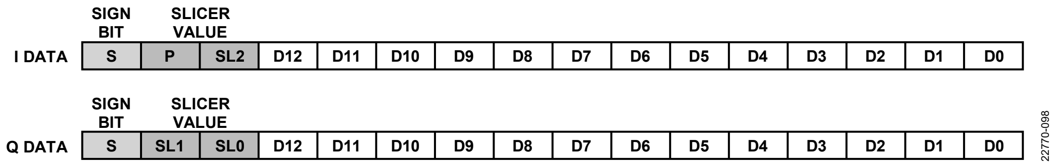


Figure 99. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_2_SLICERBITS_AT_MSB_3_BIT_SLICER)

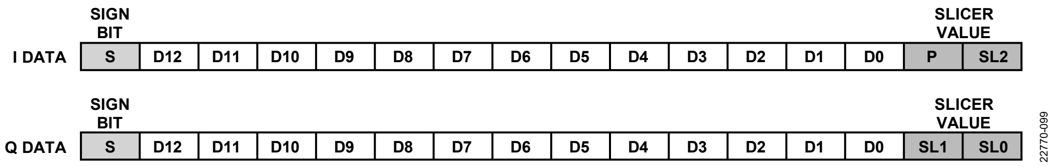


Figure 100. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_2_SLICERBITS_AT_LSB_3_BIT_SLICER)

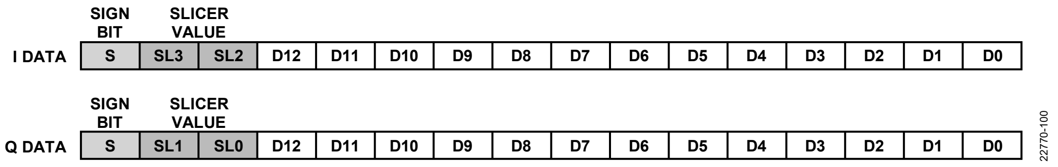


Figure 101. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_2_SLICERBITS_AT_MSB_4_BIT_SLICER)

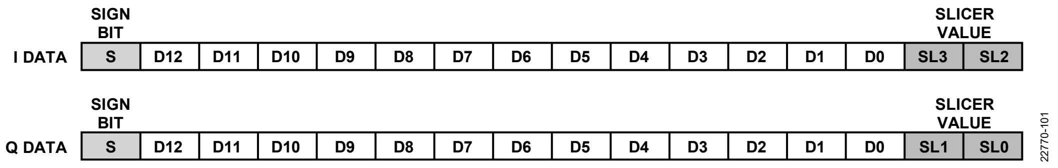


Figure 102. Encoding of Slicer Information as Control Bits (intEmbeddedBits = ADI_ADRV9025_EMBED_2_SLICERBITS_AT_LSB_4_BIT_SLICER)

Mode 4: Digital Gain Compensation and Slicer Input

In this mode, the slicer position is controlled by the user. In Mode 2 and Mode 3, the slicer can be viewed as an attenuator, which reduces the signal level a certain dB with each slicer position step in a way that it can be sent across the JESD204B and JESD204C link. This mode is similar, except the position (amount of attenuation) is controlled externally. The valid step sizes are between 1 dB and 6 dB and controlled by the extPinStepSize API parameter, as outlined in Table 168.

Table 168. 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 169. Each channel can be set to any one of the options using the rx1ExtSlicerGpioSelect, rx2ExtSlicerGpioSelect, rx3ExtSlicerGpioSelect, and rx4ExtSlicerGpioSelect API parameters. The value of these pins and the step size chosen set the level of slicer attenuation applied to the data before transmission across the JESD204BC link.

$$Slicer\ Attenuation = Slicer\ Input\ Pin\ Values \times extPinStepSize$$

For example, if the value on the slicer input pins is 0'b111, and the step size is 2 dB, the slicer applies 14 dB (7 × 2 dB) of attenuation to the data.

Table 169. adi_adrv9025_RxExtSlicerGpioSel_e Description

Value of RxExtSlicerGpioSelect	GPIOs Utilized (MSB to LSB)
ADI_ADRV9025_EXTSLICER_RX_GPIO_2_DOWNTO_0	GPIO2, GPIO1, GPIO_0
ADI_ADRV9025_EXTSLICER_RX_GPIO_5_DOWNTO_3	GPIO5, GPIO4, GPIO3

Value of RxExtSlicerGpioSelect	GPIOs Utilized (MSB to LSB)
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, GPIO_15

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 in a way so 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 a higher range, and therefore 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 103):

- Sign, exponent, significand
- Sign, significand, exponent

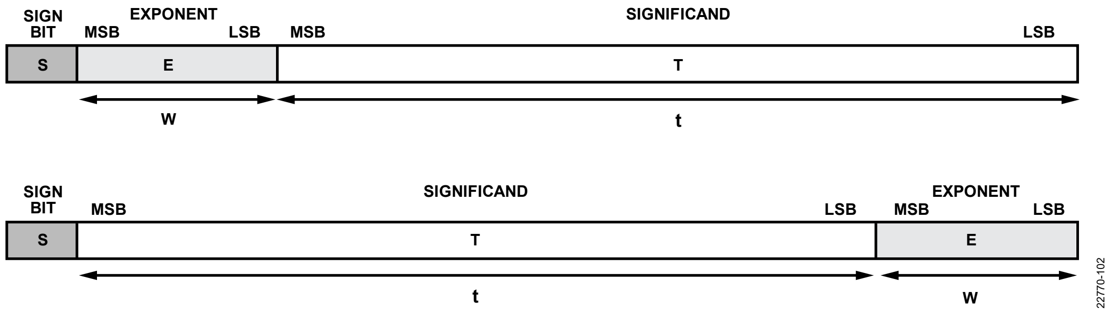


Figure 103. Floating Point Number Representation

In Figure 103, 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:

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

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.
p is the precision of the mode (p = t + 1, because the t significand bits are coupled with a sign bit).

Table 170 provides the values to use in these equations for the various IEEE754 supported modes.

Table 170. Floating-Point Formatter—Supported IEEE754 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 104 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. Because of 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 than there is at higher signal levels, preserving SNR when the received signal strength is low.

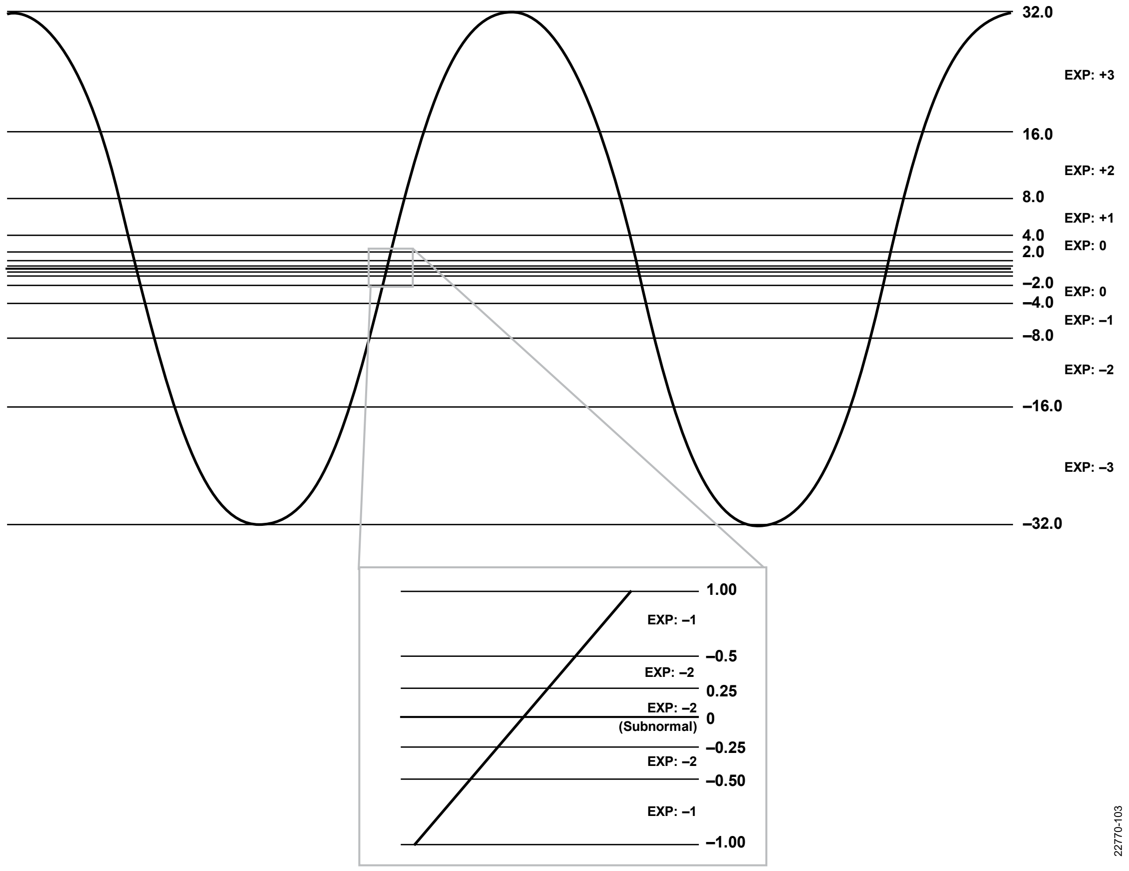


Figure 104. Visualization of the Floating-Point Formatter Values

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 because 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 171. Exponent Bit Widths of IEEE754 and Analog Devices Modes

Exponent Bit Width	IEEE754 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. However, this reduces the precision of the values.

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, it is also possible to enable a fixed digital attenuation (from 6 dB to 42 dB) prior to the floating-point formatter to ensure that the signal never exceeds the maximum range encodable over the JESD204B and JESD204C link.

RECEIVER DATA FORMAT DATA STRUCTURE

The configuration parameters for the floating-point formatter and slicer are set up in a data structure of adi_adrv9025_RxDataFormat_t type.

Table 172. 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 receivers 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 JESD204B and JESD204C 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 173). 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 JESD204B and JESD204C link (see Table 174). Data type: adi_adrv9025_IntegerConfigSettings_t.	
slicerConfigSettings	A configuration structure for the slicer functionality (see Table 175). Data type: adi_adrv9025_SlicerConfigSettings_t.	
externalLnaGain	For use in dual band modes. Not currently supported.	
tempCompensationEnable	Not currently supported.	

Table 173. 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 indicates 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 not a number (NaN) 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 174. 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 receiver 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 bit on I and 1 slicer bit on Q and the MSB position.
	ADI_ADRV9025_EMBED_1_SLICERBIT_AT_LSB	Embeds 1 slicer bit on I and 1 slicer bit 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 bits, 16 bits, or 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 175. 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

This command configures the receiver data format.

Parameters

Table 176. adi_adrv9025_RxDataFormatSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
rxDataFormat[]	An array of receiver data format structures.
arraySize	The number of receiver data format structures in rxDataFormatarray length of txPaProtectCfg[].

DIGITAL FILTER CONFIGURATION

OVERVIEW

This section describes the digital filters within the transceiver and provides a description of each filter in terms of their filter coefficients and position within the signal chain. The API structures are described and an example profile specific configuration is provided for each signal chain. The API functions that are used to configure the filters are also described in this section.

RECEIVER SIGNAL PATH

Each receive input has an independent signal path including separate I/Q mixers that feed into programmable analog transimpedance amplifiers (TIAs) that serve as LPFs in the analog data path. The signals are then converted by the ΣΔ ADCs and filtered in half-band decimation stages and the PFIR. The fixed coefficient half-band filters (FIR1, FIR2, RHB1(HR), RHB1(LP), RHB2, RHB3, and DEC5) and the PFIR is designed to prevent data wrapping and overrange conditions.

Each receive channel can convert signals down to zero IF real data using either 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 105 shows the in-phase (I) and quadrature (Q) signal paths for the Rx1, Rx2, Rx3, and Rx4 signal chain.

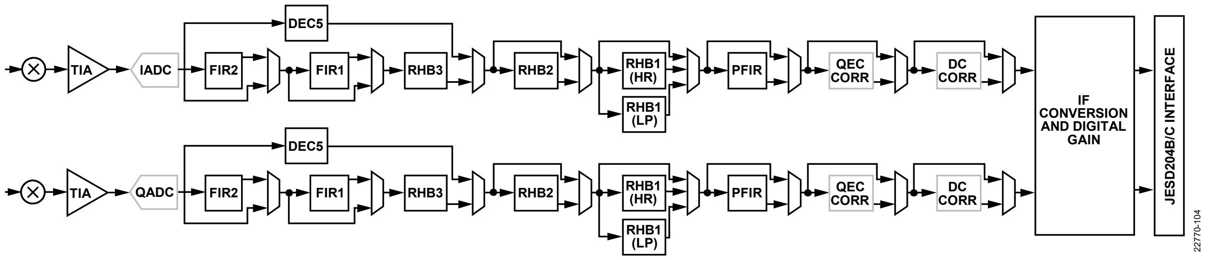


Figure 105. Receive Signal Path

TIA

The receive transimpedance amplifier is an LPF with a single real pole frequency response. The transceiver supports bandwidths up to 200 MHz and 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 3 dB bandwidth is set within the device data structure and is profile dependent. Roll-off within the receive pass band is compensated by the PFIR to ensure a maximally flat pass band frequency response.

Decimation Stages

The signal path can be configured such that either the decimate by 5 filter (DEC5) or the combination of FIR2, FIR1, and RHB3 is used in the receive 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 include the following: +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, and +0.000976563

Finite Impulse Response 2 Filter (FIR2)

The FIR2 filter is a fixed coefficient decimating filter. The FIR2 filter can decimates by a factor of 2 or the filter can be bypassed.

The FIR2 filter coefficients include the following: 0.0625, 0.25, 0.375, 0.25, and 0.0625.

Finite Impulse Response 1 Filter (FIR1)

The FIR1 filter is a fixed coefficient decimating filter. The FIR1 filter can decimate by a factor of 2 or the filter can be bypassed.

The FIR1 filter coefficients include the following: 0.0625, 0.25, 0.375, 0.25, and 0.0625.

Receive Half-Band 3 Filter (RHB3)

The RHB3 filter is a fixed coefficient decimating filter. The RHB3 filter decimates by a factor of 2.

The RHB3 filter coefficients include the following: −0.033203125, 0, +0.28125, +0.49609375, +0.28125, 0, and −0.033203125.

Receive Half-Band 2 Filter (RHB2)

The RHB2 filter is a fixed coefficient decimating filter. The RHB2 filter can decimate by a factor of 2 or the filter can be bypassed.

The RHB2 filter coefficients include the following: −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 Filter (RHB1 (HR))

The RHB1 (HR) filter is a fixed coefficient decimating filter. The RHB1 (HR) filter can decimate by a factor of 2 or the filter can be bypassed.

RHB1 (HR) filter coefficients include the following: +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 Filter (RHB1 (LP))

The RHB1 (LP) filter is a fixed coefficient decimating filter. The RHB1 (LP) filter can decimate by a factor of 2 or the filter can 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.

Receive PFIR Filter

The receive PFIR filter acts as a decimating filter. The PFIR can decimate by a factor of 1, 2, or 4, or the filter can be bypassed. The RFIR filter compensates for the roll-off of the TIA LPF. The PFIR filter can use 24, 48, or 72 filter taps. The PFIR filter 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 the divider 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 equation:

$$Rx\ PFIR\ Filter\ Taps_{max} = (DPCLK)/(Rx\_IQ\_DATARATE) \times 24$$

where:
Rx PFIR Filter Taps$_{max}$ is the maximum number of filter taps that can be used for the given clock rate.
DPCLK is the digital filter clock rate.
Rx_IQ_DATARATE is the output data rate of the filter.

IF Conversion

The IF conversion stage provides the ability to change how the received data is presented to the JESD port. Figure 106 shows a block diagram of the IF conversion stage. There are two parallel paths where data can be processed (Band A and Band B). There are two mixer stages in the circuitry of each band that allow upshifting or downshifting, interpolation and decimation stages, and a half-band filter with a pass band of 0.4 × the sample rate.

The half-band filter coefficients in this IF conversion stage include the following: -9.1553×10^{-5}, 0, $+2.4414 \times 10^{-4}$, 0, -5.7983×10^{-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, $+2.4414 \times 10^{-4}$, 0, -9.1553×10^{-5}.

The following use cases provide examples 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 the LO such that the entire signal of interest is on one side of the LO. The Band A NCO1 downshifts the signal such that the signal is centered at 0 Hz. There is a half-band filter and decimate by 2 stage that decreases the bandwidth and subsequently the IQ rate if used. This stage reduces the number of JESD lanes required, or the rate that at which the lanes must be run.

Figure 107 shows a conceptual case of a 200 MHz receive bandwidth (IQ rate 245.76 MSPS) profile 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. The channel then uses the IF conversion stage to shift the signal to be centered at about 0 Hz, filter with the half-band filter, and decimate 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 the downconverted signal exists solely on one side of the LO. The signal no longer needs to be in complex form; only I data is sent across the link and Q data is dropped. The interpolate by 2 stage can also be utilized for this scenario.

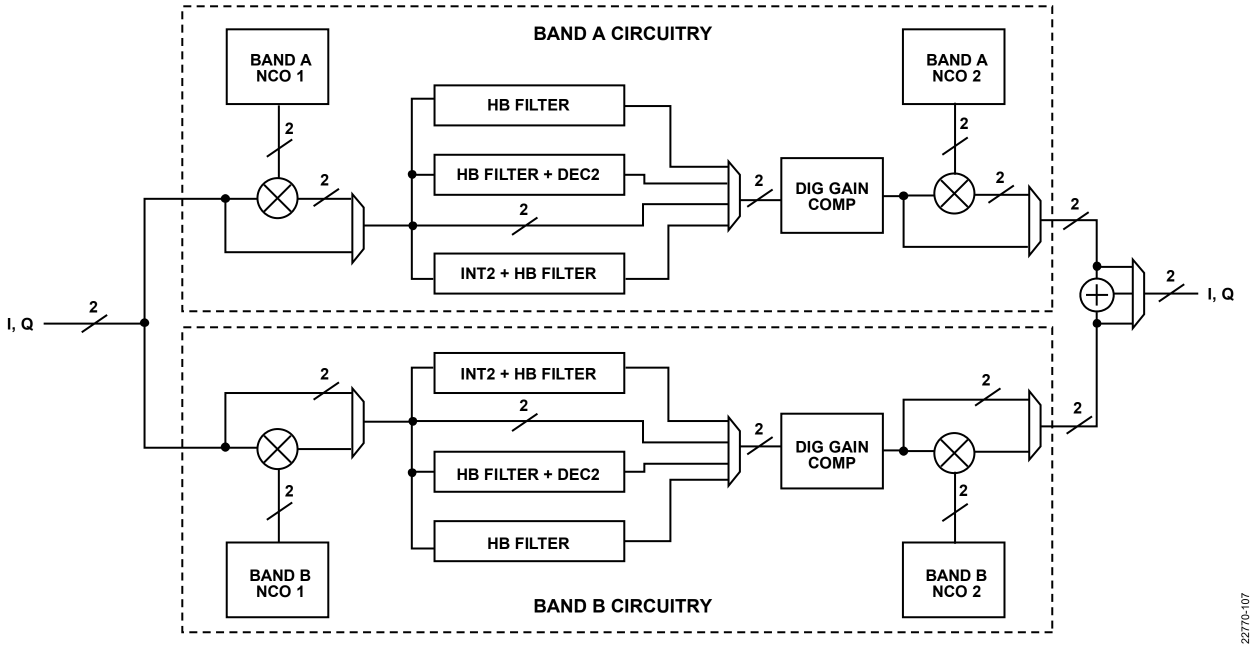


Figure 106. IF Conversion Stage Block Diagram (All Circuitry is Implemented in Quadrature, as Indicated)

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 upshifts or downshifts the data to generate a signal that is symmetrical to about 0 Hz. The result of this signal is that the spectrum no longer requires a complex representation, only I data is sent across the link, and the Q data is dropped.

DUAL BAND MODE

In this use case, multiple signals are received (Signal 1 and Signal 2). Band A circuitry can be used to process Signal 1, and Band B circuitry can be used to process Signal 2. Band A NCO1 shifts Signal 1 such that the signal is placed within the pass band of the half-band filter and filters out Signal 2. The decimate by 2 stage can also be used if the final composite bandwidth allows 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 of this procedure is that the two signals, originally located far apart in the spectrum and 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 that the signals exist on the same side of the LO. At this point, the spectrum no longer needs a complex representation, only I data can be sent across the link, and Q data is dropped. The interpolate by 2 stage can also be utilized for this scenario.

HB FILTER ONLY MODE

If there is a blocker to one side of the signal, the IF conversion stage can be used to obtain further rejection of the blocker. Band A NCO1 offsets the signal to position the signal close to the edge of the half-band filter pass band, and to position the blocker in the filter transition or stopband. The Band A NCO2 can be used to position the desired signal to its previous position within the spectrum, if required.

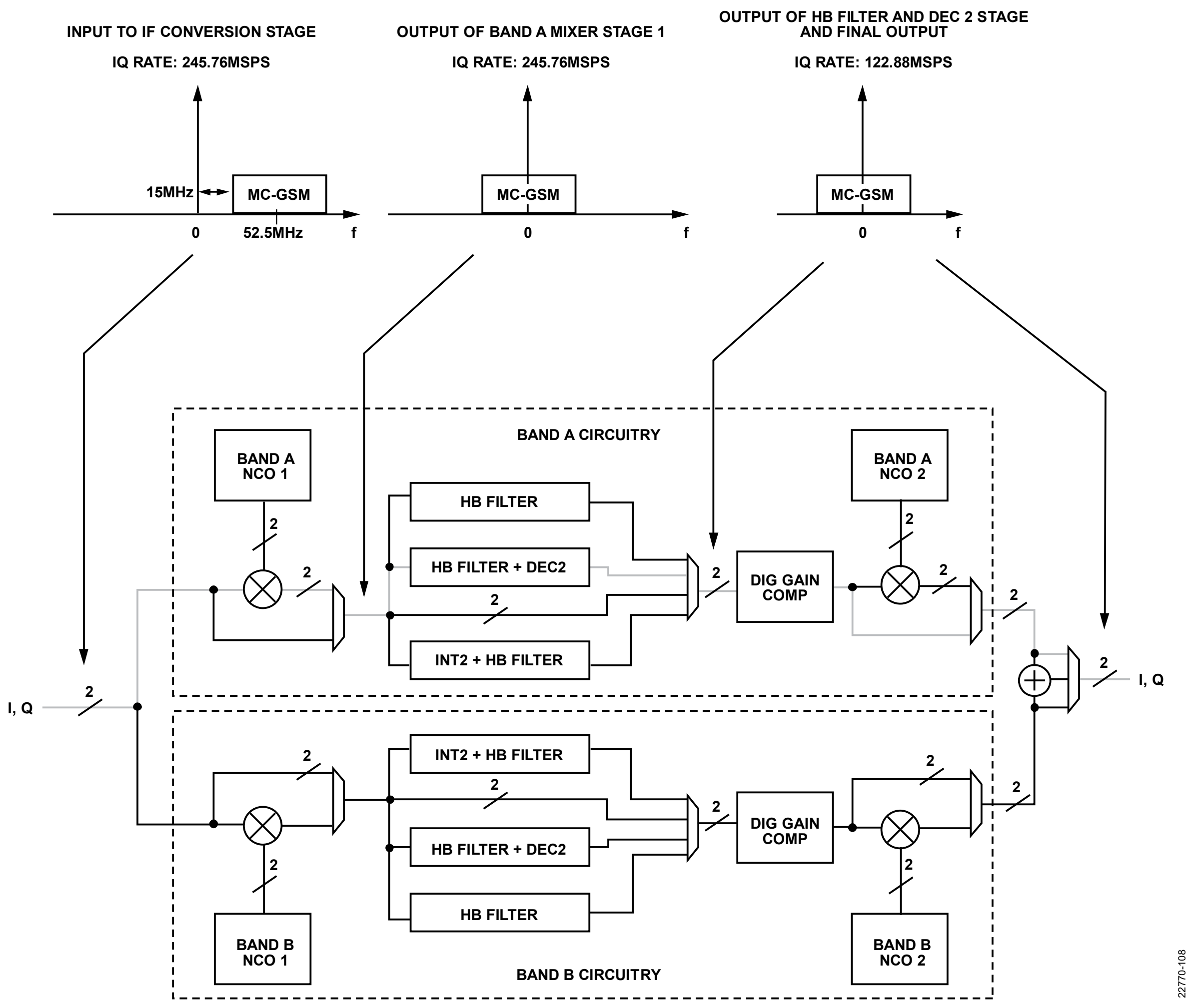
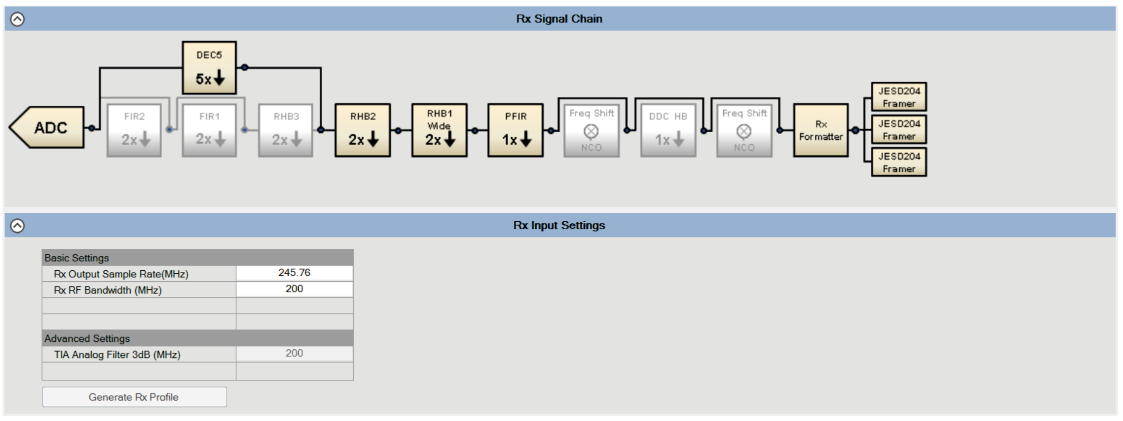


Figure 107. IF Conversion Stage in Zero-IF MC-GSM Configuration Block Diagram

RECEIVER SIGNAL PATH EXAMPLE

The TES provides an example that shows 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 receive channels. This example is a 200 MHz profile with an IQ rate of 245.76 MSPS.

Figure 108 shows the filter configuration for this example profile. The signal rate after the PFIR block is equal to the profile IQ rate.



Basic Settings	
Rx Output Sample Rate(MHz)	245.76
Rx RF Bandwidth (MHz)	200
Advanced Settings	
TIA Analog Filter 3dB (MHz)	200

Figure 108. Filter Configuration for Receive 200 MHz, IQ Rate 245.76 MSPS

The TES also provides a graph of the complete signal chain transfer function for this profile in the **Rx** tab under the **ChipConfig** dropdown (see Figure 109).

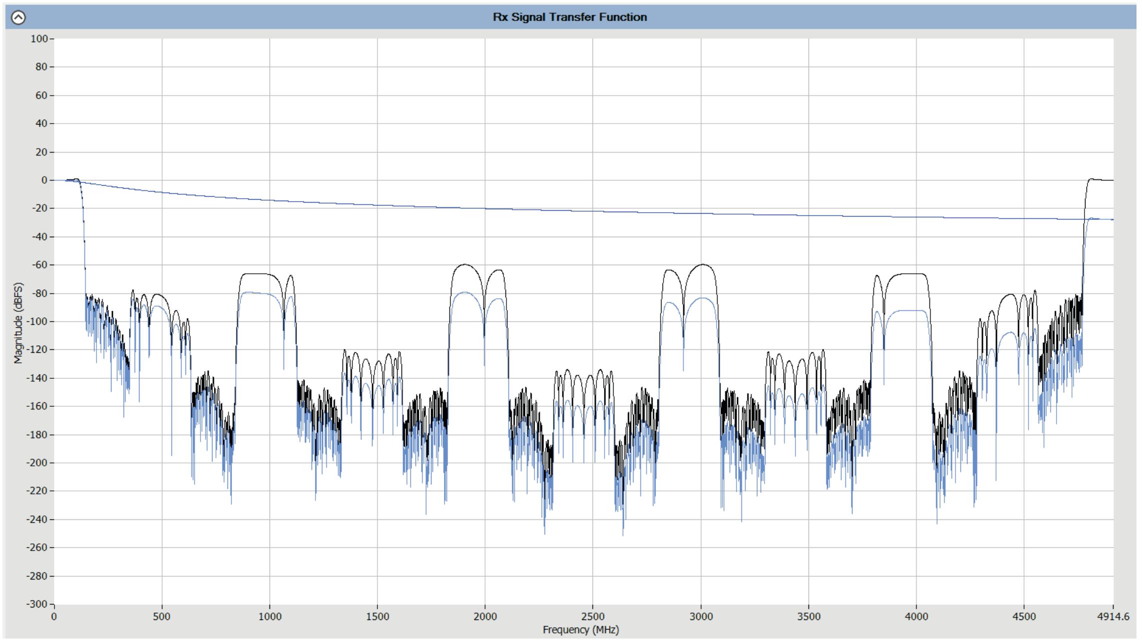


Figure 109. Receive Signal Transfer Function

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). The adi_adrv9025_RxProfile_t structure parameters are listed in Table 177.

Table 177. adi_adrv9025_RxProfile_t Structure Parameters

Name	Value	Description
channelType	A value of type adi_adrv9025_RxChannels_e	Chooses which channel is used to configure the filters described in Table 178
rxFirDecimation	1, 2, 4	Receive FIR decimation setting
rxDec5Decimation	4 = use a combination of FIR1, FIR2, and/or RHB3 5 = use DEC5	Set to use either the Dec5 or HB3 and HB2 in the Observation receive path
rhb1Decimation	1 = bypass, 2 = in use	Receive HB1 decimation setting
rhb1WideBandMode	0 – HB1 is narrow, 1 – HB1 is wider	Observation receive and loopback profiles ignore this field
rhb2Decimation	1, 2	Receive HB2 decimation factor
rhb3Decimation	1, 2	Receive HB3 decimation factor
rxFir1Decimation	1, 2	Receive FIR1 decimation factor
rxFir2Decimation	1, 2	Receive FIR2 decimation factor, observation receive 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)	Receive 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 receive FIR filter structure is described in Table 179
rxDdcMode	A value of type adi_adrv9025_RxDdc_e	The receive DDC mode settings are described in Table 180
rxNcoShifterCfg	A value of type adi_adrv9025_RxNcoShifterCfg_t	The receive NCO shifter configuration structure is described in Table 181
tiaPowerMode	0, 1, 2, 3	Four options for TIA power reduction modes (Range 0 to 3)

Table 178. adi_adrv9025_RxChannels_e Enumeration Definition

adi_adrv9025_RxChannels_e Enumeration	Enabled Channels
ADI_ADRV9025_RXOFF	No receive or observation receive 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 ORx1or ORx2channel enabled
ADI_ADRV9025_LB34	Tx3 or Tx4 internal loopback into ORx3 or ORx4 channel enabled

Receive PFIR Settings

The receive PFIR filter is specified in signed coefficients from +32767 to −32768. The gain block allows more flexibility when designing a digital filter. For example, a FIR can be designed with 6 dB gain in the pass band, and then that block can be set to −6 dB gain to give an overall 0 dB gain in the pass band. To calculate the gain of the filter coefficients, use the following equation:

$$DC\ Gain = \frac{\sum FIR\ Coefficients}{2^{15} - 1}$$

Table 179. 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 receive FIR
numFirCoefs	24, 48, 72	Number of taps to be used in the receive FIR
coefs[ADI_ADRV9025_MAX_RXPFIR_COEFS]	A pointer to an array of filter coefficients of size ADI_ADRV9025_MAX_RXPFIR_COEFS	

Receive DDC Mode

Receive 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 180.

Table 180. adi_adrv9025_RxDdc_e Enumeration Definition

adi_adrv9025_RxDdc_e Enumeration	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 stage and half-band filter stage are utilized.
ADI_ADRV9025_RXDDC_DEC2	In this mode, the half-band filter stage and decimate by 2 stage are utilized.
ADI_ADRV9025_RXDDC_BYPASS_REALIF	In this mode, the half-band filter stage and interpolation/decimation stage are bypassed. Q data is dropped at the input to the JESD core.
ADI_ADRV9025_RXDDC_FILTERONLY_REALIF	In this mode, the half-band filter stage is used, but the interpolation stage and decimation stage are bypassed. Q data is dropped at the input to the JESD core.
ADI_ADRV9025_RXDDC_INT2_REALIF	In this mode, the interpolate by 2 stage and half-band filter stage are utilized. Q data is dropped at the input to the JESD core.
ADI_ADRV9025_RXDDC_DEC2_REALIF	In this mode, the half-band filter stage and decimate by 2 stage are utilized. Q data is dropped at the input to the JESD code.

Receive NCO Shifter Configuration

The adi_adrv9025_RxNcoShifterCfg_t structure is contained within the adi_adrv9025_RxProfile_t structure. The adi_adrv9025_RxNcoShifterCfg_t structure contains the settings of the Band A and Band B NCO stages, as well as the bandwidth and baseband center frequency of the desired signal(s). These settings allows the API to ensure that the IF conversion stage is properly setup, and that the signal(s) post NCO shifting falls within the bandwidth provided by the IQ rate utilized and the pass-band bandwidth of the half-band filter, if utilized.

The NCOs can 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 which of the following modes is 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

Rule 1: Input Center Frequency Setup

Use the following relationships to ensure the center frequency is setup properly.

- inputCenterFreq + bandWidthDiv2 > primaryBwHz/2
- inputCenterFreq − bandWidthDiv2 < −primaryBwHz/2

Rule 2: Output Center Frequency Setup NCO1 (If DDC HB is Enabled)

Use the following relationships to ensure the NCO1 center frequency is setup properly.

- nco1OutputCenterFreq + bandWidthDiv2 > ddcHbCorner
- nco1OutputCenterFreq − bandWidthDiv2 < −ddcHbCorner

Rule 3: Output Center Frequency Setup NCO2

Use the following relationships to ensure the NCO2 center frequency is setup properly.

- nco2OutputCenterFreq + bandWidthDiv2 > outputRateHz/2
- nco2OutputCenterFreq − bandWidthDiv2 < −outputRateHz/2

Table 181. 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 dual-band 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 transmit signal path via the JESD high-speed serial data interface at the IQ data rate of the transmitter profile. The serial data is converted to parallel format through the JESD deframer into I and Q components. The data is processed through digital filtering and signal correction stages and input to I/Q DACs.

The DAC output is low pass filtered by the transmit LPFs and input to the upconversion mixer. The I and Q paths are identical to one another. Over-ranging is detected in the transmit digital signal path at each stage and limited to the maximum code value to prevent data wrapping. A block diagram of a transmit signal path is shown in Figure 110. Blocks that are not discussed in this section are faded.

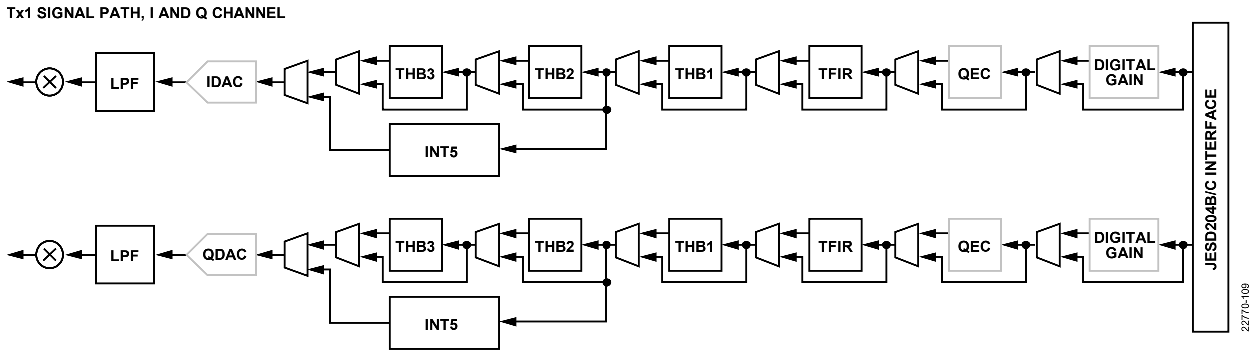


Figure 110. Transmit Signal Path Diagram

Analog LPF

The LPF is a second-order, analog Butterworth LPF with an adjustable 3 dB corner. The transmit chains of the device can support pass-band bandwidths up to 225 MHz (on I and Q). The LPF is calibrated during device initialization, which results 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 transmitter finite impulse response (TFIR) to ensure a maximally flat pass-band frequency response.

Interpolation By 5 Filter (INT5)

Either the INT5 filter or any combination of THB3 and THB2 are used in the transmit digital path. The INT5 filter interpolates by a factor of 5.

The INT5 filter coefficients include the following: +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, and +0.002929688.

Transmit Half-Band 3 Filter (THB3)

The THB3 filter is a fixed coefficient, half-band, interpolating filter. The THB3 filter can interpolate by a factor of 2 or the filter can be bypassed. The THB3 filter coefficients include the following: 0.125, 0.5, 0.75, 0.5, and 0.125.

Transmit Half-Band 2 Filter (THB2)

The THB2 filter is a fixed coefficient, half-band, interpolating filter. The THB2 filter can interpolate by a factor of 2 or the filter can be bypassed. The THB2 filter coefficients include the following: −0.08203125, 0, +0.58203125, +1, +0.58203125, 0, −0.08203125.

Transmit Half Band 1 Filter (THB1)

The THB1 filter is a fixed coefficient, half-band, interpolating filter. The THB1 interpolates by a factor of 2 or the filter can be bypassed. The THB1 filter coefficients include the following: −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 TFIR

The TFIR filter acts as an interpolating filter in the transmit path. The TFIR can interpolate by a factor of 1, 2, or 4, or the TFIR can be bypassed. The TFIR is used to compensate for roll-off caused by the post DAC analog LPF. The TFIR has a configurable number of taps that can be used including 20, 40, 60, or 80 taps. 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:

$$Tx\ PFIR\ Filter\ Taps_{MAX} = (DPCLK/Tx\_IQ\_DATARATE) \times 20$$

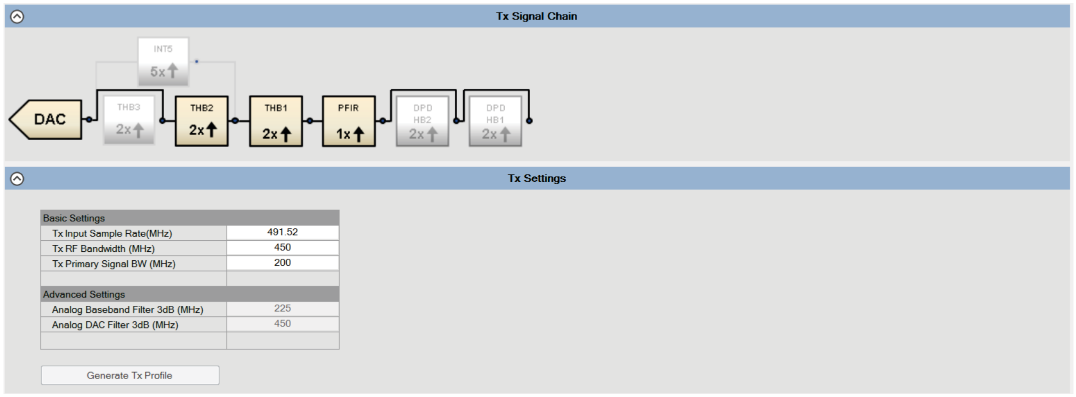
where:
Tx PFIR Filter Taps$_{max}$ is the maximum number of filter taps that can be used for the given clock rate
Tx_IQ_DATARATE is the input datarate of the filter

TRANSMIT SIGNAL PATH EXAMPLE

The TES provides an example that shows 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 transmit channels. This example is a 200 MHz/450 MHz profile with an IQ rate of 491.52 MSPS.

To explain the terminology of the 200 MHz/450 MHz profile, the 200 MHz refers to the transmit primary signal bandwidth, and the 450 MHz refers to the transmit RF bandwidth.

Figure 111 shows the filter configuration for this example profile. The signal rate after the TFIR block is equal to the profile IQ rate.



Basic Settings	
Tx Input Sample Rate(MHz)	491.52
Tx RF Bandwidth (MHz)	450
Tx Primary Signal BW (MHz)	200
Advanced Settings	
Analog Baseband Filter 3dB (MHz)	225
Analog DAC Filter 3dB (MHz)	450

Figure 111. Filter Configuration for the Transmit 200 MHz/450 MHz, 491.52 MSPS Profile

The combined transmit signal transfer function can be found in the **Tx** tab under the **ChipConfig** dropdown menu, as shown in Figure 112.

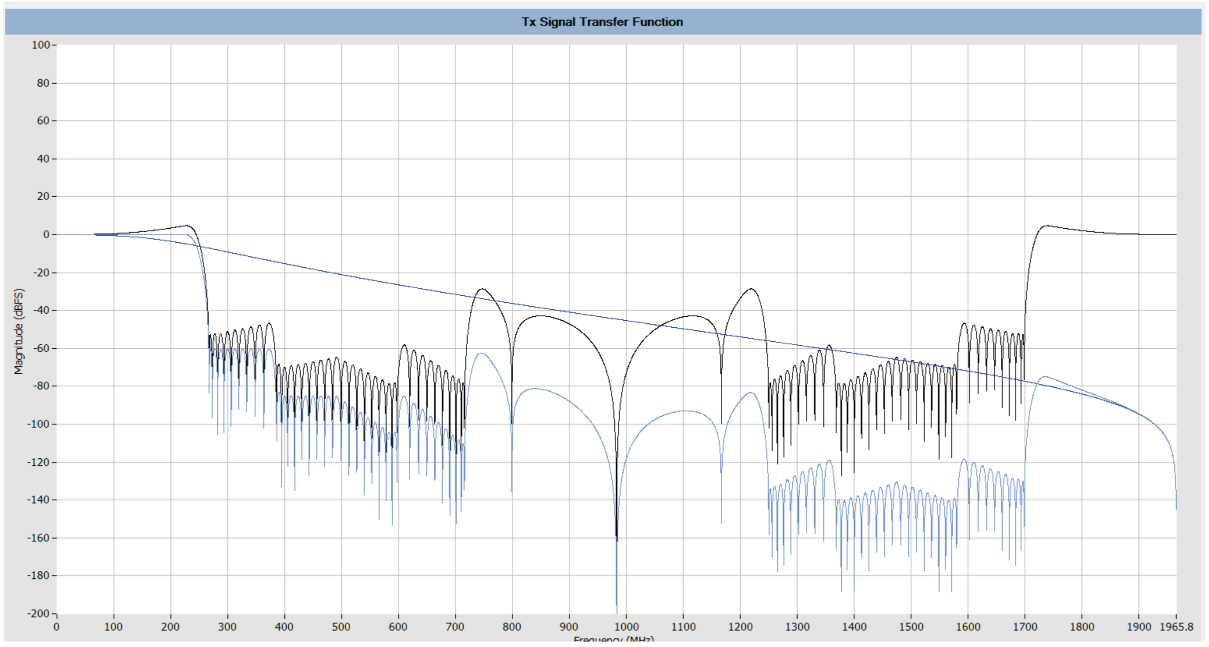


Figure 112. Transmit Signal Transfer Function

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). The adi_adrv9025_TxProfile_t structure parameters are described in Table 182.

Table 182. 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)	RF bandwidth specified in kHz
txDac3dBCorner_kHz	100000 to 450000 (based on currently defined use cases)	DAC3 dB corner specified in kHz
txBbf3dBCorner_kHz	50000 to 225000 (based on currently defined use cases)	BBF3 dB corner frequency specified in kHz
txFirInterpolation	1, 2, 4	Transmit FIR interpolation setting
thb1Interpolation	1 = bypass, 2 = in use	Transmit HB1 interpolation setting
thb2Interpolation	1 = bypass, 2 = in use	Transmit HB2 interpolation setting
thb3Interpolation	1 = bypass, 2 = in use	Transmit HB3 interpolation setting
txInt5Interpolation	1 = bypass, 5 = in use	Transmit INT5 interpolation setting
txFir	A value of type adi_adrv9025_TxFir_t	The txFir structure is explained in detail in the Transmit FIR Settings section
txBbfPowerMode	0 to 8	The transmit 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 look up tables (LUT) of power saving

Transmit FIR Settings

The adi_adrv9025_TxFir_t structure is contained within the adi_adrv9025_TxProfile_t structure. The adi_adrv9025_TxFir_t structure parameters are described in Table 183.

Table 183. 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 transmit FIR
numFirCoefs	20, 40, 60, 80	Number of taps to be used in the transmit FIR
coefs[ADI_ADRV9025_MAX_TXPFIR_COEFS]	A pointer to an array of filter coefficients of size ADI_ADRV9025_MAX_TXPRIF_COEFS	

The transmit 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 transceiver has four observation receiver inputs (ORx1, ORx2, ORx3, and ORx4) that can be used to capture data for DPD algorithms and other measurements or calibrations that require monitoring the transmitter outputs. The observation receiver can serve as an external loopback path to loop back the output of a power amplifier, provided the input level to the observation receiver is below the full-scale level of the ADC.

The observation channels have separate I and Q mixers. These mixers are identical to the receiver mixers except that the observation mixers include an LO multiplexer. The LO multiplexer allows either the RF PLL or the AUX PLL to provide the local oscillator signal source for the observation channel mixers.

The mixer feeds into a programmable TIA that serves as an LPF in the analog data path. The signal is converted by the ΣΔ ADC and filtered in half-band decimation stages and the PFIR. The fixed coefficient half-band filters (FIR1, RHB1(HR), RHB1(LP), RHB2, RHB3, and DEC5) and the PFIR are designed to prevent data wrapping and overrange conditions.

The IF conversion stage provides the ability to frequency shift or upsample/downsample digital data. Configurations supported for the observation receivers include real IF (real valued baseband data) configuration and low IF (complex data) configuration.

The diagram in Figure 113 shows the signal path for an observation receive signal chain.

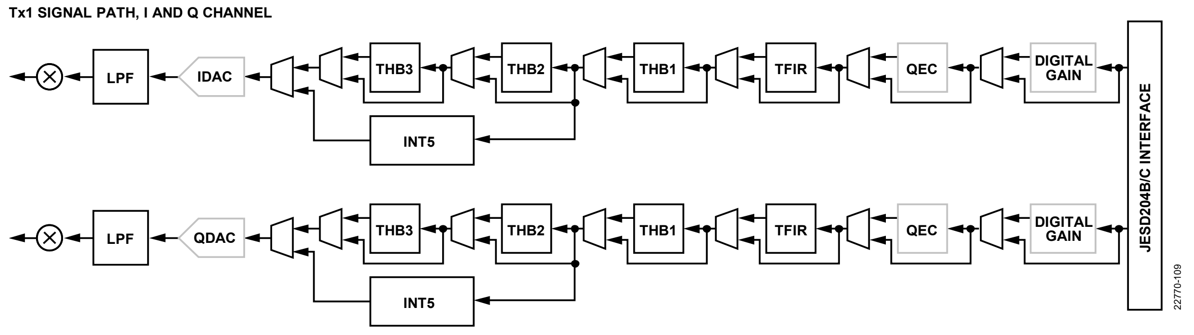


Figure 113. Observation Receive Signal Path

TIA

The observation receive TIA is an LPF 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 to ensure 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 observation receive pass band is compensated by the PFIR to ensure a maximally flat pass-band frequency response.

DEC5 Filter

Either the DEC5 filter or a combination of RHB3 and FIR1 is used in the receive digital path. The DEC5 filter decimates by a factor of 5 or the filter can be bypassed.

The DEC5 filter coefficients include the following: +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, and +0.000732422.

Finite Impulse Response 1 Filter (FIR1)

The FIR1 filter is a fixed coefficient, decimating filter. The FIR1 filter decimates by a factor of 2 or the filter can be bypassed.

The FIR1 filter coefficients include the following: 0.25, 0.75, 0.75, and 0.25.

Receive Half-Band 3 Filter (RHB3)

The RHB3 filter is a fixed coefficient, decimating filter. The RHB3 filter decimates by a factor of 2 or the filter can be bypassed.

The RHB3 filter coefficients include the following: −0.0625, 0.0078125, +0.5625, +0.984375, +0.5625, +0.0078125, and −0.0625.

Receive Half-Band 2 Filter (RHB2)

The RHB2 filter is a fixed coefficient, decimating filter. The RHB2 decimates by a factor of 2 or the filter can be bypassed.

The RHB2 filter coefficients include the following: −0.002929688, +0, 0.018554688, 0, −0.0703125, +0, 0.3046875, +0.500976563, +0.3046875, 0, −0.0703125, +0, 0.018554688, 0, and −0.002929688.

Receive Half-Band 1 High Rejection Filter (RHB1 (HR))

The RHB1 (HR) filter is a fixed coefficient, decimating filter. The RHB1 filter can decimate by a factor of 2 or the filter can be bypassed.

The RHB1 filter coefficients include the following: −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, and −0.000732422.

Receive Half-Band 1 Low Power Filter (RHB1 (LP))

The RHB1 (LP) filter is a fixed coefficient, decimating filter. The RHB1 filter can decimate by a factor of 2 or the filter can be bypassed.

The RHB1 filter coefficients include the following: −0.002685547, 0, +0.017333984, 0, −0.068359375, 0, +0.304443359, +0.501708984, +0.304443359, 0, −0.068359375, 0, +0.017333984, 0, and −0.002685547.

PFIR Filter

The PFIR filter acts as a decimating filter. The PFIR can decimate by a factor of 1, 2, or 4, or the filter can be bypassed. The PFIR filter compensates for the roll-off of the analog TIA LPF. The PFIR can use 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 divisor is 4 when using the HB2 and HB3 filters, and the divisor 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:

$$ORx\ PFIR\ Filter\ Taps_{MAX} = (DPCLK/ORx\_IQ\_DATARATE) \times 24$$

where:
$ORx\ PFIR\ Filter\ Taps_{MAX}$ is the maximum number of filter taps that can be used for the given clock rate
$ORx\_IQ\_DATARATE$ is the output datarate of the filter

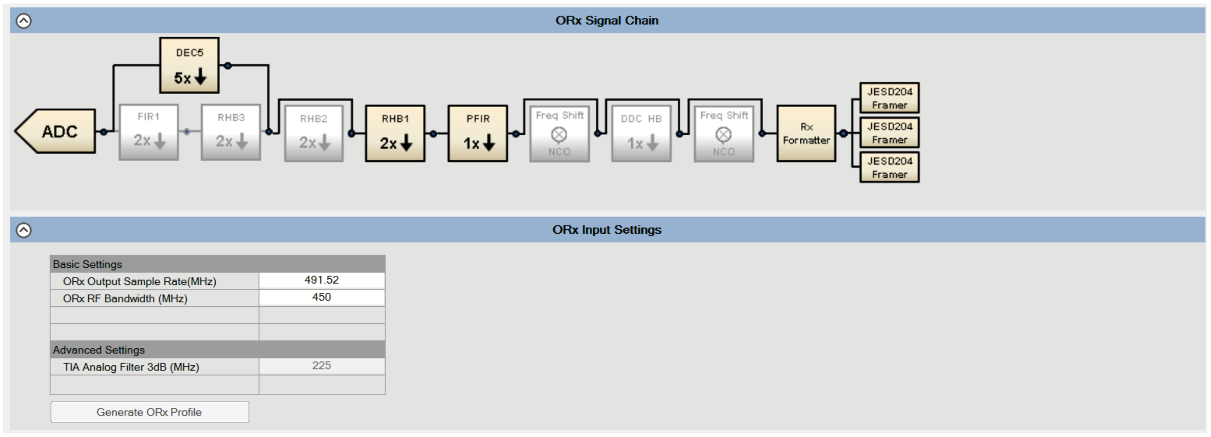
IF Conversion

Refer to the equivalent Receiver Signal Path section for information on the IF conversion stage.

OBSERVATION RECEIVER SIGNAL PATH EXAMPLE

The TES provides an example that shows how the baseband filtering stages are used in profile configurations for a signal pathway. In this example, the Observation receive 450 MHz, IQ Rate = 491.52 MSPS profile is selected for the Observation receive channels. This profile is compatible with the other examples provided in this user guide.

Figure 114 shows the filter configuration for this example profile. The clocking frequencies are noted in blue. The signal rate after the RFIR block is equal to the IQ Rate of the profile.



Basic Settings	
ORx Output Sample Rate(MHz)	491.52
ORx RF Bandwidth (MHz)	450
Advanced Settings	
TIA Analog Filter 3dB (MHz)	225

Figure 114. Filter Configuration for Observation Receive 450 MHz, IQ Rate = 491.52 MSPS

The Observation receive signal transfer function of the signal chain is in the **ORx** tab within the **ChipConfig** dropdown menu, as shown in Figure 115.

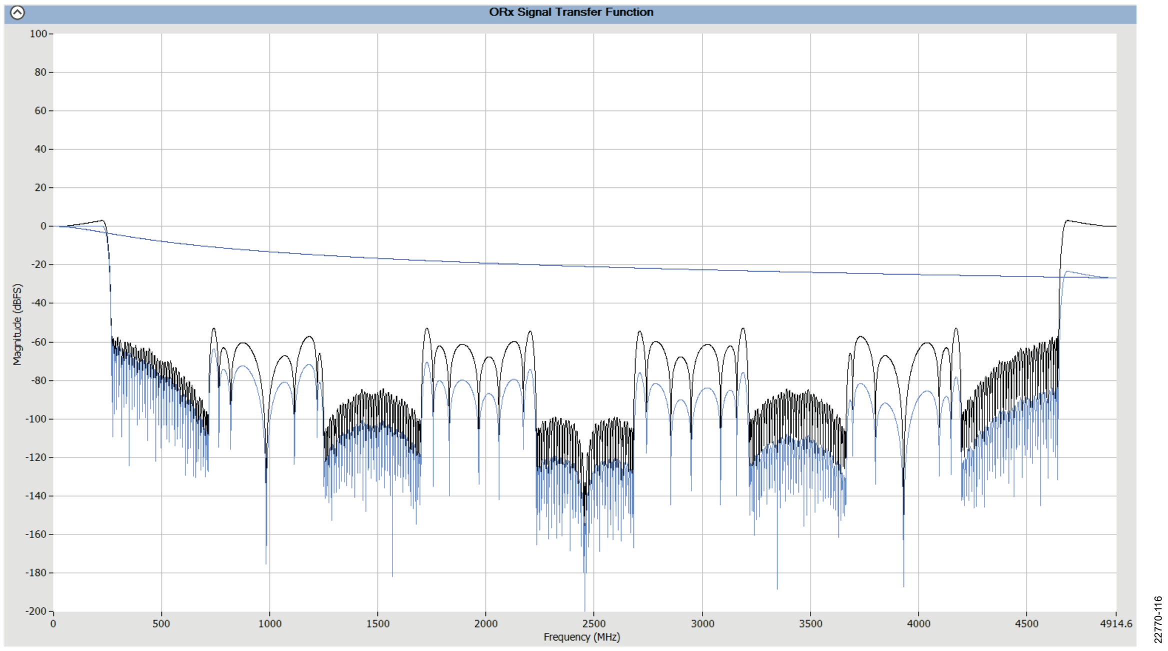


Figure 115. Observation Receive Signal Transfer Function

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). The adi_adrv9025_RxProfile_t structure contains the parameters listed in Table 184. For further details, refer to the Receiver Filter API Structure section.

Table 184. adi_adrv9025_RxProfile_t Structure Parameters

Name	Value	Description
channelType	A value of type adi_adrv9025_RxChannels_e	Choose which channel is used to configure the filters described in Table 178
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 the 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 HB2 decimation factor
rhb3Decimation	1, 2	ORx HB3 decimation factor
rxFir1Decimation	1, 2	ORx FIR decimation factor
rxFir2Decimation	1, 2	Receive FIR decimation factor, the ORx and loopback profiles ignore this field
rxOutputRate_kHz	122880 to 491520 (based on currently defined use cases)	The IQ data rate is specified in kHz (to the input of the JESD block)
rfBandwidth_kHz	112500 to 450000 (based on currently defined use cases)	The RF bandwidth is specified in kHz
rxBbf3dBCorner_kHz	112500 to 450000 (based on currently defined use cases)	The BBF 3 dB corner frequency is specified in kHz
rxAdcBandWidth_kHz	56250 to 225000 (based on currently defined use cases)	The receive 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 receive FIR filter structure is described in Table 179
rxDdcMode	A value of type adi_adrv9025_RxDdc_e	The receive DDC mode settings are described in Table 180
rxNcoShifterCfg	A value of type adi_adrv9025_RxNcoShifterCfg_t	The receive NCO shifter configuration structure is described in Table 181
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 172

DUAL BAND OVERVIEW: DUAL-BAND 2T2R SOLUTION

The normal transceiver configuration uses four transmitters, four receivers, and four observation receivers with either a common LO in TDD mode or dual LOs in FDD mode configured for a single radio band. Figure 116 shows how the transceiver is configured as a dual-band 2T2R solution by configuring half of the channels to operate in one band (Band A) and the other half of the device to operate in another band (Band B).

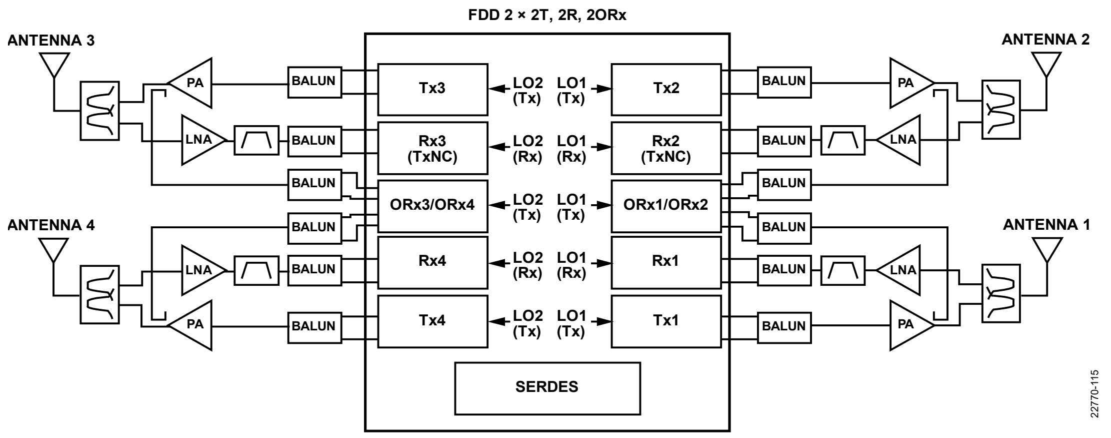


Figure 116. 2 Transmitter, 2 Receiver Dual-Band Mode

Each half of the transceiver can operate as TDD, FDD, or a mix of both. The transceiver contains two independent Los, LO1 and LO2, where LO1 is asigned to either Band A or Band B and LO2 is assigned to the other band. Reducing the number of LOs by sharing an LO minimizes the risks of LO to LO coupling issues, which is a major contributor to spurious issues in highly integrated RF ICs. Figure 117 shows one LO shared between the receive and transmit FDD bands.

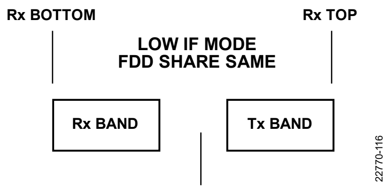


Figure 117. Dual Band LO Configuration

The frequency planning for the LO frequency selection is flexible. The receiver bandwidth remains at 200 MHz, as specified in the device data sheet. The transmitter bandwidth has been extended beyond 200 MHz to accomodate 3GPP bands that require larger receiver and transmitter bandwidths and duplex spacing. The transmitter channels have internal interpolation and NCOs to shift the input carriers beyond the 200 MHz primary bandwidth. The device data sheet has transmitter QEC performance specifications for operation beyond the 200 MHz primary bandwidth. Choose the LO frequency necessary to place the transmitter image out of the receiver band so that there is no impact to the receiver sensitivity. Duplexer rejection of the transmitter image is required to meet transmit emissions specifications. The receiver and transmitter channels both have NCOs, so the low IF configuration is transparent to the baseband.

The transceiver has three dual-band profiles to choose from that optimize power consumption versus required bandwidths based on user application. These profiles include the following:

- UC51nonLinkSharing, TDD bands with transmit and receive bandwidths greater than 100 MHz and an I/Q data rate of 245.76 MSPS.
- UC54nonLinkSharing, FDD or TDD bands with transmit and receive bandwidths less than 100 MHz and I/Q data rates of 122.88 MSPS. Reduced transmit and receive IQ rates are included to save power consumption and lower cost FPGA solution.
- UC55nonLinkSharing, similar to UC54 except the receive bandwidth is reduced to 160 MHz if additional receive channel filtering is desired.

LO Assignment

The firmware determines if the device is in dual-band mode when the TX1 and TX2 selected LO is the same as the RX1 and RX2 LO and different than the LO selected for the TX3, TX4 and RX3, and RX4 channels. The following code is an example how this is setup. Receiver channels 1 and 2 and transmitter channels 1 and 2 share LO1, and receiver channels 3 and 4 and transmitter channels 3 and 4 share LO2. It is also acceptable to use the opposite LO in this assignment.

The following init structures are used to make LO assignments:

```
initStruct = link.platform.board.Adrv9025Device.InitStructGet()
initStruct.clocks.rx12LoSelect = Types.adi_adrv9025_LoSel_e.ADI_ADRV9010_LOSEL_LO1
initStruct.clocks.rx34LoSelect = Types.adi_adrv9025_LoSel_e.ADI_ADRV9010_LOSEL_LO2
initStruct.clocks.tx12LoSelect = Types.adi_adrv9025_LoSel_e.ADI_ADRV9010_LOSEL_LO1
initStruct.clocks.tx34LoSelect = Types.adi_adrv9025_LoSel_e.ADI_ADRV9010_LOSEL_LO2
initStruct.clocks.orx12LoSelect = Types.adi_adrv9025_OrxLoSel_e.ADI_ADRV9025_ORXLOSEL_TXLO
initStruct.clocks.orx34LoSelect = Types.adi_adrv9025_OrxLoSel_e.ADI_ADRV9025_ORXLOSEL_TXLO
postMcsInit = Types.adi_adrv9025_PostMcsInit_t()
postMcsInit.radioCtrlInit.lo1PllFreq_Hz = 1810000000
postMcsInit.radioCtrlInit.lo2PllFreq_Hz = 2593000000
```

The AUX LO must remain cleared. For observation receivers, only the transmitter LO can be used. The AUX LO provides the source needed to run tracking calibrations that are controlled by the ARM processor.

For dual-band applications, it is possible for Band A (Channel 1 and Channel 2) to be in real traffic and for Band B (Channel 3 and Channel 4) to require an LO frequency change. For more information on the procedure to change the LO frequency, see the Calibration Guidelines after PLL Frequency Changes section. The dual-band considerations include the following:

- Change the LO frequency using the following process:
  - Disable all tracking on all four channels.
  - Change the LO frequency.
  - Run the internal path delay initial calibration on the desired channels.
  - Run the external LO leakage initial calibration on the desired channels.
  - Enable the tracking on all four channels.
- For devices that incorporate crest factor reduction (CFR), change the CFR correction pulses using the following process:
  - Disable all tracking on all four channels.
  - Load the CFR correction pulses.
  - Run the CFR initial calibration.
  - Enable the tracking.
- TX_EN, RX_EN, and ORX_CTRL_X can continue toggling when these initial calibrations are called.

DUAL-BAND CONFIGURATION AND EXAMPLE USE CASES

For most bands, the LO can be selected to fall within the transmit band. The widest 3GPP bands are Band 1 and the AWS-1 bands (4, 10, or 66).

An example frequency plan for Band 1 is shown in Figure 118. An LO frequency of 2015 MHz places the transmit image at the edge of the receive band so as not to impact receive sensitivity. A transmit NCO frequency of 125 MHz places a 0 Hz centered baseband input at the proper output RF frequency, which requires 310 MHz of transmit bandwidth. Using a receiver NCO frequency of 65 MHz shifts the receive band to 0 IF, if desired. A DPD correction of 3× requires 430 MHz bandwidth, which is within the 450 MHz available.

An example frequency plan for Band 3 is shown in Figure 119. With these two frequency plans, the transceiver can provide a dual-band 2-transmitter, 2-receiver, 2-observation receiver solution.

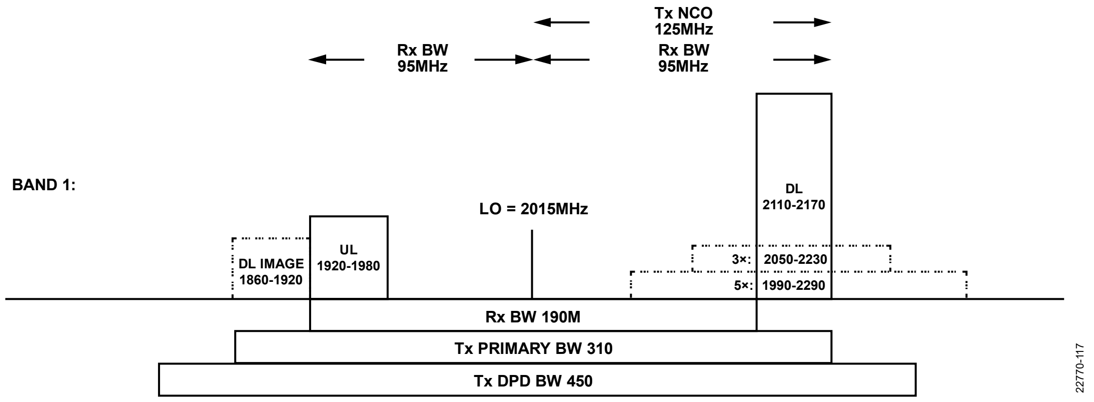


Figure 118. Band1 Frequency Plan Example (UL = uplink, DL = downlink)

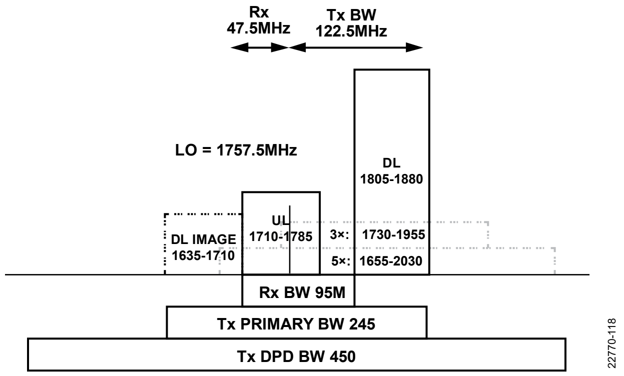


Figure 119. Band 3 Frequency Plan Example

Band 4, Band 10, and Band 66 have the largest bandwidth from the bottom of the receive bands and the top of the transmit bands. Band 66 is the largest, and Band 4 and Band 10 are subsets of Bands 66. The span from the top of the transmit band to the bottom of receive band is 490 MHz. This range is too wide a bandwidth to share a single LO. Operators in the United States that have an AWS-1 band also have Band 25. In this case, Band 25 can be paired with these AWS-1 bands (B66 or B4 or B10). Band 2 is also a subset of Band 25. These frequency plans require less bandwidth than Band 1. Because transmitters and receivers do not share an LO, transmit images do not impact the receive channel. Band 1 is the most stringent 2T2R use case. A combination of Band 66 and Band 25 can be accomodated using LO1 (1812.5 MHz) for the Band 66 and Band 25 uplink and LO2 (2042.5 MHz) for the Band 66 and Band 25 downlink.

GPIO CONFIGURATION

The transceiver features 19 digital general purpose input/output (GPIO) pins that can be used for a variety of functions. The transceiver also features eight analog GPIO (GPIO_ANA_x) pins. The GPIO pins and GPIO_ANA pins provide a real-time interface either 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 the input signal level is too high.

The GPIO_ANA pins serve as the output pins for eight auxiliary digital to analog converter (AUXDAC_x) signals. The AUXDAC can be used to provide a control voltage to peripheral devices. The AUXDAC is not a precision converter device and is recommended to be used in applications where high accuracy is not needed. It is recommended to use the AUXDAC in feedback systems rather than in open-loop control systems.

The digital GPIO supply is the VIF supply voltage. The GPIO_ANA supply is the 1.8V analog supply connected through the VANAx_1p8 pins. 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. Input mode allows the baseband processor to drive pins on the transceiver to execute specific tasks. 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

This command configures pins for input direction.

Parameters

Table 185. adi_adrv9025_GpioInputDirSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
gpioInputMask	Selects the device GPIO pins required to be set as an input in the 0x00000 to 0x7FFFF range. If a bit is set high, the GPIO pin associated with that 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

This command configures pins for output direction.

Parameters

Table 186. adi_adrv9025_GpioOutputDirSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
gpioInputMask	Selects the device GPIO pins that are required to be set as an output in the 0x00000 to 0x7FFFF range. If a bit is set high, the GPIO pin associated with that 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 can 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 187, the API automatically sets the I/O direction of the GPIO pins assigned for the feature. More details on these features are provided in the following subsections.

Table 187. Summary of Input GPIO Features

Feature	Description	GPIO Pins Available for Feature
SPI2	Secondary SPI channel for control and readback of receiver gain index and transmitter attenuation API configuration commands are adi_adrv9025_Spi2CfgSet(…) and adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(…)	GPIO_0: SPI_DIO (input or output) GPIO_1: SPI_DO (output only) GPIO_2: SPI_CLK (input) GPIO_3: SPI_$\overline{\text{CS}}$ (input) GPIO_4 through GPIO_18: transmit attenuation state select
Pin Controlled Receive/ORx Gain Index Increment and Decrement	Configure specific GPIO pins to increment or decrement the gain index on any receive or ORx channel after a rising edge on the assigned pin. API configuration command is adi_adrv9025_RxGainPinCtrlCfgSet(…)	GPIO_0 through GPIO_15: receive/ORx gain index increment pin select GPIO_0 through GPIO_15: receive/ORx gain index decrement pin select
Pin Controlled Transmit Attenuation Increment and Decrement	Configures specific GPIO pins to increment or decrement attenuation on any transmit channel after a rising edge on the assigned pin API configuration command is adi_adrv9025_TxAttenPinCtrlCfgSet(…)	GPIO_0 through GPIO_15: transmit attenuation increment pin select GPIO_0 through GPIO_15: transmit 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 three are available in input mode) API configuration command is adi_adrv9025_RxDataFormatSet(…)	GPIO_[2:0] = assign to any receive GPIO_[5:3] = assign to any receive GPIO_[8:6] = assign to any receive GPIO_[11:9] = assign to any receive GPIO_[14:12] = assign to any receive GPIO_[17:15] = assign to any receive
Transmit Observation Receiver Select	When using fewer than four ORx channels, the ORx channel requires information on which transmit channel data is presented to the ORx, if a pin interface is required to indicate the transmit to ORx mapping, the following command sets up the pins, provided that the stream file is generated with appropriate input settings API configuration command is adi_adrv9025_StreamGpioConfigSet(…)	GPIO_0 through GPIO_15.

SPI2

A complete description, including descriptions of custom data types, for the SPI2 interface can be found in the SPI2 Description section.

The SPI2 interface acts as a secondary SPI channel that operates on the digital GPIO_[3:0] pins. An optional pin can be configured for toggling the transmit attenuation between the S1 attenuation state and the S2 attenuation state on the GPIO_4 through GPIO_18 pins. The SPI2 interface uses the same SPI configuration used on the primary SPI interface. SPI2 can be used to set the gain index on the receiver and observation receiver channels, read back the gain index on those channels, and set up two distinct transmit attenuation states that the user can alternate between by switching a GPIO pin. The SPI2 interface cannot access registers available to the primary SPI interface.

When the SPI2 feature is enabled, the GPIO_[3:0] pin and the pin assigned for transmit attenuation selection (can be GPIO_4 through GPIO_18 or left unassigned) cannot be used for other purposes. When SPI2 is enabled, it overrides the functionality previously assigned to the digital GPIO_[3:0] pins. Refer to Table 187 for specific pin mapping details.

adi_adrv9025_Spi2CfgSet(…)

```
adi_adrv9025_Spi2CfgSet(adi_adrv9025_Device_t* device, uint8_t spi2Enable)
```

Description

This command enables the SPI2 feature.

Parameters

Table 188. adi_adrv9025_SpiCfgSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
Spi2Enable	Sets the state of the SPI2 bus, 1 = enable and 0 = disable.

adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(…)

```
adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(adi_adrv9025_Device_t* device,
adi_adrv9025_TxAttenSpi2PinCfg_t txAttenSpi2PinCfg[], uint8_t numTxAttenSpi2PinConfigs)
```

Description

This command assigns the transmit attenuation select pin.

Parameters

Table 189. adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
txAttenSpi2PinCfg[]	Pointer to an array of the adi_adrv9025_TxAttenSpi2PinCfg_t structure that configures the transmit attenuation SPI2 pin control. Note that multiple transmitters can share an attenuation select pin, if desired.
numTxAttenSpi2PinConfigs	This parameter determines the number of channelized transmit attenuation SPI2 pin configurations passed in the txAttenSpi2PinCfg array.

Pin-Based Receive Gain Control

A complete description of the pin-based receive gain control feature is provided in the Receiver Gain Control and Gain Compensation section.

Pin-based receive gain control is relevant for applications that require MGC and precise timing for gain change events. The pin-based control scheme offers a 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 two AGC clock cycles in the logic high state. The gain change because of 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 receive 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. The off-table gain indices (that is, gain indices below 195) can correspond to the maximum gain condition. Take 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 receive gain control feature. Note that the device must be in MGC for proper operation.

Parameters

Table 190. adi_adrv9025_RxGainPinCtrlCfgSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
rxChannel	This parameter selects which receive channel is used for configuring the pin-based receive gain control.
*rxGainPinCtrlCfg	Pointer to the adi_adrv9025_RxGainPinCfg_t structure containing the configuration values for the pin-based receive gain control.

Table 191 describes the adi_adrv9025_RxGainPinCfg_t data structure used in this command.

Table 191. Description of adi_adrv9025_RxGainPinCfg_t Data Structure

Parameter	Data Type	Comments
incStep	uint8_t	An increment in the gain index is applied when the increment gain pin is pulsed. A value from 0 to 7 applies a step size of 1 to 8.
decStep	uint8_t	A decrement in the gain index is applied when the increment gain pin is pulsed. A value from 0 to 7 applies a step size of 1 to 8.
rxGainIncPin	adi_adrv9025_GpioPinSel_e	Choose the GPIO used for the increment gain input. ADI_ADRV9025_GPIO_00 to ADI_ADRV9025_GPIO_15 can be used.
rxGainDecPin	adi_adrv9025_GpioPinSel_e	Choose the GPIO assigned for the decrement gain input. ADI_ADRV9025_GPIO_00 to ADI_ADRV9025_GPIO_15 can be used.
enable	uint8_t	Enable or disable the gain pin control. Enable = 1 and disable = 0.

Pin-Based Transmit Attenuation Control

A complete description of transmit attenuation control is provided in the Transmitter Overview and Path Control section.

Pin-based transmit attenuation control, similar to the transmit attenuation select feature of SPI2, provides an interface to make attenuation adjustments with precise timing control. The pin-based control scheme offers a 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 transmit attenuation control is that SPI2 allows switching between the programmed attenuation states (S1 and S2) and pin-based transmit attenuation control allows multiple increments or decrements of transmit attenuation.

adi_adrv9025_TxAttenPinCtrlCfgSet(…)

```
adi_adrv9025_TxAttenPinCtrlCfgSet(adi_adrv9025_Device_t* device, adi_adrv9025_TxAttenPinCfg_t
txAttenPinCfg[],uint8_t numTxAttenPinConfigs)
```

Description

This command configures the pin-based transmit gain control feature.

Parameters

Table 192. adi_adrv9025_TxAttenPinCtrlCfgSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
txAttenPinCfg[]	Pointer to an array of the adi_adrv9025_TxAttenPinCfg_t structure that configures the transmit attenuation pin control.
numTxAttenPinConfigs	This parameter determines the number of channelized transmit attenuation pin configuration passed in the txAttenPinCfg array.

Table 193 describes the adi_adrv9025_TxAttenPinCfg_t data structure used in this command.

Table 193. Description of adi_adrv9025_TxAttenPinCfg_t Data Structure

Parameter	Data Type	Comments
txChannelMask	uint32_t	Choose the bitwise channel mask that the transmit attenuation pin configuration settings are applied to. [D0] = Tx1, [D1] = Tx2, [D2] = Tx3, [D3] = Tx4.
stepSize	uint8_t	This parameter sets the change in transmit attenuation for each increment or decrement signal received in increment and decrement mode with a step size of 0.5dB/LSB. The valid range is from 0 to 31.
txAttenIncPin	adi_adrv9025_GpioPinSel_e	Choose the GPIO assigned for the increment attenuation input. ADI_ADRV9025_GPIO_00 to ADI_ADRV9025_GPIO_15 can be used
txAttenDecPin	adi_adrv9025_GpioPinSel_e	Choose the GPIO assigned for the decrement attenuation input. ADI_ADRV9025_GPIO_00 to ADI_ADRV9025_GPIO_15 can be used
enable	uint8_t	Enable or disable the gain pin control. Enable = 1 and disable = 0.

External Slicer Mode

A complete description of the external slicer use case is provided in the Receiver Gain Control and Gain Compensation section.

The receive datapath features a GPIO-based slicer used in conjunction with the digital gain compensation to digitally attenuate data sent over the JESD204B/JESD204C interface. The digital gain compensation can 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. Three GPIO pins per receiver are utilized. See Table 197 for the valid external slicer pins.

adi_adrv9025_RxDataFormatSet(…)

```
adi_adrv9025_RxDataFormatSet(adi_adrv9025_Device_t* device, adi_adrv9025_RxDataFormat_t
rxDataFormat[], uint8_t arraySize)
```

Description

This command configures the external slicer mode.

Parameters

Table 194. adi_adrv9025_RxDataFormatSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
rxDataFormat[]	Pointer to the receive data format configuration structure.
arraySize	This parameter determines the size of the rxDataFormat array that represents the number of configurations.

Table 195 describes the adi_adrv9025_RxDataFormat_t data structure.

Table 195. Description of adi_adrv9025_RxDataFormat_t Data Structure

Parameter	Data Type	Comments
rxChannelMask	uint32_t	Receive channel mask settings
formatSelect	adi_adrv9025_RxDataFormatModes_e	Receive channel format mode select
floatingPointConfig	adi_adrv9025_FloatingPointConfigSettings_t	Receive channel floating point format configuration
integerConfigSettings	adi_adrv9025_IntegerConfigSettings_t	Receive channel integer format configuration
slicerConfigSettings	adi_adrv9025_SlicerConfigSettings_t	Receive channel integer slicer configuration
externalLnaGain	uint8_t	Selects the slicer to compensate for external dual-band LNA (0 = disable, 1 = enable)
tempCompensationEnable	uint8_t	Selects the slicer to compensate for temperature variations (0 = disable, 1 = enable)

For 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 196.

Table 196. Description of adi_adrv9025_SlicerConfigSettings_t Data Structure

Parameter	Data Type	Comments
extSlicerStepSize	adi_adrv9025_ExtSlicerStepSizes_e	Enumeration selects the external pin gain step size
intSlicerStepSize	adi_adrv9025_IntSlicerStepSizes_e	Enumeration selects the internal pin gain step size
rx1ExtSlicerGpioSelect	adi_adrv9025_RxExtSlicerGpioSel_e	Enumeration selects the Rx1 Ext Ctrl GPIO configuration
rx2ExtSlicerGpioSelect	adi_adrv9025_RxExtSlicerGpioSel_e	Enumeration selects the Rx2 Ext Ctrl GPIO configuration
rx3ExtSlicerGpioSelect	adi_adrv9025_RxExtSlicerGpioSel_e	Enumeration selects the Rx3 Ext Ctrl GPIO configuration
rx4ExtSlicerGpioSelect	adi_adrv9025_RxExtSlicerGpioSel_e	Enumeration selects the Rx4 Ext Ctrl GPIO configuration

The enum adi_adrv9025_RxExtSlicerGpioSel_e structure provides the list of GPIO groupings available when using external slicer mode, as shown in Table 197.

Table 197. Description of adi_adrv9025_RxExtSlicerGpioSel_e Enumeration

Enumeration Name	Enumeration 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 receive gain slicer external, GPIO2, GPIO1, and GPIO_0
ADI_ADRV9025_EXTSLICER_RX_GPIO_5_DOWNTO_3	2	Select receive gain slicer external, GPIO5, GPIO4, and GPIO3
ADI_ADRV9025_EXTSLICER_RX_GPIO_8_DOWNTO_6	3	Select receive gain slicer external, GPIO8, GPIO7, and GPIO6
ADI_ADRV9025_EXTSLICER_RX_GPIO_11_DOWNTO_9	4	Select receive gain slicer external, GPIO11, GPIO10, and GPIO9
ADI_ADRV9025_EXTSLICER_RX_GPIO_14_DOWNTO_12	5	Select receive gain slicer external, GPIO14, GPIO13, and GPIO12
ADI_ADRV9025_EXTSLICER_RX_GPIO_17_DOWNTO_15	6	Select receive gain slicer external, GPIO17, GPIO16, and GPIO_15

Other members of the adi_adrv9025_RxDataFormatter_t structure are discussed in the Receiver Gain Control and Gain Compensation section.

Transmitter to Observation Receiver Mapping

A full description of transmitter to observation receiver mapping is provided in the Use Cases section.

For initial calibrations and tracking calibrations that require an external transmit to observation receive loopback channel for the algorithm, the ARM processor must understand the specific mapping of transmit to observation receive at that time. In the use case with four observation receivers, the mapping is typically static, and it is recommended to use the adi_adrv9025_TxToOrxMappingSet(…) API command to configure the mapping. In the use case with two observation receivers, each observation receive channel must know which transmit channel is provided as the input. An alternative to the API command interface is to use a GPIO-based interface to inform the ARM about the currently mapped transmit channels into the observation receive. To clarify, the baseband processor informs the transceiver about the channel mapping state by signaling on the GPIO, 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 the GPIO_0 pin to the GPIO_15 pin. Up to four GPIO are required to fully implement pin-based mapping controls. A partial implementation can be achieved with two GPIO. The partial implementation only indicates which transmit was mapped to the observation receive (TX_SEL signal) and does not permit the baseband processor to inform the device that the device 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 to use for the TX_SEL and TX_EN signals.

adi_adrv9025_StreamGpioConfigSet(…)

```
adi_adrv9025_ StreamGpioConfigSet(adi_adrv9025_Device_t* device,
 adi_adrv9025_StreamGpioPinCfg_t* streamGpioPinCfg);
```

Description

With the proper stream file, the user can configure the stream processor to listen to the input GPIO with the following command.

Note that this command is called as a part of the adrv9025_RadioctrlInit command, which is called during the adi_adrv9025_PostMcsInit(…) command.

This function associates a GPIO pin with the stream processor general-purpose inputs and enables the stream trigger functionality if a valid GPIO (GPIO_0 to GPIO_15) is assigned to the internal streamGpInput path.

There are 16 GPIO inputs available to trigger streams. These GPIO inputs can be mapped to one of the pins GPIO_0 to GPIO_15.

To unmap a GPIO association with a stream general-purpose input, set the GPIO input to ADI_ADRV9025_GPIO_INVALID.

Parameters

Table 198. adi_adrv9025_StreamGpioConfigSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
streamGpioPinCfg	Data structure containing the GPIO assignments for the stream processor inputs.

Table 199. Description of the adi_adrv9025_StreamGpioPinCfg_t Data Structure

Member	Data Type	Description
streamGpInput0	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 0 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput1	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose P Input 1 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput2	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 2 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput3	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 3 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput4	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 4 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput5	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 5 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput6	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 6 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput7	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 7 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput8	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 8 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput9	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 9 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput10	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 10 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput11	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 11 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput12	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 12 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput13	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 13 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput14	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 14 (valid GPIO_0 to GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.
streamGpInput15	adi_adrv9025_GpioPinSel_e	Select the desired GPIO pin input to stream the processor general-purpose Input 15 (valid GPIO_0 GPIO_15). To disable select ADI_ADRV9025_GPIO_INVALID.

Description

This command sets the source control.

Parameters

Table 200. adi_adrv9025_GpioOutSourceCtrlSet(…) Parameters

Parameter	Description
*device	Pointer to device structure.
gpioSrcCtrl	Selects 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 201.

Table 201. Description of the Nibble Groups Configured Via gpioSrcCtrl

gpioSrcCtrl[bits]	Description
gpioSrcCtrl[d3:d0]	GPIO output source for GPIO_3 to GPIO_0 pins
gpioSrcCtrl[d7:d4]	GPIO output source for GPIO_7 to GPIO_4 pins
gpioSrcCtrl[d11:d8]	GPIO output source for GPIO_11 to GPIO_8 pins
gpioSrcCtrl[d15:d12]	GPIO output source for GPIO_15 to GPIO_12 pins
gpioSrcCtrl[d19:d16]	GPIO output source for GPIO_18 to GPIO_16 pins

The values for these nibble groupings can be formed with the adi_adrv9025_GpioOutputModes_e enumeration. This enumeration is described in Table 202.

Table 202. Description of adi_adrv9025_GpioOutputModes_e Enumeration

Enumeration Name	Enumeration 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, the GPIO can be set as an input pin. For example, consider a use case where three pins in a 4-pin nibble group are dedicated for slicer output mode. The fourth 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

This command sets the output logic level of the GPIO pins after configuring the I/O direction and source control.

Parameters

Table 203. adi_adrv9025_GpioOutPinLevelSet(…) Parameters

Parameter	Description
*device	Pointer to the 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_x pins is to serve as control pins for an external control element, such as a DSA or LNA. Other features can be exposed in future software releases. A high level overview of the GPIO_ANA_x features are provided in Table 204.

Table 204. Summary of GPIO_ANA Features

Feature	Description	GPIO Pins Available for Feature
Receive Gain Table External Control Word Output	The receive gain table includes a column for 2-bit control of an external gain element. Each receive channel is associated with two fixed GPIO_ANA_x pins. The 2-bit value expressed on the pins depends on the gain index and gain table column. The API function for configuration is adi_adrv9025_RxGainTableExtCtrlPinsSet(…).	GPIO_ANA_1 and GPIO_ANA_0: Rx1 external control word
		GPIO_ANA_3 and GPIO_ANA_2: Rx2 external control word
		GPIO_ANA_5 and GPIO_ANA_4: Rx3 external control word
		GPIO_ANA_7 and GPIO_ANA_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_x pins. The configuration of the GPIO pins for the 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

This command configures the GPIO pins for the gain table external control word.

Parameters

Table 205. adi_adrv9025_RxGainTableExtCtrlPinsSet(…) Parameter

Parameter	Description
*device	Pointer to the device structure.
extCtrlGpioChannelEnable	Determines the adi_adrv9025_RxChannels_e enumeration type to select which set of gain table external control words to output on analog GPIOs.

Table 206 describes the adi_adrv9025_RxExtCtrlPinOutputEnable_e enumeration.

Table 206. Description of adi_adrv9025_RxExtCtrlPinOuputEnable_e Enumeration

Enumeration Name	Comments
ADI_ADRV9025_DISABLE_RX1_RX2_EXT_CTRL_GPIOS	Disable Rx1 and Rx2 external control words output on the analog GPIOs
ADI_ADRV9025_ENABLE_RX1_RX2_EXT_CTRL_GPIOS	Enable Rx1 and Rx2 external control words output on the analog GPIOs
ADI_ADRV9025_DISABLE_RX3_RX4_EXT_CTRL_GPIOS	Disable Rx3 and Rx4 external control words output on the analog GPIOs
ADI_ADRV9025_ENABLE_RX3_RX4_EXT_CTRL_GPIOS	Enable Rx3 and Rx4 external control words output on the analog GPIOs
ADI_ADRV9025_DISABLE_RX1_RX2_RX3_RX4_EXT_CTRL_GPIOS	Disable Rx1, Rx2, Rx3 and Rx4 external control words output on the analog GPIOs
ADI_ADRV9025_ENABLE_RX1_RX2_RX3_RX4_EXT_CTRL_GPIOS	Enable Rx1, Rx2, Rx3 and Rx4 external control words output on the analog GPIOs

GENERAL-PURPOSE INTERRUPT (GPINT)

The transceiver features two general purpose interrupt (GPINT) pins, GPINT1 and GPINT2. Note that the device 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 GPINT1 and GPINT2 conventions on the device data sheet pinout except when listed in an API code example. A summary of API commands relevant to the GPINT functionality is provided in the API Commands for GPINT section.

The GPINTx 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 207. The GPINT2 pin acts as the high priority interrupt pin, and the GPINT1 pin acts as the low priority interrupt pin. These pins can be configured with independent bitmasks that control which signals can assert GPINT1 or GPINT2. A high level block diagram of the GPINT operation is shown in Figure 120.

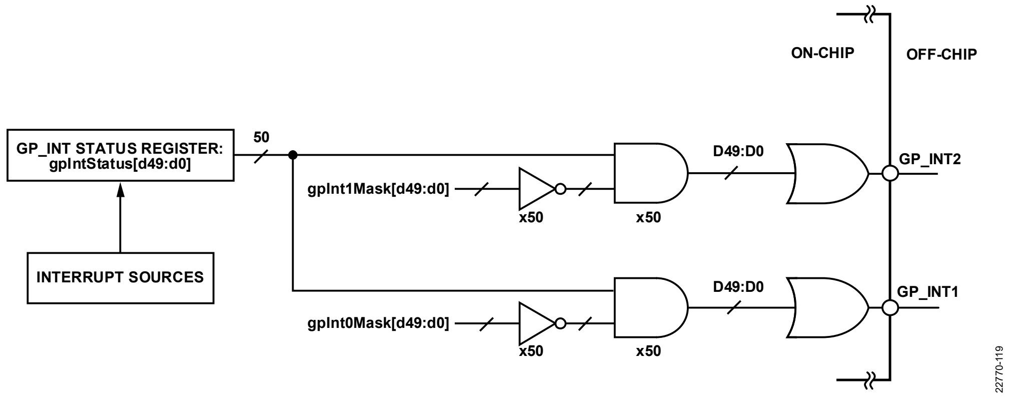


Figure 120. Block Diagram of GPINT Outputs

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 GPINTx pin asserts, the GPINT status indicates what interrupt source(s) asserted the GPINTx pin.

Note that the GPINT status and the GPINTx pins have different behaviors. The GPINTx pins are real-time indicators of error status. For example, if a power amplifier protection error occurs when power amplifier protection is configured in the autoclear mode, the GPINTx pin deasserts when the power returns to normal. The GPINT status bit fields are sticky and remain asserted until the user clears the register. If the power amplifier protection error occurs and disappears in autoclear mode, the GPINT status still indicates that a power amplifier 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 207.

Table 207. GP_INTERRUPT Bitmask Description

Bit Position	Description	Subsystem	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	Auxiliary PLL unlock		ADI_COMMON_ACT_ERR_RESET_MODULE
D44	Clock 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	Auxiliary PLL charge pump overrange		
D40	Clock PLL charge pump overrange		
D39	SERDES PLL unlock		ADI_COMMON_ACT_ERROR_RESET_FULL
D38	Deframer IRQ 9: Deframer1 JESD204B quad byte deframer (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		

Bit Position	Description	Subsystem	API Recovery Action
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		
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	Framer	
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	Power Amplifier Protection Error Tx4 (threshold exceeded)	Transmitter	ADI_ADRV9025_ACT_ERR_BBIC_LOG_ERROR
D18	Power Amplifier Protection Error Tx3 (threshold exceeded)		
D17	Power Amplifier Protection Error Tx2 (threshold exceeded)		
D16	Power Amplifier Protection Error Tx1 (threshold exceeded)		
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 or ORx4 stream processor error	Stream processor	ADI_COMMON_ACT_ERROR_RESET_FULL
D10	ORx1 or ORx2 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 r		
D0	Memory ECC error	ARM	ADI_COMMON_ACT_ERROR_RESET_FULL

Table 207 can be used to form bitmasks for the GPINT2 and GPINT1 pins. Note that in the API, GPINT1 is linked to the GPINT2 pin and GPINT0 is linked to the GPINT1 pin. Further descriptions 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 overrange events. Note that if initial calibrations are run, it is expected that some PLLs are used during this time and a PLL unlock event can appear in the GPINT status register. PLL unlocks during successful runs of initialization calibrations are expected and are 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 can unlock because of external or internal factors. There are two recovery procedures for PLL unlocks depending on the PLL that unlocks. These procedures include the following:

- If the clock PLL unlocks, reset the device. The device is not expected to recover from the loss of the primary clock within the transceiver.
- If the LO2, LO1, or auxiliary PLL unlocks, call adi_adrv9025_PllFrequencySet(…) to see if the PLL relocks.
    - If the unlocked PLL relocks, follow the procedures to rerun 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 the 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, reset the device.

The real time lock status of the PLL can be verified with the command adi_adrv9025_PllStatusGet(…).

Charge Pump Overrange Event Bits

The charge pump overrange event bits must not be unmasked for the GPINTx pins. These bits can assert intermittently but do not indicate a significant device issue.

JESD204B AND JESD204C GPINT SOURCES

The deframer and framer in both JESD204B and JESD204C modes of operation can send information to the user regarding error events over the GPINTx pin.

Because of 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 GPINTx pins when configured in JESD204B mode because there is no value provided in this configuration.

Table 208 provides additional details regarding the deframer and framer interrupts that can assert the GPINTx pin. In general, referring to JESD204B/JESD204C documentation explains these events in more detail as well as possible recovery mechanisms.

Table 208. Framer and Deframer Interrupt List

GP_INT Bits	Brief Description	Technical Description	Further Actions, If Necessary
D34, D49	Deframer JESD204 CRC error	A CRC error has been detected on one of the active deframer lanes, the transmit data is possibly corrupted.	Log the event. The user must decide how to react to the event.
D33, D48	Deframer JESD204C loss of sync	The JESD204C link layer has lost sync. This can be because of a loss of sync header alignment or multiblock alignment. Typically, the link has dropped and must be reestablished.	Log the event. If the link is down, reestablish the link.
D32, D38	Deframer JESD204B QBD IRQ	The QBD interrupt request (IRQ) indicates that a deframer IRQ source has asserted. Deframer IRQ sources include bad disparity (BD), not in table (NIT), and unexpected K (UEK). Most errors are considered minor.	Log the event. Call adi_adrv9025_DfrmIrqSourceGet(…) to retrieve the specific interrupt that asserted. Typically, this is an informational interrupt, but some cases can require the link reset.
D31, D37	Deframer SYSREF out of phase	SYSREF registered at the wrong phase in the link.	Log the event. Something is likely incorrect in the overall system timing and must be adjusted.
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 the event. Reassess the lmfcOffset value selection if deterministic latency is required.
D29, D35	Deframer lane FIFO pointer error	Lane FIFO pointers have moved in the link. This error may or may not be associated with $\overline{\text{SYNC}}$ going low.	Log the event. Reset the link.
D22, D25, and D28	Framer transport layer not sending data	The framer is not sending user data. This error occurs if the LMFC from the link layer is out of phase with the transport layer LMFC, and forces a relink by taking $\overline{\text{SYNC}}$ low.	Log the event.
D21, D24, and D27	Framer SYSREF out of phase	SYSREF is registered at the wrong phase in the framer link. If JESD is configured to attempt relink with the new phase, no action is required.	Log the event. Something is likely incorrect in the overall system timing and must be adjusted.
D20, D23, and D26	Framer lane FIFO pointer error	The lane FIFO pointer has changed.	Log the event.

These deframer interrupts can be used to assert the rampdown of transmit attenuation as described in the Transmitter Power Amplifier Protection section.

POWER AMPLIFIER PROTECTION GPINT SOURCES

The power amplifier protection feature must be enabled for these interrupts to assert. The power amplifier protection block refers specifically to the peak and average power measurement capabilities within the transmit data path and must not be misconstrued for the general transmit attenuation ramp features.

Power amplifier protection GPINT sources indicate to the user that a peak or average power measurement within the transmit 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 power amplifier protection error. Log this event and take appropriate action within the system to resolve the reason for the power increase in the transmit data path.

The user can configure the power amplifier protection block to enforce a ramp (or increase) of transmit attenuation with the adi_adrv9025_PaPllDfrmEventRampDownEnableSet(…) command. Control over whether the attenuation ramp is sticky or autoclears is determined by the 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 when a fatal error occurs within the firmware. If possible, acquire an ARM memory dump to assist in a 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 is unable to reset this timer, a fatal error occurs within the ARM. If possible, acquire an ARM memory dump to assist in a debug. Reset the device.

Slew Rate Limiter IRQ

As of SW 2.0.5 versions, this bit represents the SRL error interrupt for the transmit datapaths. If this interrupt asserts, 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 can 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 indicates 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. Reset the device if this error is detected.

SOFTWARE PROCEDURES FOR GPINT

Referring to the transceiver programming sequence in adi_adrv9025_daughter_board.c, the GPINT feature setup is one of the last steps in device initialization and occurs after both the adi_board_adrv9025_JesdBringup(…) and adi_adrv9025_TxRampDownInit(…) commands are issued. The GPINT masks for the GPINT2 and GPINT1 pins are stored in the adi_adrv9025_GpInterruptSettings_t structure and applied to the device during adi_adrv9025_GpIntInit(…). This command configures both GPINTx pins and no further action is needed for setup.

If it is necessary to reconfigure the GPINT masks after initialization, use the adi_adrv9025_GpIntMaskSet(…) command. The primary difference between the two GPINT setup commands is that the adi_adrv9025_GpIntMaskSet(…) command 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 can require user intervention to resolve. The GPINT handler functions attempts 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 occurred since the status was last cleared.

The general setup and usage for the GPINT command is as follows:

1. Initialize the device using either the call adi_adrv9025_GpIntInit(…) or adi_adrv9025_GpIntMaskSet(…) command to set up the GPINT feature.
2. Operate the device. The baseband processor monitors the GPINT2 pin and/or GPINT1 pin for rising edges that indicate an interrupt occurred.

3. If the GPINT2 pin and/or GPINT1 pin asserts, call their associated interrupt handler API command, either the adi_adrv9025_GpInt1Handler(…) or adi_adrv9025_GpInt0Handler(…) command, respectively. The interrupt handler returns information related to the interrupt source to the user. Calling this command can be sufficient to clearing the error. Either handler function returns a recovery action which suggests further action if necessary.
4. Alternatively, the user can call the adi_adrv9025_GpIntStatusGet(…) command, 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.
5. If the device does not need to be reset and the error state has been eliminated, it is necessary to call the adi_adrv9025_GPIntClearStatusRegister(…) command to clear all error bits asserted in the GPINT status register.
6. 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

This command applies the desired bitmasks to the device.

Parameters

Table 209. adi_adrv9025_GpIntMaskSet(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
maskSelect	Sets enumeration indicating which GP_INTERRUPT bitmask (GPINT1 or GPINT0) to write.
*maskArray	Pointer to the data structure holding the GP_INTERRUPT bitmasks to write.

Table 210 describes the adi_adrv9025_gpMaskSelect_e enumeration. This parameter describes which pin to write the mask to.

Table 210. Description of adi_adrv9025_gpMaskSelect_e Enumeration

Enumeration	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 211 describes the adi_adrv9025_gp_MaskArray_t data structure. Refer to Table 207 for a description of the bitmasks.

Table 211. Description of adi_adrv9025_gp_MaskArray_t Data Structure

Data Type	Parameter	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 GPINT1 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 GPINT2 pin.

When either GPINTx 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

This command sets up the GPINT2 interrupt handler.

Parameters

Table 212. adi_adrv9025_GpInt1Handler(…)

Parameter	Description
*device	Pointer to the device structure.
*gpInt1Status	Pointer to the status readback word that contains the GPINT2 source registers.

adi_adrv9025_GpInt0Handler(…)

```
adi_adrv9025_GpInt0Handler(adi_adrv9025_Device_t* device, adi_adrv9025_gpIntStatus_t
*gpInt0Status)
```

Description

This command sets up the GPINT1 interrupt handler.

Parameters

Table 213. adi_adrv9025_GpInt0Handler(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
*gpInt0Status	Pointer to the status readback word that contains 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 the GPINT2 pin or GPINT1 pin while the handler is invoked. This masking is followed by retrieval of the GPINT status. The final step in the handler is to restore the initial bitmask for the GPINT2 pin and GPINT1 pin. In some cases, reading the error is sufficient to clearing the error, which is the case for short-term, intermittent errors. If the error persists, 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

This command provides a direct readback of the GPINT status word.

Parameters

Table 214. adi_adrv9025_GpIntStatus(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
*gpIntStatus	Pointer to the status readback word. Refer to Table 207 for bitmask descriptions.

adi_adrv9025_GPIntClearStatusRegister(…)

```
adi_adrv9025_GPIntClearStatusRegister(adi_adrv9025_Device_t *device, uint64_t *gpIntStatus)
```

Description

This command clears the GPINT status register.

Parameters

Table 215. adi_adrv9025_GPIntClearStatusRegister(…) Parameters

Parameter	Description
*device	Pointer to the device structure.
*gpIntStatus	Pointer to the status readback word. Refer to Table 207 for bitmask descriptions.

AUXILIARY CONVERTERS AND TEMPERATURE SENSOR

The transceiver features auxiliary data converters including eight 12-bit auxiliary digital-to-analog converters (AUXDACs) and two 12-bit auxiliary analog-to-digital converters (AUXADCs). 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 the API command for configuration and control.

The AUXDAC and AUXADC are not precision data converters. DC offset and gain/slope errors are present and can vary on different channels. Refer to the specifications in ADRV9029 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 216.

Table 216. AUXDAC Pin Mapping and adi_adrv9025_AuxDacs_e Enumeration Description

Auxiliary DAC Number	Pin Name	Pin Number	Enumeration Name	Enumeration Value
AUXDAC[0]	GPIO_ANA_0	C13	ADI_ADRV9025_AUXDAC0	0x01
AUXDAC[1]	GPIO_ANA_1	C12	ADI_ADRV9025_AUXDAC1	0x02
AUXDAC[2]	GPIO_ANA_2	L16	ADI_ADRV9025_AUXDAC2	0x04
AUXDAC[3]	GPIO_ANA_3	L17	ADI_ADRV9025_AUXDAC3	0x08
AUXDAC[4]	GPIO_ANA_4	L2	ADI_ADRV9025_AUXDAC4	0x10
AUXDAC[5]	GPIO_ANA_5	L1	ADI_ADRV9025_AUXDAC5	0x20
AUXDAC[6]	GPIO_ANA_6	C5	ADI_ADRV9025_AUXDAC6	0x40
AUXDAC[7]	GPIO_ANA_7	C4	ADI_ADRV9025_AUXDAC7	0x80

The capacitive load of the AUXDAC pins must not exceed more than 100 pF. Otherwise, stability issues can occur.

The AUXDAC uses the GPIO_ANA pins on the device. Conflicts between GPIO_ANA and AUXDAC functionality can occur. In case of these conflicts, the AUXDAC takes precedence over all other GPIO_ANA functionality when the 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 that require 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

V_{AUXDAC} is the output voltage.

AuxDacValue is the 12-bit digital code applied to the AUXDAC.

The AUXDAC is not a precision converter and is best used in feedback systems. Figure 121 shows the AUXDAC output voltage vs. the input codes for a full range code sweep of the AUXDAC. Channel to channel variability in slope and dc offset are expected.

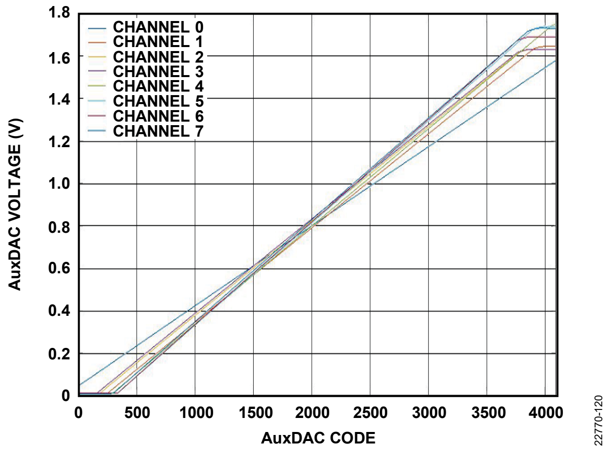


Figure 121. AUXDAC Channel Comparison over Full Range Code Sweep

AUXDAC Configuration

The AUXDAC is configured and controlled using the commands listed in this section.

adi_adrv9025_AuxDacCfgSet(…)

```
adi_adrv9025_AuxDacCfgSet(adi_adrv9025_Device_t *device, adi_adrv9025_AuxDacCfg_t
auxDacConfig[],uint8_t numberOfCfg)
```

Description

This command configures the AUXDAC settings. This command must be called when device initialization is complete to use the AUXDACs.

Clears the GPINT status register.

Parameters

Table 217. adi_adrv9025_AuxDacCfgSet(…) Parameters

Parameter	Description
*device	The 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 218.

Table 218. 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.

adi_adrv9025_AuxDacValueSet(…)

```
adi_adrv9025_AuxDacValueSet(adi_adrv9025_Device_t* device,adi_adrv9025_AuxDacValue_t
auxDacValues[], uint8_t numberOfCfg)
```

Description

This command sets the output value of one or more AUXDAC outputs.

Parameters

Table 219. adi_adrv9025_AuxDacValueSet(…) Parameters

Parameter	Description
*device	The 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 220.

Table 220. 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 the AUXDACs selected by auxDacMask

AUXILIARY ADC (AUXADC)

There are two physical AUXADCs integrated on the device. Each AUXADC has two inputs: AUXADC_0 and AUXADC_1 for the first converter, AUXADC_2 and AUXADC_3 for the second converter. The different AUXADCs are designated as ADI_ADRV9025_AUXADC_A and ADI_ADRV9025_AUXADC_B, per the adi_adrv9025_AuxAdcSelect_e enumeration.

The AUXADC is a 12-bit Δ-Σ converter and is most useful for relative voltage measurements rather than precision measurements because of 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 the VCONV1_1P0 pin for AUXADC_A and to the VCONV2_1P0 pin 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 122).

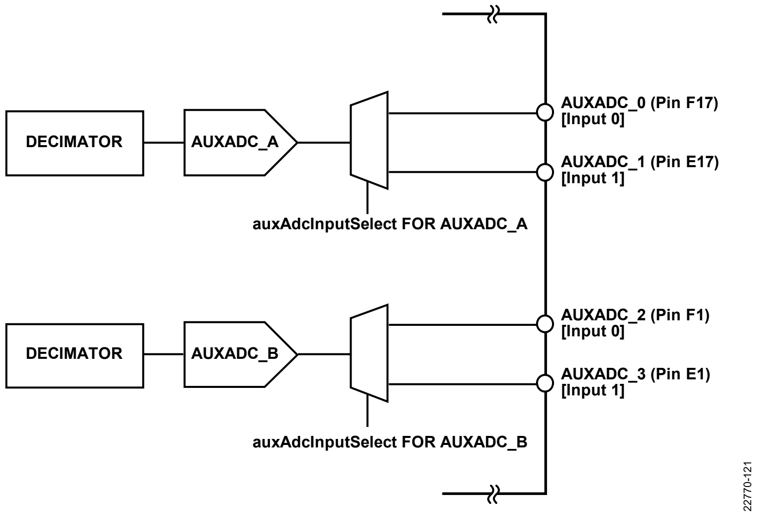

DECIMATOR
AUXADC_A
AUXADC_0 (Pin F17) [Input 0]
AUXADC_1 (Pin E17) [Input 1]
auxAdcInputSelect FOR AUXADC_A
DECIMATOR
AUXADC_B
AUXADC_2 (Pin F1) [Input 0]
AUXADC_3 (Pin E1) [Input 1]
auxAdcInputSelect FOR AUXADC_B
22770-121

Figure 122. 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.

$$D_{OUT} = 4096(V_{IN} - 0.5\text{ V}) + 2048$$

Where D_{OUT} is the output of the AUXADC for the given input voltage V_{IN}.

AUXADC Configuration

The following commands are used to configure and read the AUXADCs.

adi_adrv9025_AuxAdcCfgSet(…)

```
adi_adrv9025_AuxAdcCfgSet(adi_adrv9025_Device_t *device, adi_adrv9025_AuxAdcCfg_t *auxAdcConfig,
uint8_t arraySize)
```

Description

This command configures the AUXADC setup.

Parameters

Table 221. adi_adrv9025_AuxAdcCfgSet(…) Parameters

Parameter	Description
*device	Pointer to the device settings structure
*auxAdcConfig	Pointer to the supplied ADC configuration structure(s)
arraySize	Number of supplied configuration structures

An important data structure used in this command is adi_adrv9025_AuxAdcCfg_t. Table 222 describes the parameters used in this structure.

Table 222. 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 Table 223 through Table 226.

Table 223. Description of adi_adrv9025_AuxAdcEnable_e Enumeration

Enumeration Name	Enumeration Value	Comments
ADI_ADRV9025_AUXADC_DISABLE	0	Aux ADC disabled
ADI_ADRV9025_AUXADC_ENABLE	1	Aux ADC enabled

Table 224 provides the enumerations that describe the two physical converters on the device.

Table 224. Description of adi_adrv9025_AuxAdcSelect_e Enumeration

Enumeration Name	Enumeration Value	Comments
ADI_ADRV9025_AUXADC_A	0	Aux ADC A selection
ADI_ADRV9025_AUXADC_B	1	Aux ADC B selection

Table 225 provides the enumerations that describe the two input selections that can be applied to each converter.

Table 225. Description of adi_adrv9025_AuxAdcInputSelect_e Enumeration

Enumeration Name	Enumeration 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 226. Select the AUXADC divider setting such that the sampling clock frequency is set as low as possible without resulting in aliasing.

Table 226. Description of adi_adrv9025_AuxAdcClkDivide_e Enumeration

Enumeration Name	Enumeration 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

adi_adrv9025_AuxAdcValueGet(…)

```
adi_adrv9025_AuxAdcValueGet(adi_adrv9025_Device_t *device, adi_adrv9025_AuxAdcSelect_e
auxAdcSelect, adi_adrv9025_AuxAdcValue_t *auxAdcValue)
```

Description

This command retrieves the AUXADC readback value when the AUXADC is configured.

Parameters

Table 227. adi_adrv9025_AuxAdcValueGet(…) Parameters

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 previous command is the adi_adrv9025_AuxAdcValue_t. Table 228 describes the members within this data structure.

Table 228. 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, the temperature sensor converter 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 result in °C through an API command. The API command to readback the temperature sensor measurement is listed in this section.

adi_adrv9025_TemperatureGet(…)

```
adi_adrv9025_TemperatureGet(adi_adrv9025_Device_t *device, int16_t *temperatureDegC)
```

Description

This command returns the temperature sensor ADC output value.

Parameters

Table 229. adi_adrv9025_TemperatureGet(…) Parameters

Parameter	Description
*device	The pointer to the device settings structure
*temperatureDegC	The pointer to a single int16_t element that returns the current 12-bit temperature sensor in °C

SPI2 DESCRIPTION

The transceiver uses the primary SPI port for nearly all SPI transactions needed during operation. The device also features a secondary SPI port, SPI2, that can be used to control transmit, receive, and observation receive 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 230 are configured automatically to the correct input/output port direction to support the SPI Interface.

Table 230. 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, which bypasses the need to access the main SPI bus. The user can program four 10-bit attenuation words into registers designated as 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 GPIO_4 to GPIO_18. The GPIO selection is performed by calling the following API.

adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(…)

```
adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(adi_adrv9025_Device_t *device,
adi_adrv9025_TxAttenSpi2PinCfg_t txAttenSpi2PinCfg[], uint8_t numTxAttenSpi2PinConfigs)
```

Description

This command selects the GPIO pins used to implement the SPI2 interface.

Parameters

Table 231. adi_adrv9025_TxAttenSpi2PinCtrlCfgSet(…) Parameters

Parameter	Description
*device	Pointer to the device settings structure
txAttenSpi2PinCfg[]	An array of structures of type adi_adrv9025_TxAttenSpi2PinCfg_t detailed in Table 232
numTxAttenSpi2PinConfigs	The number of configurations passed in the array

Table 232. SPI2 Configuration Parameters

Parameter	Comments	
txChannelMask	This parameter selects the channels upon which the API acts, and is a bit mask with each bit corresponding to a channel. The desired mask can be generated by OR'ing the desired channel enumerations as follows (data type: uint32_t):	
	adi_adrv9025_TxChannels_e	**Transmit Channel**
	ADI_ADRV9025_TXOFF	No transmit 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.

Parameter	Comments	
txAttenSpi2Pin	This parameter selects which GPIO pin is used to select between transmit 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, and removes the previously selected GPIO from the list of used resources.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO4	Select GPIO 4 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO5	Select GPIO 5 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO6	Select GPIO 6 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO7	Select GPIO 7 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO8	Select GPIO 8 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO9	Select GPIO 9 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO10	Select GPIO 10 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO11	Select GPIO 11 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO12	Select GPIO 12 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO13	Select GPIO 13 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO14	Select GPIO 14 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO_15	Select GPIO 15 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO16	Select GPIO 16 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO17	Select GPIO 17 for transmit state selection.
	ADI_ADRV9025_SPI2_TXATTEN_GPIO18	Select GPIO 18 for transmit state selection.

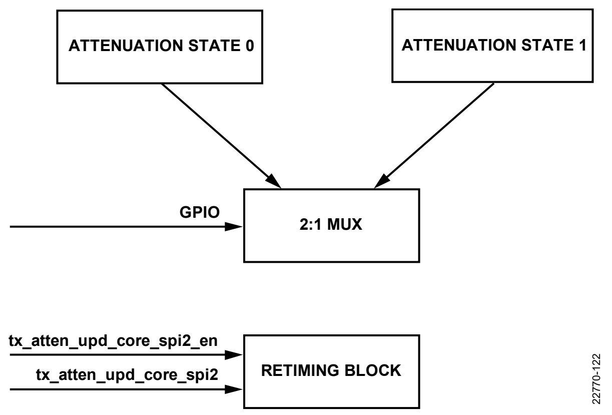


Figure 123. SPI2 Transmitter Attenuation Update Options

There are two update modes selectable for updating the attenuation applied to the transmitters, selected by Bit D0 in the SPI2 Register 0x2A. When this bit is 0, it updates to the attenuation state registers or the multiplexer select GPIO take immediate effect. When this bit 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 the SPI2 Register 0x2A, Bits[D4:D1]. Note that these bits are not self-clearing and must be written to zero before being used to latch new attenuation values.

Table 233. SPI2 Register 0x2A Details

Register 0x2A	Comments
D4	Latch this bit for Tx3 attenuation words (not self-clearing).
D3	Latch this bit for Tx2 attenuation words (not self-clearing).
D2	Latch this bit for Tx1 attenuation words (not self-clearing).
D1	Latch this bit for Tx0 attenuation words (not self-clearing).
D0	Attenuation update mode selection bit. 0 = update the attenuation when the LSB is written. 1 = update the attenuation when the latch bit is set and transitioned from low to high.

It is generally preferred to synchronize the attenuation change of all the transmit channels in one device, or across an antenna array comprising many devices. An example of how to perform this synchronization is given in the following steps:

1. Set Register 0x2A, Bit D0 low to allow immediate attenuation updates.
2. Update the attenuation values of the attenuation state that is not in use.
3. Toggle the selected attenuation state by toggling the appropriate 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 repeats, the transmit 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 234. 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 234, 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 234. 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]							

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 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 transceiver performance. This section includes data regarding the expected RF port impedance values and examples of impedance matching networks used in the evaluation platform. Some information is also provided regarding board layout techniques and balun selection guidelines.

External impedance matching networks are required on the transmitter and receiver ports to achieve the performance levels indicated in the device data sheet. Analog Devices recommends the utilization of simulation tools in the design and optimization of impedance matching networks. To achieve optimal correlation from simulation to the PCB, accurate models of the board environment, surface mounted device (SMD) components (for example, baluns and filters), and transceiver port impedances are required.

RF PORT IMPEDANCE DATA

Figure 124 through Figure 129 provide the port impedance data for all transmitters and receivers in the device obtained using ADS simulation tool modeling. Note the following:

- SEDZ stands for series equivalent differential impedance.
- PEDZ stands for parallel equivalent differential impedance.
- Z_o is defined as 50 Ω for transmit and as 100 Ω for receive/observation receive.
- 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.

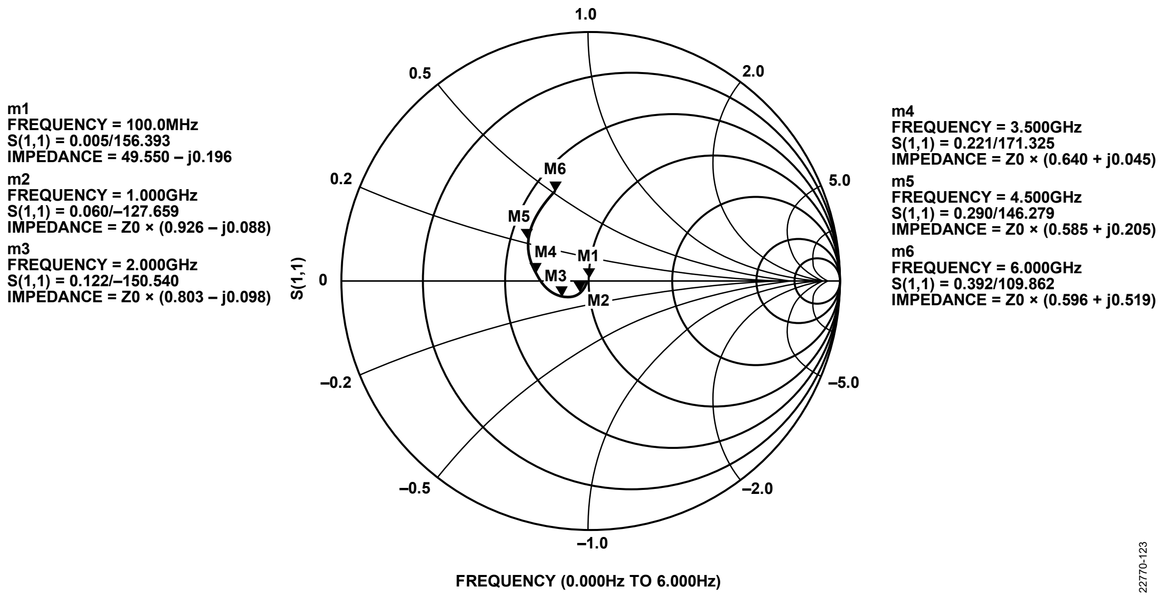


Figure 124. Tx1 and Tx4 SEDZ and PEDZ Data

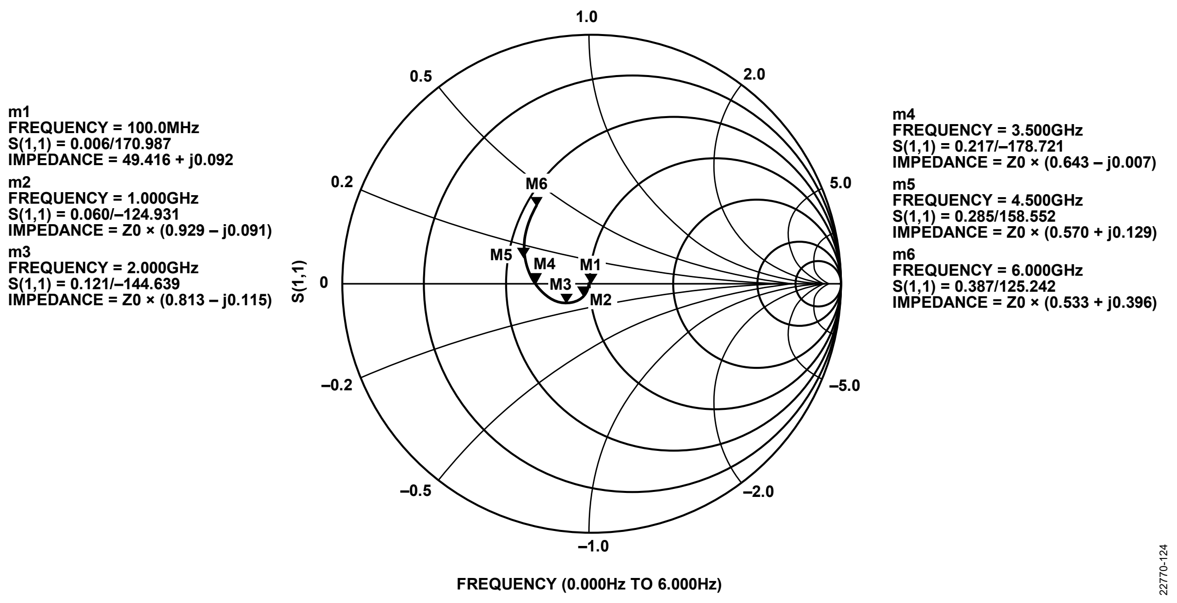

m1
FREQUENCY = 100.0MHz
S(1,1) = 0.006/170.987
IMPEDANCE = 49.416 + j0.092
m2
FREQUENCY = 1.000GHz
S(1,1) = 0.060/–124.931
IMPEDANCE = Z0 × (0.929 – j0.091)
m3
FREQUENCY = 2.000GHz
S(1,1) = 0.121/–144.639
IMPEDANCE = Z0 × (0.813 – j0.115)
m4
FREQUENCY = 3.500GHz
S(1,1) = 0.217/–178.721
IMPEDANCE = Z0 × (0.643 – j0.007)
m5
FREQUENCY = 4.500GHz
S(1,1) = 0.285/158.552
IMPEDANCE = Z0 × (0.570 + j0.129)
m6
FREQUENCY = 6.000GHz
S(1,1) = 0.387/125.242
IMPEDANCE = Z0 × (0.533 + j0.396)

Figure 125. Tx2 and Tx3 SEDZ Data

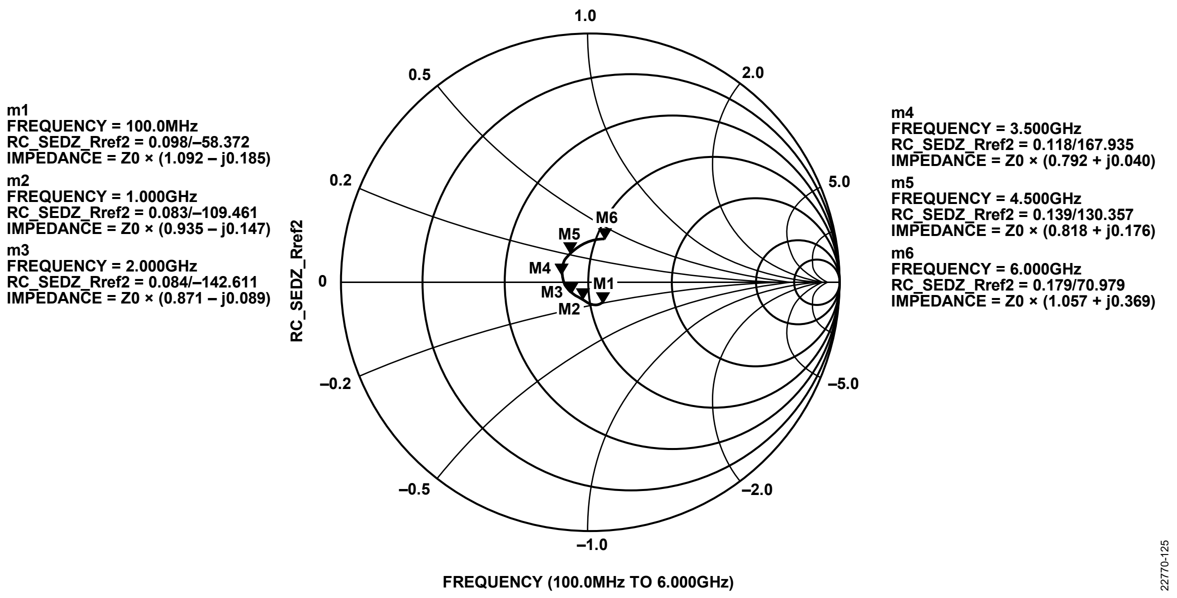

m1
FREQUENCY = 100.0MHz
RC_SEDZ_Rref2 = 0.098/–58.372
IMPEDANCE = Z0 × (1.092 – j0.185)
m2
FREQUENCY = 1.000GHz
RC_SEDZ_Rref2 = 0.083/–109.461
IMPEDANCE = Z0 × (0.935 – j0.147)
m3
FREQUENCY = 2.000GHz
RC_SEDZ_Rref2 = 0.084/–142.611
IMPEDANCE = Z0 × (0.871 – j0.089)
m4
FREQUENCY = 3.500GHz
RC_SEDZ_Rref2 = 0.118/167.935
IMPEDANCE = Z0 × (0.792 + j0.040)
m5
FREQUENCY = 4.500GHz
RC_SEDZ_Rref2 = 0.139/130.357
IMPEDANCE = Z0 × (0.818 + j0.176)
m6
FREQUENCY = 6.000GHz
RC_SEDZ_Rref2 = 0.179/70.979
IMPEDANCE = Z0 × (1.057 + j0.369)

Figure 126. Rx1 and Rx4 SEDZ Data

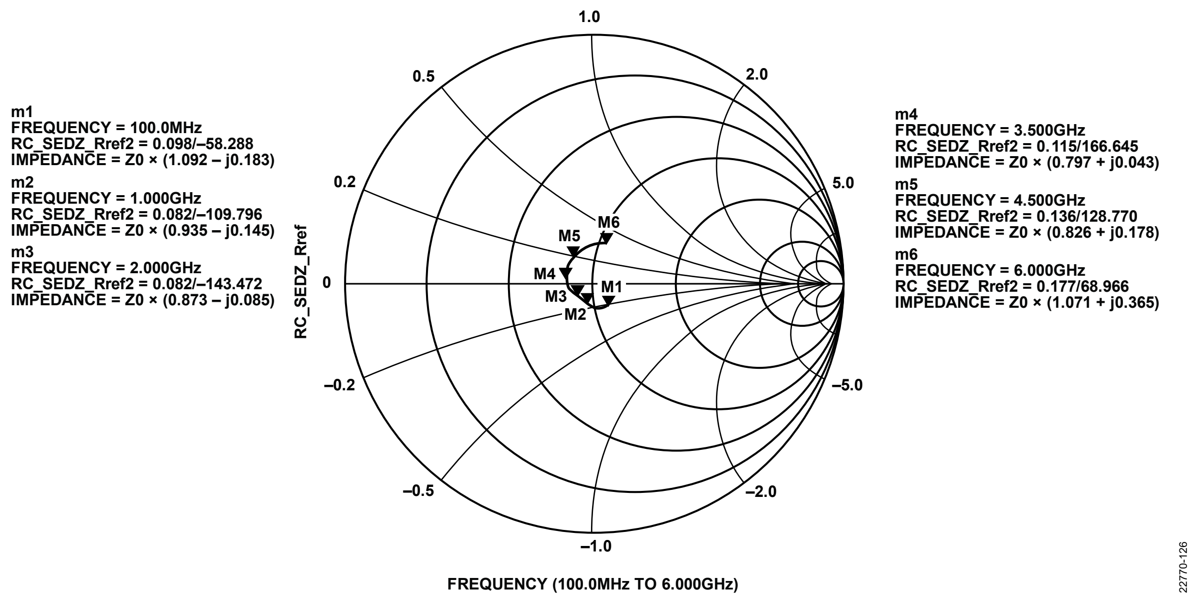


Figure 127. Rx2 and Rx3 SEDZ Data

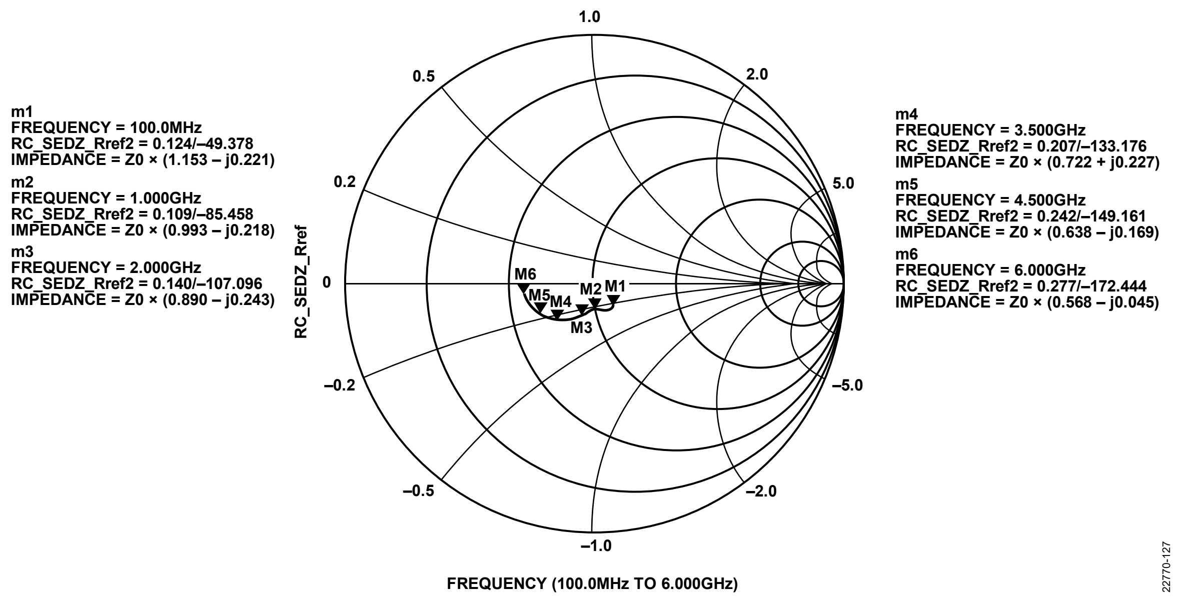


Figure 128. ORx1 and ORx4 SEDZ Data

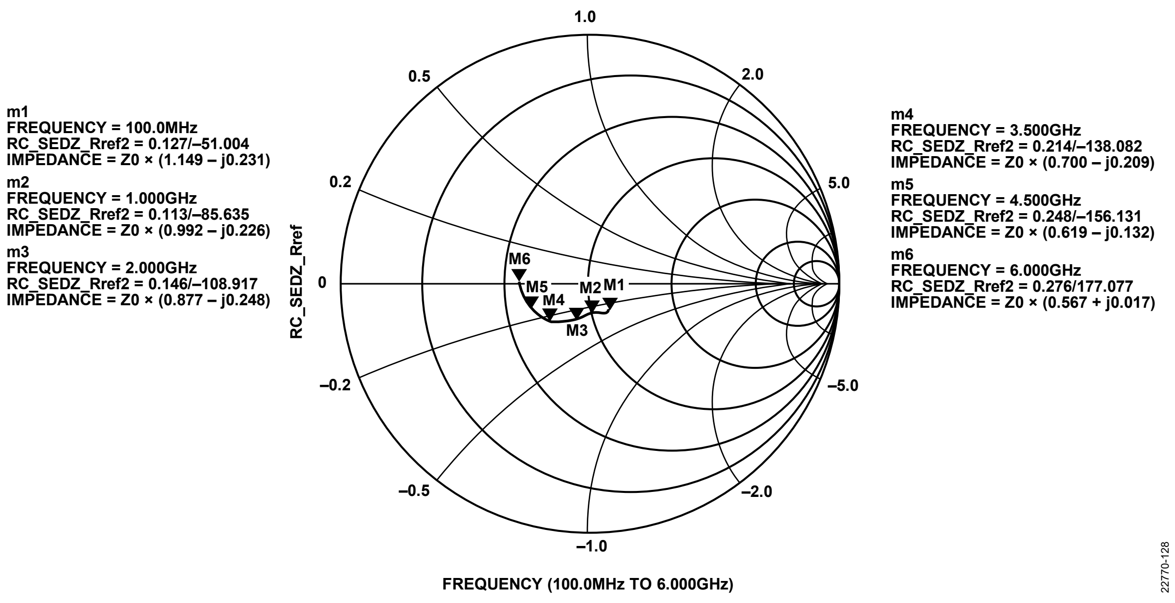


Figure 129. ORx2 and ORx3 SEDZ Data

ADS SETUP USING DATA ACCESS COMPONENT AND SEDZ FILE

The port impedances are supplied as a SEDZ file (file with a *.s1p extension. This format allows simple interface to ADS by using the data access component. In Figure 130, 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 the network can step the impedance up or down as needed with appropriate component selection.

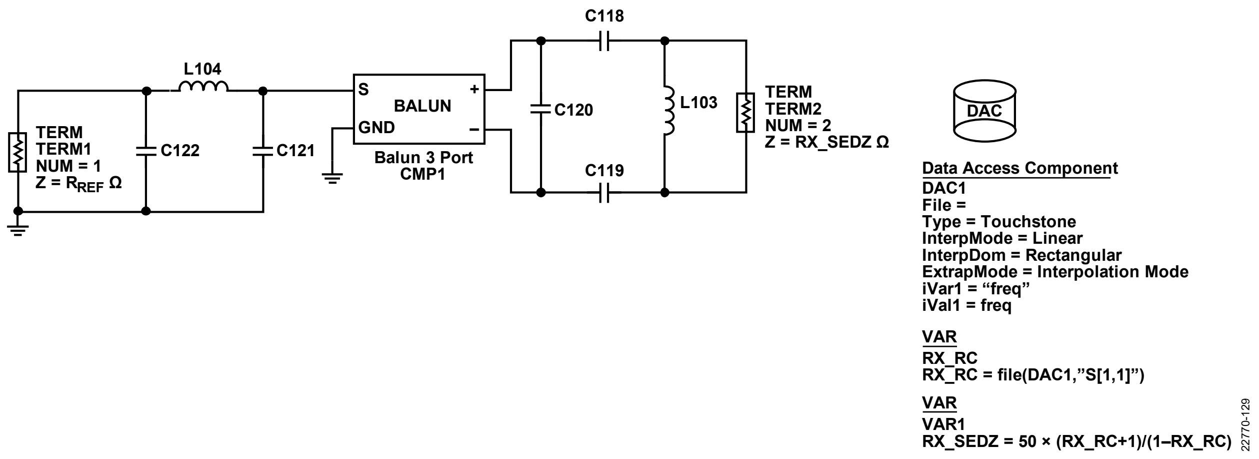


Figure 130. Simulation Setup in ADS with SEDZ s1p Files and DAC Component

This evaluation operation is as follows:

1. The data access component block reads the RF port *.s1p file, which is the device RF port reflection coefficient.
2. The two equations convert the RF port reflection coefficient to a complex impedance. The end result is the RF port calculated complex impedance (RX_SEDZ) variable.
3. The 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 the highest accuracy, use electromagnetic (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 explains the dc biasing of the transmitter outputs and how to interface to each transmit 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 transmit 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 131. 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 132. This network has fewer parasitics and fewer total components. Note that C_B signifies the bias supply decoupling capacitor, I_{BIAS} is the bias current to each transmitter output stage port, and V_{BIAS} is the bias voltage seen by each transmitter output stage port.

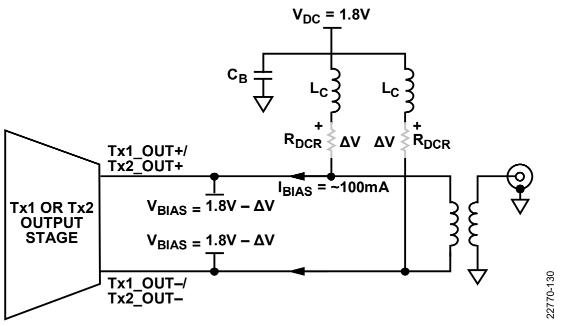


Figure 131. RF DC Bias Configurations Depicting Parasitic Losses Because of Wire Wound Chokes

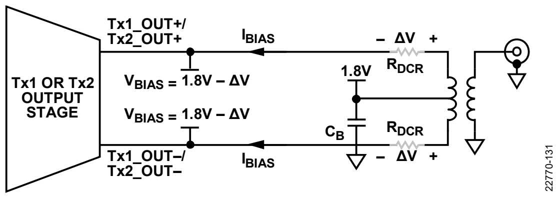


Figure 132. RF DC Bias Configurations Depicting Parasitic Losses Because of Center Tapped Transformers

Figure 133 through Figure 136 identify four basic differential transmitter output configurations. Impedance matching networks (balun single-ended port) are most likely required to achieve optimum device performance. The transmitter outputs must also be ac-coupled in most applications because of 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 133. This configuration offers the lowest component count of the options presented.

Descriptions of the transmit port interface schemes are as follows:

- The center tapped transformer passes the bias voltage directly to the transmitter outputs.
- The 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.
- The RF chokes are used to bias the differential transmitter output lines and connect into a transformer.
- The RF chokes are used to bias the differential output lines that are ac-coupled into the input of a driver amplifier.

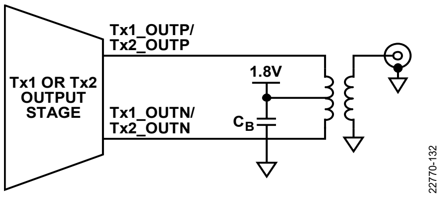


Figure 133. RF Transmitter Interface Configuration A

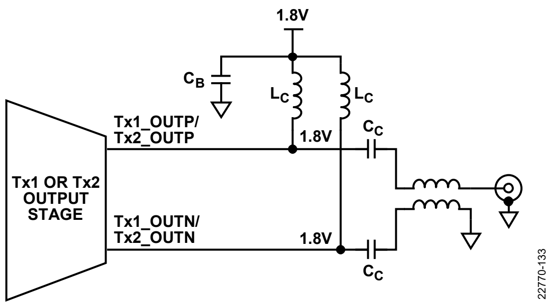


Figure 134. RF Transmitter Interface Configuration B

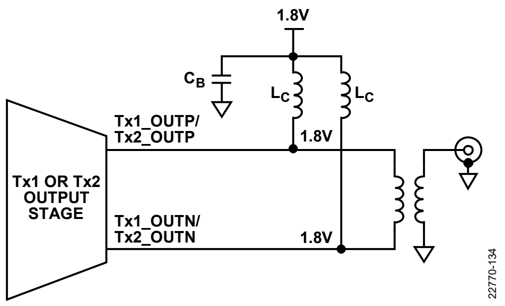


Figure 135. *RF Transmitter Interface Configuration C*

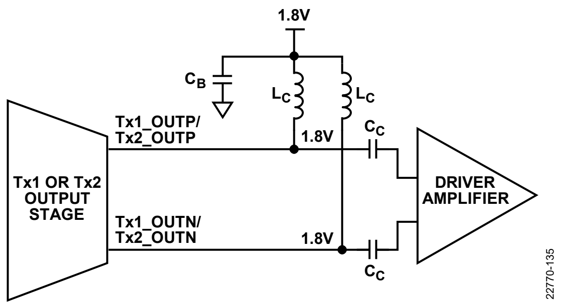


Figure 136. RF Transmitter Interface Configuration D

If a chosen transmit balun requires a set of external dc bias choke inductors, 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, the resistance decreases as the size increases. However, as the choke inductance increases, the resistance increases. Therefore, it is not recommended to use physically small chokes with high inductance as these chokes exhibit the greatest resistance. Table 235 lists some sample choke inductor resistances for different package sizes to help determine how much dc loss to expect. For example, the voltage drop of a 500 nH, 0603 choke inductor at 100 mA is roughly 45 mV.

Table 235. Sample Wire Wound DC Bias Choke Resistance vs. Size

Inductance (nH)	0603 Package Size Resistance (Ω)	01206 Package Size Resistance (Ω)
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 receive and observation receive channels are designed for differential use only.

The receivers support a wide range of operation frequencies. In the case of the receive and observation receive 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 include the following:

- What device will be interfaced to the transceiver? Examples are filter, balun, transmit/receive switch, external LNA, and external power amplifier. Determine if this device represents a short to ground at dc.
- Receive and observation receive maximum safe input power is 18 dBm (peak).
- Receive and observation receive optimum dc bias voltage is 0.7 V bias to ground.
- Board Design characteristics, including reference planes, transmission lines, and impedance matching must be carefully planned.

Figure 137 shows the possible differential receiver port interface circuits. The options in Figure 137 and Figure 138 are valid for all receiver inputs operating in differential mode, but only the Rx1 signal names are indicated. Impedance matching can be necessary to obtain the performance levels in the ADRV9029 data sheet.

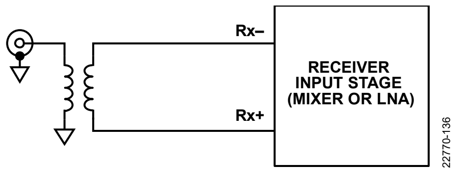


Figure 137. Differential Receiver Input Interface Circuits

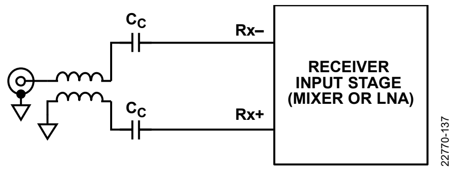


Figure 138. Differential Receiver Input Interface Circuits

Given wide RF bandwidth applications, SMD balun devices function well. Decent loss and differential balance are available in a relatively small (0603 or 0805) package.

IMPEDANCE MATCHING NETWORK EXAMPLES

Impedance matching networks are required to achieve performance levels noted in the ADRV9029 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 can be obtained from an EM simulator (for example, Momentum). The balun and SMD component models can typically be obtained from the device vendors. Figure 139 shows a typical matching circuit topology used to connect single-ended signal sources to the transceiver's differential inputs.

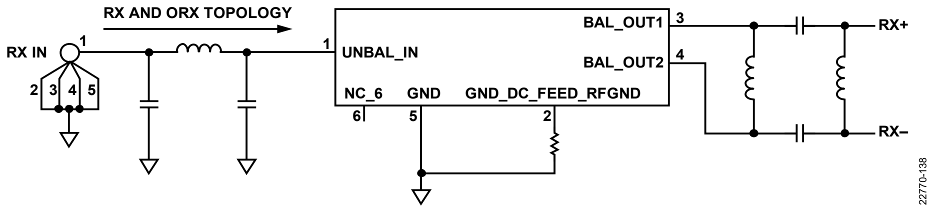


Figure 139. Impedance Matching Topology

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 140, Figure 141, and Figure 142 show two or three circuit elements in parallel marked do not include (DNI), which 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, the 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 can be a resistor, inductor, or capacitor, and the shunt elements can be either an inductor or a capacitor. Only one component of each parallel combination is placed in a practical application. Note 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.

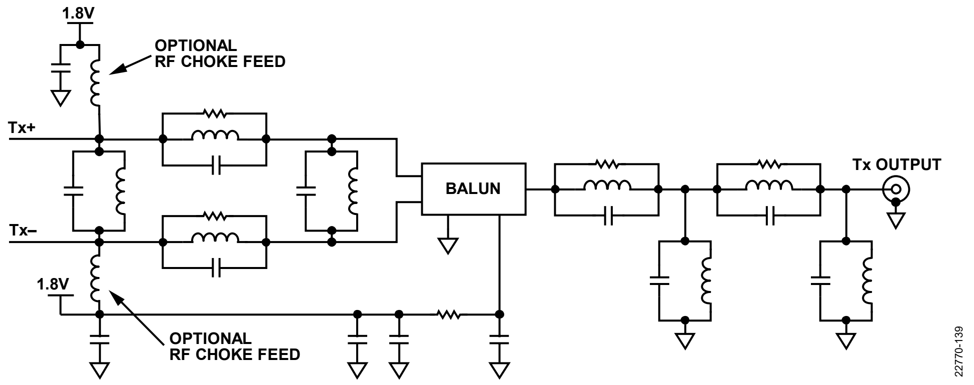


Figure 140. Transmitter Generic Matching Network Topology from CE Board

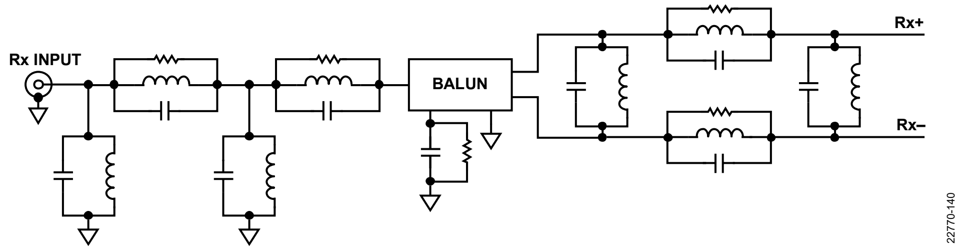


Figure 141. Receiver Generic Matching Network Topology from CE Board

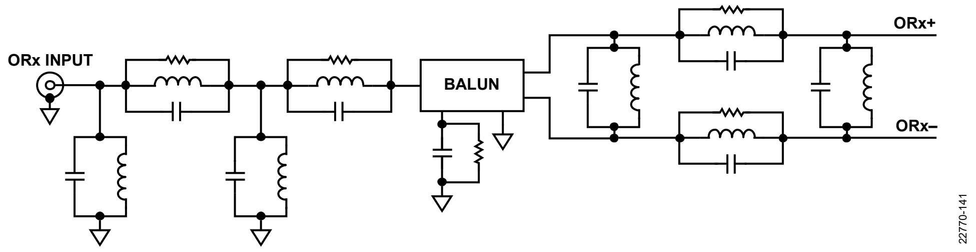


Figure 142. Observation Receiver Generic Matching Network Topology from CE Board

MATCHING COMPONENT RECOMMENDATIONS

Table 236 through Table 241 show the balun and matching components used on the CE boards. Leave the DNI components open. Note that all tolerances are ±3% unless otherwise noted. 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 different variant CE boards.

Table 236. Receiver Matching Components, Rx1 and Rx4

	Component Location on PCB (All Tolerances Are ±3% Unless Otherwise 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**
75 to 1000	DNI	0 Ω	DNI	0Ω	0.01 µF	DNI	0 Ω	0 Ω	DNI	TDK ATB2012-50011
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 237. Receiver Matching Components, Rx2 and Rx3

	Component Location on PCB (All Tolerances Are ±3% Unless Otherwise 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
75 to 1000	DNI	0 Ω	DNI	0Ω	0.01 µF	DNI	0 Ω	0 Ω	DNI	TDK ATB2012-50011
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 238. Observation Receiver Matching Components, ORx2 and ORx4

	Component Location on PCB (All Tolerances Are ±3% Unless Otherwise 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
75 to 1000	DNI	0 Ω	DNI	0Ω	0.01 µF	DNI	0 Ω	0 Ω	DNI	TDK ATB2012-50011
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 239. Observation Receiver Matching Components, ORx1 and ORx3

	Component Location on PCB (All Tolerances Are ±3% Unless Otherwise 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
75 to 1000	DNI	0 Ω	DNI	0Ω	0.01 µF	DNI	0 Ω	0 Ω	DNI	TDK ATB2012-50011
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 240. Transmitter Matching Components, Tx1 and Tx4

	Component Location on PCB (All Tolerances Are ±3% Unless Otherwise 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
75 to 1000	DNI	0.01 µF	DNI	0 Ω	DNI	0 Ω	0 Ω	DNI	0.01 µF	TDK ATB2012-50011
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 241. Transmitter Matching Components, Tx2 and Tx3

	Component Location on PCB (All Tolerances Are ±3% Unless Otherwise 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
75 to 1000	DNI	0.01 µF	DNI	0 Ω	DNI	0 Ω	0 Ω	DNI	0.01 µF	TDK ATB2012-50011
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 transceiver requires the following five different power supply domains:

- 1.0 V digital: this supply is connected to the device through the three VDIG_1P0 pins. This supply 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%, however, 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, and it is critical that the input traces for these three inputs be balanced (same impedance for the 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 the inputs isolated from digital domains inside the transceiver. The tolerance on these supply inputs is ±5%.
- 1.3 V analog: these supplies connect to all functional blocks in the transceiver through 14 different input pins, and 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. The tolerance on these supply inputs is ±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 SPI interface, GPIO, and digital control inputs.

SUPPLY CAPACITY

During operation, supply currents can vary significantly, especially if operating in TDD mode. The supply must have adequate capacity to provide the necessary current (as indicated in the device data sheet) so that performance criteria over all process and temperature variations are maintained. Analog Devices recommends adding 900 mA to the digital supply maximum and 30% margin to all analog supply maximums to ensure proper operation under all conditions.

POWER SUPPLY SEQUENCE

The transceiver 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, these inputs can power up at the same time as the VDIG_1P0 supply. When 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 transceiver. 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, disable the sources of all supplies simultaneously to ensure that there is no back feeding circuits that have been powered down.

POWER SUPPLY DOMAIN CONNECTIONS

Table 242 lists the pin name, the pin number, the recommended routing technique for that pin from the main supply rail (if applicable), and a brief description of the block the pin powers in the chip.

The information listed in Table 242 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. This input can be powered from the other 1.3 V analog supply. Noise from this supply can affect the JESD link performance directly.

Table 242. 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 that 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 is 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, isolate by ground from the other transmitter supplies, connect to the pins using RF chokes (part depends on the frequency range).	Alternative transmit 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, isolate by ground from the other transmitter supplies, connect to the pins using RF chokes (part depends on the frequency range).	Alternative transmit 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, isolate by ground from the other transmitter supplies, connect to the pins using RF chokes (part depends on the frequency range).	Alternative transmit output supply if power is not supplied via a centertapped balun
TX4± (RF Choke Feed)	P1, N1	Analog	1.8	Star connect from the 1.8 V plane, isolate by ground from the other transmitter supplies, connect to the pins using RF chokes (part depends on the frequency range).	Alternative transmit 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 the other 1.8 V inputs with a ferrite bead if necessary.	1.8 V supply for the following circuit blocks: Tx1 analog output, Rx1 LO buffer, RF synthesizer 1, AUXADC_0, AUXADC_1, Rx1 TIA, ORx1 mixer, and Converter1 LDO
VANA2_1P8	B14	Analog	1.8	Star connect from the 1.8 V plane. Isolate from the other 1.8 V inputs with a ferrite bead if necessary.	1.8 V supply for the following circuit blocks: Tx2 analog output, Tx1_2 LO buffers and LO delay, analog SPI, DEVCLK, AUX PLL and AUX LO generation, receive LO multiplexer and master bias, Rx2 LO buffer, RFPLL1 and LO generator 1, 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 the other 1.8 V inputs with a ferrite bead if necessary.	1.8 V supply for the following circuit blocks: Tx3 analog output, auxilliary synthesizer, Tx3_4 LO, analog SPI, Rx3 LO buffer, RF PLL2 and LO generator 2, ORx3_4 LO buffers and TxLB3_4 LO buffers, Rx3 TIA, ORx3 mixer, and ORx3_4 TIA
VANA4_1P8	N2	Analog	1.8	Star connect from the 1.8 V plane. Isolate from the other 1.8 V inputs with a ferrite bead if necessary.	1.8 V supply for the following circuit blocks:Tx4 analog output, RF synthesizer 2, Rx4 LO buffer, clock PLL and CLKGEN, clock synth, AUXADC_2, AUXADC_3, Rx4 TIA, ORx4 mixer, and Converter2 LDO
VCONV1_1P8	H15	Analog	1.8	Star connect from the 1.8 V plane. Isolate from the other 1.8 V inputs with a ferrite bead if necessary.	1.8 V supply for Tx1 and Tx2 DACs, Rx1 and Rx2 ADCs, and ORx1 and ORx2 ADCs
VCONV2_1P8	H3	Analog	1.8	Star connect from the 1.8 V plane. Isolate from the other 1.8 V inputs with a ferrite bead if necessary.	1.8 V supply for Tx3 and4 DACs, Rx3 and 4 ADCs, and ORx3 and 4 ADCs

Pin Name	Pin No.	Type	Voltage (V)	Recommended Routing/Notes	Description
VJVCO_1P8	P11	Analog	1.8	Star connect from the 1.8 V plane. Isolate from the other 1.8 V inputs with a ferrite bead if necessary.	1.8 V 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 the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary.	1.3 V supply for Tx1 phase detector, BBF, Tx2 phase detector, Rx1, Rx2 TIA, ORx1_2 TIA, and analog SPI
VANA2_1P3	D3	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from hte other 1.3 V inputs with a ferrite bead if necessary.	1.3 V supply for Tx3 phase detector, BBF, Tx4 phase detector, Rx3,4 TIA, ORx3_4 TIA, and analog SPI
VCONV1_1P3	J15	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary.	1.3 V supply for Tx1,2 DACs, Rx1,2 ADCs, and ORx1_2 ADCs
VCONV2_1P3	J3	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary.	1.3 V supply for Tx3,4 DACs, Rx3, Rx4 ADCs, and ORx3_4 ADCs
VRFVCO1_1P3	G15	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for RF VCO1 and LO generator 1
VRFVCO2_1P3	G3	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for RF VCO2 and LO generator 2
VRFSYN1_1P3	J13	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for RF1 synthesizer
VRFSYN2_1P3	J5	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for RF1 synthesizer
VAUXVCO_1P3	C12	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for auxilliary VCO, auxilliary LO generator 1 and 2, and auxilliary LO generator 3 and 4
VAUXSYN_1P3	C6	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This	1.3 V supply for auxilliary synthesizer

Pin Name	Pin No.	Type	Voltage (V)	Recommended Routing/Notes	Description
				pin is very sensitive to aggressors.	
VCLKSYN_1P3	R7	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for DEVCLK and clock synthesizer
VCLKVCO_1P3	N5	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for clock VCO, clock generation, and clock distribution
VRXLO_1P3	A9	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for Rx1 and Rx2 LO multiplexer, Rx3 and Rx4 LO multiplexer
VTXLO_1P3	A7	Analog	1.3	Star connect from the 1.3 V plane. Use wide traces/shapes to minimize the trace resistance as much as possible. Isolate from the other 1.3 V inputs with a ferrite bead if necessary. This pin is very sensitive to aggressors.	1.3 V supply for Tx1and Tx2 LO multiplexer, Tx3 and Tx4 LO multiplexer
VSER_1P0	R3, R4	Analog	1.0	Connect these pins directly to VDIG_1P0 or to a 1.0 V regulator using a separate wide trace to minimize the resistance as much as possible. Connect to these pins using a ferrite bead if digital noise is a concern.	1.0 V supply for JESD serializer
VDES_1P0	P12, P13	Analog	1.0	Connect these pins directly to VDIG_1P0 or to a 1.0 V regulator using a separate wide trace to minimize the resistance as much as possible. Connect to these pins using a ferrite bead if digital noise is a concern.	1.0 V supply for JESD deserializer
VTT_DES	P14	Analog	1.0	Connect directly to VDIG_1P0 or to a 1.0 V regulator. Connect to this pin using a ferrite bead if digital noise is a concern.	1.0 V 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 to this pin using a ferrite bead if digital noise is a concern.	1.0 V supply for the JESD synth
VCONV1_1P0	K12	Analog	1.0	Connect a 4.7 µF bypass capacitor from this pin to ground.	Bypass connection point for internal converter regulator
VCONV2_1P0	K3	Analog	1.0	Connect a 4.7 µF bypass capacitor to from this pin ground.	Bypass connection point for internal converter regulator
VAUXVCO_1P0	B11	Analog	1.0	Connect a 4.7 µF bypass capacitor to from this pin ground.	Bypass connection point for internal AUXVCO regulator
VCLKVCO_1P0	P5	Analog	1.0	Connect a 4.7 µF bypass capacitor to from this pin ground.	Bypass connection point for internal CLKVCO regulator
VRFVCO1_1P0	G13	Analog	1.0	Connect a 4.7 µF bypass capacitor to from this pin ground.	Bypass connection point for internal RFVCO1 regulator
VRFVCO2_1P0	G5	Analog	1.0	Connect a 4.7 µF bypass capacitor to from this pin ground.	Bypass connection point for internal RFVCO2 regulator

POWER SUPPLY ARCHITECTURE

The diagram in Figure 143 outlines the power supply configuration used on the CE board. This configuration follows the recommendations outlined in Table 242. This diagram includes the use of ferrite beads for additional RF isolation and 0 Ω resistors. These devices accomplish the following functions:

- The ferrite beads provide RF noise filtering. They may be necessary when users encounter RF spurs or noise coupling in their application and additional isolation is required. If they are found to be unnecessary, they can be replaced by 0 Ω resistors.
- The resistors and ferrite beads ensure that the layout follows power routing guidelines, which recommends the traces to be star connected to a central supply.
- The resistors and ferrite beads provide a place in the circuit where the current can be monitored and measured for debugging purposes. For this function, the components can be replaced by low impedance shunt resistors and the voltage measured to determine the total current to the specified input ball.
- The 0 Ω resistors provide connection options for the JESD204B and JESD204C power supply inputs so that these power inputs can be isolated from the digital supply if digital noise corrupts the JESD204B and JESD204C performance.

For more details on recommended power supply implementation, refer to the CE board schematic included in the design support package.

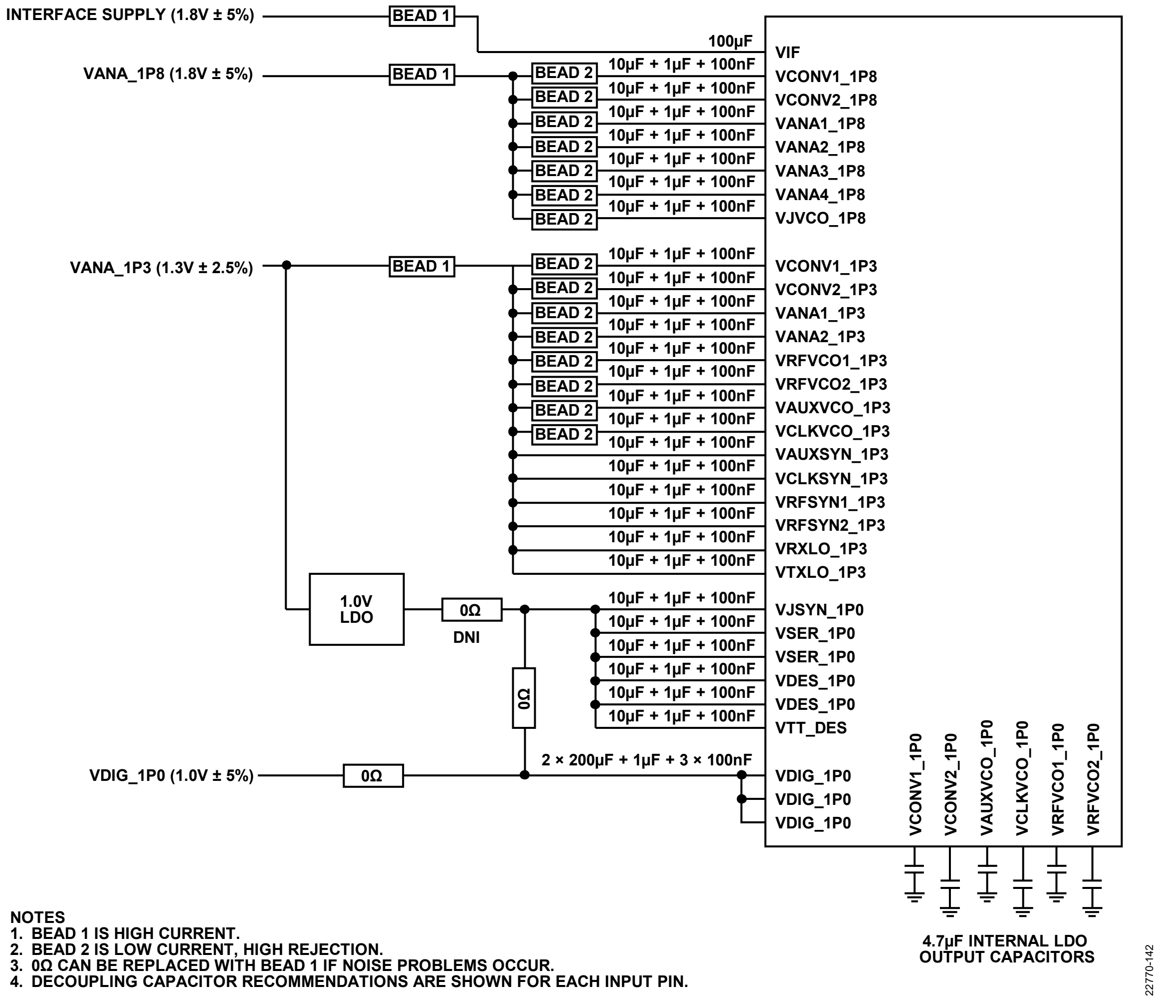


Figure 143. Power Supply Connection Diagram

CURRENT CONSUMPTION

The current consumption in each block can vary depending on the device configuration for the profile in use. Clock frequencies, data rates, calibrations, and the number of channels in operation all influence the amount of current required for transceiver operation. The following section gives a sample of a typical use case profile and the resulting current consumption in different modes. This example is a typical example, but do not consider the values maximums for design purposes. Follow the design margins noted previously in Supply Capacity the section when sizing power supplies.

Current Measurements: Use Case 26 Non Link Sharing Profile

The setup parameters for this use case are as follows:

- Transmit channels: 4
- Receive channels: 4
- Observation receive channels: 1
- Device clock: 491.52 MHz
- Transmit/Receive primary signal bandwidth: 200 MHz
- Transmit/Observation receive synthesis bandwidth: 450 MHz
- Receive data sample rate: 245.76 MSPS
- Transmit/Observation Receive data sample rate: 491.52 MSPS
- JESD lane rate: 24.33024 Gbps

Table 243. 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	959	1068	1600
VSER_1P0	R3, R4	Analog	1.0	190	191	193
VDES_1P0	P12, P13	Analog	1.0	361	362	364
VTT_DES	P14	Analog	1.0	4	4	4
VJSYN_1P0	R9	Analog	1.0	6	7	7
VIF	N9	Analog	1.8	8	8	8
VANA1_1P8	N16	Analog	1.8	6	132	131
VANA2_1P8	B14	Analog	1.8	16	134	134
VANA3_1P8	B4	Analog	1.8	8	141	142
VANA4_1P8	N2	Analog	1.8	9	134	134
VCONV1_1P8	H15	Analog	1.8	99	26	100
VCONV2_1P8	H3	Analog	1.8	102	65	142
VJVCO_1P8	P11	Analog	1.8	2	2	2
VANA1_1P3	D15	Analog	1.3	329	325	438
VANA2_1P3	D3	Analog	1.3	331	381	496
VCONV1_1P3	J15	Analog	1.3	382	125	494
VCONV2_1P3	J3	Analog	1.3	382	306	679
VRFVCO1_1P3	G15	Analog	1.3	188	188	189
VRFVCO2_1P3	G3	Analog	1.3	187	188	190
VRFSYN1_1P3	J13	Analog	1.3	11	11	11
VRFSYN2_1P3	J5	Analog	1.3	10	10	10
VAUXVCO_1P3	C12	Analog	1.3	187	216	217
VAUXSYN_1P3	C6	Analog	1.3	8	8	8
VCLKSYN_1P3	R7	Analog	1.3	23	23	24
VCLKVCO_1P3	N5	Analog	1.3	110	110	141
VRXLO_1P3	A9	Analog	1.3	176	19	179
VTXLO_1P3	A7	Analog	1.3	9	186	189

Table 244. 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	242	2333	1520
Tx + ORx Enabled	634	2096	1632
Rx + Tx + ORx Enabled	785	3265	2168

PCB LAYOUT CONSIDERATIONS

OVERVIEW

Because of the complexity of this transceiver device and the high pin count, careful PCB layout is important to obtain optimal performance. This user guide provides a checklist of issues to look for and general guidelines on how to optimize the PCB to mitigate performance issues. Use this user guide to help achieve the best performance from the transceiver while reducing board layout effort. This section assumes that the user is an experienced analog/RF engineer who understands RF PCB layout as well as RF and high speed transmission lines.

The evaluation board represents one of the most complex implementations of the transceiver. All RF inputs and outputs, JESD serial data lanes, and digital control and monitoring signals are implemented in this design and are expected to operate over the full frequency range of the device. Advanced PCB technology is used to achieve maximum device 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 meet the design goals. These techniques include but are not limited to a traditional BGA fanout, fewer layers, through-hole vias only, and lower grade PCB materials.

The PCB Layout Considerations section discusses the following issues and provides guidelines for system designers to get the optimal performance out of the transceiver:

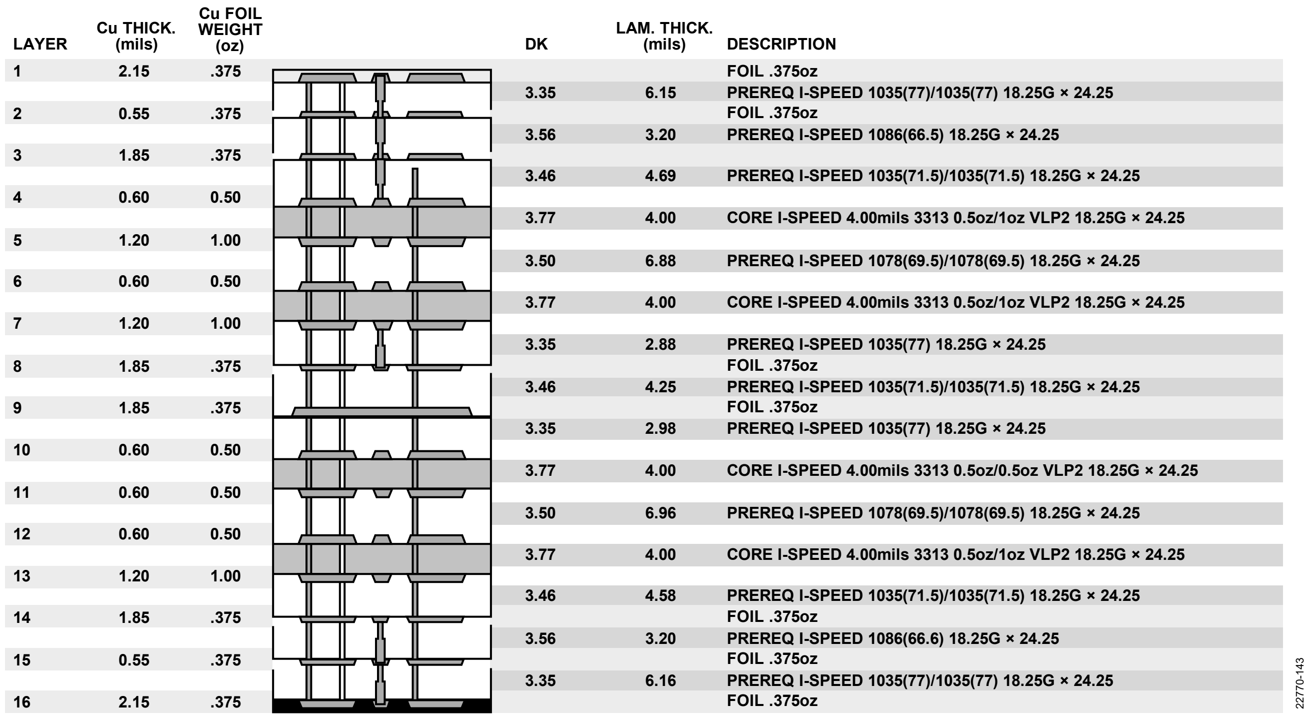
- PCB material and stack up selection
- Fanout and trace space layout guidelines
- Component placement and routing priorities
- RF and JESD transmission line layout
- Isolation techniques used on the CE board
- Power management routing considerations
- Analog signal routing recommendations
- Digital signal routing recommendations
- Unused pin instructions

PCB MATERIAL AND STACK UP SELECTION

The evaluation board utilizes Isola I-Speed dielectric material, and was selected for its low loss tangent and low dielectric constant characteristics. On previous evaluation systems, Analog Devices chose a combination of low loss, RF capable dielectric material for the outer edge layers and standard FR4-370 HR dielectric material for interior layers. RF signal routing on these boards was confined to the top and bottom layers. 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 because of 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 this transceiver. The key comparison metrics for these materials are 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 144 shows the PCB stack up used for the evaluation board. Layer 1 and Layer 16 are primarily used for RF input/output 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 Management Layout Design 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 245 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 short at the exposed board edge.



LAYER	Cu THICK. (mils)	Cu FOIL WEIGHT (oz)	DK	LAM. THICK. (mils)	DESCRIPTION
1	2.15	.375			FOIL .375oz
			3.35	6.15	PREREQ I-SPEED 1035(77)/1035(77) 18.25G × 24.25
2	0.55	.375			FOIL .375oz
			3.56	3.20	PREREQ I-SPEED 1086(66.5) 18.25G × 24.25
3	1.85	.375			
			3.46	4.69	PREREQ I-SPEED 1035(71.5)/1035(71.5) 18.25G × 24.25
4	0.60	0.50			
			3.77	4.00	CORE I-SPEED 4.00mils 3313 0.5oz/1oz VLP2 18.25G × 24.25
5	1.20	1.00			
			3.50	6.88	PREREQ I-SPEED 1078(69.5)/1078(69.5) 18.25G × 24.25
6	0.60	0.50			
			3.77	4.00	CORE I-SPEED 4.00mils 3313 0.5oz/1oz VLP2 18.25G × 24.25
7	1.20	1.00			
			3.35	2.88	PREREQ I-SPEED 1035(77) 18.25G × 24.25
8	1.85	.375			FOIL .375oz
			3.46	4.25	PREREQ I-SPEED 1035(71.5)/1035(71.5) 18.25G × 24.25
9	1.85	.375			FOIL .375oz
			3.35	2.98	PREREQ I-SPEED 1035(77) 18.25G × 24.25
10	0.60	0.50			
			3.77	4.00	CORE I-SPEED 4.00mils 3313 0.5oz/0.5oz VLP2 18.25G × 24.25
11	0.60	0.50			
			3.50	6.96	PREREQ I-SPEED 1078(69.5)/1078(69.5) 18.25G × 24.25
12	0.60	0.50			
			3.77	4.00	CORE I-SPEED 4.00mils 3313 0.5oz/1oz VLP2 18.25G × 24.25
13	1.20	1.00			
			3.46	4.58	PREREQ I-SPEED 1035(71.5)/1035(71.5) 18.25G × 24.25
14	1.85	.375			FOIL .375oz
			3.56	3.20	PREREQ I-SPEED 1086(66.6) 18.25G × 24.25
15	0.55	.375			FOIL .375oz
			3.35	6.16	PREREQ I-SPEED 1035(77)/1035(77) 18.25G × 24.25
16	2.15	.375			FOIL .375oz

Figure 144. PCB Material Stack Up Diagram

Table 245. 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	33.00	84.06		
3	8	Mech	Via	Resin fill	7.90	27.06		
9	14	Mech	Via	Resin fill	7.90	26.51		
1	16	Mech	Via	Nonconductive via fill	7.90	84.06		
15	14	Laser	Microvia	CuVF_Button pattern	5.90	3.74		Y
16	15	Laser	Microvia	Nonconductive via fill	9.10	6.54		Y
8	7	Laser	Microvia	Dummy drill	7.90	3.37		Y
1	2	Laser	Microvia	Non-Conductive via fill	9.10	6.53		Y
2	3	Laser	Via	CuVF_Button pattern	5.90	3.74		Y
3	4	Laser	Microvia	CuVF_Button pattern	7.90	5.16		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 246 describes details about trace impedance controls used in the evaluation board and types of line structures used to obtain desired impedance and performance on and for given layers and impedances.

Table 246. 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	51.02	9.75
1	Edge coupled differential	50.00	±5	27.00	32.00	(2)	27.50	50.54	9.75
1	Edge coupled differential	100.00	±10	7.50	14.50	(2)	8.25	101.83	9.62
2	Edge coupled differential	100.00	±10	4.25	12.00	(1, 3)	3.75	101.01	12.25
4	Edge coupled differential	100.00	±10	3.75	9.50	(3, 5)	3.75	100.67	12.00
4	Single-ended	50.00	±5	4.50	0.00	(3, 5)	4.25	50.05	12.13
6	Single-ended	50.00	±5	4.50	0.00	(5, 7)	4.75	50.94	11.88
8	Single-ended	38.00	±3.8	5.00	0.00	(9, 7)	5.00	37.90	5.00
10	Single-ended	50.00	±5	3.00	0.00	(9, 11)	3.25	51.20	11.88
10	Edge coupled differential	100.00	±10	3.00	11.00	(9, 11)	3.25	100.87	11.88
11	Single-ended	50.00	±5	4.50	0.00	(12, 10)	4.75	51.02	11.88
11	Edge coupled differential	100.00	±10	4.00	9.00	(12, 10)	4.00	100.19	12.01
12	Single-ended	50.00	±5	4.50	0.00	(11, 13)	4.75	51.02	11.88
12	Edge coupled differential	100.00	±10	4.00	9.00	(11, 13)	4.00	100.19	12.00
16	Single-Ended	50.00	±5	11.00	0.00	(15)	11.50	51.06	9.75
16	Edge coupled differential	100.00	±10	7.50	14.50	(15)	8.25	101.85	9.62

FANOUT AND TRACE SPACING GUIDELINES

The package used for these transceivers is a 14 mm × 14 mm 289-ball BGA 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 are placed on the perimeter rows of the BGA to minimize complexity of the routing of these critical signals. Via in pad technology is used to escape all other signals to the 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 145 illustrates the fanout of RF differential channels from the transceiver 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 transceiver. 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.

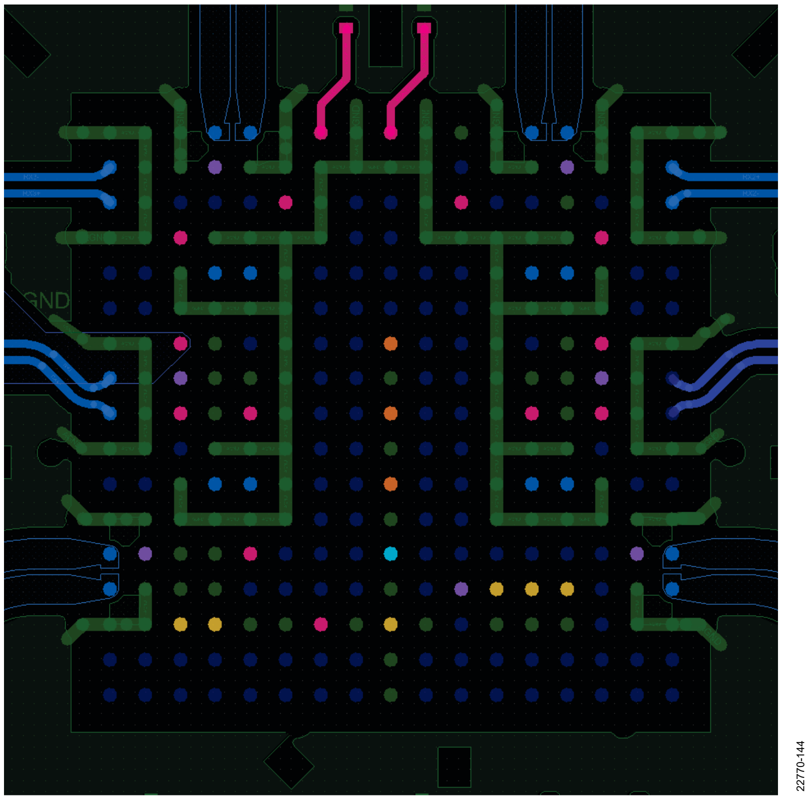


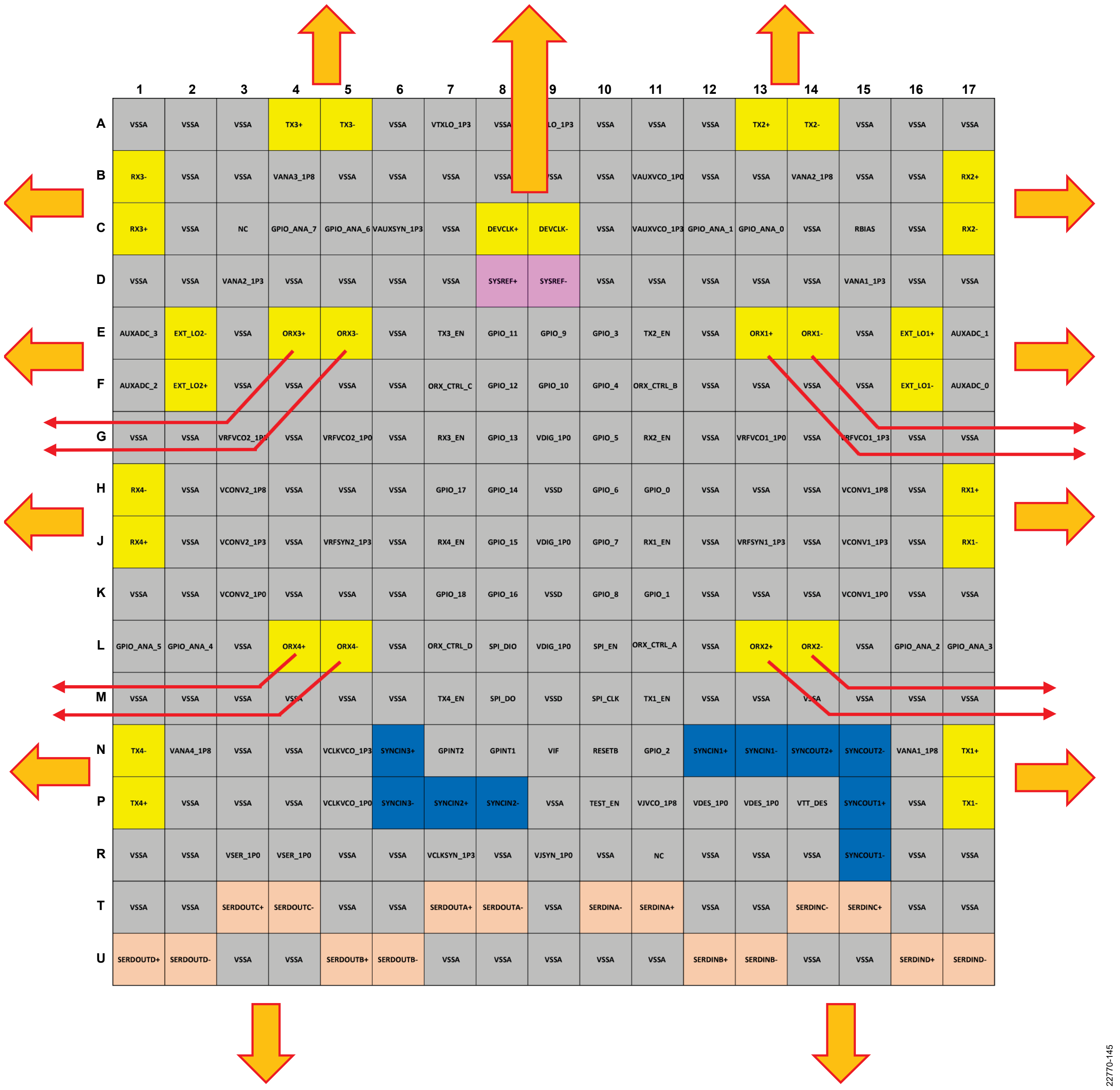
Figure 145. CE Board RF Receiver and Transmitter Fanout and Layout

COMPONENT PLACEMENT AND ROUTING GUIDELINES

The transceiver requires few external components to function. Those components that are required must be carefully placed and routed to optimize performance. This section provides instructions 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 146 shows the general directions in which each of the signals must be routed so that they can be effectively isolated from aggressor signals. It can 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 either on the 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.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	VSSA	VSSA	VSSA	TX3+	TX3-	VSSA	VTXLO_1P3	VSSA	LO_1P3	VSSA	VSSA	VSSA	TX2+	TX2-	VSSA	VSSA	VSSA
B	RX3-	VSSA	VSSA	VANA3_1P8	VSSA	VSSA	VSSA	VSSA	VSSA	VSSA	VAUXVCO_1P0	VSSA	VSSA	VANA2_1P8	VSSA	VSSA	RX2+
C	RX3+	VSSA	NC	GPIO_ANA_7	GPIO_ANA_6	VAUXSYN_1P3	VSSA	DEVCLK+	DEVCLK-	VSSA	VAUXVCO_1P3	GPIO_ANA_1	GPIO_ANA_0	VSSA	RBIAS	VSSA	RX2-
D	VSSA	VSSA	VANA2_1P3	VSSA	VSSA	VSSA	VSSA	SYSREF+	SYSREF-	VSSA	VSSA	VSSA	VSSA	VSSA	VANA1_1P3	VSSA	VSSA
E	AUXADC_3	EXT_LO2-	VSSA	ORX3+	ORX3-	VSSA	TX3_EN	GPIO_11	GPIO_9	GPIO_3	TX2_EN	VSSA	ORX1+	ORX1-	VSSA	EXT_LO1+	AUXADC_1
F	AUXADC_2	EXT_LO2+	VSSA	VSSA	VSSA	VSSA	ORX_CTRL_C	GPIO_12	GPIO_10	GPIO_4	ORX_CTRL_B	VSSA	VSSA	VSSA	VSSA	EXT_LO1-	AUXADC_0
G	VSSA	VSSA	VRFVCO2_1P3	VSSA	VRFVCO2_1P0	VSSA	RX3_EN	GPIO_13	VDIG_1P0	GPIO_5	RX2_EN	VSSA	VRFVCO1_1P0	VSSA	VRFVCO1_1P3	VSSA	VSSA
H	RX4-	VSSA	VCONV2_1P8	VSSA	VSSA	VSSA	GPIO_17	GPIO_14	VSSD	GPIO_6	GPIO_0	VSSA	VSSA	VSSA	VCONV1_1P8	VSSA	RX1+
J	RX4+	VSSA	VCONV2_1P3	VSSA	VRFSYN2_1P3	VSSA	RX4_EN	GPIO_15	VDIG_1P0	GPIO_7	RX1_EN	VSSA	VRFSYN1_1P3	VSSA	VCONV1_1P3	VSSA	RX1-
K	VSSA	VSSA	VCONV2_1P0	VSSA	VSSA	VSSA	GPIO_18	GPIO_16	VSSD	GPIO_8	GPIO_1	VSSA	VSSA	VSSA	VCONV1_1P0	VSSA	VSSA
L	GPIO_ANA_5	GPIO_ANA_4	VSSA	ORX4+	ORX4-	VSSA	ORX_CTRL_D	SPI_DIO	VDIG_1P0	SPI_EN	ORX_CTRL_A	VSSA	ORX2+	ORX2-	VSSA	GPIO_ANA_2	GPIO_ANA_3
M	VSSA	VSSA	VSSA	VSSA	VSSA	VSSA	TX4_EN	SPI_DO	VSSD	SPI_CLK	TX1_EN	VSSA	VSSA	VSSA	VSSA	VSSA	VSSA
N	TX4-	VANA4_1P8	VSSA	VSSA	VCLKVCO_1P3	SYNCIN3+	GPINT2	GPINT1	VIF	RESETB	GPIO_2	SYNCIN1+	SYNCIN1-	SYNCOUT2+	SYNCOUT2-	VANA1_1P8	TX1+
P	TX4+	VSSA	VSSA	VSSA	VCLKVCO_1P0	SYNCIN3-	SYNCIN2+	SYNCIN2-	VSSA	TEST_EN	VJVCO_1P8	VDES_1P0	VDES_1P0	VTT_DES	SYNCOUT1+	VSSA	TX1-
R	VSSA	VSSA	VSER_1P0	VSER_1P0	VSSA	VSSA	VCLKSYN_1P3	VSSA	VJSYN_1P0	VSSA	NC	VSSA	VSSA	VSSA	SYNCOUT1-	VSSA	VSSA
T	VSSA	VSSA	SERDOUTC+	SERDOUTC-	VSSA	VSSA	SERDOUTA+	SERDOUTA-	VSSA	SERDINA-	SERDINA+	VSSA	VSSA	SERDINC-	SERDINC+	VSSA	VSSA
U	SERDOUTD+	SERDOUTD-	VSSA	VSSA	SERDOUTB+	SERDOUTB-	VSSA	VSSA	VSSA	VSSA	VSSA	SERDINB+	SERDINB-	VSSA	VSSA	SERDIND+	SERDIND-

Figure 146. RF IO, DEVCLK, EXT LO, and SERDES Routing Guidelines

Transmit, receive, and observation receive 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 the desired levels of isolation between the RF signal paths, use the considerations and techniques described in the Isolation Techniques section in designs.

For RF transmit outputs, install a 10 μF capacitor near the transmit balun(s) VANAx_1P8 dc feed(s). This capacitor acts as a reservoir for the transmit supply current. The Transmit Bias Supply Guidelines section discusses the transmit 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 147. 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.

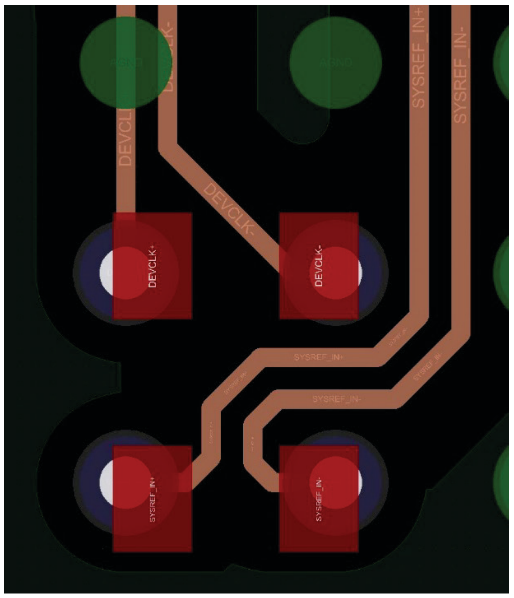


Figure 147. DEVCLK and SYSREF Termination

Route SERDES 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 SERDES signals. Provide adequate isolation between interface differential pairs.

Signals with Second Routing Priority

Power supply routing and quality have a direct impact on overall system performance. The Power Management Layout section provides recommendations for how to best route the power supplies to minimize loss as well as interference between RF channels. Follow the recommendations provided in the Power Management Layout Design section to ensure optimal RF and isolation performance.

Signals with Lowest Routing Priority

Route the remaining low frequency digital inputs and outputs, the auxiliary ADCs and DACs, and the SPI signals. It is important to route all digital signals bounded between Row E and Row 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 146 for the ball diagram). The 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 input/output and high speed digital, but are still on the analog side of the PCB.

RF AND JESD TRANSMISSION LINE LAYOUT

RF Routing Guidelines

The evaluation boards use both surface coplanar waveguide and surface edge coupled coplanar waveguide transmission lines for transmit, receive, and observation receive 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 transceiver and the RF balun as short as possible. Keep the length of the single-ended transmission lines for RF signals as short as possible. Keeping signal paths as short as possible reduces susceptibility to undesired signal coupling and reduces the effects of parasitic capacitance, inductance, and loss on the transfer function of the transmission line and impedance matching network system. The routing of these signal paths is the most critical factor in optimizing performance and, therefore, must be routed prior to any other signals and maintain the highest priority in the PCB layout process.

All 12 RF ports are impedance matched using π-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, DEVCLK can require impedance matching to ensure optimal performance. Figure 148 depicts the path from the device to the 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.

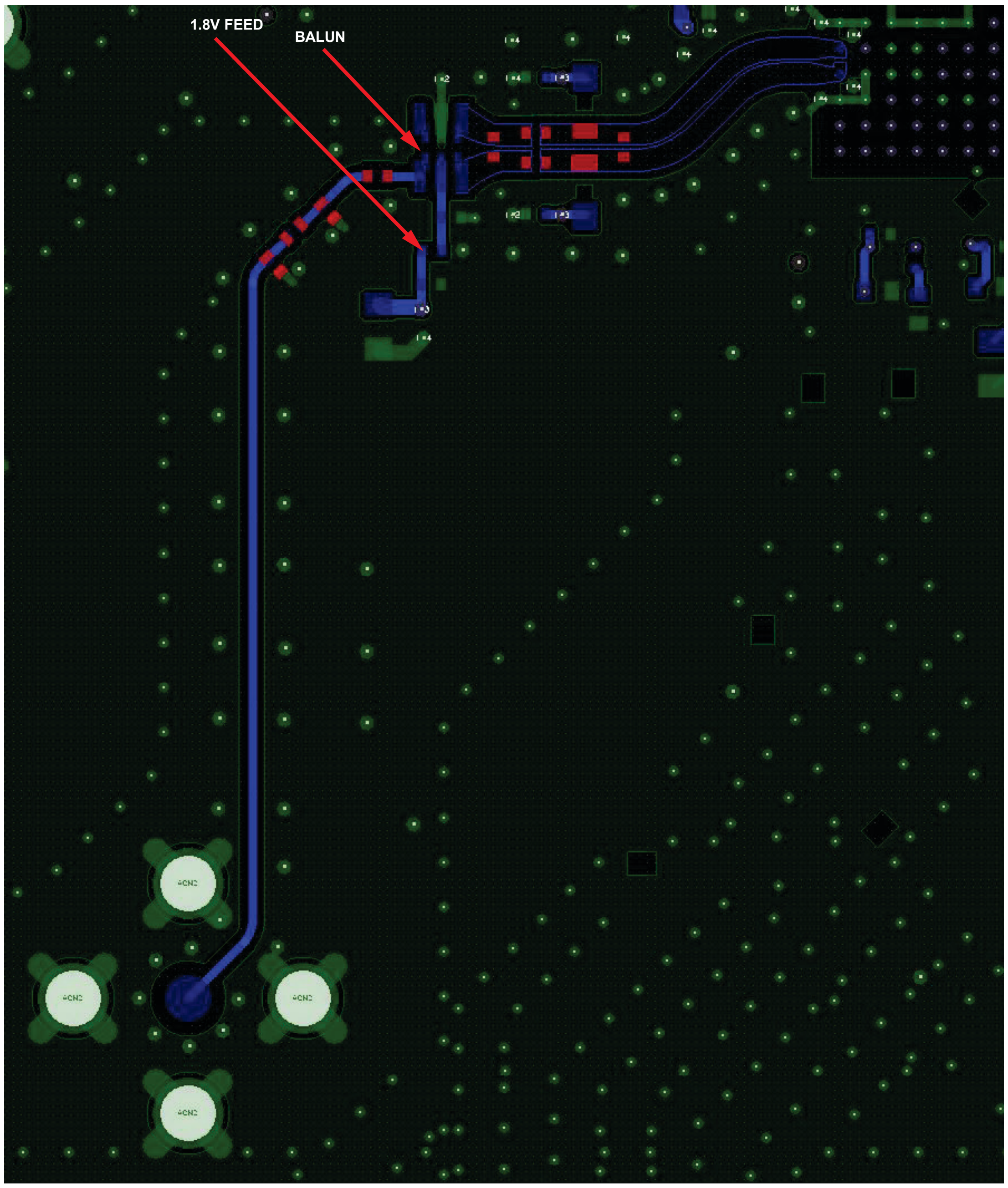


Figure 148. Transmit RF Routing and Matching Network

All 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.

Transmit Bias Supply Guidelines

Each transmitter requires approximately 125 mA supplied through an external connection. In the CE board, bias voltages are supplied at the dc feed of a center tapped balun in the RF signal path, as shown in Figure 149.

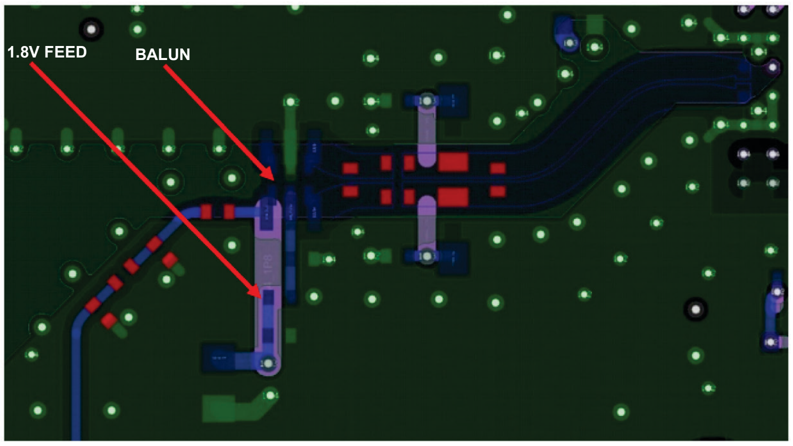


Figure 149. 1.8V Transmit Bias Routing at Balun

To reduce switching transients because of 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 150 shows the 1.8 V supply distribution on the 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 transmit input.

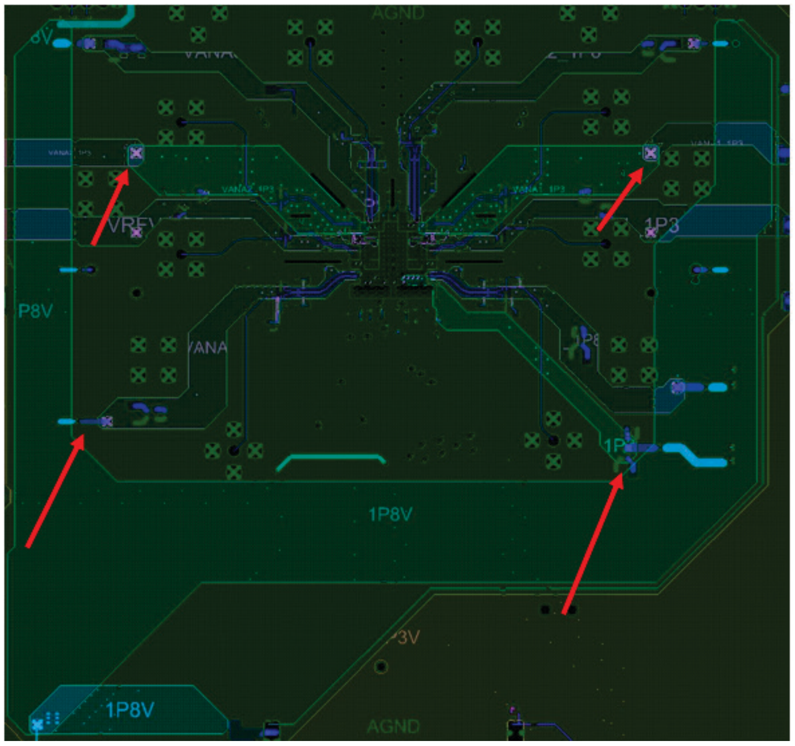


Figure 150. 1.8 V Supply Distribution

The evaluation board couples the supply into the transmitter via a center tapped balun, but is also provisioned for an external choke feed inductor with an ac decoupling capacitor. This topology helps improve transmitter to transmitter isolation.

When a balun is selected that does not have a dc feed capability, RF chokes must be used to supply the current to the transmitters. Chokes are connected from the 1.8 V supply to each transmit output. Note that in this scenario, the transmit 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 transmit output pins. Take care to match both chokes and their layout to avoid peaking because of current transients.

JESD204B/JESD204C Routing Recommendations

The transceiver 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 transceiver 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 and target 100 Ω differential. Ensure that the pair is loosely coplanar edge coupled. The CE board uses 4 mil wide traces and a separation of approximately 10 mil. These parameters can vary depending on the stack up and selected dielectric material. Minimize the pad area for all connector and passive components to reduce parasitic capacitance effects on the transmission lines, which can negatively impact signal integrity. Minimize using vias to route these signals 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, which reduces 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 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 the pad of the component footprint. AC coupling capacitors (100 nF) are placed in series near the FMC connector away from the chip. The JESD interface can operate at frequencies up to 16 GHz.

Figure 151 and Figure 152 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 JESD204 standard.

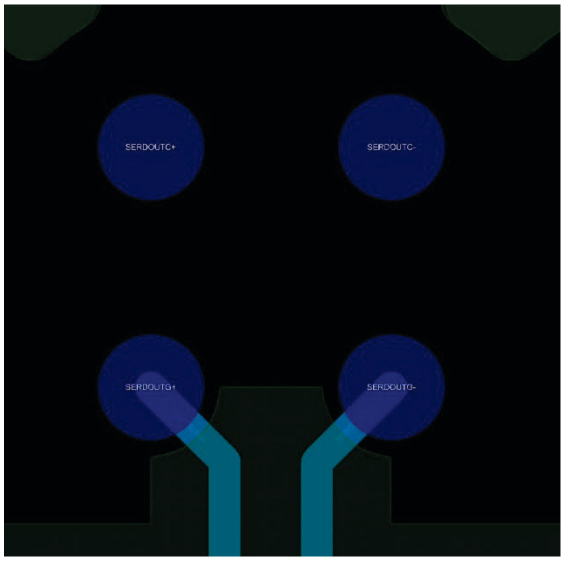


Figure 151. JESD Signal Launch on Layer 2

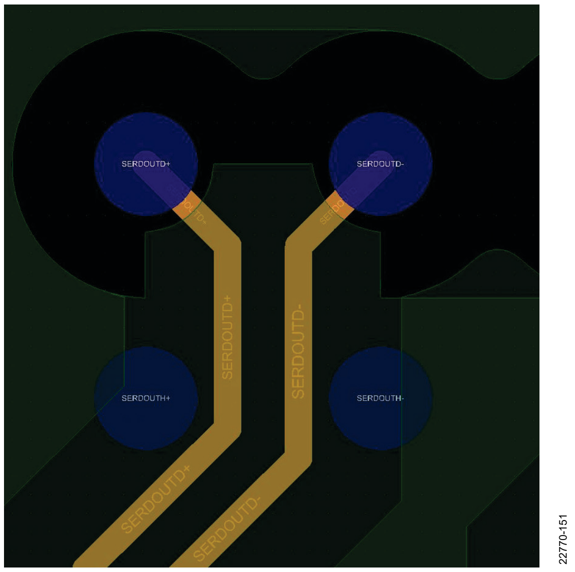


Figure 152. JESD Signal Launch on Layer 4

ISOLATION TECHNIQUES

Given the density of sensitive and critical signals, significant isolation challenges are faced when designing a PCB for the transceiver. The isolation requirements listed in Table 247 were followed to accurately evaluate transceiver performance. Analytically determining aggressor to victim isolation in a system is complex and involves considering vector combinations of aggressor signals and coupling mechanisms.

Isolation Goals

Table 247 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 CE board.

Table 247. Port to Port Isolation Goals

Port	650 MHz to 4 GHz	4 GHz to 6 GHz
Transmit to Transmit	65 dB	60 dB
Transmit to Receive	70 dB	65 dB
Transmit to Observation Receive	70 dB	65 dB
Receive to Receive	65 dB	60 dB
Receive to Observation Receive	70 dB	65 dB

Isolation Between RF I/O Ports

The primary coupling mechanisms between the RF IO paths on the evaluation board include the following:

- Magnetic field coupling
- Surface propagation
- Cross domain coupling via ground

To reduce the impact of these coupling mechanisms on the CE board, several strategies were used. Large slots are opened in the ground plane between the RF IO paths. These discontinuities prevent surface propagation. A careful designer can 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 153.

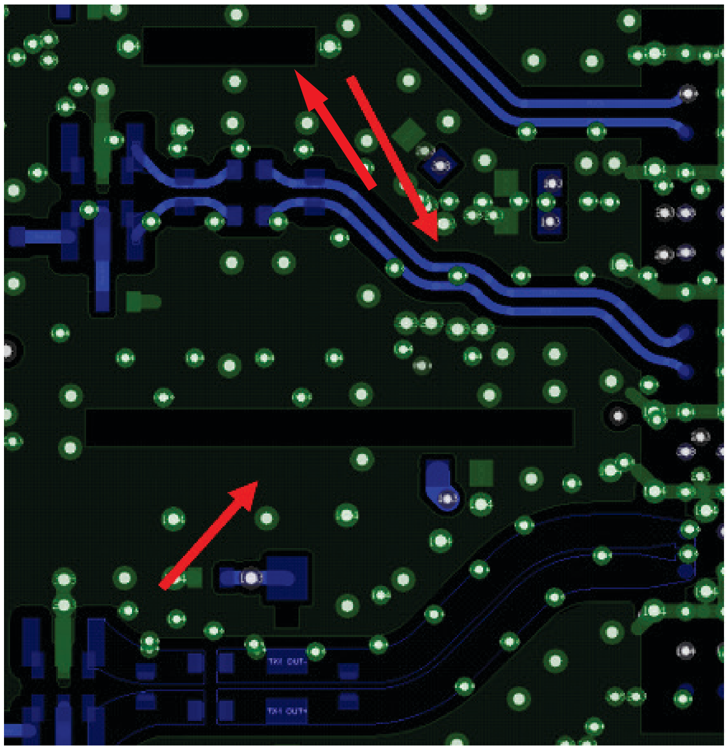


Figure 153. RF IO Isolation Structures

Additional shielding is provided using connecting VSSA balls under the device to form a shield around the RF IO ball pairs. This ground provides a termination for stray electric fields. Figure 154 shows how this layout is done for Tx1. The same layout approach is used for each set of sensitive RF IO ports. Ground vias are used along the 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 because of practical layout considerations. The wavelength is dependent on the dielectric material relative permittivity (ε_r) and can be calculated using the following equation:

$$wavelength\,(m) = \frac{300}{frequency\,(MHz) \times \sqrt{\varepsilon_r}}$$

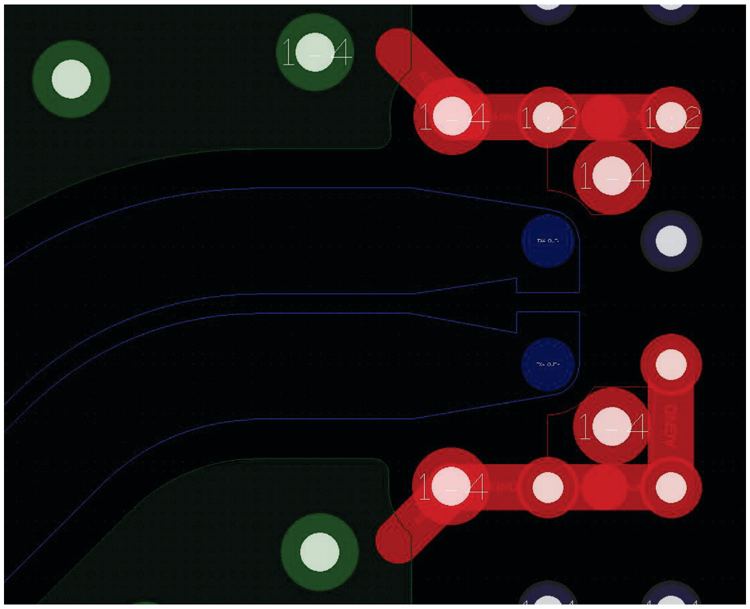


Figure 154. Transmit Launch Shielding

RF IO baluns are spaced and aligned to reduce magnetic coupling from the structures in the balun package. Care must be taken to reduce crosstalk over shared grounds between baluns. Another precaution taken involves placing and orienting SMA connectors to minimize connector to connector coupling between ports.

Isolation Between SERDES Lines

The SERDES interface uses 16 lane pairs that can operate at speeds up to 16 GHz. Take care when creating the PCB layout to ensure that those lines are routed following the guidelines described in the JESD204B/JESD204C Routing Recommendations section. In addition, use isolation techniques to prevent crosstalk between the different SERDES lane pairs. Via fencing is the primary technique used on the CE board.

Figure 155 illustrates the via fencing technique. Ground vias are placed along and between each pair of traces to provide isolation and decrease crosstalk. The spacing between vias, shown as Label A in Figure 155, follows the rule provided in the Isolation Between RF I/O Ports section. For the most accurate spacing of fencing vias, use layout simulation software.

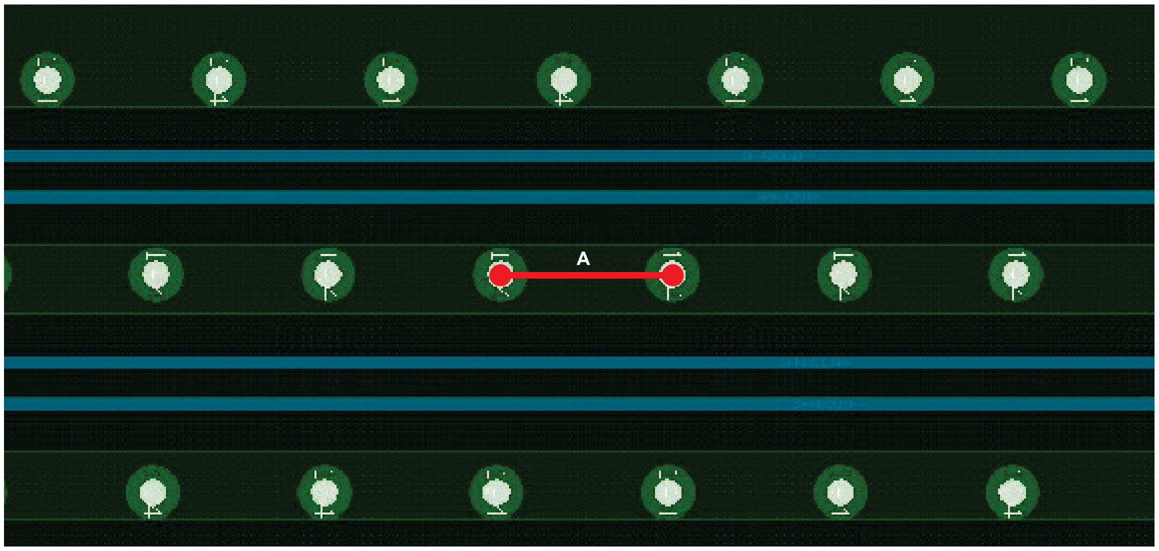


Figure 155. SERDES Lane Via Fencing

POWER MANAGEMENT LAYOUT DESIGN

Because of the complexity and high level of integration in the transceiver, power supply routing is critical to achieve optimum RF performance. The transceiver 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 design provides an improved level of isolation compared to previous products, but 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 CE 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 156 shows this layout approach on the CE board. The inner purple “U” shape labeled 1P8 is the 1.8 V supply and the outer pink “U” shape labeled 1P3V is the 1.3 V supply. Note that neither shape forms a complete ring, which was done to better control the current path for each supply and avoid current loops and coupling between the two hemispheres of the board. There is also a thin strip of ground plane that is routed between the two supply rings to maintain some separation and prevention of direct coupling on the same layer.

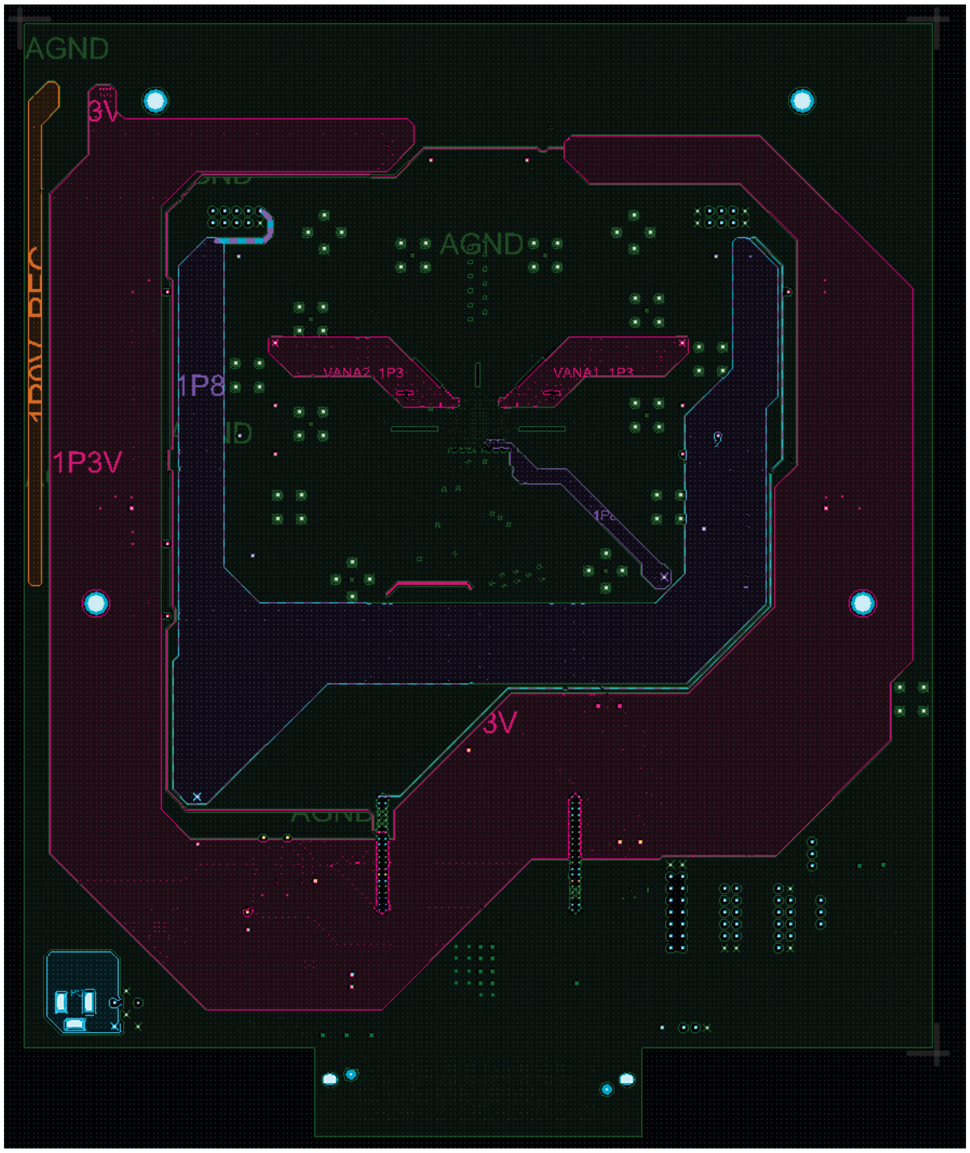


Figure 156. Analog Power Ring Layout Approach

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 157 and Figure 158 illustrate how these star connections are made to the individual supply balls of the transceiver. Some of the connections are made directly to the corresponding supply ring and 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 stack up drawing are used for the inputs with the highest current consumption values.

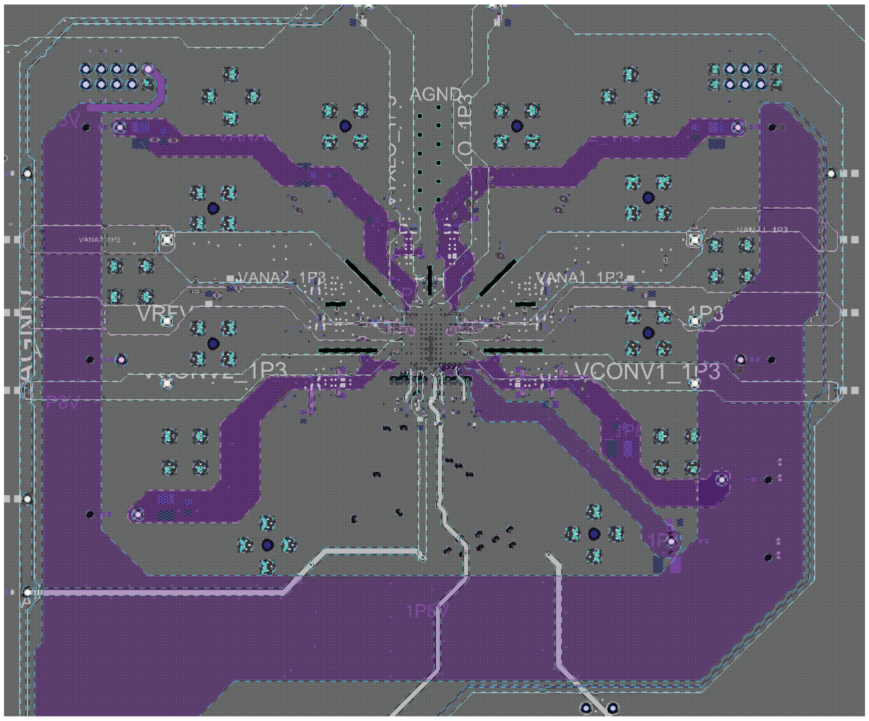


Figure 157. 1.8 V Supply Routing Using Star Connections

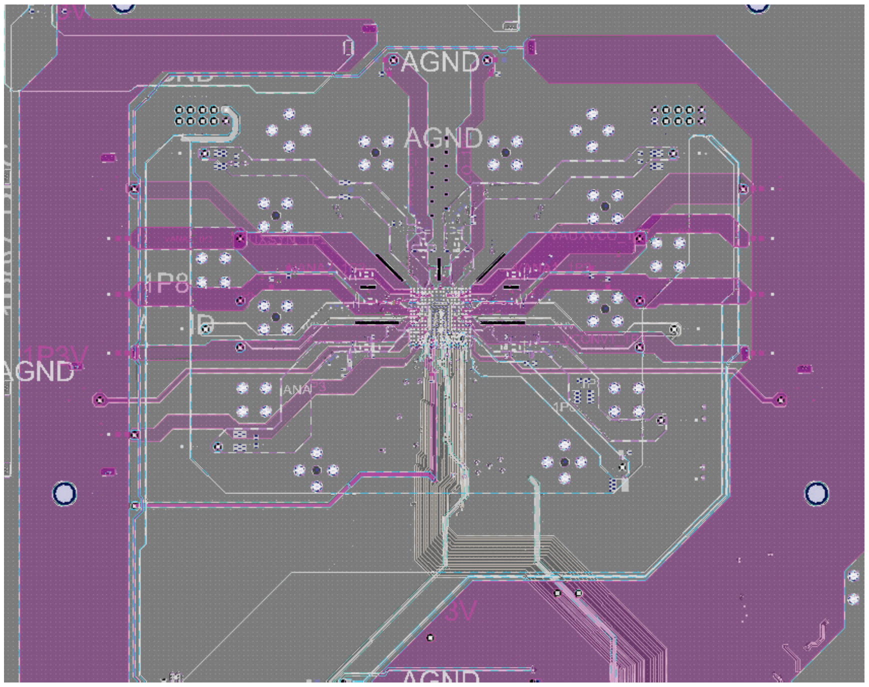


Figure 158. 1.3 V Supply Routing Using Star Connections

Digital Power Routing

The digital 1.0 V supply is the noisiest supply in the system and it is important to keep this supply shielded from the other supplies. This supply is also the highest current supply and the thickness of the traces must 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 and 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 that there is no additional voltage drop in one connection compared to the others. Figure 159 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 the channel is isolated from other signals. Figure 160 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 and their voltage drops are equal when the device is active.

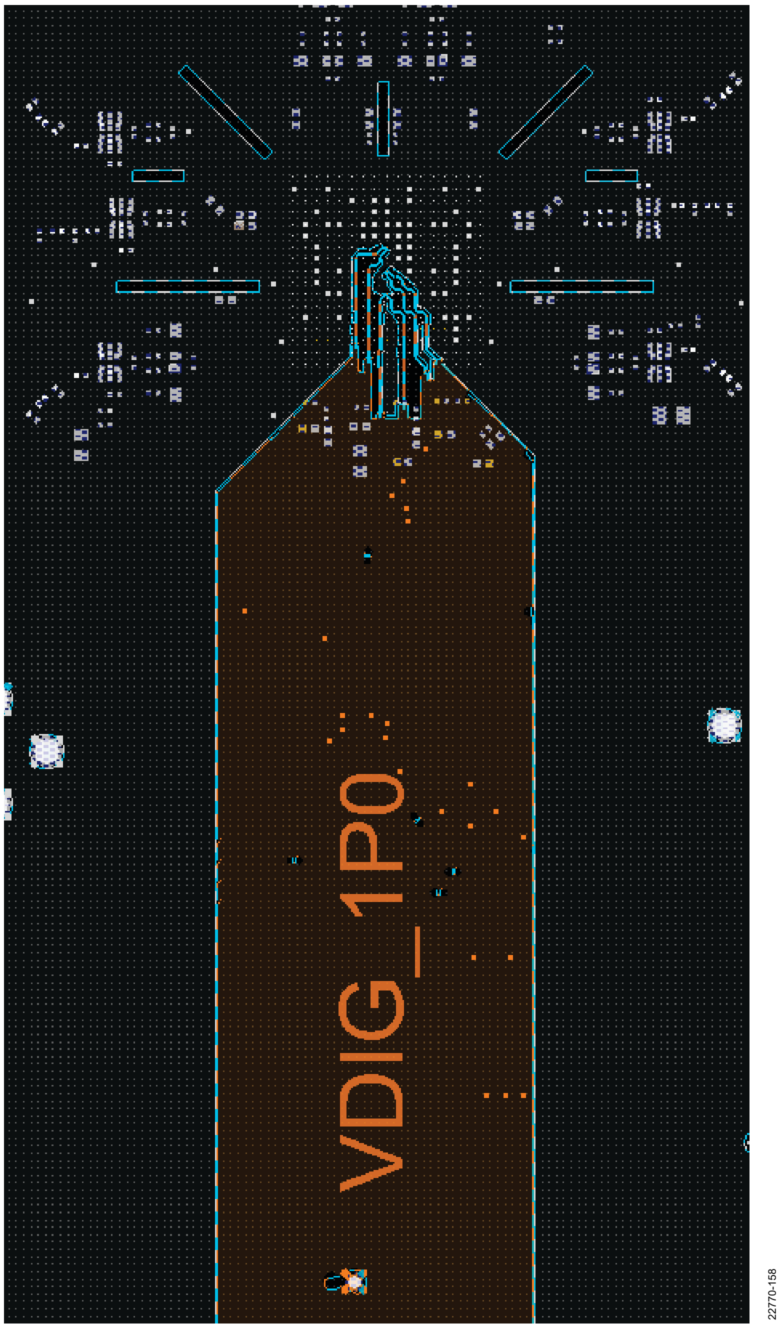


Figure 159. Digital Supply Routing

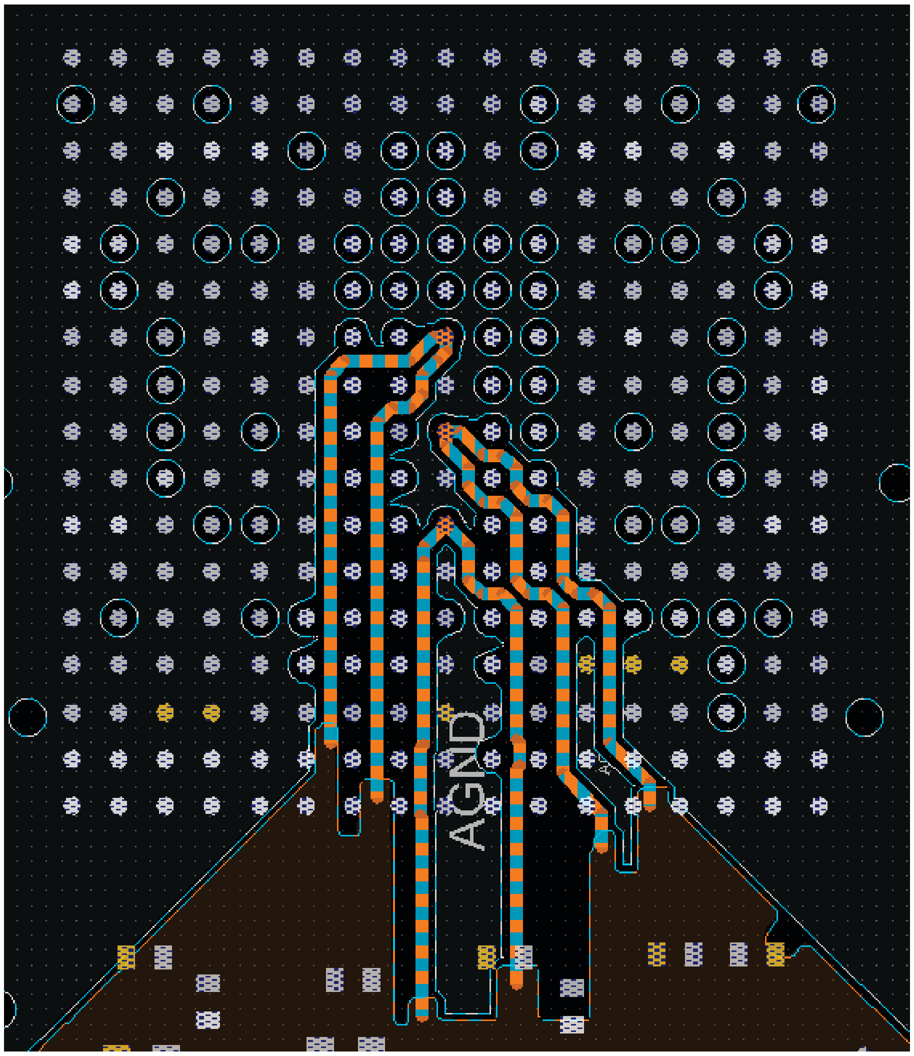


Figure 160. Digital Supply Connection to Three VDIG Input Pins of the Device

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 and these traces are made by creating filled areas as wide as possible to minimize voltage drop. The VTT_DES and VJSYN_1P0 traces carry little current and are, therefore, routed using standard traces. Figure 161 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.

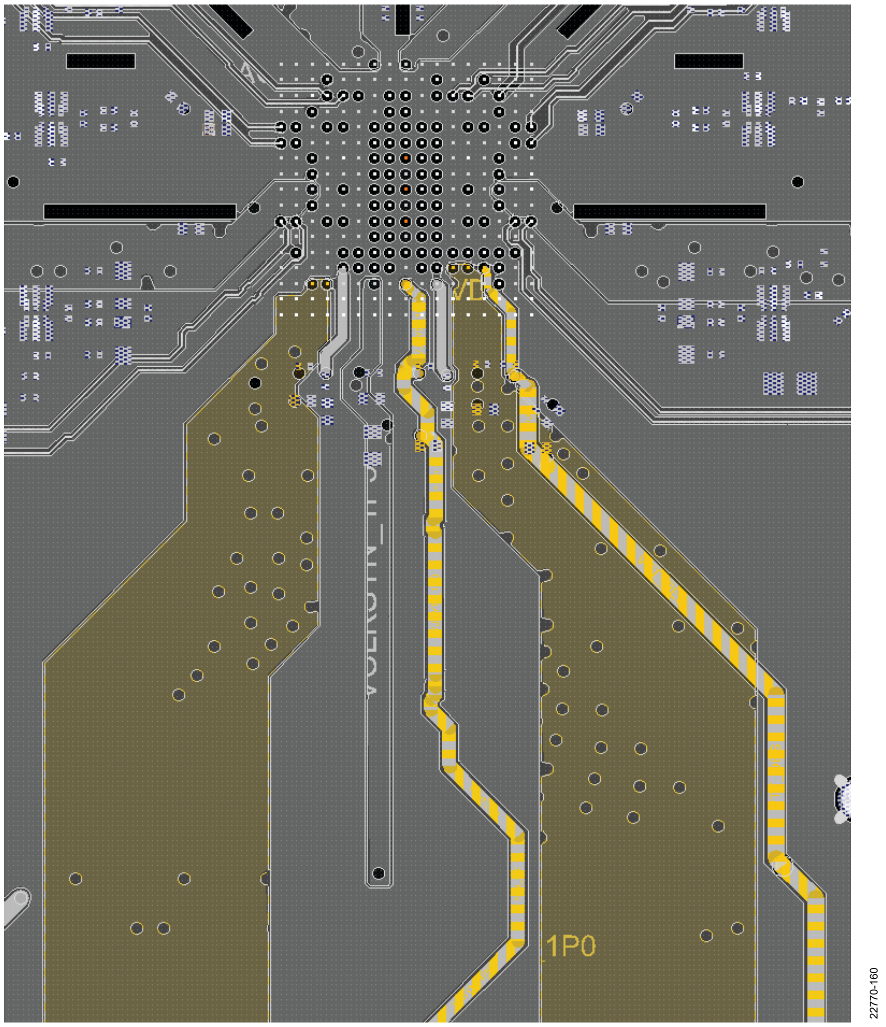


Figure 161. SERDES Power Supply Input Routing

Interface Supply Input

The VIF interface supply is a low current input that provides the supply reference for the SPI serial interface. This supply 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 similar to the VDIG_1P0 supply and keep the trace isolated from other signals to ensure that the trace 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 such as what is required for this transceiver. Another option is to connect all ground to the same plane system and use cutouts and channeling similar to 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. When placing components on power supply inputs, use the following guidelines:

- Each power supply pin requires at a minimum a 0.1 μF bypass capacitor near the pin. For inputs that require a large current step, a 10 μF capacitor in parallel is recommended. Place the ground side of the bypass capacitor(s) so that the 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 either a ferrite bead or a 0 Ω place holder as a series component. Ensure that the device is sized properly to handle the current load for the particular power supply input of concern.
- Figure 162 and Figure 163 illustrate an example of how the power supply routing from the common power ring to a bypass capacitor and into the transceiver 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.

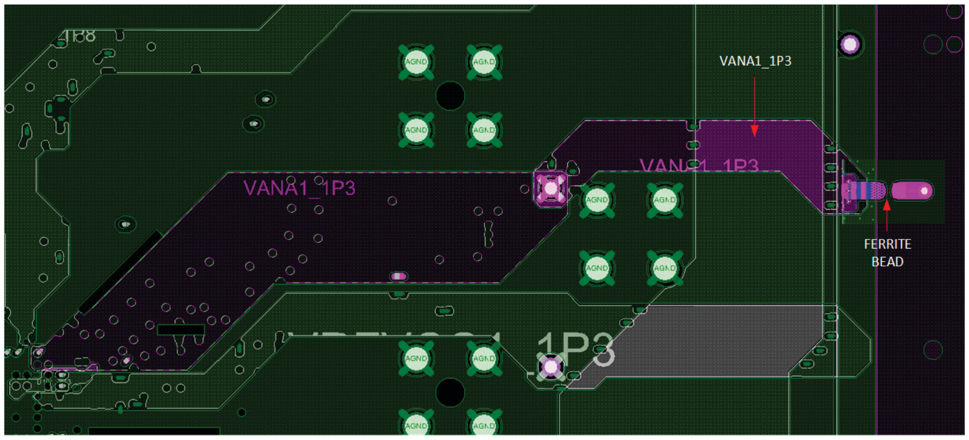


Figure 162. Power Supply Routing Example with Ferrite Bead at the Input (VANA1_1P3)

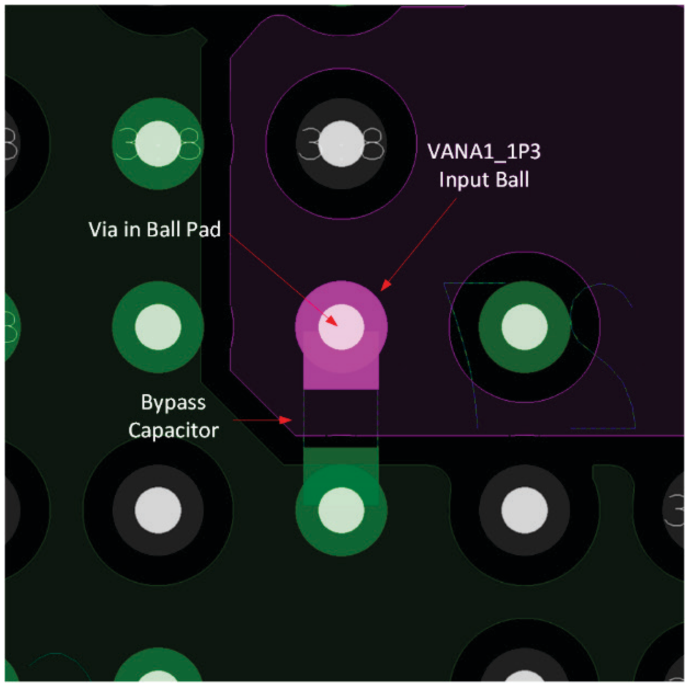


Figure 163. Power Connection to Supply Ball with Bypass Capacitor Between Vias

ANALOG SIGNAL ROUTING CONSIDERATIONS

Other analog signals in and out of the transceiver 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 can 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 is nevertheless 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 routing 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 transceiver. Most of these signals are static or infrequently change state and once signals are routed out of the device, they can be fanned out to other parts of the PCB without interfering with the radio functions. Figure 164 illustrates how the signals are routed out of the device following the same path as the VDIG supply (the brown area labeled "Digital Routing Region"). Note that there are designated layers on the customer evaluation PCB for digital routing and that these layers are isolated by ground layers from other sensitive signals such as the JESD lanes and the RF inputs and outputs. It is recommended to keep any traces that are nonstatic, such as the SPI or SPI2 buses, isolated from the sensitive analog, RF, and JESD signals by ensuring that there is ample ground between the traces and that there is no overlap of signals from layer to layer.

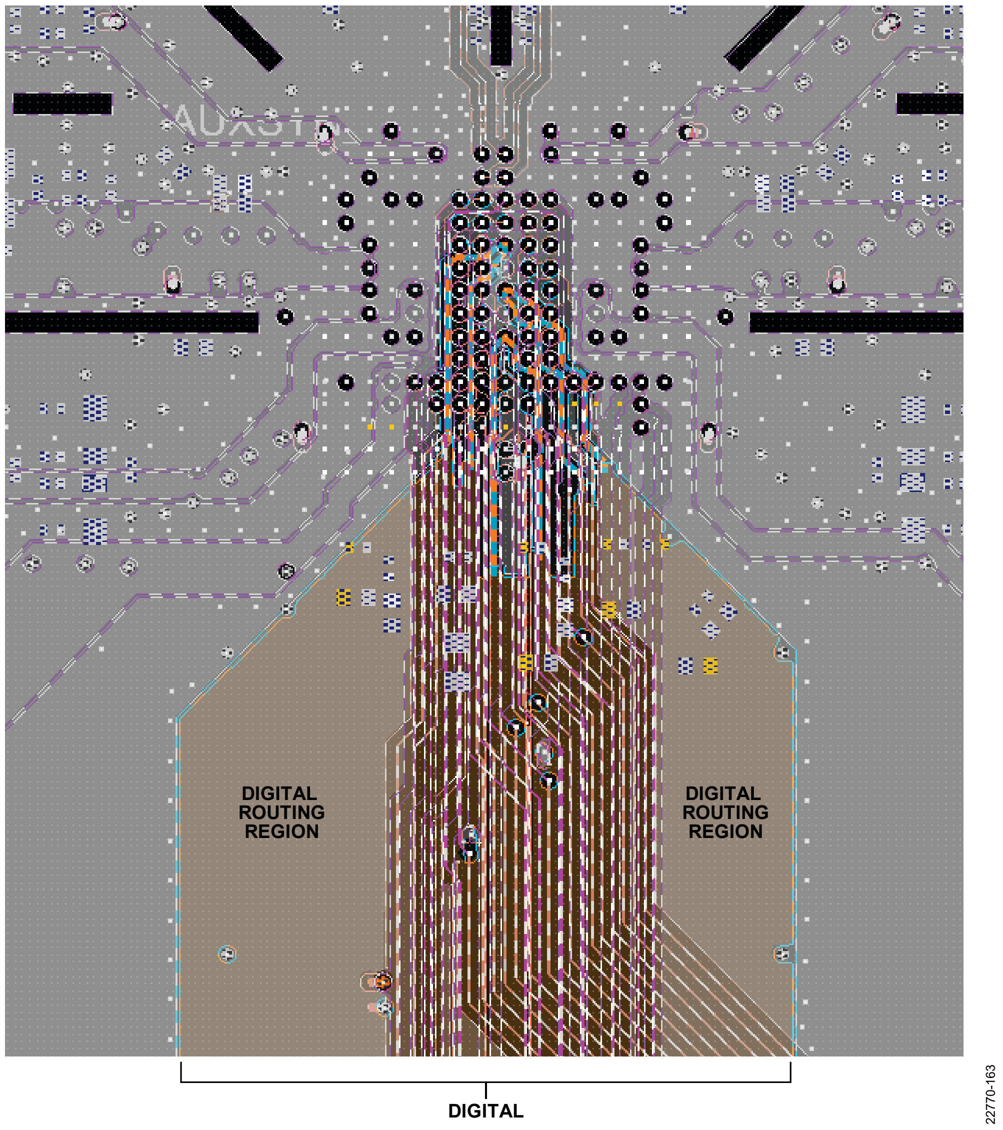


Figure 164. Digital Routing Out of the Transceiver

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 248 for unused pins.

Table 248. 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 transceiver demonstration system enables the user to evaluate the transceiver without having to develop custom software or hardware. The system comprises a radio daughtercard, an ADS9 motherboard, a microSD card with an 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. The GUI can run with or without evaluation hardware connected. When the TES runs without the hardware connected, the TES can be fully configured for a particular operating mode. If the evaluation hardware is connected, the desired operating parameters can be setup with the TES and then the software can program the evaluation hardware. When the transceiver 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 the TES if customized scripts are desired.

HARDWARE KIT

The transceiver demonstration system kit contains the following:

- The CE board in the form of a daughter card with an FMC connector
- One 12 V wall connector power supply cable
- One SD card that contains an image of a Linux operating system with the required evaluation software (SD card is 16 GB size, Type 10)

The ADS9 demonstration system kit contains the following:

- The ADS9 motherboard with an FMC connector
- One 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 transceiver demonstration system kit (six options available).
- 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 (local area network) 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 165 and Figure 166.

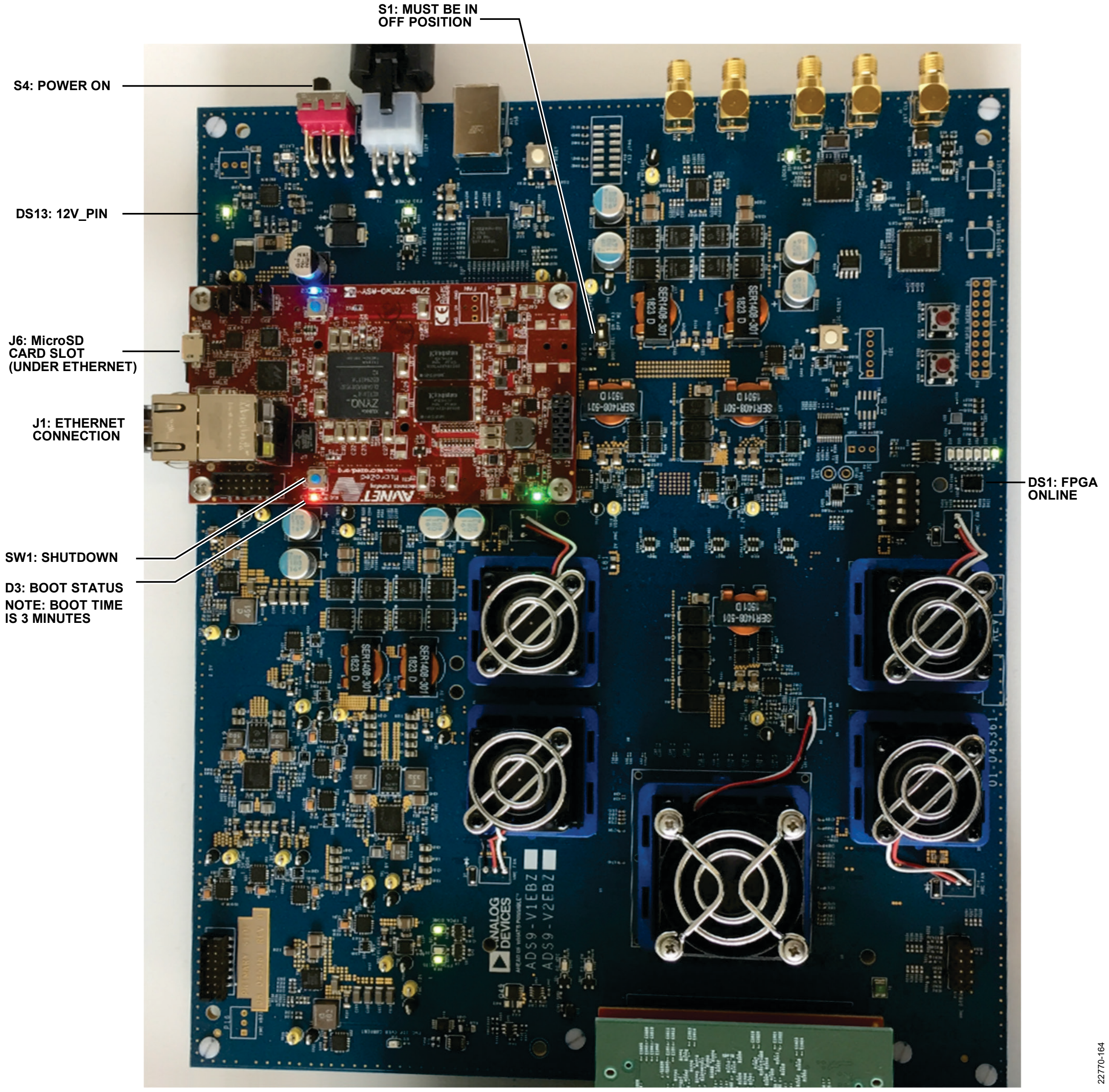


Figure 165. Analog Devices ADS9 Motherboard Configured to Work with Transceiver Evaluation Boards

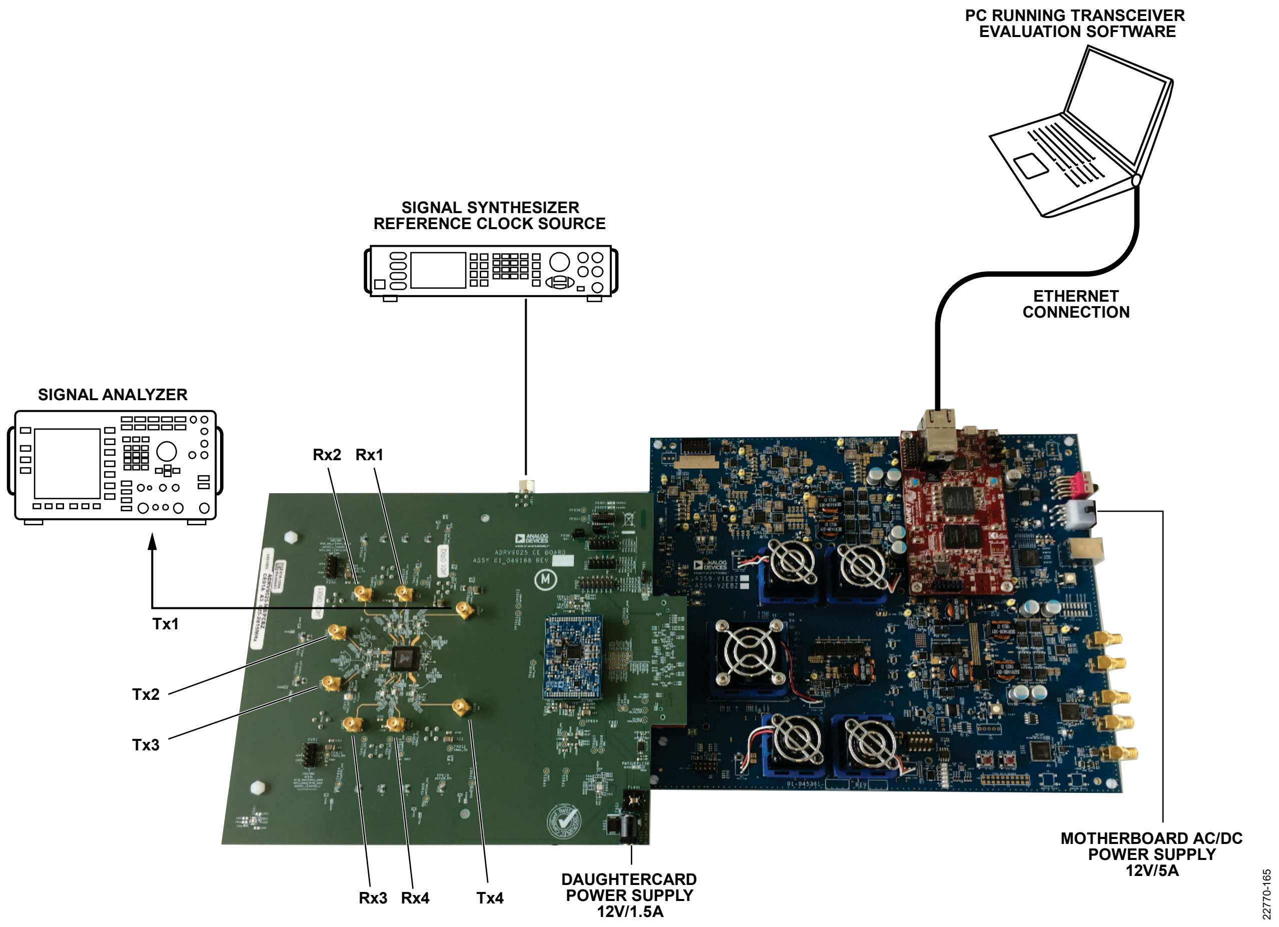

PC RUNNING TRANSCEIVER EVALUATION SOFTWARE
SIGNAL SYNTHESIZER REFERENCE CLOCK SOURCE
ETHERNET CONNECTION
SIGNAL ANALYZER
Rx2
Rx1
Tx1
Tx2
Tx3
Rx3
Rx4
Tx4
DAUGHTERCARD POWER SUPPLY 12V/1.5A
MOTHERBOARD AC/DC POWER SUPPLY 12V/5A
22770-165

Figure 166. CE Board and ADS9 Motherboard with Connections Required for Transmit Testing

To set up the evaluation board for testing, take the following steps:

1. Connect the transceiver evaluation board and the ADS9 evaluation platform together as shown in Figure 166. Use the HPC FMC connector (P1001/P2). Ensure that the connectors are properly aligned.
2. Insert the SD card that came with the evaluation kit into the ADS9 microSD card slot (J6).
3. On the transceiver 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 the 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.

If the Ethernet port is already occupied by another connection, use an USB to Ethernet adapter.

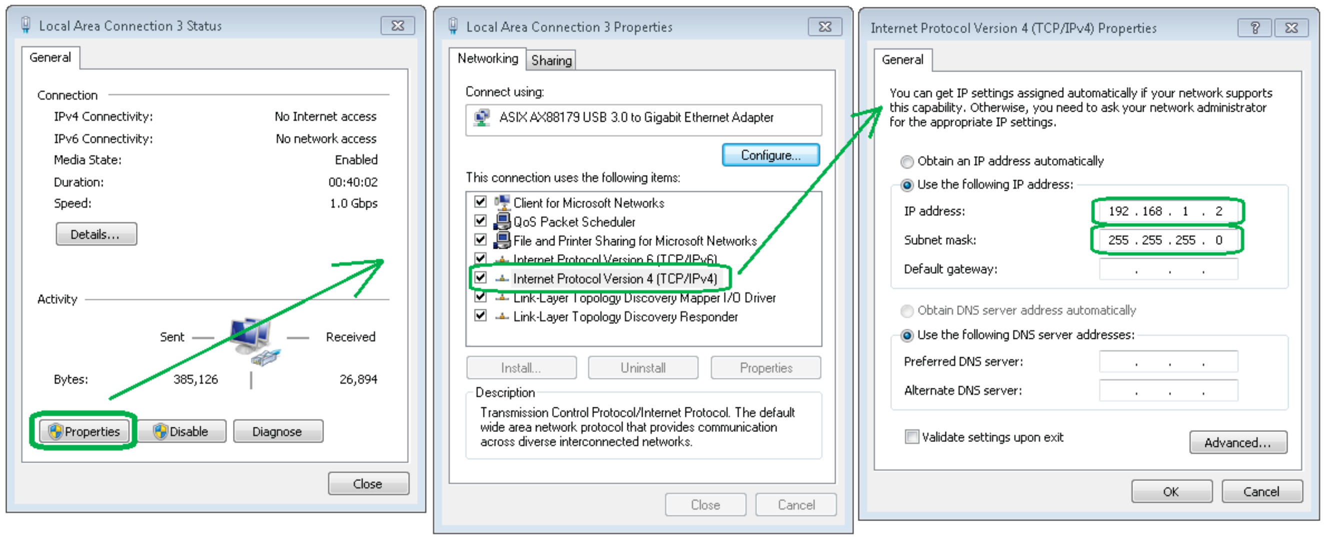
On an Ethernet connection dedicated to the ADS9 platform, the user must manually set the following:

- IPv4 address: 192.168.1.2
- IPv4 subnet mask: 255.255.255.0

Refer to Figure 167 for more details. Ensure that the following ports are not blocked by firewall software on the PC:

- Port 22: SSH protocol
- Port 55556: access to the evaluation software on the ADS9 platform

Note that the ADS9 IP address is set by default to 192.168.1.10.



Local Area Connection 3 Status

General

Connection

IPv4 Connectivity:	No Internet access
IPv6 Connectivity:	No network access
Media State:	Enabled
Duration:	00:40:02
Speed:	1.0 Gbps

Details...

Activity

	Sent	Received
Bytes:	385,126	26,894

Properties Disable Diagnose

Close

Local Area Connection 3 Properties

Networking Sharing

Connect using:

ASIX AX88179 USB 3.0 to Gigabit Ethernet Adapter

Configure...

This connection uses the following items:

- ✓ Client for Microsoft Networks
- ✓ QoS Packet Scheduler
- ✓ File and Printer Sharing for Microsoft Networks
- ✓ Internet Protocol Version 6 (TCP/IPv6)
- ✓ Internet Protocol Version 4 (TCP/IPv4)
- ✓ Link-Layer Topology Discovery Mapper I/O Driver
- ✓ Link-Layer Topology Discovery Responder

Install... Uninstall Properties

Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

Close Cancel

Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

○ Obtain an IP address automatically

● Use the following IP address:

IP address:	192 . 168 . 1 . 2
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .

○ Obtain DNS server address automatically

● Use the following DNS server addresses:

Preferred DNS server:	. . .
Alternate DNS server:	. . .

☐ Validate settings upon exit

Advanced...

OK Cancel

Figure 167. IP Settings for Ethernet Port Dedicated for ADS9 Platform

HARDWARE OPERATION

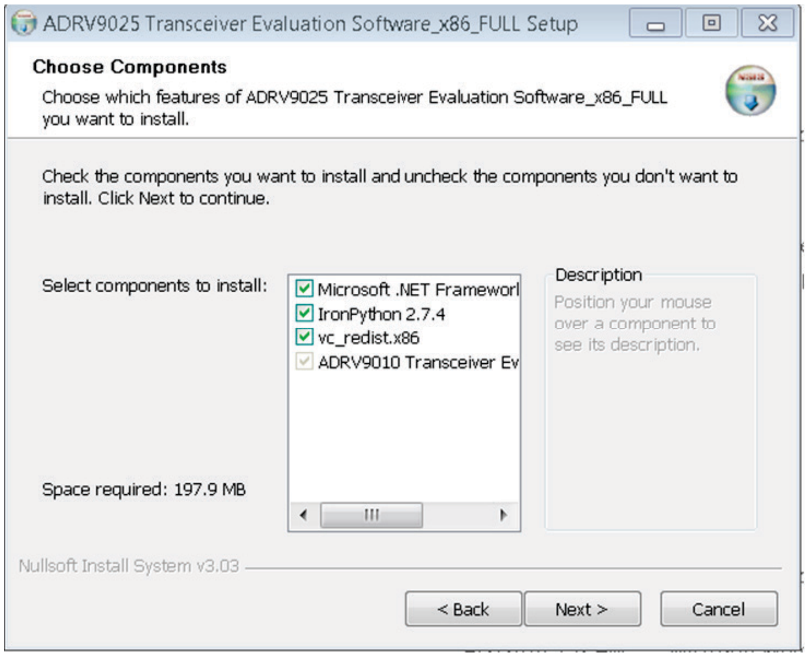
The following steps should be used to setup the evaluation platform for use with the Transceiver Evaluation Software.

1. Switch the ADS9 motherboard power switch (S4) to the on position to turn on the evaluation system. If hardware is connected correctly, the green LED (DS13) on the ADS9 motherboard illuminates.
2. The ADS9 motherboard uses a Linux operating system. Wait approximately 3 minutes before the system is ready for operation and can accept commands from PC software. Boot status can be observed on the ADS9 LED (D3, on the MicroZED daughtercard). This LED illuminates red for approximately 3 minutes after power-on. When the LED 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 continuous wave 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 transmit output ports (J501 to J504) are properly terminated.
5. Connect power from the 12 V wall adapter to the CE board. When power is applied, DS801 and DS802 illuminate on the CE board.
6. For transmitter testing, connect a spectrum analyzer to any transmit 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 transmit paths, either into spectrum analyzers or into 50 Ω if unused.
7. Unplug the wall adaptor to power the CE board off before the motherboard.
8. When power is removed from the CE board, click Disconnect in the TES window and then press and hold SW1 on the ADS9 evaluation board (MicroZED daughter card) until LED D3 illuminates. When LED D3 starts to blink, it is safe to turn off the ADS9 power using Switch S4.

TES INSTALLATION

Contact an Analog Devices representative to obtain access to the TES. When the initial software download completes, 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_x64_FULL.exe** (there is a x86 version available if the computer does not support x64 operation).

Administrator privileges are required to install the TES. When an executable file is run, 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 168 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 updating the version of the TES, these boxes can be left cleared to save installation time.


ADRV9025 Transceiver Evaluation Software_x86_FULL Setup

Choose Components

Choose which features of ADRV9025 Transceiver Evaluation Software_x86_FULL you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [✓] Microsoft .NET Frameworl
- [✓] IronPython 2.7.4
- [✓] vc_redist.x86
- [✓] ADRV9010 Transceiver Ev

Description

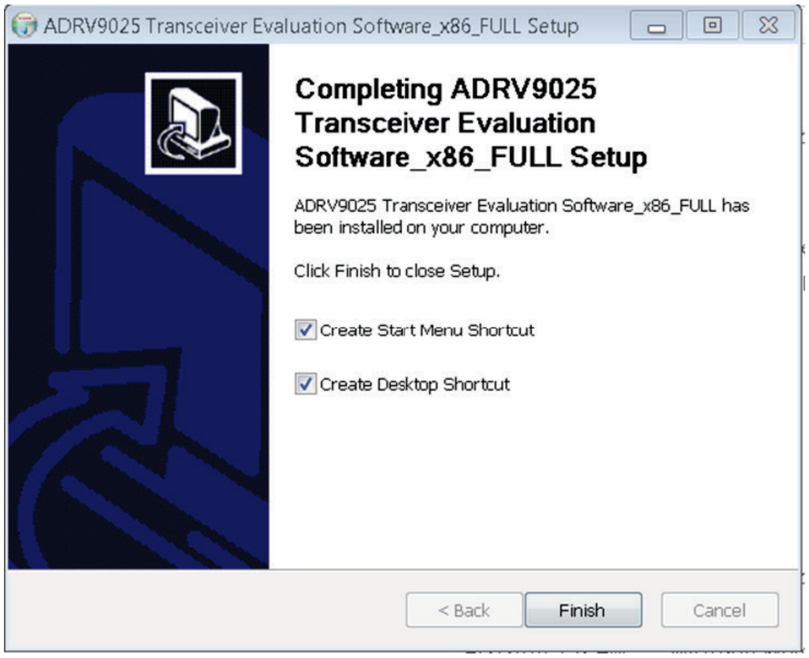
Position your mouse over a component to see its description.

Space required: 197.9 MB

Nullsoft Install System v3.03

Figure 168. Software Installation Components

The last step of the instalation process is to select the shortcut configuration, as shown in Figure 169. The user can select a shortcut to be placed in the Windows **Start** menu and/or on the Windows desktop.


ADRV9025 Transceiver Evaluation Software_x86_FULL Setup

Completing ADRV9025 Transceiver Evaluation Software_x86_FULL Setup

ADRV9025 Transceiver Evaluation Software_x86_FULL has been installed on your computer.

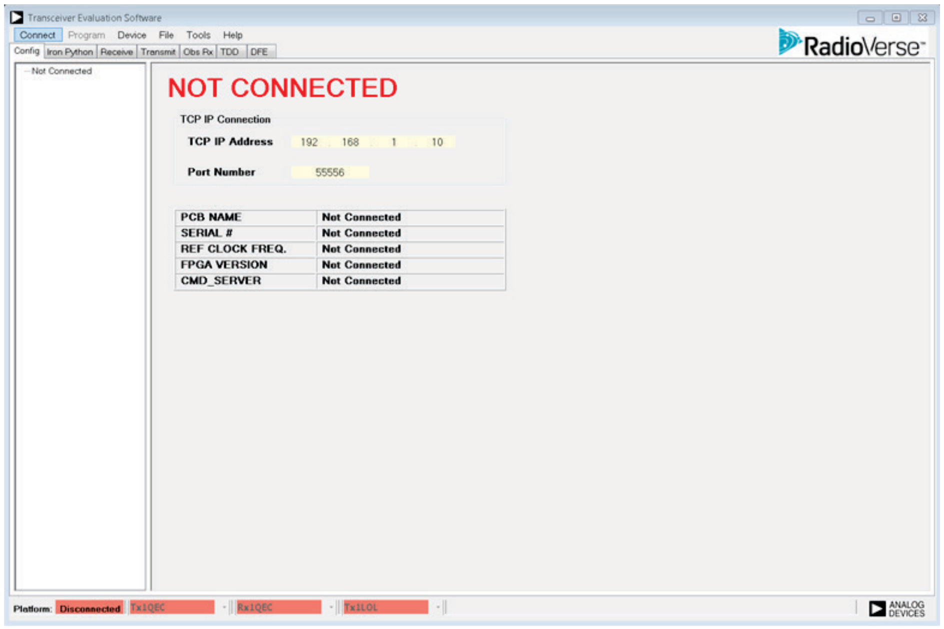
Click Finish to close Setup.

- [✓] Create Start Menu Shortcut
- [✓] Create Desktop Shortcut

Figure 169. Transceiver Evaluation Software Shortcut Configuration

STARTING THE TRANSCEIVER EVALUATION SOFTWARE

Depending on the user selection during the installation process (see Figure 169), 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 labeled **ADRV9025 Transceiver Evaluation Software**. Figure 170 shows the opening page of the TES when the software is activated.



NOT CONNECTED

TCP IP Connection

TCP IP Address 192 . 168 . 1 . 10

Port Number 55556

PCB NAME	Not Connected
SERIAL #	Not Connected
REF CLOCK FREQ.	Not Connected
FPGA VERSION	Not Connected
CMD_SERVER	Not Connected

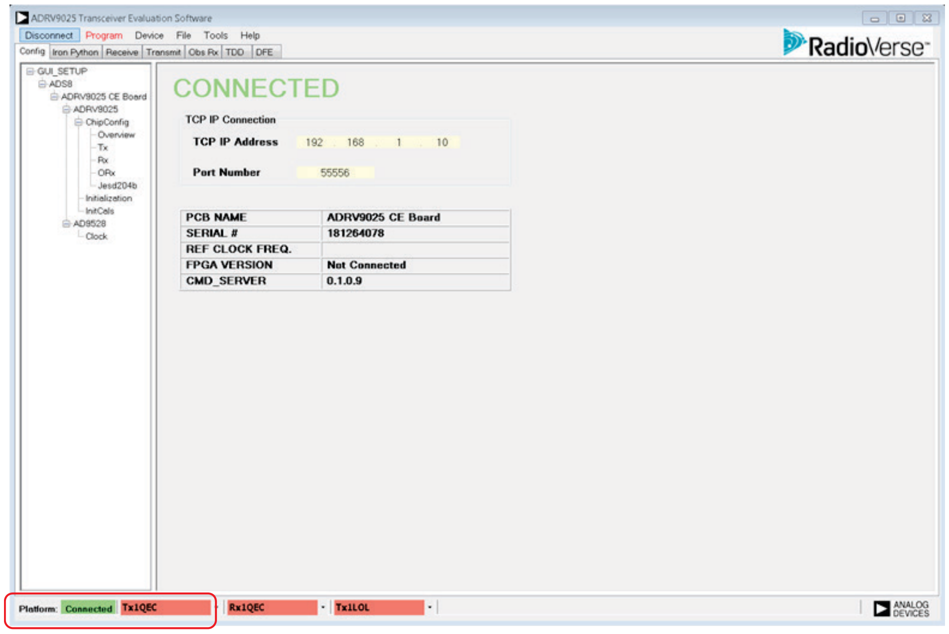
Figure 170. TES Interface

Demo Mode

Figure 170 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 clicking **Connect** (top left corner of the window). 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, the TES establishes a connection with the ADS9 system via Ethernet when the **Connect** option in the dropdown menu is clicked. When proper connection is established, the user can click the **DaughterCard** position in the device tree on the left side of the window. When **DaughterCard** is clicked, 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 the port number set to 55556. Figure 171 shows an example of correct connection between a PC and an ADS9 system with a daughter card connected.



CONNECTED

TCP IP Connection

TCP IP Address 192 . 168 . 1 . 10

Port Number 55556

PCB NAME	ADRV9025 CE Board
SERIAL #	181264078
REF CLOCK FREQ.	
FPGA VERSION	Not Connected
CMD_SERVER	0.1.0.9

Figure 171. Setup Revision Information

Configuring the Device

Contained within the **Config** tab are subtabs that contain setup options for the transceiver. The first subtab displayed is the **Overview** tab. Figure 172 shows the initial window for the device. In this window the user can select the following:

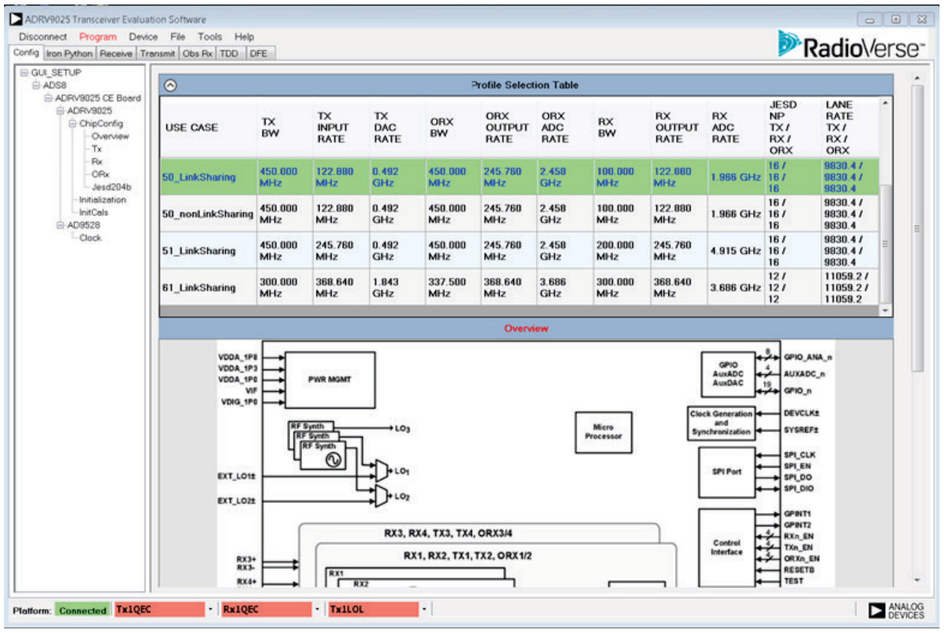
- Device to be programmed
- Select profiles

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 transceiver for different transmit, receive, and observation receive bandwidths, sample rates, and clock rates. The profiles also set different JESD configurations and lane rates. By default, the platform boots to JESD204B mode. Note that the available use cases can vary based on the version of the software being run.


ADRV9025 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Iron Python Receive Transmit Obs Rx TDD DFE
RadioVerse
GUI_SETUP
ADS8
ADRV9025 CE Board
ADRV9025
ChipConfig
Overview
Tx
Rx
ORx
Jesd204b
Initialization
InitCals
AD9528
Clock
Overview
PWR MGMT
GPIO AuxADC AuxDAC
Clock Generation and Synchronization
Micro Processor
SPI Port
Control Interface
RF Synth
RX3, RX4, TX3, TX4, ORX3/4
RX1, RX2, TX1, TX2, ORX1/2
Platform: Connected Tx1QEC Rx1QEC Tx1LOL
ANALOG DEVICES
22770-171

Profile Selection Table

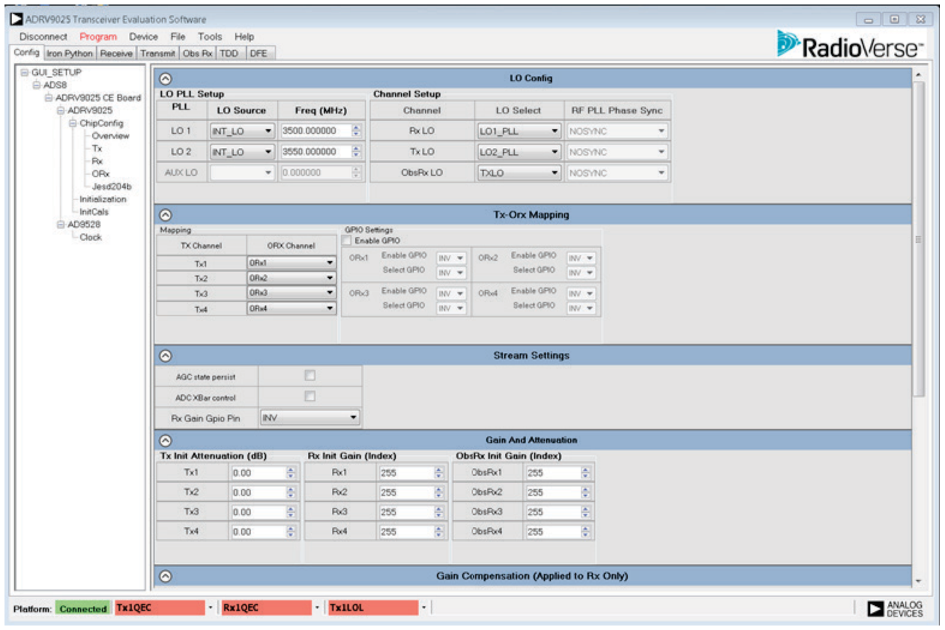
USE CASE	TX BW	TX INPUT RATE	TX DAC RATE	ORX BW	ORX OUTPUT RATE	ORX ADC RATE	RX BW	RX OUTPUT RATE	RX ADC RATE	JESD NP TX / RX / ORX	LANE RATE TX / RX / ORX
50_LinkSharing	450.000 MHz	122.880 MHz	0.492 GHz	450.000 MHz	245.760 MHz	2.458 GHz	100.000 MHz	122.880 MHz	1.966 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4
50_nonLinkSharing	450.000 MHz	122.880 MHz	0.492 GHz	450.000 MHz	245.760 MHz	2.458 GHz	100.000 MHz	122.880 MHz	1.966 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4
51_LinkSharing	450.000 MHz	245.760 MHz	0.492 GHz	450.000 MHz	245.760 MHz	2.458 GHz	200.000 MHz	245.760 MHz	4.915 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4
61_LinkSharing	300.000 MHz	368.640 MHz	1.843 GHz	337.500 MHz	368.640 MHz	3.686 GHz	300.000 MHz	368.640 MHz	3.686 GHz	12 / 12 / 12	11059.2 / 11059.2 / 11059.2

Figure 172. ***Overview*** *Tab*

Additional 204C use cases are also available. Switch the platform to 204C mode and then the available 204C profiles are displayed. To switch the platform, click **Device** > **FPGA switch JESD** > **Jesd 204C**, at which point the platform reboots (which takes approximately 3 minutes). Upon reconnecting, the 204C profiles are available.

Initialization

The **Initialization** subtab provides access to the settings that are used to configure the transceiver at startup. This window allows the user to set the LO frequencies used, the initial transmit attenuation settings, the initial receive gain index settings, and the initial gain index for ORx1 (the only observation receive channel available at this stage in development). These and other settings that are provided in future revisions are shown in Figure 173.



LO Config

LO PLL Setup

PLL	LO Source	Freq (MHz)
LO 1	INT_LO	3500.000000
LO 2	INT_LO	3550.000000
AUX LO		0.000000

Channel Setup

Channel	LO Select	RF PLL Phase Sync
Rx LO	LO1_PLL	NOSYNC
Tx LO	LO2_PLL	NOSYNC
ObsRx LO	TXLO	NOSYNC

Tx-Orx Mapping

Mapping

TX Channel	ORX Channel
Tx1	ORx1
Tx2	ORx2
Tx3	ORx3
Tx4	ORx4

GPIO Settings

☐ Enable GPIO

ORx1	Enable GPIO	INV	ORx2	Enable GPIO	INV
	Select GPIO	INV		Select GPIO	INV
ORx3	Enable GPIO	INV	ORx4	Enable GPIO	INV
	Select GPIO	INV		Select GPIO	INV

Stream Settings

AGC state persist	☐
ADC XBar control	☐
Rx Gain Gpio Pin	INV

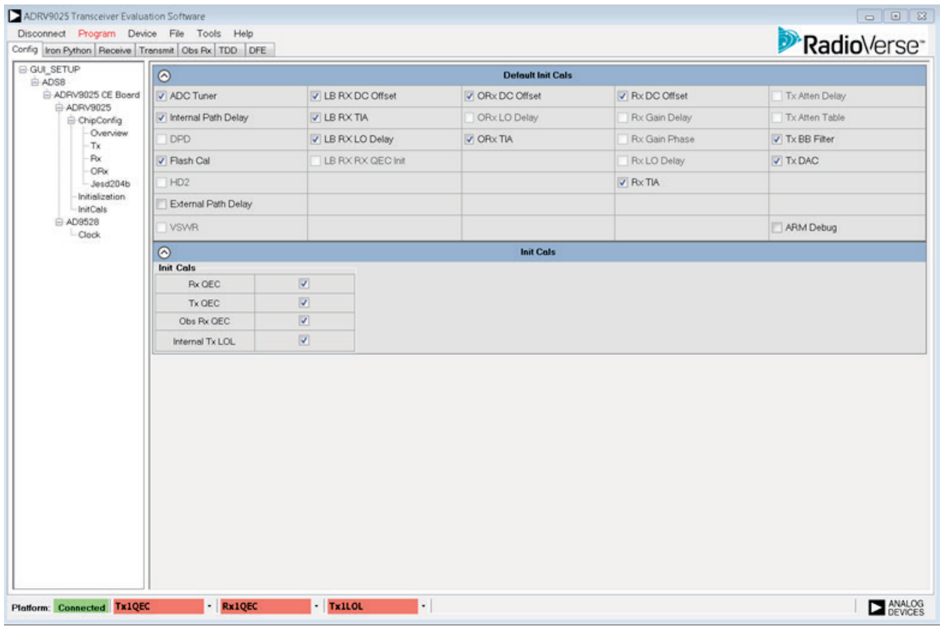
Gain And Attenuation

Tx Init Attenuation (dB)		Rx Init Gain (Index)		ObsRx Init Gain (Index)	
Tx1	0.00	Rx1	255	ObsRx1	255
Tx2	0.00	Rx2	255	ObsRx2	255
Tx3	0.00	Rx3	255	ObsRx3	255
Tx4	0.00	Rx4	255	ObsRx4	255

Figure 173. ***Initialization*** *Configuration Tab*

InitCals

The **InitCals** subtab sets which calibrations take place at initialization. The default settings are shown in Figure 174. To enable or disable these settings, select the checkbox next to the calibration. A check mark indicates that the calibration is run at startup.



Default Init Cals

✓ ADC Tuner	✓ LB RX DC Offset	✓ ORx DC Offset	✓ Rx DC Offset	☐ Tx Atten Delay
✓ Internal Path Delay	✓ LB RX TIA	☐ ORx LO Delay	☐ Rx Gain Delay	☐ Tx Atten Table
☐ DPD	✓ LB RX LO Delay	✓ ORx TIA	☐ Rx Gain Phase	✓ Tx BB Filter
✓ Flash Cal	☐ LB RX RX QEC Init		☐ Rx LO Delay	✓ Tx DAC
☐ HD2			✓ Rx TIA	
☐ External Path Delay				
☐ VSWR				☐ ARM Debug

Init Cals

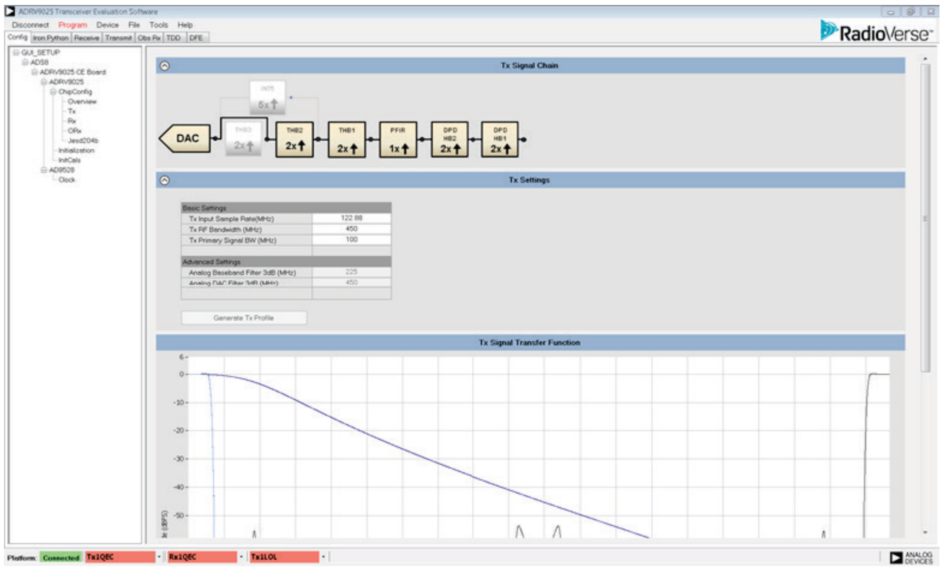
Init Cals

Rx QEC	✓
Tx QEC	✓
Obs Rx QEC	✓
Internal Tx LOL	✓

Figure 174. ***InitCals*** *Tab with Default Settings*

Transmit Configuration

The **Tx** tab is primarily informative and is based on the profile selection in the **Overview** tab (Figure 172). 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 175 shows an example of the **Tx** tab with the resulting composite filter response for the chosen profile.

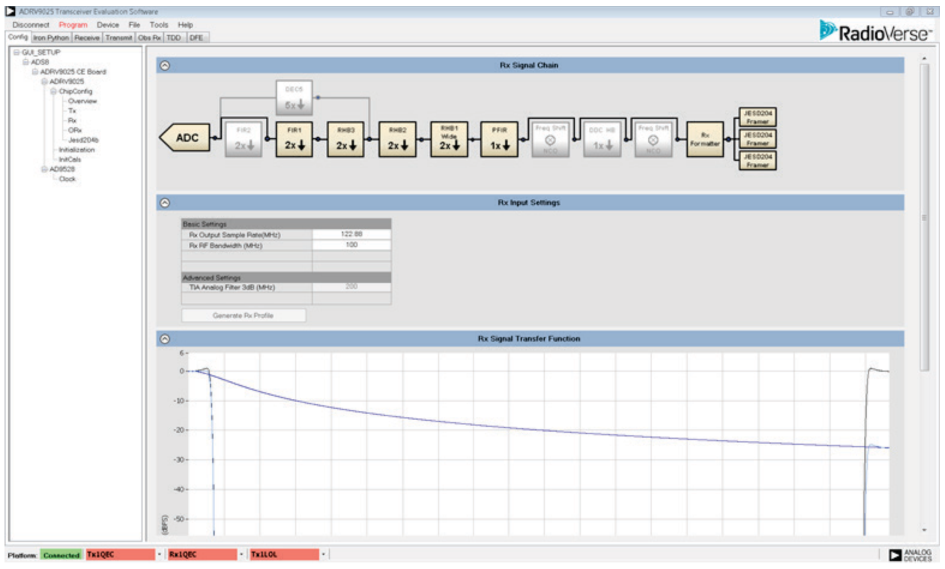


Basic Settings	
Tx Input Sample Rate(MHz)	122.88
Tx RF Bandwidth (MHz)	450
Tx Primary Signal BW (MHz)	100
Advanced Settings	
Analog Baseband Filter 3dB (MHz)	225
Analog DAC Filter 3dB (MHz)	450

Figure 175. **Tx** Summary Tab

Receive Configuration

The **Rx** tab is primarily informative and is based on the profile selection in the **Overview** tab (see Figure 172). 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 176 shows an example of the **Rx** tab with the resulting composite filter response for the chosen profile.

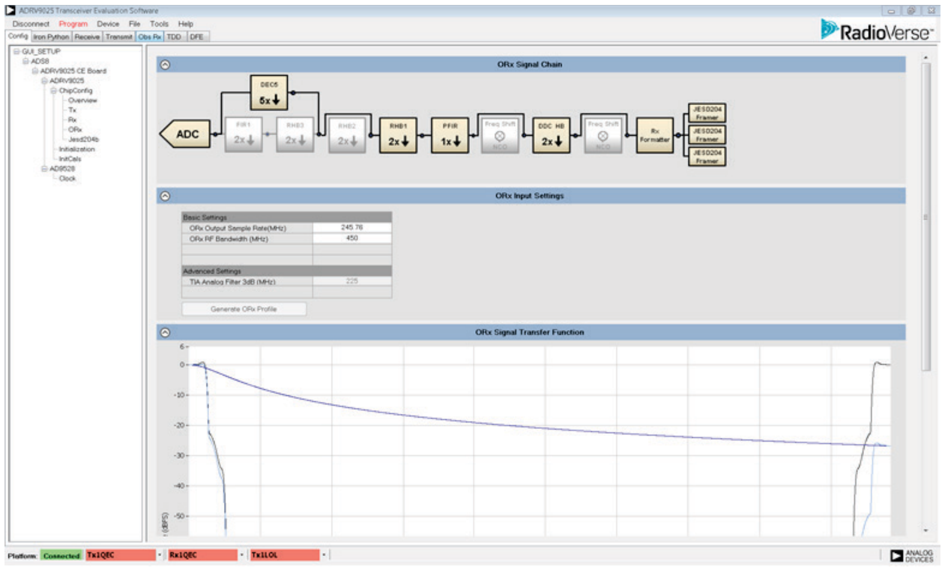


Basic Settings	
Rx Output Sample Rate(MHz)	122.88
Rx RF Bandwidth (MHz)	100
Advanced Settings	
TIA Analog Filter 3dB (MHz)	200

Figure 176. **Rx** Summary Tab

Observation Receive Configuration

The **ORx** tab is primarily informative and is based on the profile selection in the Overview tab (Figure 172). 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 177 shows an example of the **ORx** tab with the resulting composite filter response for the chosen profile.

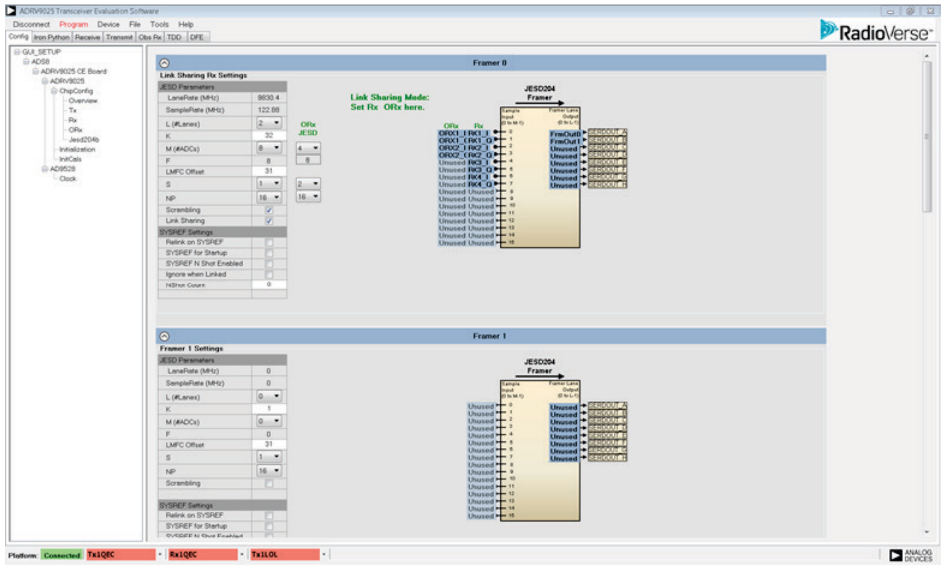


Basic Settings	
ORx Output Sample Rate(MHz)	245.76
ORx RF Bandwidth (MHz)	450
Advanced Settings	
TIA Analog Filter 3dB (MHz)	225

Figure 177. **ORx** Summary Tab

JESD Configuration

The **JESDb** tab is primarily informative and is based on the profile selection in the **Overview** tab (see Figure 172). In this tab, the user can check the transmit deframer settings and the receive and observation receive framer settings. Figure 178 shows an example of the **JESDb** tab with the settings for Use Case 13.



Link Sharing Rx Settings

JESD Parameters		ORx JESD
LaneRate (MHz)	9830.4	
SampleRate (MHz)	122.88	
L (#Lanes)	2	
K	32	
M (#ADCs)	8	4
F	8	8
LMFC Offset	31	
S	1	2
NP	16	16
Scrambling	✓	
Link Sharing	✓	
SYSREF Settings		
Relink on SYSREF		
SYSREF for Startup		
SYSREF N Shot Enabled		
Ignore when Linked		
NShot Count	0	

Framer 1 Settings

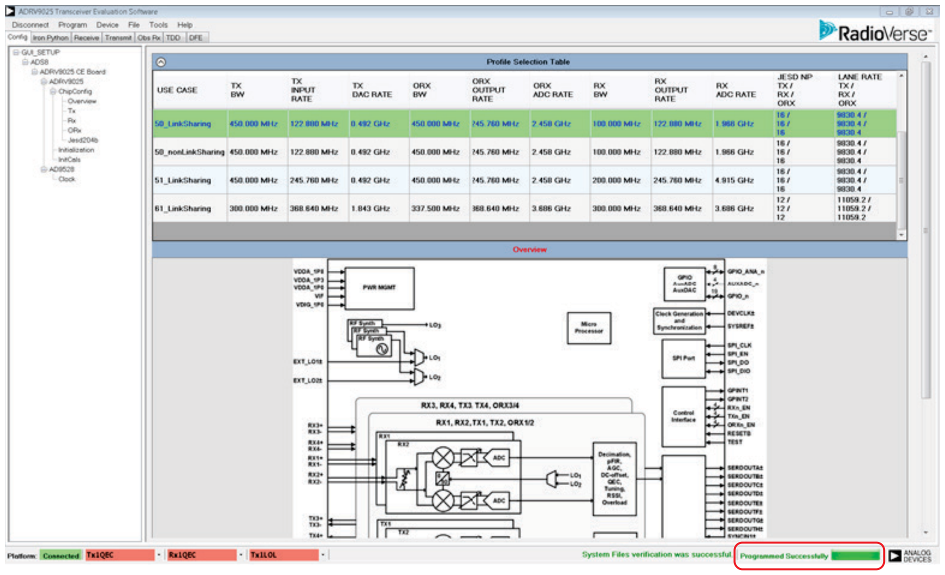
JESD Parameters	
LaneRate (MHz)	0
SampleRate (MHz)	0
L (#Lanes)	0
K	1
M (#ADCs)	0
F	0
LMFC Offset	31
S	1
NP	16
Scrambling	
SYSREF Settings	
Relink on SYSREF	
SYSREF for Startup	

Figure 178. JESD204B Summary Tab

Programming the Evaluation System

When all tabs are configured, press the Program button in the top menu line (highlighted in red text) to start initialization programming. The TES sends a series of API commands that are executed by a dedicated Linux application on the ADS9 platform.

When programming is complete, the system is ready to operate. There is a progress bar at the bottom of the window. Figure 179 shows the window with the progress bar and message when the device has been programmed.


Profile Selection Table

USE CASE	TX BW	TX INPUT RATE	TX DAC RATE	ORX BW	ORX OUTPUT RATE	ORX ADC RATE	RX BW	RX OUTPUT RATE	RX ADC RATE	JESD NP TX / RX / ORX	LANE RATE TX / RX / ORX
50_LinkSharing	450.000 MHz	122.880 MHz	0.492 GHz	450.000 MHz	245.760 MHz	2.458 GHz	100.000 MHz	122.880 MHz	1.966 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4
50_nonLinkSharing	450.000 MHz	122.880 MHz	0.492 GHz	450.000 MHz	245.760 MHz	2.458 GHz	100.000 MHz	122.880 MHz	1.966 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4
51_LinkSharing	450.000 MHz	245.760 MHz	0.492 GHz	450.000 MHz	245.760 MHz	2.458 GHz	200.000 MHz	245.760 MHz	4.915 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4
61_LinkSharing	300.000 MHz	368.640 MHz	1.843 GHz	337.500 MHz	368.640 MHz	3.686 GHz	300.000 MHz	368.640 MHz	3.686 GHz	12 / 12 / 12	11059.2 / 11059.2 / 11059.2

System Files verification was successful

Programmed Successfully

Figure 179. Device Programmed

Initialization Script

The 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 chosen location for future use. The 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 185.

The commands ultilized by the TES to initialize the transceiver are different than those described in the System Initialization section. These commands perform the same underlying task as the API initialization procedure. The three main commands used to initialize using the TES are shown in Figure 180. Help for these commands can be found in the client dynamic link library (DLL) help file. The first command loads the profile, the second command configures the AD9528, and the third command runs all the initialization API commands. It is not possible to initialize the transceiver in any other way using the TES platform.

```

    # MAKE SURE ARM/STREAM FILES ARE PROGRAMMED. PROGRAM THE DEVICE USING GUI TO LOAD DEFAULT ARM/STREAM FILES.

    link.platform.board.Adrv9010Device.ConfigFileLoad()

    # Ad9528 Config
    link.platform.board.ClockConfig(245760, 245760, 245760, 245760)

    # Program the Part
    link.platform.board.Program()

```

Figure 180 Iron Python Initialization Script Example

TRANSMITTER OPERATION

Select the **Transmit** tab to open a page as shown in Figure 181. The upper plot displays the FFT of the digital input data and the lower plot shows its time domain waveform. When multiple transmit 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.

When the **Transmit** tab opens, the user can enter the RF transmit center frequency in MHz for transmit LO1 (used for transmit operation), change attenuation level, and transmit continuous wave tones.

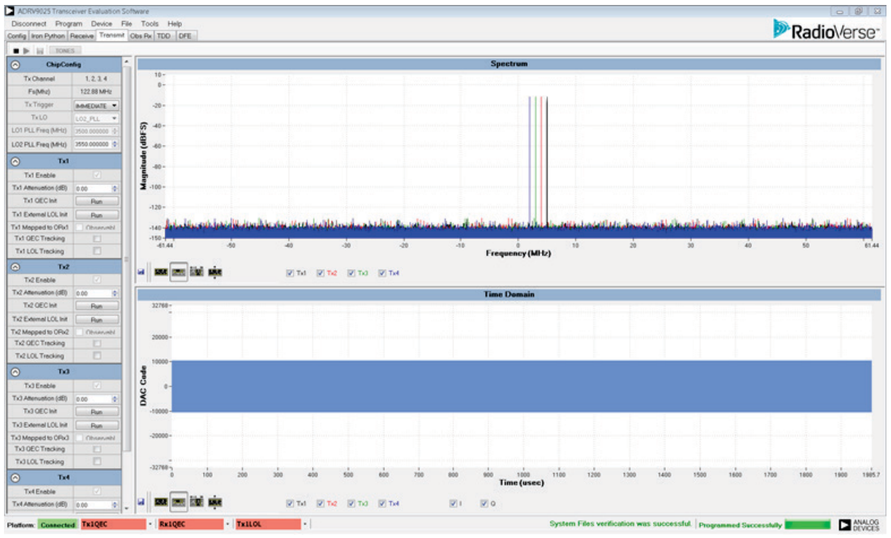
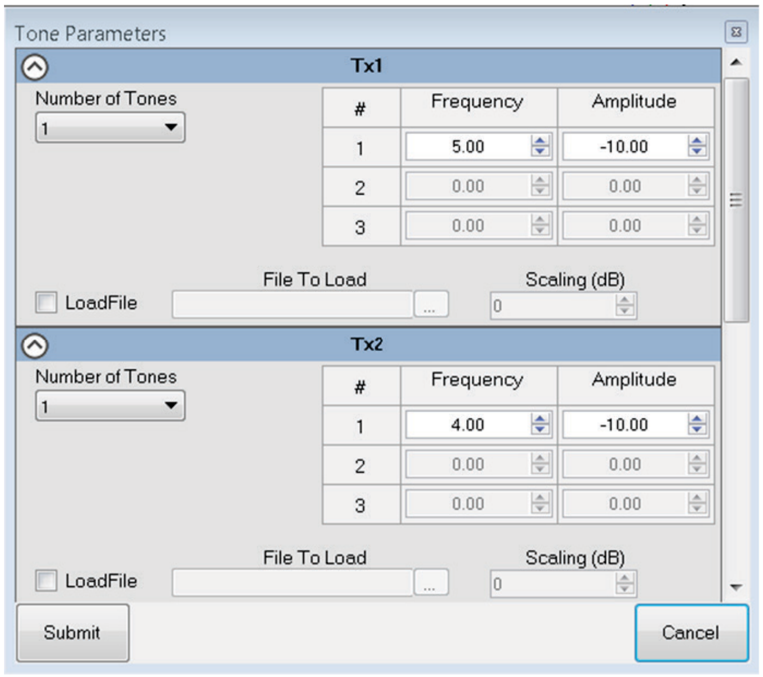


Figure 181. Transmit Data Tab

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** window shown in Figure 182. TO acces this window, click the **TONES** button near the upper left of the **Transmit** page. In that window, the user can enable the tone (**Number of Tones** value = 0 to 3) to be transmitted on the selected transmit output.
- The user can also choose to input a waveform file instead of using the internal NCO by selecting the **LoadFile** checkbox and entering the path to the waveform file.



Tone Parameters

Tx1

Number of Tones: 1

#	Frequency	Amplitude
1	5.00	-10.00
2	0.00	0.00
3	0.00	0.00

LoadFile ☐ File To Load: Scaling (dB): 0

Tx2

Number of Tones: 1

#	Frequency	Amplitude
1	4.00	-10.00
2	0.00	0.00
3	0.00	0.00

LoadFile ☐ File To Load: Scaling (dB): 0

*Figure 182. Transmit **Tone Parameters** Setup Menu*

Press the play symbol in Figure 181 to move the transceiver to the transmit state and start a process where the NCO generated continuous wave data is enabled.

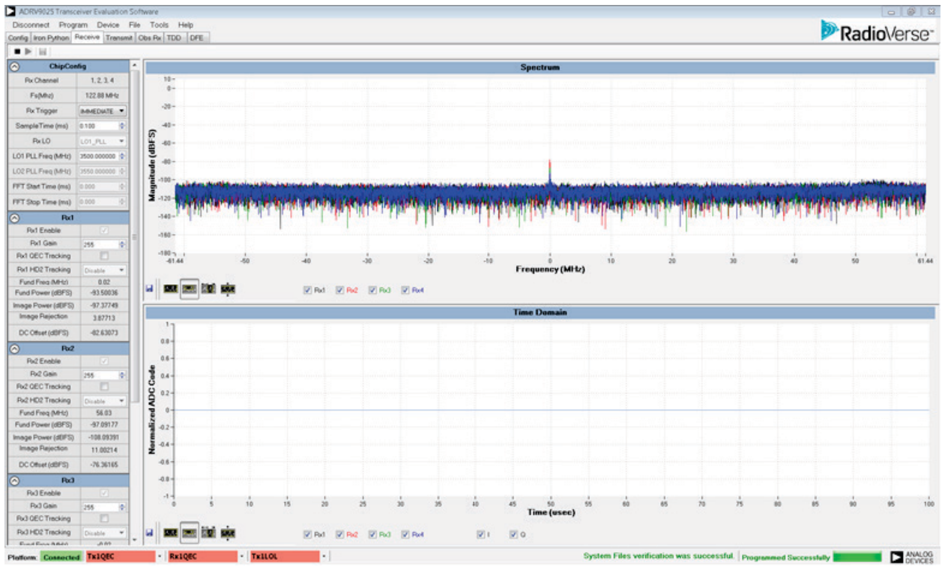
The **Tx2 Attenuation (dB)** input allows the user to control analog attenuation in the Tx2 channel. The input provides 0.05 dB of attenuation control accuracy. The **Tx3 Attenuation (dB)** and **Tx4 Attenuation (dB)** perform the same operation on the Tx3 and Tx4 channels.

RECEIVER OPERATION

Receive Signal Chain

The **Receive** tab opens the window shown in Figure 183. The upper plot displays the FFT of the received input data and the lower plot shows its time domain waveform. When multiple receive 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.

When the **Receive** tab is open, the user can enter the RF LO frequencies for the LO1 and LO2 PLLs, select which LO is used by Rx1 and Rx2, select which LO is used by Rx3 and Rx4, select the receive trigger type, enter the sample time for the data, and select the gain levels and tracking calibrations for each channel. Press the play symbol to enable the selected receivers and display their waveform data.



ChipConfig	
Rx Channel	1, 2, 3, 4
Fs(Mhz)	122.88 MHz
Rx Trigger	IMMEDIATE
SampleTime (ms)	0.100
Rx LO	LO1_PLL
LO1 PLL Freq (MHz)	3500.000000
LO2 PLL Freq (MHz)	3550.000000
FFT Start Time (ms)	0.000
FFT Stop Time (ms)	0.000
Rx1	
Rx1 Enable	✓
Rx1 Gain	255
Rx1 QEC Tracking	☐
Rx1 HD2 Tracking	Disable
Fund Freq (MHz)	0.02
Fund Power (dBFS)	-93.50036
Image Power (dBFS)	-97.37749
Image Rejection	3.87713
DC Offset (dBFS)	-82.63073
Rx2	
Rx2 Enable	✓
Rx2 Gain	255
Rx2 QEC Tracking	☐
Rx2 HD2 Tracking	Disable
Fund Freq (MHz)	56.03
Fund Power (dBFS)	-97.09177
Image Power (dBFS)	-108.09391
Image Rejection	11.00214
DC Offset (dBFS)	-76.36165
Rx3	
Rx3 Enable	✓
Rx3 Gain	255
Rx3 QEC Tracking	☐
Rx3 HD2 Tracking	Disable

System Files verification was successful
Programmed Successfully

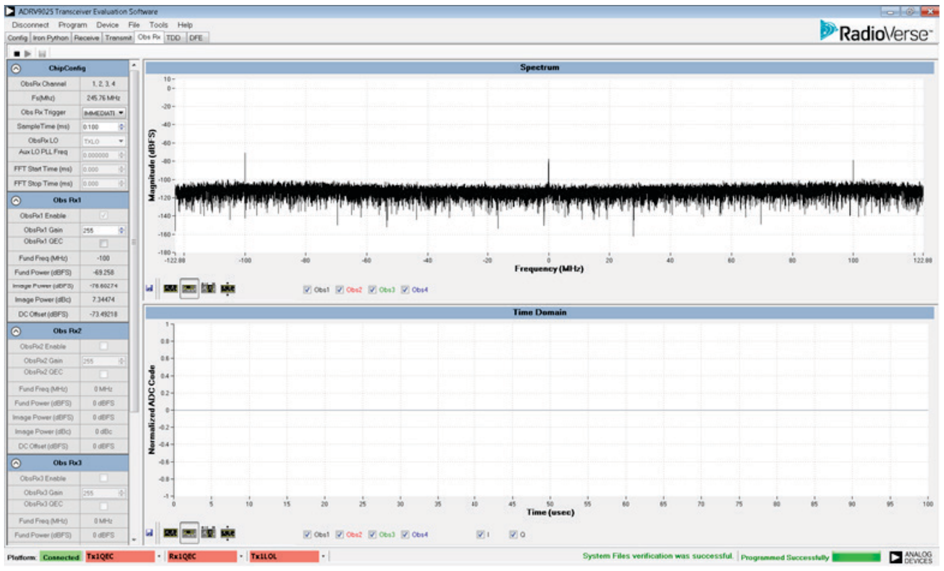
Figure 183. Receive Data Tab

Observation Receive Signal Chain

The **Obs Rx** tab opens the window shown in Figure 184. The upper plot displays the FFT of the received input data and the lower plot shows its time domain waveform. When multiple observation receive 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.

When the **Obs Rx** tab is open, the user can enter the RF LO frequencies for the 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 observation receive trigger type, enter the sample time for the data, and select the gain levels and QEC for each channel. Press the play symbol to enable the selected receivers and display their waveform data.

Note that only Rx1 is enabled for use at this stage in product development. All four channels are available in future software revisions.


ADRV9025 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Iron Python Receive Transmit Obs Rx TDD DFE
RadioVerse
Spectrum
Magnitude (dBFS)
Frequency (MHz)
Time Domain
Normalized ADC Code
Time (usec)
Obs1 Obs2 Obs3 Obs4
I Q
22770-182

ChipConfig	
ObsRx Channel	1, 2, 3, 4
Fs(MHz)	245.76 MHz
Obs Rx Trigger	IMMEDIATI
SampleTime (ms)	0.100
ObsRx LO	TXLO
Aux LO PLL Freq	0.000000
FFT Start Time (ms)	0.000
FFT Stop Time (ms)	0.000
Obs Rx1	
ObsRx1 Enable	✓
ObsRx1 Gain	255
ObsRx1 QEC	
Fund Freq (MHz)	-100
Fund Power (dBFS)	-69.258
Image Power (dBFS)	-76.60274
Image Power (dBc)	7.34474
DC Offset (dBFS)	-73.49218
Obs Rx2	
ObsRx2 Enable	
ObsRx2 Gain	255
ObsRx2 QEC	
Fund Freq (MHz)	0 MHz
Fund Power (dBFS)	0 dBFS
Image Power (dBFS)	0 dBFS
Image Power (dBc)	0 dBc
DC Offset (dBFS)	0 dBFS
Obs Rx3	
ObsRx3 Enable	
ObsRx3 Gain	255
ObsRx3 QEC	
Fund Freq (MHz)	0 MHz
Fund Power (dBFS)	0 dBFS

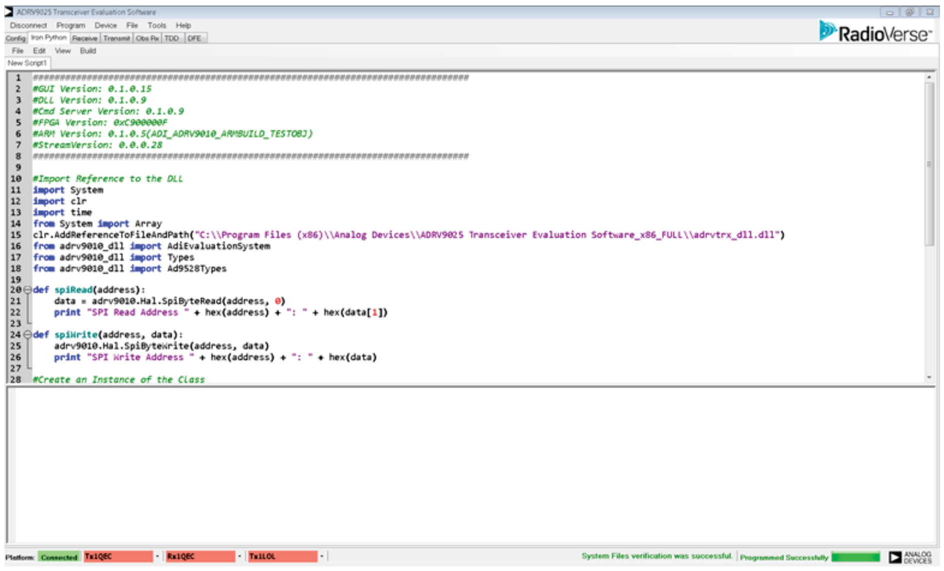
Platform: Connected Tx1QEC Rx1QEC Tx1LOL System Files verification was successful. Programmed Successfully

*Figure 184. **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 185 shows the **Iron Python** tab after executing the **File** > **New** script function. The top part of the window contains the IronPython commands while the bottom part of the window displays the script output. Scripts are run by clicking **Build** > **Run**. To save scripts that provide useful functions that may be useful in the future, click **File** > **Save** and enter the path and file name for saving the script.

When 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. Click **Tools** > **Create Script** > **Python** to accomplish this task. This function generates a script with the initialization sequence and opens a dialogue box to save the file. Basic script with no initializaion sequence can be generated by clicking **File** > **New** option.


ADRV9025 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Iron Python Receive Transmit Obs Rx TDD DFE
File Edit View Build
New Script1
RadioVerse
22770-183

```
##########################################################################################
#GUI Version: 0.1.0.15
#DLL Version: 0.1.0.9
#Cmd Server Version: 0.1.0.9
#FPGA Version: 0xC900000F
#ARM Version: 0.1.0.5(ADI_ADRV9010_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 adrv9010_dll import AdiEvaluationSystem
from adrv9010_dll import Types
from adrv9010_dll import Ad9528Types

def spiRead(address):
    data = adrv9010.Hal.SpiByteRead(address, 0)
    print "SPI Read Address " + hex(address) + ": " + hex(data[1])

def spiWrite(address, data):
    adrv9010.Hal.SpiByteWrite(address, data)
    print "SPI Write Address " + hex(address) + ": " + hex(data)

#Create an Instance of the Class
```

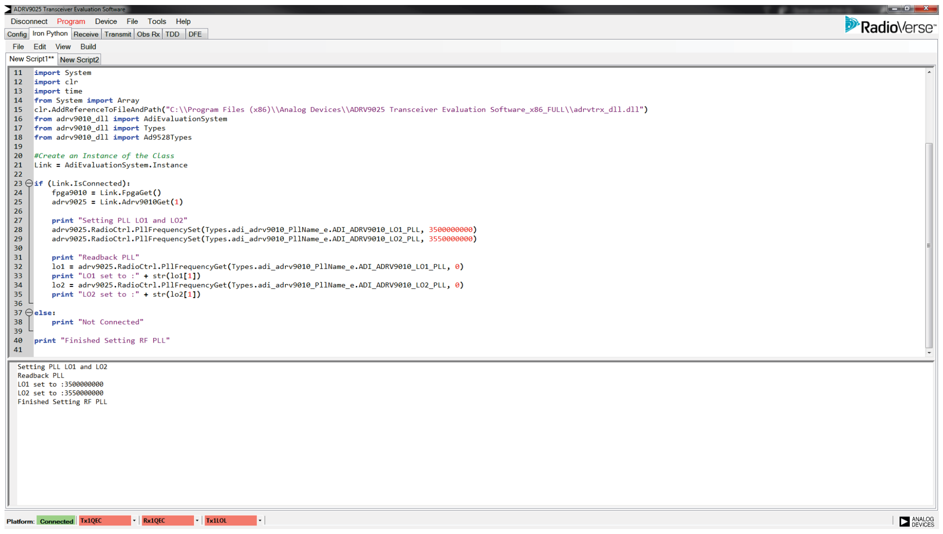
Platform: Connected Tx1QEC Rx1QEC Tx1LOL System Files verification was successful. Programmed Successfully

Figure 185. 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"
```


ADRV9025 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Iron Python Receive Transmit Obs Rx TDD DFE
File Edit View Build
New Script1** New Script2
RadioVerse
Platform: Connected Tx1QEC Rx1QEC Tx1LOL
22770-184

```
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 adrv9010_dll import AdiEvaluationSystem
from adrv9010_dll import Types
from adrv9010_dll import Ad9528Types

#Create an Instance of the Class
Link = AdiEvaluationSystem.Instance

if (Link.IsConnected):
    fpga9010 = Link.FpgaGet()
    adrv9025 = Link.Adrv9010Get(1)

    print "Setting PLL LO1 and LO2"
    adrv9025.RadioCtrl.PllFrequencySet(Types.adi_adrv9010_PllName_e.ADI_ADRV9010_LO1_PLL, 3500000000)
    adrv9025.RadioCtrl.PllFrequencySet(Types.adi_adrv9010_PllName_e.ADI_ADRV9010_LO2_PLL, 3550000000)

    print "Readback PLL"
    lo1 = adrv9025.RadioCtrl.PllFrequencyGet(Types.adi_adrv9010_PllName_e.ADI_ADRV9010_LO1_PLL, 0)
    print "LO1 set to :" + str(lo1[1])
    lo2 = adrv9025.RadioCtrl.PllFrequencyGet(Types.adi_adrv9010_PllName_e.ADI_ADRV9010_LO2_PLL, 0)
    print "LO2 set to :" + str(lo2[1])

else:
    print "Not Connected"

print "Finished Setting RF PLL"
```

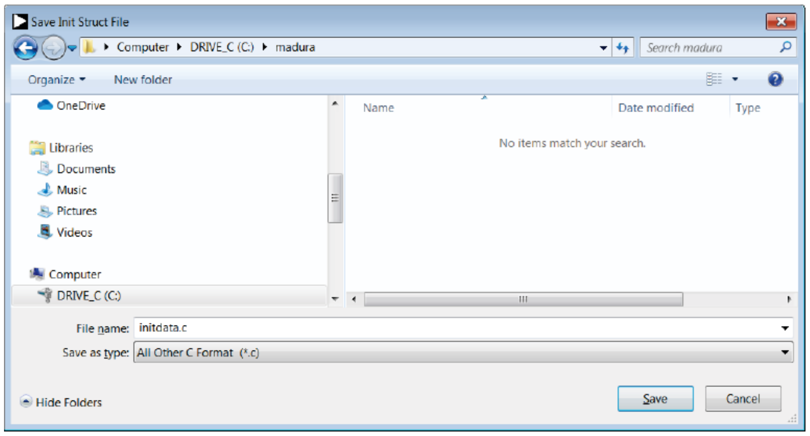
```
Setting PLL LO1 and LO2
Readback PLL
LO1 set to :3500000000
LO2 set to :3550000000
Finished Setting RF PLL
```

Figure 186. Python Script Example

When using the **Iron Python** tab window, the user can execute any API command that is available in the loaded software build.

C CODE GENERATION

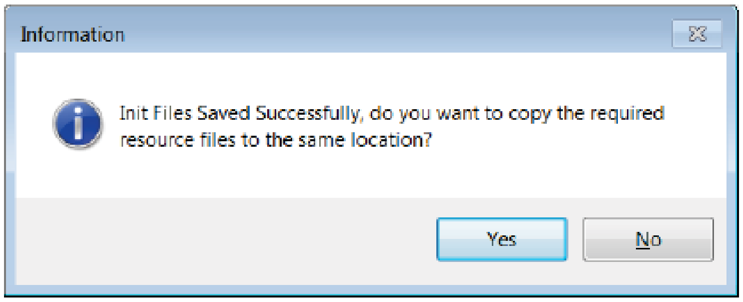
It is possible to generate C initialization structure from the GUI. To generate this code, click **Tools** > **Create Script** > **Init c files**.



File name: initdata.c
Save as type: All Other C Format (*.c)

Figure 187. C Initialization Structure Save Screen

When this option is selected, the GUI opens a dialogue box to select a location and file name to store this code. The preferred file name is **initdata.c**. The user can choose to store resource files at the same location or another location by clicking **Yes** or **No** on the prompt (see Figure 188).

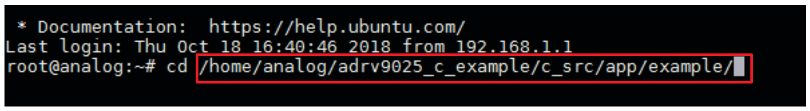


Init Files Saved Successfully, do you want to copy the required resource files to the same location?

Figure 188. C Initialization Structure File Storage Confirmation Screen

To use this code, take the following steps:

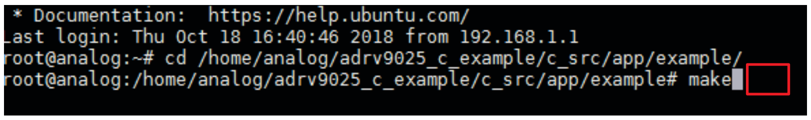
1. Copy the c_src folder from Adi.ADRV9025.Api\public\src to the /home/analog/adrv9025_c_example/ location on the platform. (create the adrv9025_c_example directory if not present).
2. Copy the generated resources folder to the platform at the same /home/analog/adrv9025_c_example/ location.
3. Copy the generated files, initdata.h, initdata.c, and main.c, to /home/analog/adrv9025_c_example/c_src/app/example/.
4. Use the terminal to navigate to the example directory and run following the command to enter the directory (see Figure 189).



```
 * Documentation:  https://help.ubuntu.com/
Last login: Thu Oct 18 16:40:46 2018 from 192.168.1.1
root@analog:~# cd /home/analog/adrv9025_c_example/c_src/app/example/
```

Figure 189. Linux Command Line to Enter Example Directory

5. Run the make command at this location to compile the code. If no errors occur, this command generates an executable called main in /home/analog/adrv9025_c_example/c_src/app/example/.



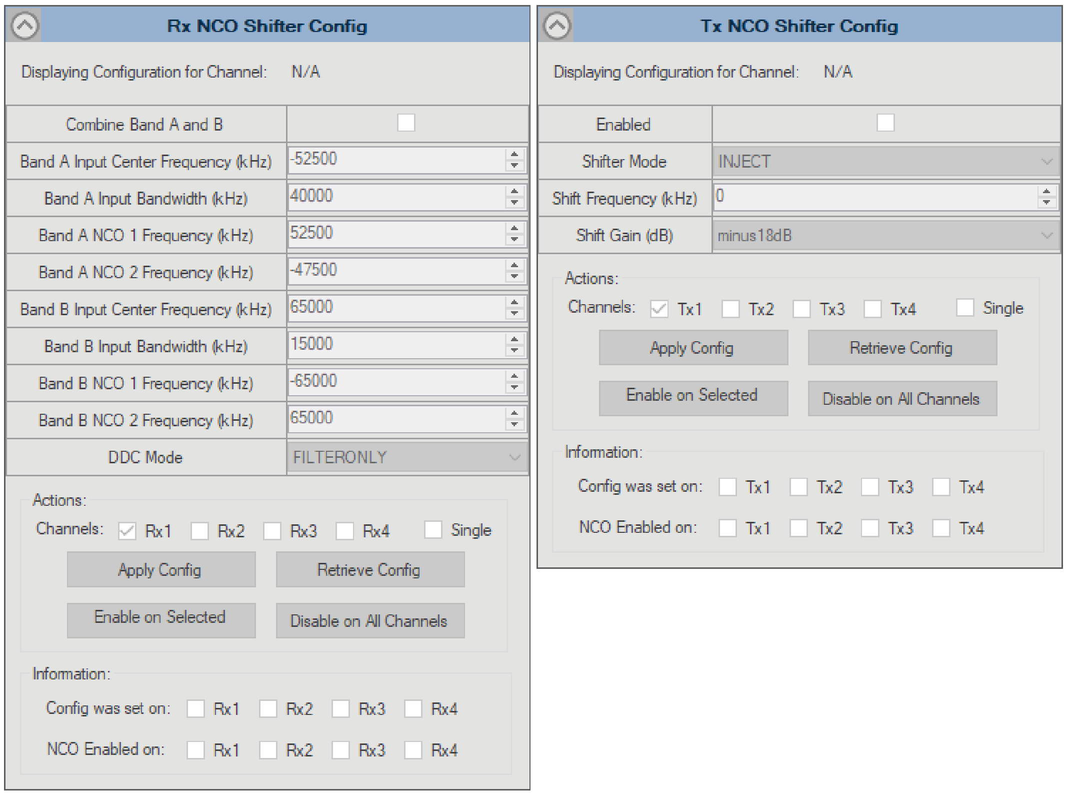
```
 * Documentation:  https://help.ubuntu.com/
Last login: Thu Oct 18 16:40:46 2018 from 192.168.1.1
root@analog:~# cd /home/analog/adrv9025_c_example/c_src/app/example/
root@analog:/home/analog/adrv9025_c_example/c_src/app/example# make
```

Figure 190. Linux Command Line make Execution Example

6. Run the ./main command in the same folder to run the initialization sequence.

NCO SETUP

The NCO is a digital block that can be used to provide an offset center frequency in the digital domain. This function can be used to offset signals in the frequency domain to allow multiple frequency bands to be processed when received on a single channel. The setup of these functions can be configured in the TES using the **NCO** tab Figure 191 shows the options and settings available for the NCO control for both receivers and transmitters.



Rx NCO Shifter Config

Displaying Configuration for Channel: N/A

Combine Band A and B	☐
Band A Input Center Frequency (kHz)	-52500
Band A Input Bandwidth (kHz)	40000
Band A NCO 1 Frequency (kHz)	52500
Band A NCO 2 Frequency (kHz)	-47500
Band B Input Center Frequency (kHz)	65000
Band B Input Bandwidth (kHz)	15000
Band B NCO 1 Frequency (kHz)	-65000
Band B NCO 2 Frequency (kHz)	65000
DDC Mode	FILTERONLY

Actions:
Channels: ✓ Rx1 ☐ Rx2 ☐ Rx3 ☐ Rx4 ☐ Single

Information:
Config was set on: ☐ Rx1 ☐ Rx2 ☐ Rx3 ☐ Rx4
NCO Enabled on: ☐ Rx1 ☐ Rx2 ☐ Rx3 ☐ Rx4

Tx NCO Shifter Config

Displaying Configuration for Channel: N/A

Enabled	☐
Shifter Mode	INJECT
Shift Frequency (kHz)	0
Shift Gain (dB)	minus18dB

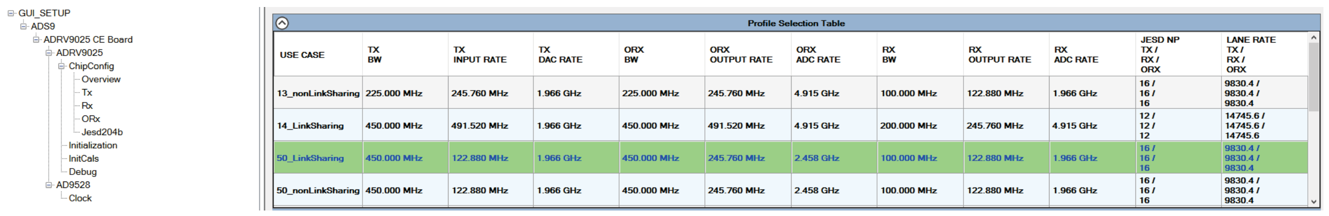
Actions:
Channels: ✓ Tx1 ☐ Tx2 ☐ Tx3 ☐ Tx4 ☐ Single

Information:
Config was set on: ☐ Tx1 ☐ Tx2 ☐ Tx3 ☐ Tx4
NCO Enabled on: ☐ Tx1 ☐ Tx2 ☐ Tx3 ☐ Tx4

Figure 191. NCO Setup Options

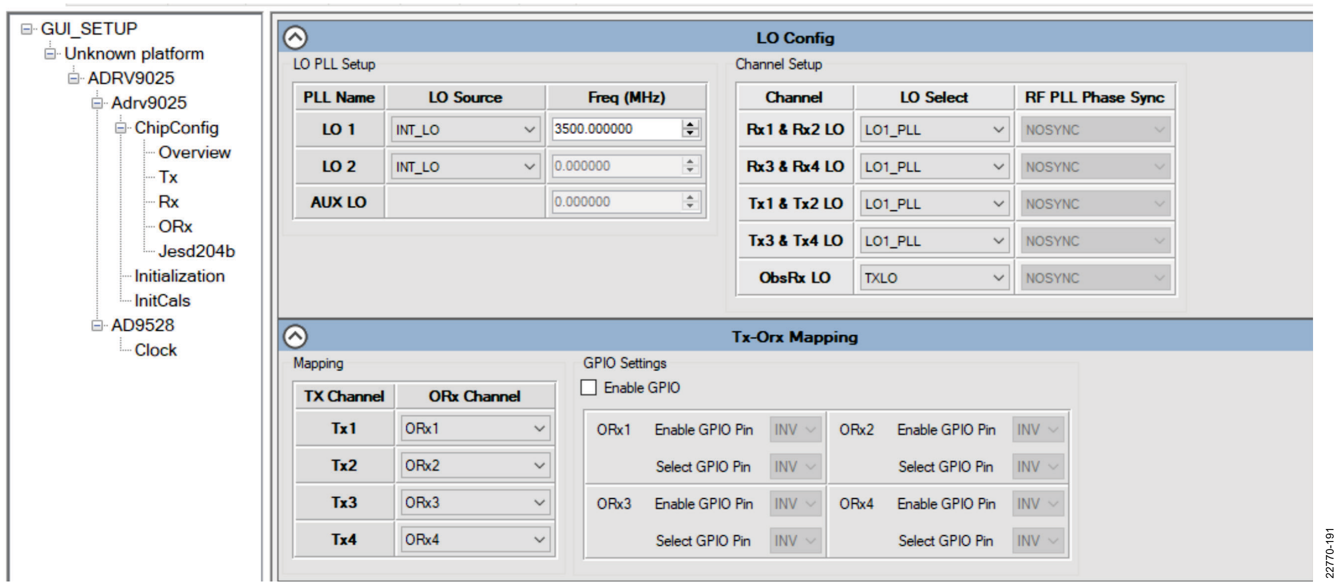
DIGITAL FRONT END TAB SETUP

The ADRV9029 variant provides additional digital front end (DFE) functions to linearize power amplifier performance using digital predistortion and cress factor reduction (see the Digital Predistortion and Crest Factor Reduction (CFR) sections for details). The TES determines which variant is connected to the system when the platform is connected. If the ADRV9029 is detected, the TES automatically allows access to the DFE features tab for setup and evaluation. To use these features, select a use case that includes DPD capability prior to programming. In the **Overview** tab, highlight a use case profile such as the 50_LinkSharing option. This use case supports a transmitter sample rate of 122.88 Msps and a DPD actuator rate of 491.52 Msps, which translates into a 450 MHz DPD bandwidth. When this setting is selected, switch to the **Initialization** tab and ensure that all transmitter and receiver channels are running with the same LO. On this menu, setup the transmit channel to observation receive channel mapping and ensure that the observation receive LO is set to TXLO. Figure 192 and Figure 193 illustrate where these selections are made. When these settings are confirmed, click on the **Program** menu option to program the device with the appropriate settings.



USE CASE	TX BW	TX INPUT RATE	TX DAC RATE	ORX BW	ORX OUTPUT RATE	ORX ADC RATE	RX BW	RX OUTPUT RATE	RX ADC RATE	JESD NP TX / RX / ORX	LANE RATE TX / RX / ORX
13_nonLinkSharing	225.000 MHz	245.760 MHz	1.966 GHz	225.000 MHz	245.760 MHz	4.915 GHz	100.000 MHz	122.880 MHz	1.966 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4
14_LinkSharing	450.000 MHz	491.520 MHz	1.966 GHz	450.000 MHz	491.520 MHz	4.915 GHz	200.000 MHz	245.760 MHz	4.915 GHz	12 / 12 / 12	14745.6 / 14745.6 / 14745.6
50_LinkSharing	450.000 MHz	122.880 MHz	1.966 GHz	450.000 MHz	245.760 MHz	2.458 GHz	100.000 MHz	122.880 MHz	1.966 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4
50_nonLinkSharing	450.000 MHz	122.880 MHz	1.966 GHz	450.000 MHz	245.760 MHz	2.458 GHz	100.000 MHz	122.880 MHz	1.966 GHz	16 / 16 / 16	9830.4 / 9830.4 / 9830.4

Figure 192. TES **Overview** Selection Tab



LO PLL Setup

PLL Name	LO Source	Freq (MHz)
LO 1	INT_LO	3500.000000
LO 2	INT_LO	0.000000
AUX LO		0.000000

Channel Setup

Channel	LO Select	RF PLL Phase Sync
Rx1 & Rx2 LO	LO1_PLL	NOSYNC
Rx3 & Rx4 LO	LO1_PLL	NOSYNC
Tx1 & Tx2 LO	LO1_PLL	NOSYNC
Tx3 & Tx4 LO	LO1_PLL	NOSYNC
ObsRx LO	TXLO	NOSYNC

Mapping

TX Channel	ORx Channel
Tx1	ORx1
Tx2	ORx2
Tx3	ORx3
Tx4	ORx4

GPIO Settings

☐ Enable GPIO

ORx1	Enable GPIO Pin	INV	ORx2	Enable GPIO Pin	INV
	Select GPIO Pin	INV		Select GPIO Pin	INV
ORx3	Enable GPIO Pin	INV	ORx4	Enable GPIO Pin	INV
	Select GPIO Pin	INV		Select GPIO Pin	INV

Figure 193. TES **Initialization** Tab

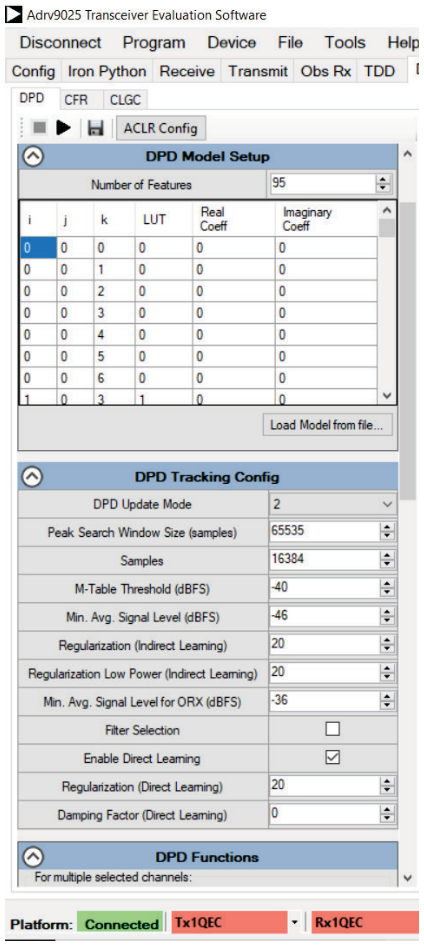
When programming is complete, click the **Tx** tab and setup the transmit data file and signal level. Ensure that the signal level is adequate to produce a signal at the observation receiver input with a power level between −20 dBm and −25 dBm. When the transmitter levels are set, click the Obs Rx tab and clear the enable box for each observation receive channel so that these channels can be controlled by the DPD tracking calibrations. When the observation receiver signal level is confirmed to be in the proper range, go back to the **Tx** tab and disable the transmitter outputs so that they can be controlled by the DFE functions.

DPD Setup

There are three subtabs on the DFE setup tab. Select the DPD tab to select the configuration page for DPD functions. To evaluate DPD performance, take the following steps (see Figure 194 and Figure 195):

1. Select the DPD model by clicking the **Load Model from file…** button.
2. Configure the DPD tracking parameters (it is recommended to start with the default settings).
3. Select which transmit channels will be evaluated.
4. Click **Apply Tracking** Config.
5. Click **Run Path Delay Init Cal**.

6. Check the signal level reported. If the level is not in the −20 dBm to −25 dBm range, check the cable connections. If still not at the expected level, try reprogramming the device with the desired settings.
7. Click Apply Model on Device from M Table.
8. Click **Apply Model on Device from C Table**.
9. Click **Enable DPD on select channels (only).**
10. Click the **Transmit** tab and click the play button to send data.
11. Click the **DFE** tab and then the **DPD** tab, and click **Reset DPD**.
12. Click **Get Status & Statistics** to evaluate the performance.


Adrv9025 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Iron Python Receive Transmit Obs Rx TDD
DPD CFR CLGC
ACLR Config
DPD Model Setup
Load Model from file...
DPD Tracking Config
DPD Functions
For multiple selected channels:
Platform: Connected Tx1QEC Rx1QEC

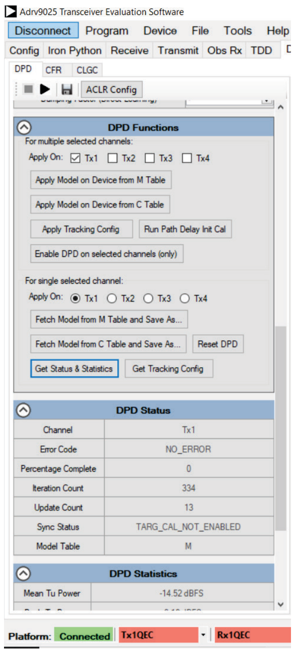
Number of Features: 95

i	j	k	LUT	Real Coeff	Imaginary Coeff
0	0	0	0	0	0
0	0	1	0	0	0
0	0	2	0	0	0
0	0	3	0	0	0
0	0	4	0	0	0
0	0	5	0	0	0
0	0	6	0	0	0
1	0	3	1	0	0

DPD Update Mode	2
Peak Search Window Size (samples)	65535
Samples	16384
M-Table Threshold (dBFS)	-40
Min. Avg. Signal Level (dBFS)	-46
Regularization (Indirect Learning)	20
Regularization Low Power (Indirect Learning)	20
Min. Avg. Signal Level for ORX (dBFS)	-36
Filter Selection	☐
Enable Direct Learning	☑
Regularization (Direct Learning)	20
Damping Factor (Direct Learning)	0

22770-192

Figure 194. DPD Model and Tracking Configuration Setup


Adrv9025 Transceiver Evaluation Software
Disconnect
Program
Device
File
Tools
Help
Config
Iron Python
Receive
Transmit
Obs Rx
TDD
DPD
CFR
CLGC
ACLR Config
DPD Functions
For multiple selected channels:
Apply On: ☑ Tx1 ☐ Tx2 ☐ Tx3 ☐ Tx4
Apply Model on Device from M Table
Apply Model on Device from C Table
Apply Tracking Config
Run Path Delay Init Cal
Enable DPD on selected channels (only)
For single selected channel:
Apply On: ● Tx1 ○ Tx2 ○ Tx3 ○ Tx4
Fetch Model from M Table and Save As...
Fetch Model from C Table and Save As...
Reset DPD
Get Status & Statistics
Get Tracking Config
DPD Status
DPD Statistics
Platform:
Connected
Tx1QEC
Rx1QEC

DPD Status

Channel	Tx1
Error Code	NO_ERROR
Percentage Complete	0
Iteration Count	334
Update Count	13
Sync Status	TARG_CAL_NOT_ENABLED
Model Table	M

DPD Statistics

Mean Tu Power	-14.52 dBFS

22770-193

Figure 195. DPD Function Setup and Performance Statistics

DIGITAL PREDISTORTION (DPD)

This section provides an overview of the DPD function provided in the ADRV9029 transceiver, including a hardware architecture overview, an overview of the DPD algorithm with references to features that enhance DPD robustness and performance in dynamic signaling conditions, and a summary of API software interface functions to configure these DPD functions. This functionality is only available in the ADRV9029 variant. The DPD enables users to achieve higher power amplifier efficiency by extending the linear operating region of the power amplifier, while still meeting adjacent channel leakage ratio (ACLR) requirements in the transmit signal chain for compliance with 3GPP and European Telecommunications Standards Institute (ETSI) standards for 5GNR, LTE, and other technologies. The ADRV9029 DPD supports a carrier bandwidth of up to 200 MHz.

DFE SYSTEM LEVEL OVERVIEW

The transceiver provides digital signal processing capabilities in the embedded ARM processor using closed-loop feedback signals from the observation receiver channels. These functions improve transmitter performance, measure system output, and reduce system power consumption. The list of functions includes the following: DPD, closed-loop gain control (CLGC), and crest factor reduction (CFR). These functions are collectively grouped together as the transceiver DFE.

Figure 196 is a simplified system level overview of the transceiver signal chain with DFE processing blocks highlighted. There are five main DFE processing blocks that include the following:

1. The CFR and hard clipper are used to reduce peak to average power ratio (PAPR) of the baseband signal, especially for multicarrier waveforms such as orthogonal frequency division multiplexing (OFDM). With reduced PAPR, the PA can operate at a higher output power, increasing the power amplifier efficiency. This function is explained in the Crest Factor Reduction (CFR) section.
2. There are two half-band filters with a total interpolation factor of 4× before the DPD actuator. These blocks can provide a total of 1×, 2×, or 4× interpolation.
3. There are three DPD capture buffers, which include a pre-DPD actuator, a post DPD actuator, and observation buffers. Each buffer can capture a maximum of 4096 samples.
4. The DPD actuator applies the inverse power amplifier model to the baseband signal for power amplifier linearization.
5. There is a dual core embedded ARM processor in which the DPD and CLGC algorithms reside. One of the dual core ARM processors is a control processor (ARM-C), which is the master, and the second core is a dedicated ARM core for DPD processing (ARM-D).

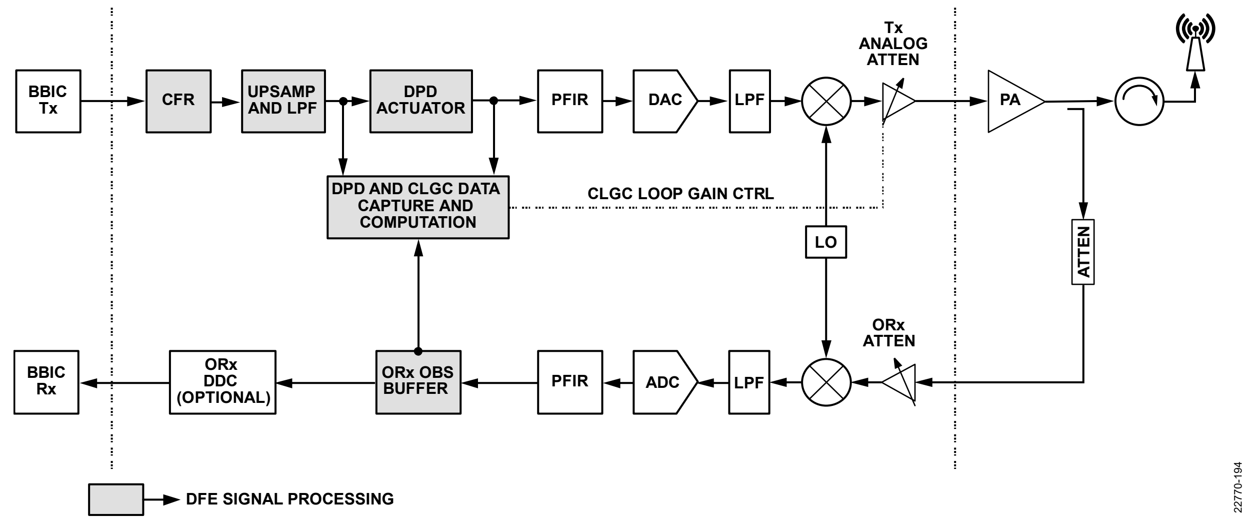


Figure 196. ADRV9029 Signal Chain with DFE Processing Blocks Highlighted

DPD INTRODUCTION AND PRINCIPLE OF OPERATION

DPD is a technique for improving the linearity of a nonlinear system such as a power amplifier by introducing precise antidistortion into the input waveform that compensates for the power amplifier in-band nonlinear products. DPD works on the principle of predistorting the transmit data in the digital domain to cancel the distortion caused by power amplifier compression in the analog domain. In this way, DPD can improve power amplifier power added efficiency by double or more to allow the power amplifier to be pushed further into saturation while maintaining linearity requirements.

A baseband model of the power amplifier is created and trained on the input and output digital baseband samples that pass through the power amplifier, as shown in Figure 197. The predistorter then applies an inverse of the power amplifier model function to input samples before passing them to the transmitter output. The cascade of the predistorter response and the power amplifier response becomes a nearly linear system.

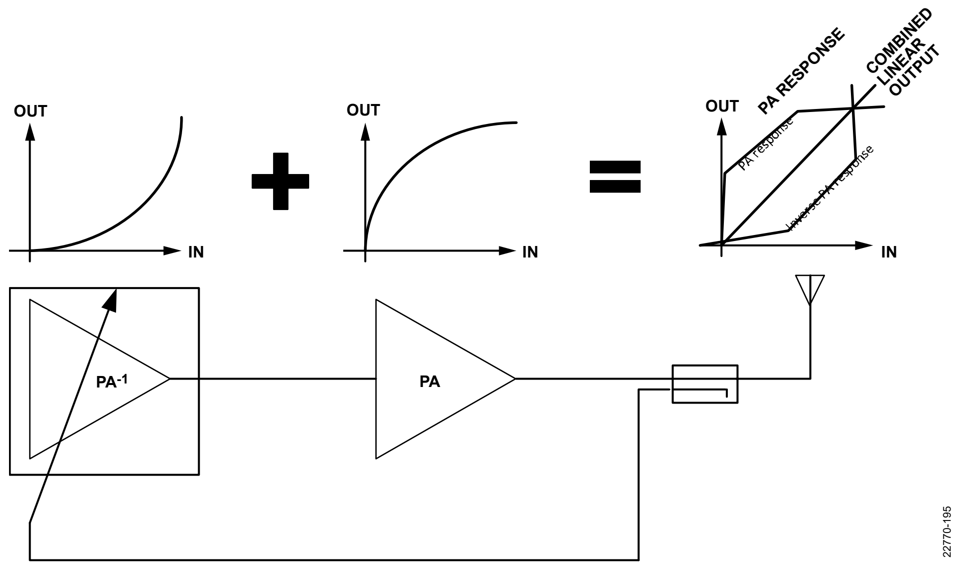
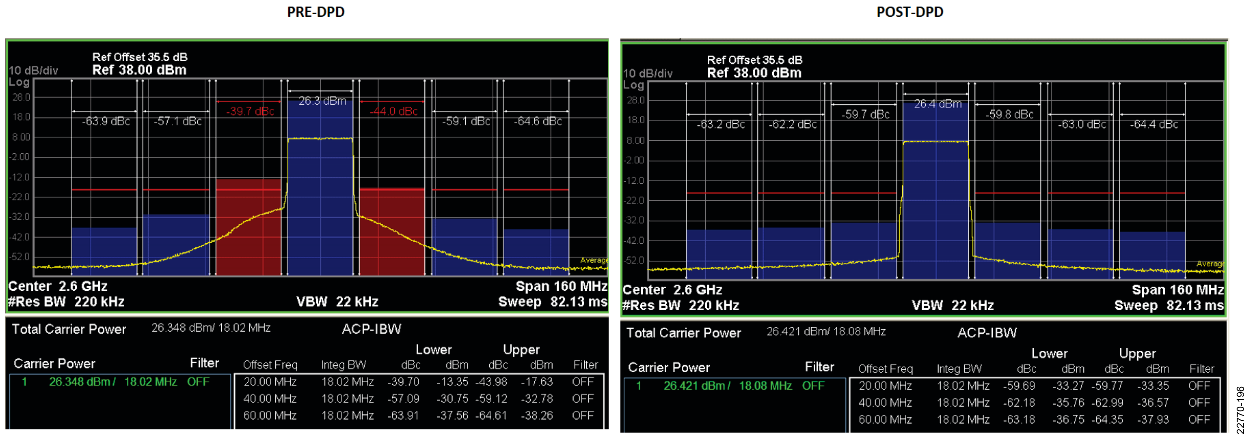


Figure 197. Concept of Digital Predistortion for Linearizing the Power Amplifier Response

The intermodulation distortion products between various subcarriers because of power amplifier nonlinearities in a wideband transmission protocol such as LTE/NR manifest as power leaked into adjacent channels. ACLR is defined as the ratio of the transmitted power on the assigned channel to the power leaked in the adjacent radio channel. The ACLR performance improvement following the application of DPD to the baseband data is captured in Figure 198. These plots illustrate how the out of band nonlinearities because of intermodulation products of an LTE 20 MHz signal are reduced by 15 dB to 20 dB after the application of DPD.



PRE-DPD

Center 2.6 GHz Span 160 MHz
#Res BW 220 kHz VBW 22 kHz Sweep 82.13 ms

Total Carrier Power 26.348 dBm/ 18.02 MHz ACP-IBW

Carrier Power			Filter	Offset Freq	Integ BW	Lower dBc	Lower dBm	Upper dBc	Upper dBm	Filter
1	26.348 dBm /	18.02 MHz	OFF	20.00 MHz	18.02 MHz	-39.70	-13.35	-43.98	-17.63	OFF
				40.00 MHz	18.02 MHz	-57.09	-30.75	-59.12	-32.78	OFF
				60.00 MHz	18.02 MHz	-63.91	-37.56	-64.61	-38.26	OFF

POST-DPD

Center 2.6 GHz Span 160 MHz
#Res BW 220 kHz VBW 22 kHz Sweep 82.13 ms

Total Carrier Power 26.421 dBm/ 18.08 MHz ACP-IBW

Carrier Power			Filter	Offset Freq	Integ BW	Lower dBc	Lower dBm	Upper dBc	Upper dBm	Filter
1	26.421 dBm /	18.08 MHz	OFF	20.00 MHz	18.02 MHz	-59.69	-33.27	-59.77	-33.35	OFF
				40.00 MHz	18.02 MHz	-62.18	-35.76	-62.99	-36.57	OFF
				60.00 MHz	18.02 MHz	-63.18	-36.75	-64.35	-37.93	OFF

Figure 198. Power Spectral Density Showing Improvement in ACLR After Application of DPD for a 20 MHz LTE Signal

TRANSCEIVER DPD OVERVIEW

The DPD feature on this transceiver enables users to offload power amplifier linearization tasks from the baseband processor to the transceiver. With the DPD implemented on the transceiver, the user does not need to allocate JESD serializer/deserializer resources for observing power amplifier feedback data through the observation receiver channels, which results in significant system power savings. Interpolators leading to the DPD actuator allow the baseband processor to transmit data at a lower rate on the JESD204B and JESD204C link than is needed for the full DPD correction bandwidth. The lower data rate at JESD translates directly into power savings and less lanes. Integration of the DPD into the transceiver chip results in significant system level cost, space, and power savings when compared to conventional FPGA/ASIC-based implementations.

A simplified block diagram of the transceiver DPD system is shown in Figure 199. The individual blocks include the following:

- Transmit datapath. The digital baseband signal from the JESD deframer output goes through an optional CFR block for reduction of the overall PAPR of the signal, followed by a digital interpolation filter that interpolates the baseband signal by a factor of 1×, 2×, or 4× for analyzing the baseband signal over the DPD analysis bandwidth. The inverse power amplifier model is applied by the DPD engine, followed by the rest of the transmit signal chain including digital to analog conversion and upconversion by a mixer before the signal is fed into the actual amplifier.
- Observation datapath. The DPD algorithm relies on observing the nonlinearities via a feedback path. The feedback path is realized using an integrated observation receiver. The power amplifier output data is sampled through the observation receiver, downconverted, and digitized for further analysis by the firmware.
- DPD processing. The DPD engine is based on an abbreviated implementation of generalized memory polynomial (GMP) that is a generalized subset of the well-known Volterra series. The simplified polynomial models a large number of power amplifier characteristics such as weak nonlinearities, temperature variation, and memory effects. The inverse power amplifier model is applied on the interpolated digital baseband samples through DPD actuator hardware. A dedicated embedded ARM processor (ARM-D) is used for computation of the GMP coefficients.

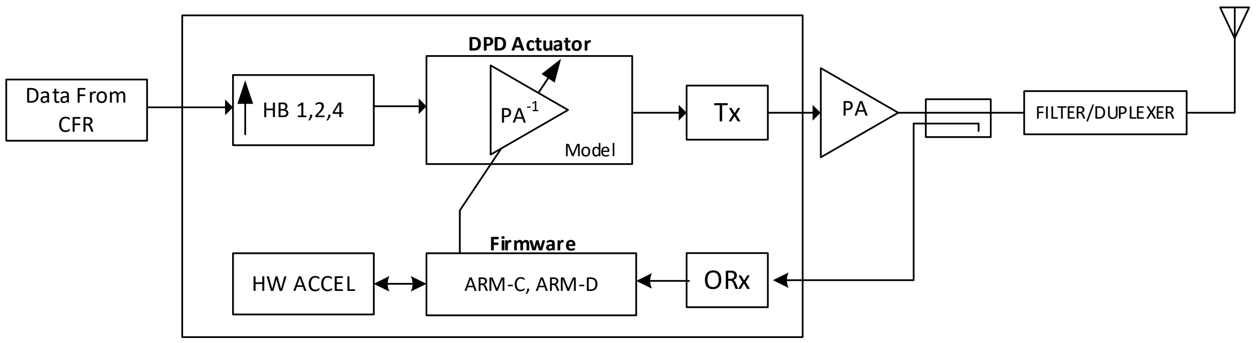


Figure 199. Simplified Block Diagram of Transceiver DPD

DPD Actuator Overview

The DPD actuator implements a programmable GMP calculator using the following equation:

$$x_{GMP}(n) = \sum_{i=0}^{i<16} \sum_{j=0}^{j<16} \sum_{k=0}^{k<16} c_{i,j,k} \, |u(n-i)|^k u(n-j) \quad (3)$$

where:
x_{GMP} is the output of the actuator.
u is the input of the actuator.
i is the memory term.
j is the cross term.
k is the order term.
$c_{i,j,k}$ is the complex valued coefficient of the GMP terms.

To compensate for memory effects in a large bandwidth signal, a higher order polynomial is required. The DPD actuator can be programmed to support up to 190 coefficients for wide bandwidth signals. The structure of the DPD actuator is shown in Figure 200. For every predistorted output, the GMP model calculates the sum of product expression. The product terms consist of a modeling coefficient ($c_{i,j,k}$), a magnitude power term ($|u(n-i)|^k$), and cross memory term $u(n-j)$. Each DPD model consisting of GMP terms can utilize up to 31 LUTs on one specified bank. The LUTs are 1k samples deep and organized in four banks of 8-bit tables (256 entries). Refer to the GMP Model and Look Up Table section for more information on the mapping GMP terms to LUTs.

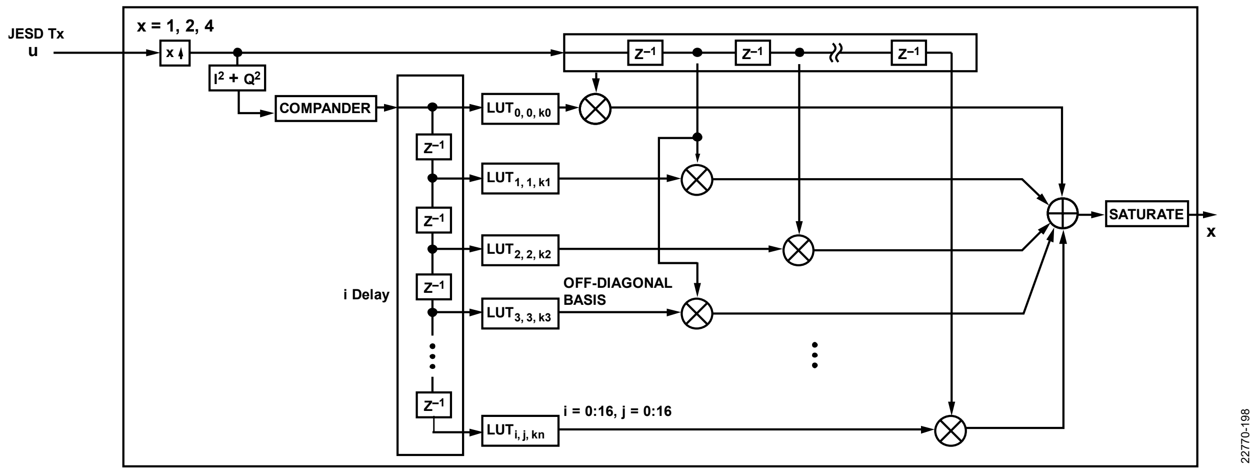


Figure 200. ADRV9029 DPD Actuator Functional Diagram

DPD Half-Band Filters

There are two half-band filters that can be enabled based on the input data rate and the desired DPD actuator rate. Two important characteristics of the half-band filters are that the passband and stopband ripples are the same, and the passband edge and stopband edge frequencies are equidistant from the half-band frequency, $F_S/4$.

Each DPD half-band filter provides either a 1× or 2× interpolation rate. A maximum of 4× interpolation can be achieved by cascading two DPD half-band filters. DPD Half-Band 1 (HB1) supports a bandwidth of approximately 82% with respect to the input data rate. For example, DPD HB1 supports a 100 MHz bandwidth signal at a 122.88 MSPS input rate. DPD Half-Band 2 (HB2) supports a bandwidth of approximately 41% with respect to the input data rate. For example, DPD HB2 supports a 100 MHz bandwidth signal at a 245.76 MSPS input rate. These two characteristics of the half-band filters are shown in Figure 201 and Figure 202 for 2× interpolation, and in Figure 203 and Figure 204 for 4× interpolation.

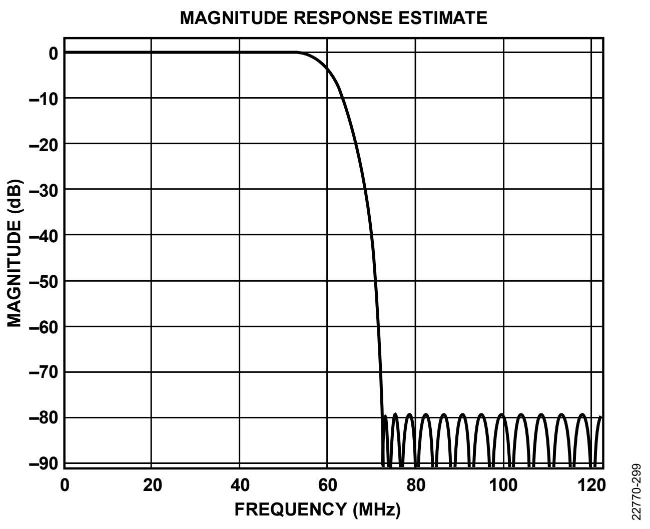


Figure 201. Magnitude Response for HB1 Enabled (×2), 82% of F_S

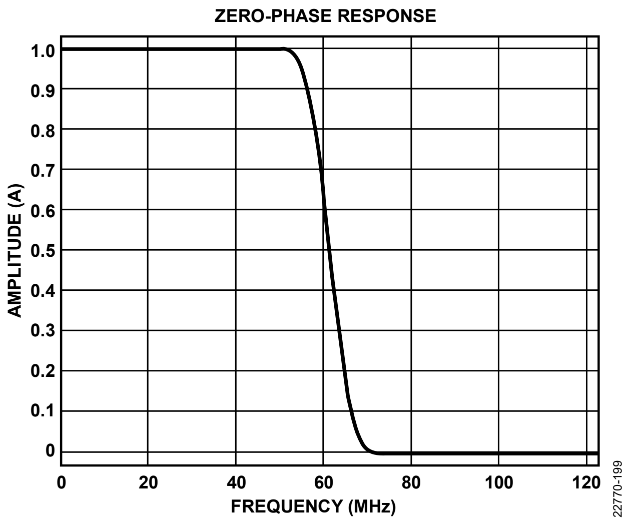


Figure 202. Zero Phase Response for HB1 Enabled (×2), 82% of F_S

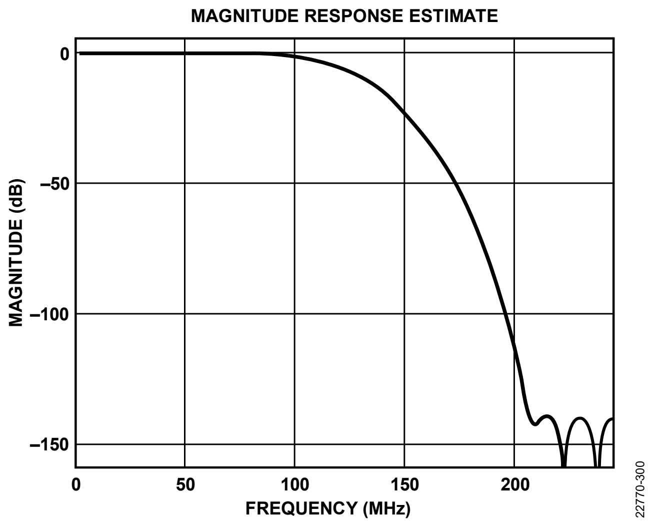


Figure 203. Magnitude Response for Both HB1 and HB2 Enabled (×4), 41% of F_S

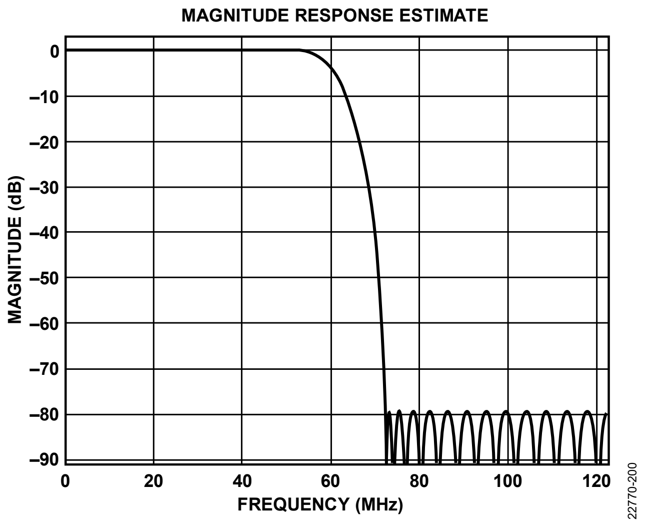


Figure 204. Zero Phase Response for Both HB1 and HB2 Enabled (×4), 41% of F_S

DPD ALGORITHM OVERVIEW

The ADRV9029 DPD algorithm supports both indirect learning and direct learning DPD mechanisms for extracting DPD model coefficients. The details of direct and indirect DPD learning mechanisms are provided in the following sections.

The user can configure the transceiver DPD learning algorithm through the adi_adrv9025_DpdTrackingConfigSet() API using the settings listed in Table 249.

Table 249. DPD Direct Learning Setting

adi_adrv9025_DpdTrackingConfig_t. enableDirectLearning	DPD Learning Mechanism Selected
0	Indirect learning
1	Direct learning

Indirect Learning

Indirect learning involves using the observation receiver data (power amplifier output data) as a reference for predicting the input samples corresponding to the reference. The function used for predicting the input samples is known as the inverse power amplifier model. When the prediction of input samples corresponding to the observed data is good, the estimated inverse power amplifier model is used to predistort the transmit data. In Figure 205, Y represents the observed samples at the output of the power amplifier and X represents the input samples to the power amplifier. The estimation engine computes the inverse power amplifier model that is applied to transmit data (U) in the DPD actuator.

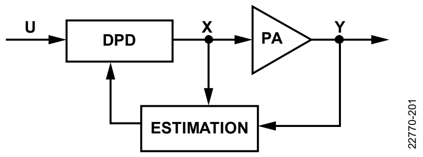


Figure 205. DPD Indirect Learning Architecture

The mathematical representation of the DPD coefficient estimation is shown in Figure 206. The DPD engine observes N samples of power amplifier input samples (X) and power amplifier output samples (Y), and computes M coefficients (c) corresponding to the inverse power amplifier function F(x).

$$\begin{array}{ccc} \mathbf{Y} & & \mathbf{F(x)} & & \mathbf{c} \\ \begin{pmatrix} y1 \\ y2 \\ y3 \\ y4 \\ \vdots \\ yN \end{pmatrix} & = & \begin{pmatrix} f1(x1) & f2(x1) & f3(x1) & f4(x1) & \dots & fM(x1) \\ f1(x2) & f2(x2) & f3(x2) & f4(x2) & \dots & fM(x2) \\ f1(x3) & f2(x3) & f3(x3) & f4(x3) & \dots & fM(x3) \\ f1(x4) & f2(x4) & f3(x4) & f4(x4) & \dots & fM(x4) \\ & & \vdots & & & \\ f1(xN) & f2(xN) & f3(xN) & f4(xN) & \dots & fM(xN) \end{pmatrix} & & \begin{pmatrix} c1 \\ c2 \\ c3 \\ c4 \\ \vdots \\ cM \end{pmatrix} \end{array}$$

22770-202

Figure 206. DPD Indirect Learning Coefficient Computation

The coefficient set (c) is estimated through a least squares approximation as described in matrix multiplication equations in Figure 207.

$$Y = F \times C \qquad \text{(1) INITIAL STEP}$$

$$F^H Y = (F^H F)C \qquad \text{(2) MULTIPLY BY COMPLEX CONJUGATE OF F ON BOTH SIDES}$$

$$\underbrace{(F^H F)^{-1}}_{\text{AUTO-CORRELATION}} \underbrace{(F^H Y)}_{\text{CROSS-CORRELATION}} = C \qquad \text{(3) TAKE THE INVERSE OF THE AUTOCORRELATION FUNCTION TO OBTAIN C}$$

22770-203

Figure 207. Formulas to Estimate the DPD Coefficient Set

DPD Direct Learning

DPD direct learning involves using the pre-DPD actuator transmit signal (U) as reference to minimize the error between the observed (Y) and reference data (U), as shown in Figure 208.

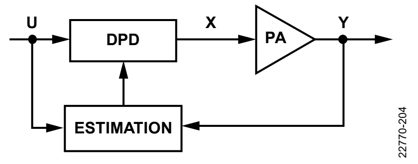


Figure 208. DPD Direct Learning Architecture

The mathematical representation of the DPD coefficient estimation via direct learning is described as an error, E, defined as the difference between the observed (Y) and pre-DPD actuator data (U) as follows:

$$E = Y - U$$

The power amplifier is modeled as the function, Fx, multiplied by adaptive coefficients, C, through the error matrix, E, as shown in Figure 209.

$$\begin{array}{cccc} \mathbf{E} & & \mathbf{f(\times)} & \mathbf{C} \\ \begin{pmatrix} e1 \\ e2 \\ e3 \\ e4 \\ . \\ . \\ . \\ eN \end{pmatrix} & = & \begin{pmatrix} f1(\times1) & f2(\times1) & f3(\times1) & f4(\times1) & \ldots fM(\times1) \\ f1(\times2) & f2(\times2) & f3(\times2) & f4(\times2) & \ldots fM(\times2) \\ f1(\times3) & f2(\times3) & f3(\times3) & f4(\times3) & \ldots fM(\times3) \\ f1(\times4) & f2(\times4) & f3(\times4) & f4(\times4) & \ldots fM(\times4) \\ & & . & & \\ & & . & & \\ & & . & & \\ f1(\times N) & f2(\times N) & f3(\times N) & f4(\times N) & \ldots fM(\times N) \end{pmatrix} & \begin{pmatrix} c1 \\ c2 \\ c3 \\ c4 \\ . \\ . \\ . \\ cM \end{pmatrix} \end{array}$$

Figure 209. DPD Direct Learning Coefficient Computation

The direct learning outcome is an iterative solution, where the current coefficients are based on the memory of previously computed coefficients and currently estimated coefficients (C).

The direct learning approach uses a parameter, μ, which is the convergence factor that defines the step size for learning coefficients. The convergence factor μ lies in the range 0% to 100% and is user configurable in the transceiver using the adi_adrv9025_DpdTrackingConfigSet() through the parameter adi_adrv9025_DpdTrackingConfig_t.dpdMu API. A higher value of the convergence factor, μ, results in faster convergence. However, a significantly high value of μ can result in an unstable system. The user can start with a convergence factor of 50% and tune the value based on characterization of the system for convergence time and stability.

Comparison Between DPD Indirect Learning and Direct Learning

The DPD indirect learning algorithm is time efficient because it estimates coefficients through inversion in a single update. The DPD indirect learning algorithm is preferred when a quicker adaptive response is required by the system.

The DPD direct learning algorithm is more accurate but iterative in nature as described in the previous section, and requires a longer time to converge compared to indirect learning. The direct learning algorithm is less sensitive to bandwidth mismatches and is preferred when the signal bandwidth is more than 100 MHz (for example, 2xNR100 or 8xLTE20 systems).

DPD Coefficient Estimation

The maximum number of coefficients is limited to 190 (M = 190), and the number of samples used to calculate the coefficients is typically 16384 samples (N = 16384). Although the number of samples, N, is user configurable using the adi_adrv9025_DpdTrackingConfigSet() API, it is recommended to set the number of samples to 16384, which provides a balance between estimation time and sample size.

The DPD algorithm runs on an ARM-D and calculates the coefficients corresponding to GMP terms of the inverse power amplifier model. This model predistorts the digital baseband signal before digital-to-analog conversion and transmission of samples to the transmitter upconverter (this output becomes the RF input to the power amplifier). The power amplifier output is sampled using an external loopback to an observation receiver channel input.

The DPD engine then correlates the observation receiver and transmitter samples to calculate the latest set of coefficients. The DPD engine performs a brief check on model error before updating the LUTs that feed the correction coefficients into the DPD actuator hardware. Details regarding the GMP model, the actuator and the LUTs are provided in sections to follow. Because of the relatively simple implementation of this algorithm, the overall time taken to react to sudden changes in transmit waveforms is relatively short and is typically less than 1 second per transmitter channel (actual time depends on the configurable parameters of the DPD and ARM scheduling). Certain protection criteria are designed into the algorithm to prevent damage to the power amplifier because of large model errors. The DPD algorithm is scheduled once every second per transmitter channel in the firmware, which means the coefficients are updated once every second per transmitter channel.

GMP Model and Look Up Table

For wideband signals such as LTE and 5GNR, power amplifiers begin to exhibit short term memory effects, which are effectively nonuniform frequency responses in certain components such as the biasing network, decoupling capacitors, or supply circuitry. A given output of the power amplifier depends not only on the current input, but also on past input values (similar to an FIR filter model). These memory effects can usually be captured with memory taps in the model if the power amplifier frequency response is locally smooth over the band. As bandwidth increases, more memory taps are required to accurately model the power amplifier.

In this DPD implementation, the GMP is used to model the power amplifier in the baseband. A GMP model is represented using Equation 3.

The DPD supports a sparse GMP model that consists of a maximum of 190 GMP terms and coefficients in which the memory term (i), the cross term (j) and the power term (k) are each restricted to a value from 0 to 15. A more complete equation for the GMP model with the limits on GMP terms is shown in the Figure 210.

The GMP model is user programmable through the adi_adrv9025_DpdModelConfigSet() API. Details pertaining to programming the DPD model are captured later in this section. The GMP model for a particular operating point of the power amplifier is determined by the user and programmed into the transceiver during initialization.

The GMP model is mapped to LUTs in the DPD actuator. Each feature is specified with a unique combination of i, j, and k indices. In general, low index values are more significant in sparse GMP models. Therefore, restrictions are placed on the actuator datapath accordingly. The LUT restrictions are shown in Figure 210. Note that the range of i and j are labeled per LUT. Each row shares the same multiplier, therefore, the same j value. Roaming A, B, C, and D LUTs are assigned to the top or bottom half of the table.

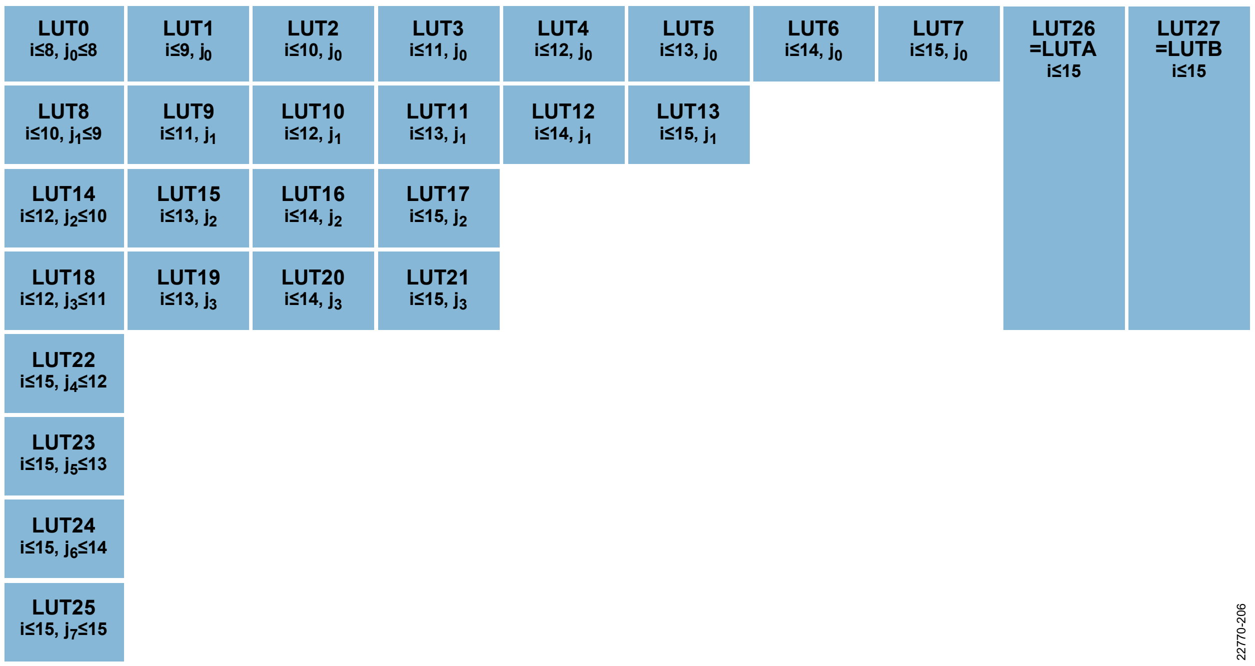
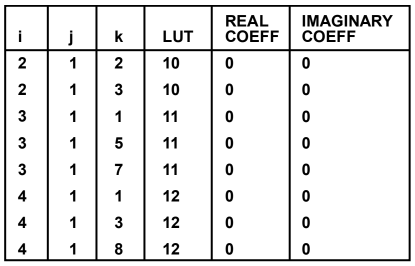


Figure 210. GMP Mapping to DPD Actuator LUTs

The restrictions placed on the GMP terms mapped to the LUTs shown in Figure 210 include the following:

- More LUTs are available for smaller i and j values clustered at the top of the LUT banks compared to fewer LUTs for high i and j values towards the bottom of the LUT bank. In general, low index values are more significant in sparse GMP models, therefore the user has more LUTs that can be mapped to lower i and j memory terms.
- Each row in the LUT bank shown in Figure 210 share the same j values, which means that the GMP terms mapped to LUT0 to LUT7 must have the same j value. Similarly, GMP terms mapped to LUT8 to LUT13, LUT14 to LUT17, and LUT18 to LUT25 must have the same j values, respectively.
- LUT28, LUT29, LUT30, and LUT31 are reserved for internal use. These LUTs are not available for the user to program GMP terms.
- LUT26 and LUT27 are floating LUTs, which means that these two LUTs can take a j value that is assigned to one of the four rows in the upper half of the LUT bank. For example, if LUT0 to LUT7 have j = 1, LUT8 to LUT13 have j = 2, LUT14 to LUT17 have j = 3 and LUT18 to LUT21 have j = 4, the GMP terms mapped to LUT26 and LUT27 can have a j value in the range 1 to 4.

A part of the user programmed GMP model is shown in Figure 211. Each row of the GMP model table consisting of the i,j,k LUT and coefficients is called a feature.



i	j	k	LUT	REAL COEFF	IMAGINARY COEFF
2	1	2	10	0	0
2	1	3	10	0	0
3	1	1	11	0	0
3	1	5	11	0	0
3	1	7	11	0	0
4	1	1	12	0	0
4	1	3	12	0	0
4	1	8	12	0	0

22770-207

Figure 211. Example of User Programmed GMP Model

The following equations represent the GMP terms that are mapped to LUT10, LUT11, and LUT12, as described in Figure 211.

$$X_{lut10} = |x(n-2)|^2 \times x(n-1) + |x(n-2)|^3 \times x(n-1)$$

$$Y_{lut11} = |x(n-3)| \times x(n-1) + |x(n-3)|^5 \times x(n-1) + |x(n-3)|^7 \times x(n-1)$$

$$Y_{lut12} = |x(n-4)| \times x(n-1) + |x(n-4)|^3 \times x(n-1) + |x(n-4)|^8 \times x(n-1)$$

where Y_{lutxx} is the output of the LUTxx.

The user can program the complex coefficients to 0. The DPD tracking calibration determines the coefficients and applies them to the GMP terms on each update.

The GMP model can be programmed using the adi_adrv9025_DpdModelConfigSet() API through the data structure adi_adrv9025_DpdModelConfig_t. The total number of features in the model is conveyed through the adi_adrv9025_DpdModelConfig_t.dpdNumFeatures variable. Each row of the table or the 'feature' is programmed through the adi_adrv9025_DpdModelConfig_t.dpdFeatures array. Each element of the array corresponds to one row in the table shown in Figure 211.

The values to load the adi_adrv9025_DpdModelConfig_t structure for the example in Figure 211 include the following (only values for feature [0] are shown):

- adi_adrv9025_DpdModelConfig_t.dpdNumFeatures = 8
- adi_adrv9025_DpdModelConfig_t.dpdFeatures[0].i = 2
- adi_adrv9025_DpdModelConfig_t.dpdFeatures[0].j = 1
- adi_adrv9025_DpdModelConfig_t.dpdFeatures[0].k = 2
- adi_adrv9025_DpdModelConfig_t.dpdFeatures[0].lut = ADI_ADRV9025_DPD_LUT10
- adi_adrv9025_DpdModelConfig_t.dpdFeatures[0].coeffReal = 0
- adi_adrv9025_DpdModelConfig_t.dpdFeatures[0].coeffImaginary = 0

INITIALIZING PRECALIBRATED COEFFICIENTS DURING STARTUP

The DPD functionality on the transceiver provides a mechanism for loading precalibrated coefficients into the GMP model to prevent emissions and to guide DPD updates at the beginning during startup. There are three different use cases for loading precalibrated coefficients.

Single Frequency Band Use Case

In the single frequency band use case, all four transmit channels work in the same frequency band, as shown in Figure 212. Note that the power amplifier model is the same for all four channels.,

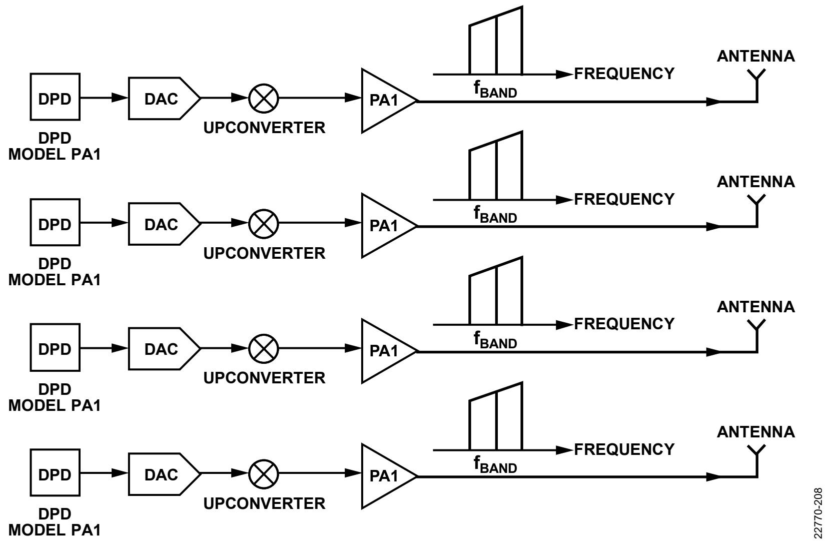


Figure 212. Single Frequency Band (f_{BAND}) Use Case Configuration

For the API sequence for programming DPD models in a single frequency band use case, the factory calibrated coefficients can be programmed into the transceiver using the adi_adrv9025_DpdModelConfigSet API, as described in the previous section. The DPD reset with the LUT restore option must be exercised consecutively on all four channels to program the coefficients into the DPD actuator hardware. The API sequence for programming DPD models in a single frequency band use case is as follows:

```
adi_adrv9025_DpdModelConfigSet()
adi_adrv9025_DpdReset(ADI_ADRV9025_TX1, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_adrv9025_DpdReset(ADI_ADRV9025_TX2, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_adrv9025_DpdReset(ADI_ADRV9025_TX3, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_drv9025_DpdReset(ADI_ADRV9025_TX4, ADI_ADRV9025_DPD_LUT_RESTORE)
```

Dual Frequency Band Use Case

In the dual frequency band use case, the signals of Transmit Channel 1 and Transmit Channel 2 are centered at f_{band1} and those of Transmit Channel 3 and Transmit Channel 4 at f_{band2}. Power amplifier characteristics are band dependent, so the DPD model that is loaded into Transmit Channel 1 and Transmit Channel 2 must be different than the model loaded into Transmit Channel 3 and Transmit Channel 4, as shown in Figure 213.

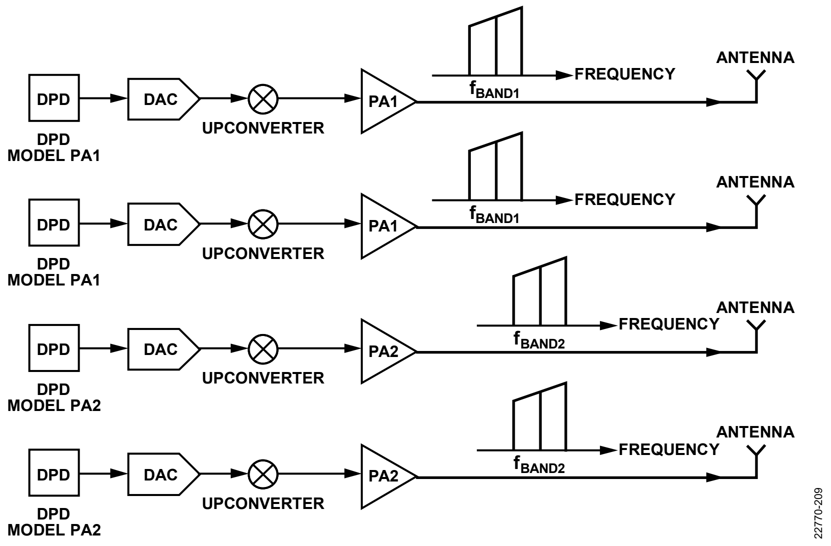


Figure 213. Case 2 Dual Frequency Band Use Case Configuration

For the API sequence for programming DPD models in a dual frequency band use case, the factory calibrated coefficients can be programmed into the transceiver through the adi_adrv9025_DpdModelConfigSet() API for the two pairs of transmit channels, as described in the previous section. The DPD reset with the LUT restore option must be exercised for two channels at a time. The API sequence for programming DPD models in a dual frequency band use case are as follows:

```
adi_adrv9025_DpdModelConfigSet() /* Load model for Tx12 */
adi_adrv9025_DpdReset(ADI_ADRV9025_TX1, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_adrv9025_DpdReset(ADI_ADRV9025_TX2, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_adrv9025_DpdModelConfigSet()/* Load model for Tx34 */
adi_adrv9025_DpdReset(ADI_ADRV9025_TX3, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_adrv9025_DpdReset(ADI_ADRV9025_TX4, ADI_ADRV9025_DPD_LUT_RESTORE)
```

Unique GMP Model Per Transmit Channel

In addition to the use cases previously described, the user can initialize each transmit channel with a unique GMP model. This approach is typically used when each power amplifier exhibits a slightly different nonlinearity, which requires a different set of coefficients, as shown in Figure 214.

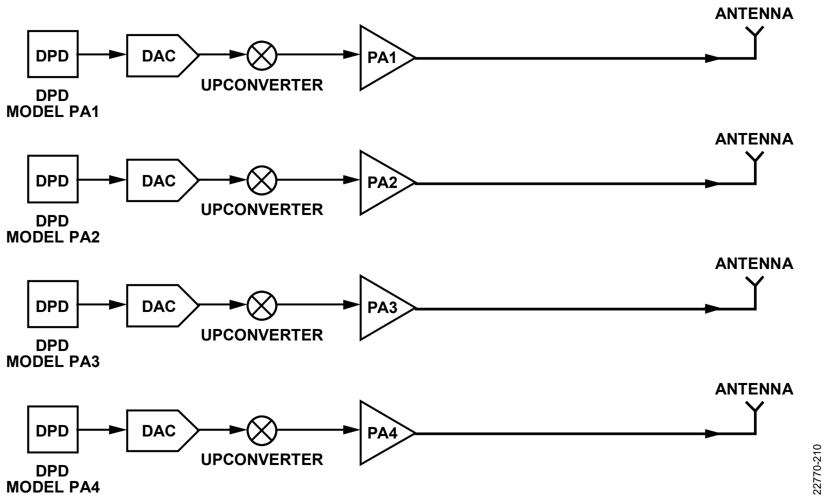


Figure 214. Use Case Where each Transmit Path Requires a Different Set of DPD Coefficients

For the API sequence for programming a unique DPD model per transmit channel, the factory calibrated coefficients can be programmed into the transceiver through the adi_adrv9025_DpdModelConfigSet() API for the two pairs of transmit channels, as described in the previous section. The DPD reset with the LUT restore option must be exercised for each channel. The API sequence for programming a unique DPD model per transmit channel is as follows:

```
adi_adrv9025_DpdModelConfigSet() /* Load model for Tx1 */
adi_adrv9025_DpdReset(ADI_ADRV9025_TX1, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_adrv9025_DpdModelConfigSet() /* Load model for Tx2 */
adi_adrv9025_DpdReset(ADI_ADRV9025_TX2, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_adrv9025_DpdModelConfigSet() /* Load model for Tx3 */
adi_adrv9025_DpdReset(ADI_ADRV9025_TX3, ADI_ADRV9025_DPD_LUT_RESTORE)
adi_adrv9025_DpdModelConfigSet() /* Load model for Tx4 */
adi_adrv9025_DpdReset(ADI_ADRV9025_TX4, ADI_ADRV9025_DPD_LUT_RESTORE)
```

DPD SAMPLE CAPTURE

The DPD algorithm relies on observing the samples distorted by the power amplifier through an observation channel to estimate the DPD coefficients. The DPD algorithm captures the observation samples after the samples have been processed by the observation receiver channel, and before and after the DPD actuator in batches of 4096 samples. The total number of samples that the DPD algorithm must capture is configured using the adi_adrv9025_DpdTrackingConfigSet() API through the adi_adrv9025_DpdTrackingConfig_t.dpdSamples parameter. The number of samples must be a multiple of 4096. Increasing the number of samples increases the processing time and computation load. Conversely, a decreased number of samples can impact the accuracy of coefficient estimation. It is recommended that the number of samples be set to 16384, which provides a balance between accuracy of estimation of coefficients and processing time.

For successful captures, the transmitter to observation channel external signal routing must be conveyed to the firmware through the adi_adrv9025_TxToOrxMappingSet() API.

For accurate estimation of predistortion coefficients, the transmitter and observation receiver samples are aligned in time by the sample capture engine in the transceiver. To align the samples, the external path delay initial calibration must be executed using the adi_adrv9025_InitCalsRun() API with a mask value of ADI_ADRV9025_EXTERNAL_PATH_DELAY (= 0x00200000) along with the requisite transmit channel mask.

DPD Sample Capture Process

The DPD algorithm implements a peak detection-based capture strategy because the high power signal levels contain more useful information for deriving DPD coefficients. The device calibration scheduler can initiate DPD capture at any available point in time when the transmit signal chain is enabled. The sequence of events involved in DPD sample capture process is shown in Figure 215.

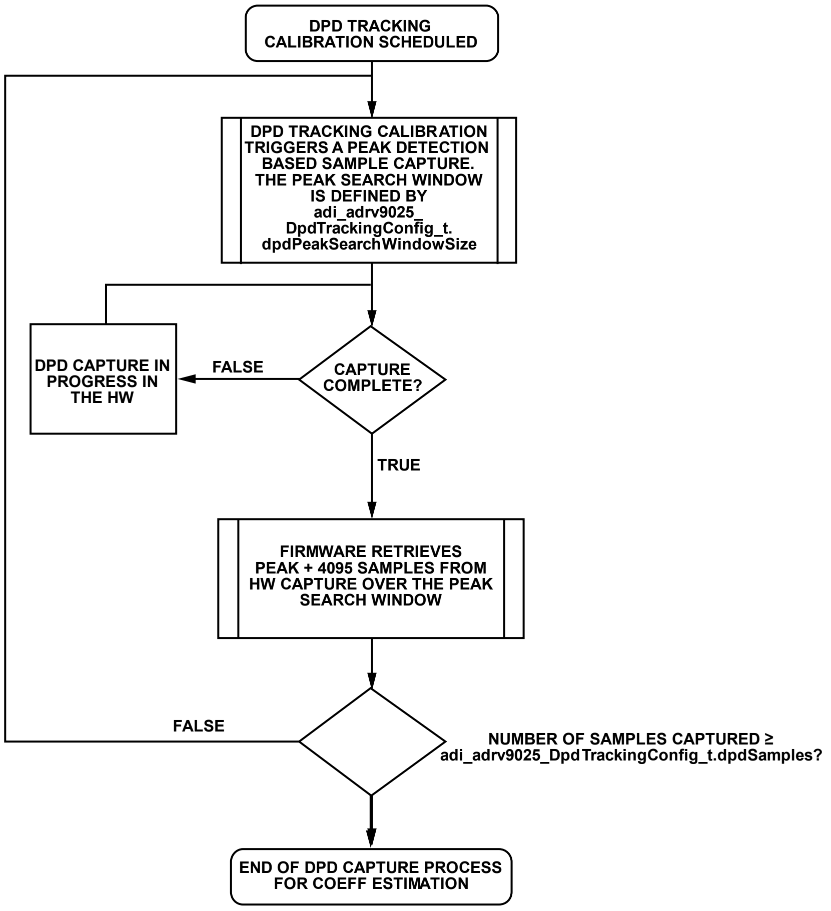

DPD TRACKING CALIBRATION SCHEDULED
DPD TRACKING CALIBRATION TRIGGERS A PEAK DETECTION BASED SAMPLE CAPTURE. THE PEAK SEARCH WINDOW IS DEFINED BY adi_adrv9025_ DpdTrackingConfig_t. dpdPeakSearchWindowSize
DPD CAPTURE IN PROGRESS IN THE HW
FALSE
CAPTURE COMPLETE?
TRUE
FIRMWARE RETRIEVES PEAK + 4095 SAMPLES FROM HW CAPTURE OVER THE PEAK SEARCH WINDOW
FALSE
NUMBER OF SAMPLES CAPTURED ≥ adi_adrv9025_DpdTrackingConfig_t.dpdSamples?
END OF DPD CAPTURE PROCESS FOR COEFF ESTIMATION
22770-211

Figure 215. DPD Tracking Calibration Sample Capture Sequence

Peak Search Window and Peak Detection-Based Capture

The DPD capture engine contains a peak detector that triggers captures on the largest peak seen in a specified time window, as shown in Figure 216. Peak detection occurs at the DPD input and after CFR correction is applied.

The peak detection time window is specified in the number of samples at the DPD actuator rate using the adi_adrv9025_ DpdTrackingConfgSet() API through the adi_adrv9025_DpdTrackingConfig_t.dpdPeakSearchWindowSize parameter. While longer peak search windows result in more accurate high power DPD modeling, the capture process also takes longer to complete. The goal of peak search window optimization is to increase the probability of capturing high power data without penalizing the rest of the system operation. To do so, the system designer must study the worst case signal statistics. This DPD contains only one adaptation engine, so the capture mechanism is shared between all active channels.

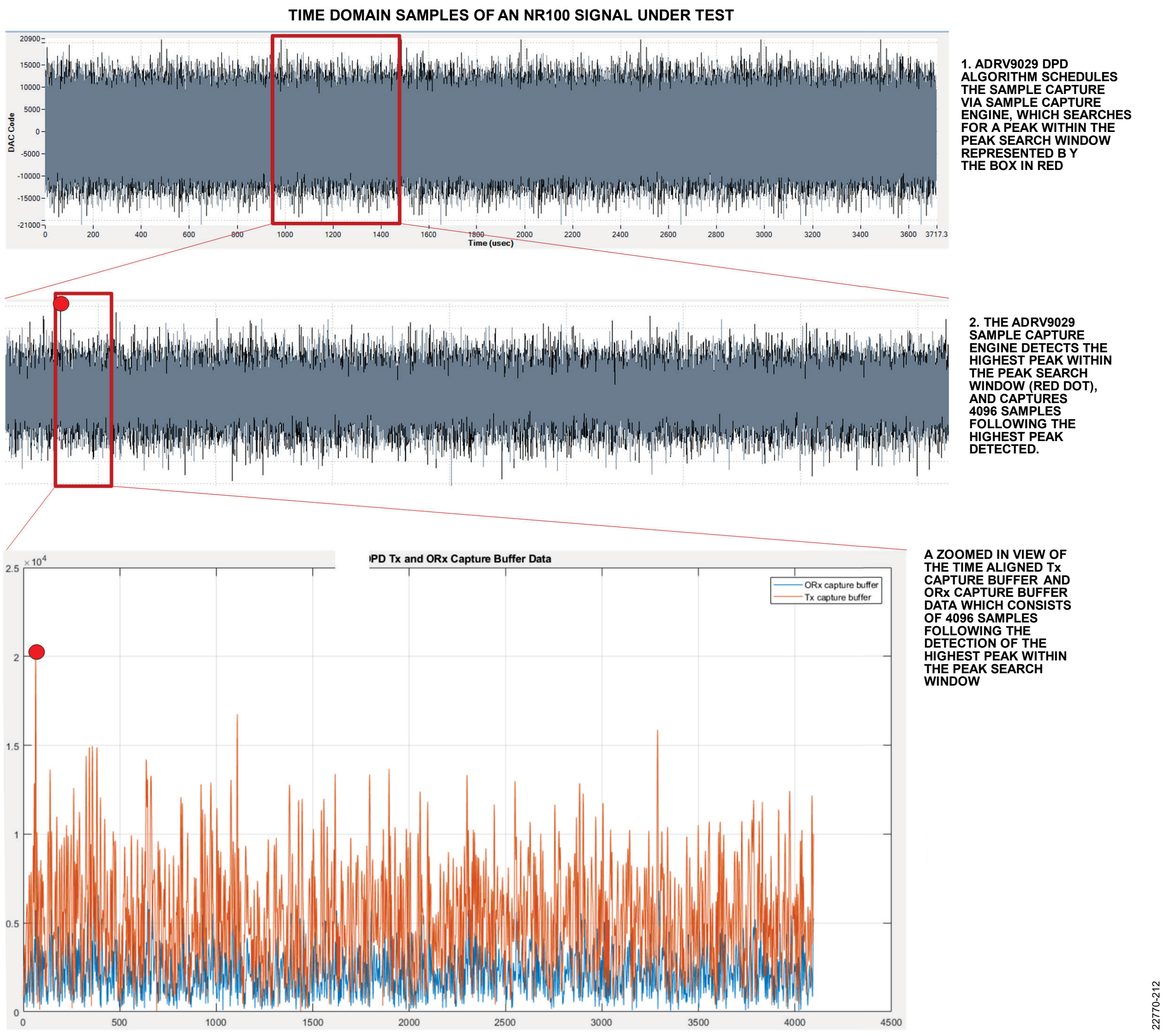


Figure 216. DPD Peak Detection-Based Sample Capture Process for DPD Adaptation

DPD Sample Capture in TDD Mode

There are additional considerations that a system developer must take into account to configure the DPD in TDD mode. In TDD mode, the DPD sample capture process spans multiple TDD downlink slot periods (transmit on periods), with each batch of 4096 samples captured during one TDD downlink slot period (transmit on period) through the peak detection process described in the previous section. The peak search window size is specified by the adi_adrv9025_DpdTrackingConfig_t.dpdPeakSearchWindowSize parameter and configured using the adi_adrv9025_DpdTrackingConfgSet() API. The search window size is restricted to a maximum of one TDD downlink slot period (one transmit on period) – 4096 samples.

An example of a typical sample capture process in TDD mode is illustrated in Figure 217. In this example, the total number of DPD samples specified by the adi_adrv9025_DpdTrackingConfig_t.dpdSamples parameter is set to 16384 samples or four batches of 4096 samples. Each batch of 4096 samples corresponding to the P1, P2, P3, and P4 peaks, respectively, are captured over four TDD downlink slot periods (four transmit on periods). If each TDD downlink slot periods consists of M samples at the DPD actuator rate, the maximum peak search window size that can be configured using the adi_adrv9025_DpdTrackingConfig_t.dpdPeakSearchWindowSize parameter is restricted to (M – 4096) samples. At the end of four TDD downlink slot periods, a composite capture data consisting of four batches of 4096 samples corresponding to the P1, P2, P3 and P4 peaks, respectively, are used to estimate the DPD coefficients, as described in the Indirect Learning section.

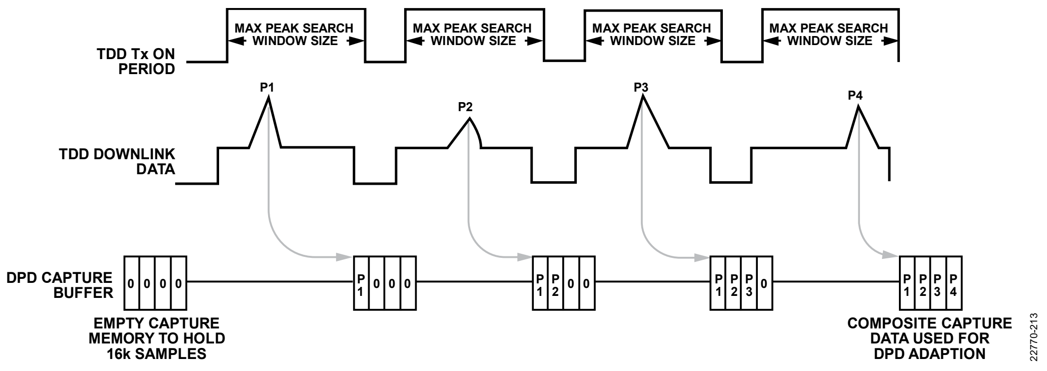

MAX PEAK SEARCH WINDOW SIZE
TDD Tx ON PERIOD
P1
P2
P3
P4
TDD DOWNLINK DATA
DPD CAPTURE BUFFER
EMPTY CAPTURE MEMORY TO HOLD 16k SAMPLES
COMPOSITE CAPTURE DATA USED FOR DPD ADAPTION
22770-213

Figure 217. DPD Sample Capture Process in TDD Mode

DPD DYNAMICS

The transceiver DPD is designed to react to dynamic signaling conditions. The algorithm defines four models that can be implemented depending on the power levels to achieve the best dynamic performance. The four DPD models are defined in Table 250.

Table 250. DPD Models Explained

DPD Model	Description
M Table (Maximum Power Table) (ADI_ADRV9025_DPD_MODEL_TABLE_M)	Default model in all three DPD modes. The conditions for updating this table in different DPD modes include the following: DPD MODE0. The model defined by the M table updates on every DPD iteration if the update criteria is met, as described in the ADI_ADRV9025_DPD_MODE0 section in Table 251. The update criteria is described in the following section. DPD MODE1. The model defined by the M table updates when the rms power of the DPD capture samples exceeds previously recorded maximum rms power, as described in the ADI_ADRV9025_DPD_MODE1 section in Table 251. DPD MODE2. The model defined by the M table updates only when the rms power of DPD capture samples exceeds the M threshold specified by the adi_adrv9025_DpdTrackingConfig_t.dpdMThreshold configuration, and the rms power of DPD capture samples exceeds the previously recorded maximum rms power. Refer to Table 251 for more details on DPD Mode 2 operation.
C Table (Current Table) (ADI_ADRV9025_DPD_MODEL_TABLE_C)	The model defined by the C table is a low power model only applicable in DPD MODE2 when the rms power of DPD capture samples is below the M threshold value specified by adi_adrv9025_DpdTrackingConfig_t.dpdMThreshold configuration, as described in the ADI_ADRV9025_DPD_MODE2 section in Table 251.
R Table (Recovery Table) (ADI_ADRV9025_DPD_MODEL_TABLE_R)	The R table, or recovery table, is a recovery model that stores the coefficients generated from the highest power data captured by the DPD. The maximum power recorded by the recovery model does not decay unlike the maximum power recorded by the M table, which decays by 0.2 dB per update period.
U Table (Unity Gain Table) (ADI_ADRV9025_DPD_MODEL_TABLE_U)	The unity gain model in which the output is equal to the input. This model is usually activated in low power conditions where predistortion is not necessary.

DPD Modes of Operation

The DPD functionality supports three modes of operation that are listed in Table 251. The user can select one of the three modes based on the application. The DPD engine must react to changing signal conditions, and Analog Devices has developed proprietary algorithms that address this requirement. Within a cost bounded implementation, there is no solution that achieves absolute performance on any time scale of measurement. The Analog Devices solution is an optimized compromise between performance and complexity.

The DPD mode of operation can be configured through the adi_adrv9025_DpdTrackingConfigSet() API using the adi_adrv9025_DpdTrackingConfig_t.dpdUpdateMode parameter. The DPD update mode can be set to one of the three enumerated lists of options represented by adi_adrv9025_DpdTrackingUpdateMode_e.

Table 251. DPD Modes of Operation

DPD Mode of Operation	Description
ADI_ADRV9025_DPD_MODE0	DPD coefficients corresponding to the GMP model are updated once every second. This mode offers the best sustained performance for any signal at the expense of transient emissions when the signal changes rapidly.

DPD Mode of Operation	Description
	Figure 218 shows the DPD updates in Mode 0. The DPD updates coefficients once every update period independent of the rms power measured by the DPD captures for coefficient computation.
ADI_ADRV9025_DPD_MODE1	DPD coefficients corresponding to the GMP model are updated only if the rms power measured by the DPD exceeds the previously recorded maximum rms power by the DPD algorithm. The rms power is calculated on the samples captured by the DPD for coefficient computation. The number of samples to capture for a DPD update is specified by adi_adrv9025_DpdTrackingConfig_t.dpdSamples. Typically, this number is set to 16384 samples. The recorded maximum power decays at a fixed rate of 0.2 dB per update. Figure 219 shows an illustration of DPD updates in Mode 1. The DPD algorithm updates coefficients only when the rms power measured by the DPD during the update exceeds the previously recorded maximum power. In this example, there is no update between Update 1 and Update 2 because the rms power is below the maximum power recorded. DPD Mode 1 offers the best mitigation of transient emissions when the signal changes rapidly at the expense of sustained performance in certain low power signal conditions.
ADI_ADRV9025_DPD_MODE2	In this mode, the DPD algorithm maintains two separate look up tables, one for the low power region and the other for the high power region. Depending on the RMS power measured by the DPD on the samples captured for coefficient computation, the DPD algorithm either switches to the high power look up table (M table) or the low power look up table (C table), and the same look up table is active until the next DPD update. The rms power threshold separating the low power and high power region is user configurable through the adi_adrv9025_DpdTrackingConfig_t. dpdMThreshold parameter. The rms power is calculated on the samples captured by the DPD for coefficient computation. The number of samples to capture for a DPD update is specified by adi_adrv9025_DpdTrackingConfig_t.dpdSamples. Typically, this number is set to 16384 samples, which offers a compromise between Mode 0 and Mode 1. There is some mitigation of transient emissions when the signal changes rapidly and sustained performance in many signaling conditions.

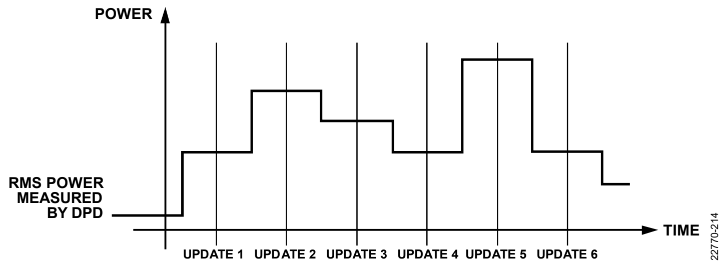


Figure 218. DPD Mode 0 Update Example

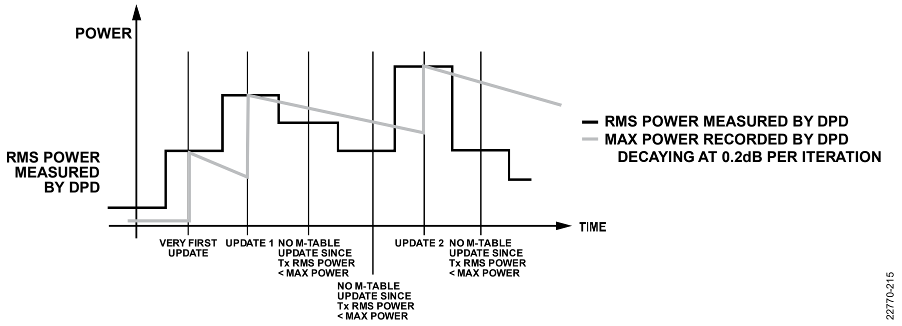


Figure 219. DPD Mode 1 Update Example

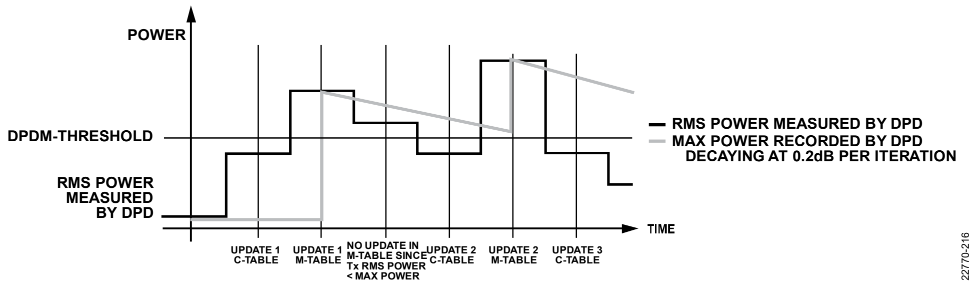


Figure 220. Illustration of DPD Updates in Mode 2

Transmitter Low Power Threshold

The DPD continuously integrates the baseband power level at the input of the DPD actuator so that the DPD can switch between the different models described in Table 251. The power is measured for a period of 10 ms through a leaky integrator that runs continuously in the background. If the 10 ms integrated rms power of the DPD input samples is below the transmitter low power threshold specified by adi_adrv9025_DpdTrackingConfig_t.minAvgSignalLevel in linear scale, the unity gain table is activated, If the 10 ms integrated rms power of the DPD input samples is higher than the transmit low power threshold specified by adi_adrv9025_DpdTrackingConfig_t.minAvgSignalLevel in linear scale, the DPD model defined by the M table is activated in DPD Mode 0 and Mode 1. In DPD Mode 2, there is an additional threshold specified by adi_adrv9025_DpdTrackingConfig_t.dpdMThreshold described in the next section. The dynamics of the DPD based on the transmit baseband input level is shown in Figure 221.

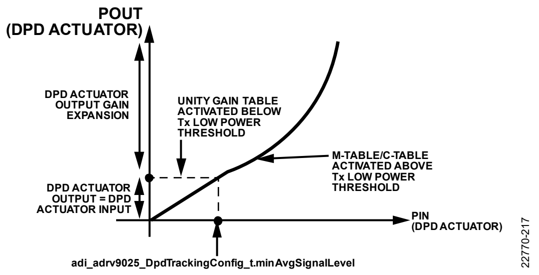


Figure 221. DPD Dynamics and Transmit Low Power Threshold

Transmitter M Threshold

The M threshold is a maximum power threshold specified by adi_adrv9025_DpdTrackingConfig_t.dpdMThreshold that is valid only in DPD Mode 2 operation (adi_adrv9025_DpdTrackingConfig_t.dpdUpdateMode = ADI_ADRV9025_DPD_MODE2). There are two DPD models (M table, C table) that the DPD tracking calibration maintains and updates. The DPD model update mechanism in DPD Mode 2 operation is described in Table 251. Note that the switching mechanism between the M table (high power), C table (low power), and U table (unity gain) models is based on the 10 ms integrated rms power of the DPD actuator input samples.

The dynamics of the DPD based on the transmit baseband input level with the M threshold taken into consideration is shown in Figure 222.

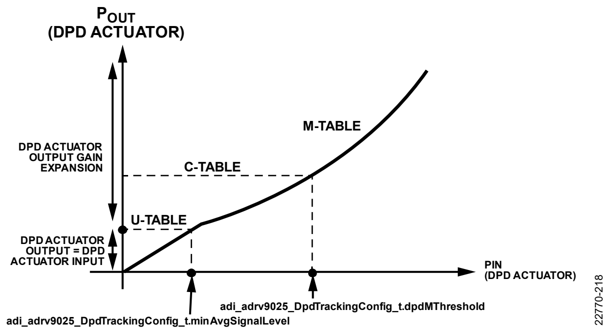


Figure 222. ADRV9025 DPD Dynamics in DPD Mode 2

Observation Receiver Low Power Threshold

The observation receiver low power threshold can be used to compare observation samples against the threshold specified by adi_adrv9025_DpdTrackingConfig_t.minAvgSignalLevelOrx in linear scale to determine if a DPD update must be applied. This comparison can help avoid DPD updates when the signal level is low and, consequently, the signal-to-noise ratio of the observed samples is poor. If the rms power of the DPD capture samples is greater than the observation channel low power threshold, an update is applied to the appropriate DPD model if the transmit low power threshold condition described in the previous section is also satisfied. Table 252 captures the various conditions related to the power levels and updates of transmitter and observation receiver captured samples.

Table 252. DPD Update Criteria Based on the Signal Level of Captured Samples

DPD Capture Samples RMS Power Condition	DPD Captured Transmit samples > adi_adrv9025_DpdTrackingConfig_tminAvgSignalLevel	DPD Captured Transmit samples < adi_adrv9025_DpdTrackingConfig_tminAvgSignalLevel
DPD Captured Observation Receive Samples > adi_adrv9025_DpdTrackingConfig_t.minAvgSignalLevelOrx	DPD Update applied to M table in DPD Mode 0 and Mode 1	No DPD update applied.
	DPD Update applied to C table/M table depending on adi_adrv9025_DpdTrackingConfig_t.dpdMThreshold in DPD Mode 2	
DPD Captured Observation Receive Samples < adi_adrv9025_DpdTrackingConfig_t.minAvgSignalLevelOrx	No DPD update applied	No DPD update applied

DPD REGULARIZATION

DPD regularization is used to make the DPD coefficient estimation less sensitive to missing data and prevent overfitting. The DPD is essentially a curve fitting process, and Figure 223 outlines the optimum fitting to achieve in a system. A higher regularization prevents overfitting, which improves stability but limits the ACLR improvement. On the other hand, a low regularization allows better ACLR improvement, but stability of the DPD must be kept in check.

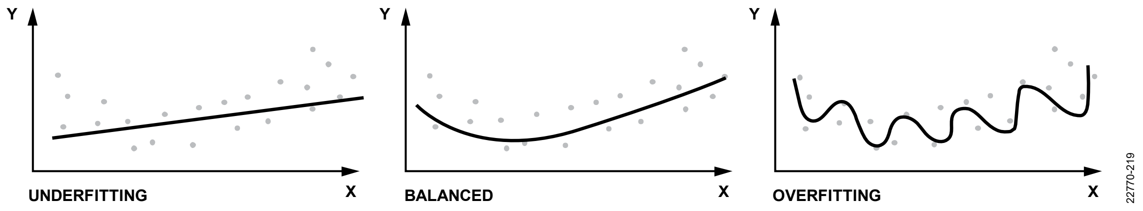


Figure 223. Comparison of Underfitting, Overfitting, and Balanced Fitting

The AM to AM characteristics of a power amplifier for a case where there is sparse data in the high power region is shown in Figure 224. In this case, a low regularization value results in overfitting and causes instability.

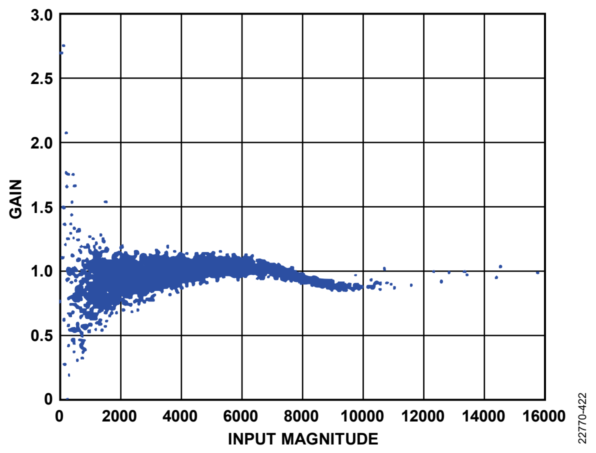


Figure 224. Gain vs Input Magnitude of an Example PA. DPD Regularization helps Decrease Sensitivity to Sparse

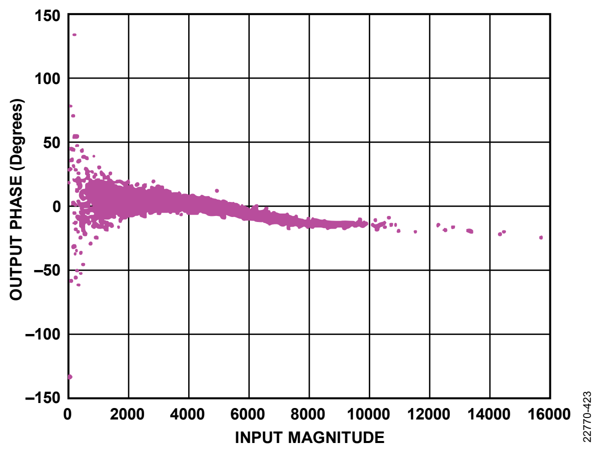


Figure 225. Output Phase vs Input Magnitude of an Example PA. DPD Regularization helps Decrease Sensitivity to Sparse Data

For the AM to AM characteristics shown in Figure 224, the effect of low regularization and optimum regularization on DPD is shown in Figure 226. With low regularization, the DPD algorithm has a tendency to overfit resulting in high power scattering. With the optimum regularization, the sensitivity of DPD algorithm to sparse data in high power is minimized.

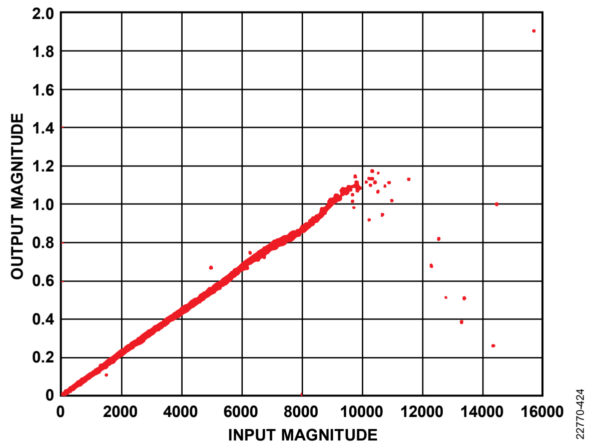


Figure 226. Effect of Low DPD Regularization on DPD Stability

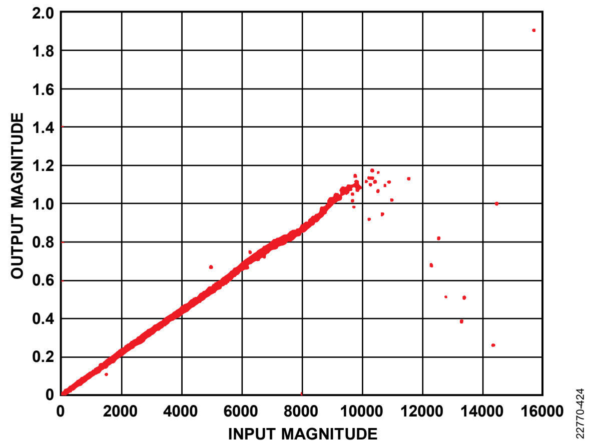


Figure 227. Effect of Optimum DPD Regularization on DPD Stability

The DPD provides user configuration for regularization via adi_adrv9025_DpdModelConfig_t. dpdIndirectRegularizationValue and adi_adrv9025_DpdModelConfig_t.dpdDirectRegularizationValue for indirect learning and direct learning mechanisms configured via the adi_adrv9025_DpdTrackingConfigSet() API.

DPD Regularization in DPD Mode 2

For DPD Mode 2 operation where separate predistortion coefficients are maintained for low power (C table) and high power (M table) data, a separate regularization value can also be applied to low power (C table) and high power (M table) models. This separate value is typically intended to be used with GaN PA applications where the power amplifier nonlinearity characteristics can vary for low power and high power signals.

Table 253. DPD Regularization Parameters in DPD Mode 2

Regularization Parameter	Target DPD Actuator Model
adi_adrv9025_DpdModelConfig_t.dpdIndirectRegularizationValue	M table
adi_adrv9025_DpdModelConfig_t.dpdIndirectRegularizationLowPowerValue	C table

Note that separate low power and high power regularization values are only applicable using the DPD indirect learning mechanism.

DPD ROBUSTNESS

This section provides an overview of the features that enhance DPD robustness. These features include DPD stability metrics and flexible architecture for setting up recovery actions based on these predefined DPD metrics. The user can optionally turn on the DPD robustness feature to protect the DPD against erroneous adaptations in abnormal conditions.

In the transceiver DPD, abnormal conditions are detected by monitoring the following metrics:

- Indirect error that indicates if the predistorted samples match the expected result after experiencing power amplifier distortion
- Transmit capture rms power
- Transmit capture peak power
- Observed capture rms power
- Observed capture peak power
- Post DPD capture transmit to observation receive EVM (indirect EVM), which is a measure of nonlinearity in the gain line up
- Pre-DPD capture transmit to observation receive EVM (direct EVM), which is a measure of DPD linearization performance

From a user point of view, there are two sets of actions that are required to be taken:

- Define the fault conditions via the adi_adrv9025_DpdFaultConditionSet() API.
- Provide user configurable thresholds and actions for fault conditions defined in the previous step via the adi_adrv9025_DpdRecoveryActionSet() API.

For example, measure the transmit capture power, and if the power < the threshold, abandon the adaptation.

Calculation of Metrics

When the DPD coefficients are estimated, the error between the predicted and measured predistortion is computed to determine the expected DPD performance. Detection of a large error prevents the application of bad coefficients, and can be calculated using the following equation:

$$IndirectError = ||x_{tx} - F_y c||/||x_{tx}||$$

where:
x_{tx} is a vector of the transmit samples after DPD actuator (post DPD data).
F_y is a matrix of features (such as items in GMP) formulated by observation receive samples.
c is the DPD coefficients vector.
|| ||is the operator and the (Euclidean) norm of vectors.

Similarly, indirect EVM and direct EVM are calculated using the following equations:

$$IndirectEVM = ||x_{tx} - y||/||x_{tx}||$$

$$DirectEVM = ||x_{tu} - y||/||x_{tx}||$$

where:
x_{tu} is a vector of the transmit samples before the DPD actuator (pre-DPD data).
y is a vector of observation receive samples.

All samples are time aligned and gain and phase equalized. The size of the vectors is the number of samples used in each update of the DPD coefficients.

The transmit signal mean and peak power are calculated from the pre-DPD samples in x_{tu} and post DPD samples in x_{tx}, respectively.

The observation receiver mean and peak power are calculated by the samples in y.

Defining Fault Conditions

The user can define fault conditions through the adi_adrv9025_DpdFaultCondition_t data structure described in Table 254. The fault conditions are programmed through the adi_adrv9025_DpdFaultConditionSet() API.

Table 254. DPD Fault Condition Definition Data Structure adi_adrv9025_DpdFaultCondition_t

Member	Data Type	Description
dpdMetric	adi_adrv9025_DpdMetric_e	This member is one of the seven DPD stability metrics described in the previous section.
comparator	adi_adrv9025_DpdComparator_e	The user can select a greater than or less than comparator for defining a fault condition. For example, the user can define a fault condition where the transmit rms power is greater than a certain threshold and the observation receive rms power is lesser than a certain threshold.
threshold0	Int16	This member defines a threshold for a lower severity level
threshold1	Int16	This member defines a threshold for a higher severity level
persistentCount	UInt16	The persistence count is defined with the expectation that more aggressive recovery actions are required in case the fault conditions persist.

The list of currently available actions is shown in Table 255. The fault conditions defined in Table 254 can be associated with a recovery action programmed through the adi_adrv9025_DpdRecoveryActionSet() API.

Table 255 ADRV902x Recovery Action Bit Mask Definition

Action Index	Description
ADI_ADRV9025_DPD_RECOVERY_ACTION_SKIP_LUTS_UPDATE	Abandon DPD adaptation on this iteration.
ADI_ADRV9025_DPD_RECOVERY_ACTION_REVERT_LUTS_TO_UNITY	Revert all DPD Models to unity gain coefficients
ADI_ADRV9025_DPD_RECOVERY_ACTION_RESET_ADAPTATION_STATE	Reset DPD firmware internal state variables that includes max power recorded by DPD applicable to M-Table in DPD Mode 1 and DPD Mode 2 that is relevant to the user.
ADI_ADRV9025_DPD_RECOVERY_ACTION_SWITCH_LUTS_TO_M	Switch to the max Power model. Please refer to DPD models section for more details on M-Table
ADI_ADRV9025_DPD_RECOVERY_ACTION_SWITCH_LUTS_TO_R	Switch to Recovery model defined as the model estimated on the highest power data seen by the DPD tracking calibration since reset.
ADI_ADRV9025_DPD_RECOVERY_ACTION_6DB_DIG_ATTEN	Attenuate transmit signal by 6dB. NOTE: Currently this action is NOT Supported.
ADI_ADRV9025_DPD_RECOVERY_ACTION_RESET_FIRST_DPD_FLAG	Resetting the first DPD flag runs 3 back to back DPD updates in indirect learning mode similar to the DPD behavior on very first update to speed up convergence.

The DPD fault conditions configured in the firmware on startup are listed in Table 256.

Table 256. Default Definition of Fault Condition Matrix

Metric	Comparator	Low Threshold	High Threshold	Persistent Count
Mean TU Power (Pre-DPD)	Less Than	−37 dBFS	−46 dBFS	10
Peak TU Power (Pre-DPD)	Less Than	−26 dBFS	−35 dBFS	10
Mean Transmit Power (Post DPD)	Less Than	−37 dBFS	−46 dBFS	10
Peak Transmit Power (Post DPD)	Less Than	−26 dBFS	−35 dBFS	10
Mean Observation Receive power (Post DPD)	Less Than	−34 dBFS	−43 dBFS	10
Peak Observation Receive power (Post DPD)	Less Than	−23 dBFS	−32 dBFS	10
Pre-DPD Capture Transmit to Observation Receive EVM (Direct EVM)	Greater Than	5%	15%	10
Post DPD Capture Transmit to Observation Receive EVM (Indirect EVM)	Greater Than	8%	15%	10
Indirect Error	Greater Than	3%	12%	10
Select Error	Greater Than	3%	13%	10

The DPD recovery actions configured in the firmware on startup are listed in Table 257.

Table 257. Default Recovery Action Matrix

Metric	Recover Action if Low Threshold is Violated Once	Recover Action if Low Threshold Violation Persists	Recover Action if High Threshold is Violated Once	Recover Action if High Threshold Violation Persists
Mean TU Power (Pre-DPD)	Abandon DPD LUT update	Take no action	Take no action	Take no action
Peak TU Power (Pre-DPD)	Abandon DPD LUT update	Take no action	Take no action	Take no action
Mean Transmit Power (Post DPD)	Abandon DPD LUT update	Take no action	Take no action	Take no action
Peak Transmit Power (Post DPD)	Abandon DPD LUT update	Take no action	Take no action	Take no action
Mean Observation Receive Power (Post DPD)	Abandon DPD LUT update	Take no action	Take no action	Take no action
Peak Observation Receive power (Post DPD)	Abandon DPD LUT update	Take no action	Take no action	Take no action
Pre-DPD Capture Transmit to Observation Receive EVM (Direct EVM)	Take no action	Take no action	Take no action	Take no action
Post DPD Capture Transmit to Observation Receive EVM (Indirect EVM)	Take no action	Take no action	Take no action	Take no action
Indirect Error	Abandon DPD LUT update	Abandon the DPD LUT update, revert the LUTs to unity, reset DPD adaptation state, hardware reset the DPD actuator	Take no action	Take no action
Select Error	Abandon DPD LUT update	Abandon the DPD LUT update, revert the LUTs to unity, reset DPD adaptation state, hardware reset the DPD actuator	Take no action	Take no action

Note that the recovery actions follow a many to one mapping. Each of the four columns in the table indicate error conditions and each error condition has a unique metrics mask to which multiple fault condition metrics can be mapped by ‘OR’ing the metrics.

For example, in the matrix shown in Table 258, Persistent0 looks for violation of lower thresholds from either indirect EVM, mean TU power, or peak transmit power, whereas Persistent1 looks for high threshold violations from indirect EVM only to trigger a recovery action.

Table 258. Example settings for Recovery Actions

Error Condition	Error0, Fault Condition Threshold0 Violated Once	Persistent0, Fault Condition Threshold0 Violated Persistently	Error1, Fault Condition Threshold0 Violated Once	Persistent1, Fault Condition Threshold0 Violated Persistently
Recovery Action	Do nothing	Skip the DPD LUT updates	Do nothing	Skip the DPD LUT updates and reset the DPD coefficients to unity gain (actuator output = actuator input)
Fault Condition		Indirect EVM, mean TU power, or peak TU power		Indirect EVM only

The following block of code shows an example of how to setup the recovery actions shown in Table 258.

```
#Setup Recovery Action
dpdRecoverActionArr = Array.CreateInstance(Types.adi_adrv9025_DpdRecoveryActionConfig_t, 2)

#Recovery Action for violation of threshold0 - Only skip LUT updates for violation of soft threshold
Thresh0ActionMask = int(Types.adi_adrv9025_DpdRecoveryAction_e.ADI_ADRV9025_DPD_RECOVERY_ACTION_SKIP_LUTS_UPDATE)

#Recovery Action for violation of threshold1 -skip LUT updates, restore LUT to unity and reset adaptation state for
violation of hard threshold
thresh1ActionMask = 0
thresh1ActionMask |= int(Types.adi_adrv9025_DpdRecoveryAction_e.ADI_DPD_RECOVERY_ACTION_SKIP_LUTS_UPDATE)
thresh1ActionMask |= int(Types.adi_adrv9025_DpdRecoveryAction_e.ADI_DPD_RECOVERY_ACTION_REVERT_LUTS_TO_UNITY)
thresh1ActionMask |= int(Types.adi_adrv9025_DpdRecoveryAction_e.ADI_DPD_RECOVERY_ACTION_RESET_ADAPTATION_STATE)

#Configure Recovery Action for persistent violation of thershold0
dpdRecoveryActionArr[0] = Types.adi_adrv9025_DpdRecoveryActionConfig_t()
dpdRecoveryActionArr[0].dpdErrorState = Types.adi_adrv9025_DpdErrorState_e.ADI_DPD_ERR_STATE_PERSISTENT_0
dpdRecoveryActionArr[0].dpdRecoveryAction.dpdMetricsMask =
                                  int(Types.adi_adrv9025_DpdMetric_e.ADI_ADRV9025_DPD_METRIC_INDIRECT_EVM) |
                                  int(Types.adi_adrv9025_DpdMetric_e.ADI_ADRV9025_DPD_METRIC_MEAN_TU_POWER) |
                                  int(Types.adi_adrv9025_DpdMetric_e.ADI_ADRV9025_DPD_METRIC_PEAK_TX_POWER)
dpdRecoveryActionArr[0].dpdRecoveryAction.dpdActionMask = thresh0ActionMask

#Configure Recovery Action for persistent violation of thershold1
dpdRecoveryActionArr[1] = Types.adi_adrv9025_DpdRecoveryActionConfig_t()
dpdRecoveryActionArr[1].dpdErrorState = Types.adi_adrv9025_DpdErrorState_e.ADI_DPD_ERR_STATE_PERSISTENT_1
dpdRecoveryActionArr[1].dpdRecoveryAction.dpdMetricsMask = int(Types.adi_adrv9025_DPD_METRIC_INDIRECT_EVM)
dpdRecoveryActionArr[1].dpdRecoveryAction.dpdActionMask = thresh1ActionMask

#Program the recovery action in the device
Link.platform.board.Adrv9025Device.Dfe.DpdRecoveryActionSet(txChannelMask, dpdRecoveryActionArr, 2)
```

DPD ACTUATOR GAIN MONITORING FOR ROBUSTNESS

Principle of Operation

The DPD gain monitoring mechanism uses power meters at the input and output of the DPD actuator to determine when to switch between DPD models if the actuator gain violates a programmable threshold. The gain monitoring mechanism can be used to monitor gain over range as well as gain under range. Gain over range can occur when DPD is trying to expand gain to compensate for gain compression. A gain under range condition can occur because of bad coefficients or gain compression.

Gain can be monitored either sample by sample or averaged over a number of samples. The maximum number of samples that can be averaged is 128k samples (~266 μs worth of samples at a 491.52 MSPS rate). The average gain over several thousands of samples should not typically exceed 1 dB. The gain monitoring mechanism can be setup to switch to a unity gain or any other model if the gain/attenuation across the actuator is in the range of several dBs. Figure 228 represents a high level overview of the DPD actuator hardware in the signal chain.

Table 250 explains the DPD models implemented in the transceiver. The user has an option to switch to either one of these four DPD models in case the actuator output experiences high gain or high attenuation as explained in the previous section.

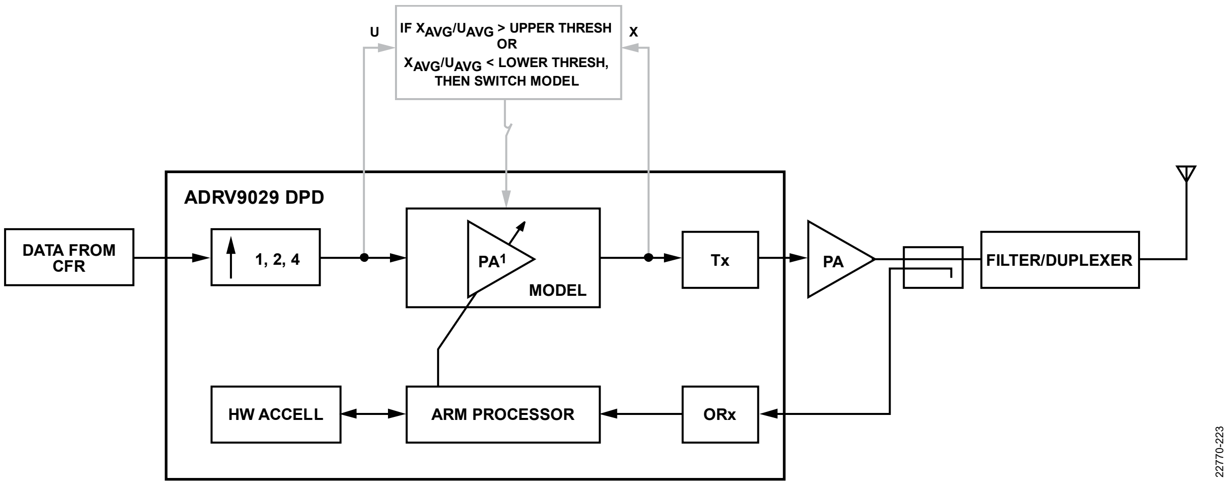


Figure 228. DPD Actuator Gain Monitoring Functional Diagram

DPD Actuator Gain Monitoring Configurations

The adjustments listed in Table 259 can be used to configure the DPD actuator gain monitoring feature.

Table 259. DPD Gain Monitoring Configurations

Gain Monitor Configuration	Description
adi_adrv9025_DpdActGainMonitorCtrl_t.dpdGainMonitorEnable	Enable/Disable gain monitoring.
adi_adrv9025_DpdActGainMonitorThresh_ t.dpdGainMonitorQualThresh	Minimum signal level above which gain monitoring is exercised. MSB 16 bits of $I^2 + Q^2$ 32 bit data.
adi_adrv9025_DpdActGainMonitorCtrl_ t.highGainModelAutoLoadEnable	Configuration to explicitly enable/disable high gain over range detection.
adi_adrv9025_DpdActGainMonitorCtrl_t	Configuration to explicitly enable/disable low gain under range detection.
adi_adrv9025_DpdActGainMonitorThresh_ t.dpdGainMonitorUpperThresh	High gain threshold above which a model switch can be initiated by the gain monitoring hardware. This configuration is only applicable if gain overrange detection is enabled.
adi_adrv9025_DpdActGainMonitorThresh_ t.dpdGainMonitorLowerThresh	Low gain threshold(attenuation) below which a model switch can be initiated by the gain monitoring hardware. This configuration is only applicable if gain under range detection is enabled.
adi_adrv9025_DpdActGainMonitorCtrl_ t.dpdGainMonitorUpperThreshModelSel	Switch to this model upon violation of the high gain threshold.
adi_adrv9025_DpdActGainMonitorCtrl_ t.dpdGainMonitorLowerThreshModelSel	Switch to this model upon violation of the low gain threshold.
adi_adrv9025_DpdActGainMonitorCtrl_t. dpdGainMonitorIIRDecay	This configuration controls the number of samples over which gain is averaged. The decay rate can be calculated as N = (65536/(averaging_ window_size/2)) where the decay rate is equal to $\log_2(N)$.

DPD Actuator Gain Monitoring API

The API functions used to control gain monitoring are listed in Table 260.

Table 260. DPD Actuator Gain Monitoring API

API Function	Description
adi_adrv9025_DpdActuatorGainMonitorConfigSet()	This function sets the DPD gain monitor configuration described in Table 261. Gain monitoring can be used to automatically switch to selected models when the actuator gain overrange or underrange is seen. Gain violation thresholds are user configurable. This function takes in as argument, a pointer to the device data structure, and a pointer to an adi_adrv9025_DpdActGainMonitorConfig_t structure that contains the configurations shown in Table 261.
adi_adrv9025_DpdActuatorGainMonitorConfigGet()	This function returns the DPD gain monitor configuration applied in the device for the requested transmit channel.

Table 261. Gain Monitor Configuration Parameters

Configuration	Value
Gain Monitor Enable	Enabled
Gain Detection Qualifying Threshold	−42 dBFS
Gain Overrange Detection Enable	Enabled
Gain Underrange Detection Enable	Enabled
Gain Overrange Threshold	6 dB
Gain Underrange Threshold	−6 dB
Gain Overrange Model Select	Unity gain (Model 3)
Gain Underrange Model Select	Recovery table (Model 2)
Decay Rate	1

The following code is an example python script to set up the gain monitor with the configuration described in Table 261.

```
def setupDpdGainMonitor():
    dpdGainMonitorCfg = Types.adi_adrv9025_DpdActGainMonitorConfig_t()
    dpdGainMonitorCfgGet = Types.adi_adrv9025_DpdActGainMonitorConfig_t()
    dpdGainMonitorCfg.txChannelMask = 0x01
    dpdGainMonitorCfg.dpdGainMonitorCtrl.dpdGainMonitorEnable = 1 #Enable Gain Monitoring
    dpdGainMonitorCfg.dpdGainMonitorCtrl.dpdGainMonitorIIREnable = 1 #Enable IIR for averaging samples. If set to 0, sample by sample gain
        detection is carried out
    dpdGainMonitorCfg.dpdGainMonitorCtrl.dpdGainMonitorIIRDecay = 1 #Sets the sample averaging window size to 128K samples
    dpdGainMonitorCfg.dpdGainMonitorCtrl.lowGainModelAutoLoadEnable = 1 #Enable DPD model switching on low gain threshold violation
    dpdGainMonitorCfg.dpdGainMonitorCtrl.highGainModelAutoLoadEnable = 1 #Enable DPD model switching on high gain threshold violation
    dpdGainMonitorCfg.dpdGainMonitorCtrl.dpdGainMonitorLowerThreshModelSel = Types.adi_adrv9025_DpdModelSel_e.ADI_ADRV9025_DPD_MODEL2 #Switch to
        R-Table on low gain violation
    dpdGainMonitorCfg.dpdGainMonitorCtrl.dpdGainMonitorUpperThreshModelSel = Types.adi_adrv9025_DpdModelSel_e.ADI_ADRV9025_DPD_MODEL3 #Switch to
        unity gain table on high gain violation
    dpdGainMonitorCfg.dpdGainMonitorThresh.dpdGainMonitorQualThresh = 1 #Upper 16 bits of I^2 +Q^2 32 bit data. A value of 1 equals -42dBFS
    dpdGainMonitorCfg.dpdGainMonitorThresh.dpdGainMonitorLowerThresh = 0x20 #Low gain threshold in 2.6 integer linear scale format
    dpdGainMonitorCfg.dpdGainMonitorThresh.dpdGainMonitorUpperThresh = 0x80 #High gain threshold in 2.6 integer linear scale format
    link.platform.board.Adrv9025Device.Dfe.DpdActuatorGainMonitorConfigSet(dpdGainMonitorCfg) |
    retVal = link.platform.board.Adrv9025Device.Dfe.DpdAtuatorGainConfigGet(Types.adi_adrv9025_TxChannels_e.ADI_ADRV9025_TX1,
        dpdGainMonitorCfgGet)
    dpdGainMonitorCfgGet = retVal[1]
    print "DPD Gain Monitor Enable = ", dpdGainMonitorCfgGet.dpdGainMonitorCtrl.dpdGainMonitorEnable
    print "DPD Gain Monitor IIR Enable = ", dpdGainMonitorCfgGet.dpdGainMonitorCtrl.dpdGainMonitorIIREnable
    print "DPD Gain Monitor IIR Decay = ", dpdGainMonitorCfgGet.dpdGainMonitorCtrl.dpdGainMonitorIIRDecay
    print "DPD Gain Monitor Low Gain Auto Model Ld En = ", dpdGainMonitorCfgGet.dpdGainMonitorCtrl.lowGainModelAutoLoadEnable
    print "DPD Gain Monitor High Gain Auto Model Ld En = ", dpdGainMonitorCfgGet.dpdGainMonitorCtrl.highGainModelAutoLoadEnable
    print "DPD Gain Monitor Qual Thresh = ", dpdGainMonitorCfgGet.dpdGainMonitorThresh.dpdGainMonitorQualThresh
    print "DPD Gain Monitor Low Gain Thresh = ", dpdGainMonitorCfgGet.dpdGainMonitorThresh.dpdGainMonitorLowerThresh
```

DPD Actuator Gain Monitoring and Model Switching State Machine Representation

The flow chart in Figure 229 describes the function of the gain monitoring state machine. The DPD gain monitoring, when enabled, runs independently from the DPD actuator. The DPD gain monitoring monitors the gain of the signal across the actuator until the DPD is turned off, as shown in the state diagram.

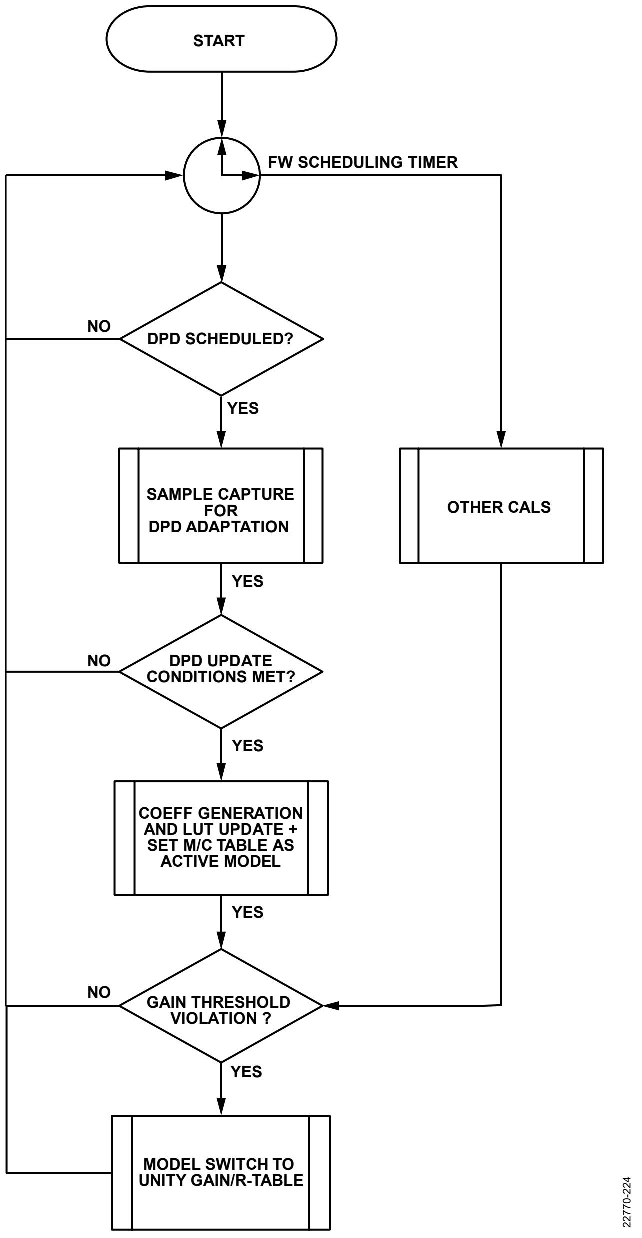

START
FW SCHEDULING TIMER
NO
DPD SCHEDULED?
YES
SAMPLE CAPTURE FOR DPD ADAPTATION
OTHER CALS
YES
NO
DPD UPDATE CONDITIONS MET?
YES
COEFF GENERATION AND LUT UPDATE + SET M/C TABLE AS ACTIVE MODEL
YES
NO
GAIN THRESHOLD VIOLATION ?
YES
MODEL SWITCH TO UNITY GAIN/R-TABLE
22770-224

Figure 229. DPD Gain Monitoring State Machine

DPD ACTUATOR BYPASS

The ADRV9029 DPD provides a mechanism to bypass predistortion through GPIO control. The GPIO input is level sensitive and activates the unity gain model for the duration of time where the GPIO level is high. The GPIO-based DPD actuator bypass is typically used for antenna calibrations in M-MIMO applications, where the predistortion must be disabled for the duration of antenna calibration to prevent predistortion from affecting antenna calibration accuracy. Figure 230 shows how this process is implemented.

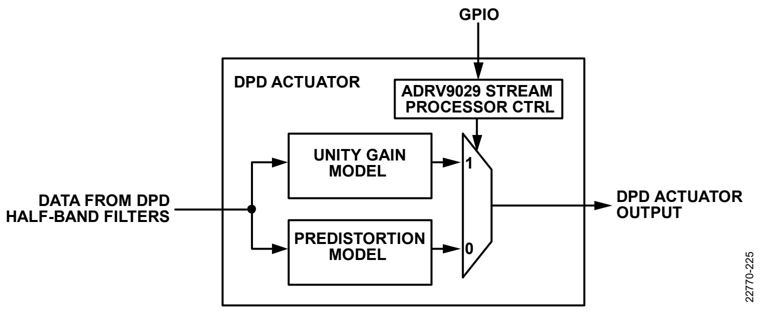
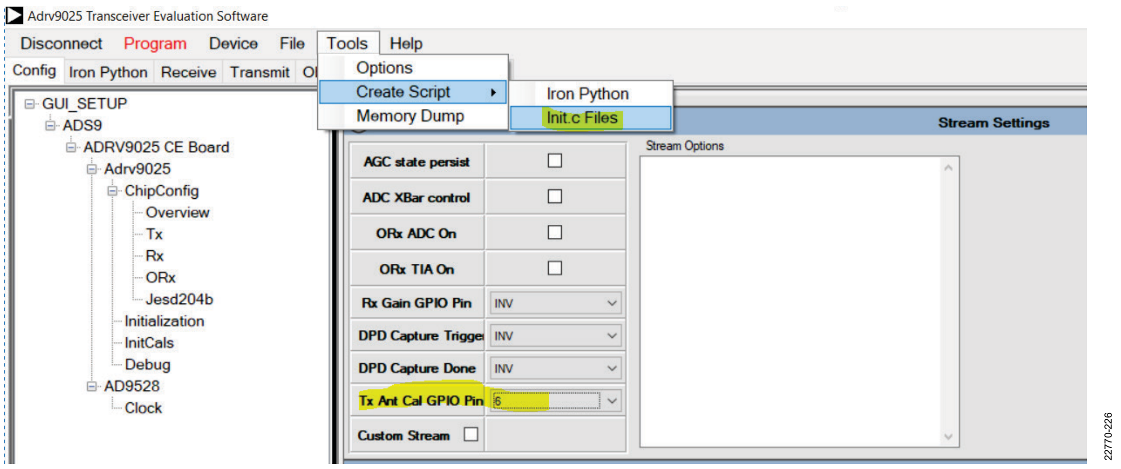


Figure 230. DPD Actuator Bypass through GPIO

The GPIO control for DPD actuator bypass is managed by the stream processor in the transceiver. To enable this feature, take the following steps:

1. Configure the stream to associate a GPIO with DPD actuator bypass control and generate a stream binary. The TES can be used to configure the GPIO for DPD actuator bypass and generate the stream binary. The **Tx Ant Cal GPIO Pin** value in the **Stream Settings** in the initialization page can be used to configure the GPIO, and the stream binary can be generated using the **Tools** > **Create Script** function shown in Figure 231.



AGC state persist	☐
ADC XBar control	☐
ORx ADC On	☐
ORx TIA On	☐
Rx Gain GPIO Pin	INV
DPD Capture Trigge	INV
DPD Capture Done	INV
Tx Ant Cal GPIO Pin	6
Custom Stream ☐	

Figure 231. DPD Actuator Bypass GPIO Pin Stream Configuration Using TES

2. Convey this stream configuration to the TES software by assigning the stream general-purpose input pins in the post MCS init structure. The following code is an example configuration where GPIO_06 is assigned to Stream GP Input 6 in the post MCS init structure.

```
{ // streamGpioCfg
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput0
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput1
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput2
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput3
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput4
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput5
    ADI_ADRV9025_GPIO_06,  // streamGpInput6
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput7
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput8
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput9
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput10
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput11
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput12
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput13
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput14
    ADI_ADRV9025_GPIO_INVALID,  // streamGpInput15
},
```

Ensure that the GPIO assigned for controlling the DPD actuator bypass is driven to a low state during initialization to prevent interference with initial calibrations.

DPD STATUS

The user can obtain the status of the DPD tracking calibration during runtime through the adi_adrv9025_DpdStatusGet() API command, which updates a adi_adrv9025_DpdStatus_t data structure supplied by the user. The DPD status data structure returns the information shown in Table 262.

Table 262. DPD Recovery Action Bit-mask Definition

Member	Data Type	Description
dpdErrorCode	adi_adrv9025_DpdError_e	DPD error status. Refer to the **adi_adrv9025_dfe_types.h** file for a full list of DPD error codes returned by the DPD status. If the DPD is functioning correctly, this parameter returns ADI_ADRV9025_DPD_NO_ERROR.
dpdPercentComplete	UInt32	Percentage of DPD update completed.
dpdIterCount	UInt32	Number of DPD updates scheduled.
dpdUpdateCount	UInt32	Number of successful DPD updates.
dpdSyncStatus	adi_adrv9025_ TrackingCalSyncStatus_e	DPD and CLGC tracking calibrations are synchronized through a semaphore mechanism. This member returns SYNC_OK if both the DPD and CLGC are synchronized. If the CLGC is not enabled, this field can be ignored.
dpdModelTable	adi_adrv9025_DpdModelTableSel_e	Currently active DPD model (M table, C table, or U table) described in Table 250.
dpdStatistics	adi_adrv9025_DpdStatistics_t	Current values of DPD stability metrics described in Table 256.

RECOMMENDED SEQUENCE FOR ENABLING THE DPD TRACKING CALIBRATION

The sequence for running DPD tracking calibrations is shown in Table 263.

Table 263. DPD Tracking Calibration Bring Up Sequence

Step	Action	APIs used
1	Program the device and run initial calibrations(including transmit QEC initial calibration) with the power amplifier turned off.	Utility function adi_daughterboard_Program() can be used to program the device
2	Setup external transmit to observational receive mapping.	adi_adrv9025_TxToOrxMappingSet
3	Adjust observational receive gain to an appropriate value to avoid saturation. The default gain index in ADRV9025 is 255 (0dB attenuation).	adi_adrv9025_RxGainSet
4	Turn on the power amplifier and run the external path delay initial calibration. Alternatively, factory calibrated path delay values can be programmed via ExternalPathDelaySet() commands.	adi_adrv9025_InitCalsRun for calibrating path delay, adi_adrv9025_ExternalPathDelaySet(), and adi_adrv9025_ClgcExternalPathDelaySet()
5	Run the transmit external LO Leakage initial calibration.	adi_adrv9025_InitCalsRun
6	If using ADRV9025 CFR, configure the CFR settings.	adi_adrv9025_CfrConfigSet, adi_adrv9025_CfrEnableSet, and adi_adrv9025_CfrCorrectionPulseWrite_v2
7	If using ADRV9025 CFR, run the CFR initial calibration.	adi_adrv9025_InitCalsRun
8	Load the DPD model.	adi_adrv9025_DpdModelConfigSet
9	Assert DPD reset.	adi_adrv9025_DpdReset
	If a unique DPD model is required to be applied to each Model, proceed to follow Step 10 to Step 17.	
10	Setup the DPD mode of operation, DPD peak search window size, and low power threshold.	adi_adrv9025_DpdTrackingConfigSet
11	Set up the DPD fault conditions and recovery actions(optional).	adi_adrv9025_DpdFaultConditionSet, adi_adrv9025_DpdRecoveryActionSet
12	Set up CLGC configurations and target loop gain.	adi_adrv9025_ClgcConfigSet
13	Enable the transmit QEC and transmit LO Leakage tracking calibrations.	adi_adrv9025_TrackingCalsEnableSet
14	Enable DPD tracking calibration.	adi_adrv9025_TrackingCalsEnableSet
15	Enable CLGC tracking calibration.	adi_adrv9025_TrackingCalsEnableSet
16	Monitor DPD tracking calibration status.	adi_adrv9025_DpdStatusGet
17	Monitor CLGC tracking calibration status.	adi_adrv9025_ClgcStatusGet

DPD STABILITY METRICS CHARACTERIZATION

The following stability metrics are at the disposal of the user for defining stability:

- Transmit power thresholds
- Observed power thresholds
- Direct EVM, the difference between measured pre-DPD transmitted and observed samples
- Indirect EVM, the difference between measured post DPD transmitted and observed samples
- Indirect error, the difference between measured post DPD transmitted and predicted samples (predicted samples are obtained by applying the DPD model to the observed samples)
- Select error, the same as indirect error but computed for post DPD transmitted samples whose amplitude is greater than –30 dBFS

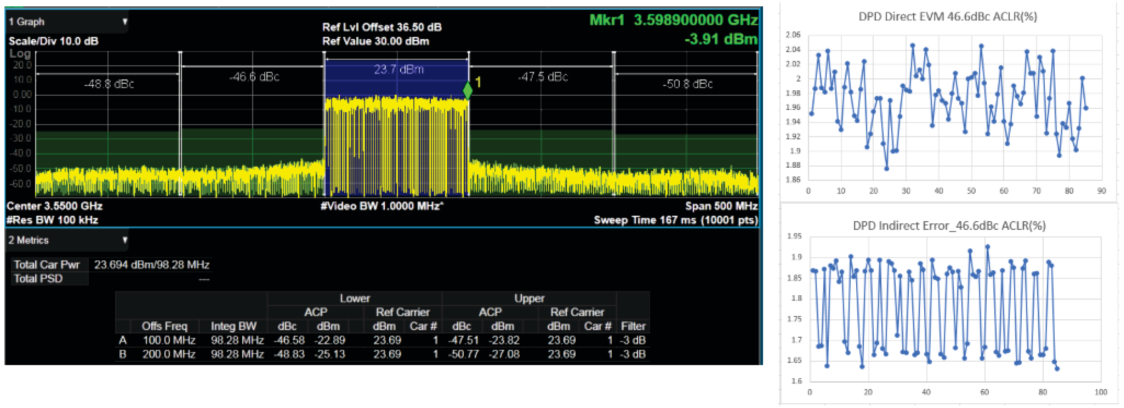
All of these metrics are user configurable. The following section provides characterization data that can provide some guidance regarding factors that can influence the configuration of the stability metrics defined.

Measuring DPD Adaptation Performance Through Direct EVM and Indirect Error

The following are three cases where the stability metrics direct EVM and indirect error parameters are compared for different ACLR performance levels of an NR100 signal TM3.1 signal.

Case 1

The ACLR performance is ~46.6 dBc in L3 channel and corresponds to a direct EVM of 2%.



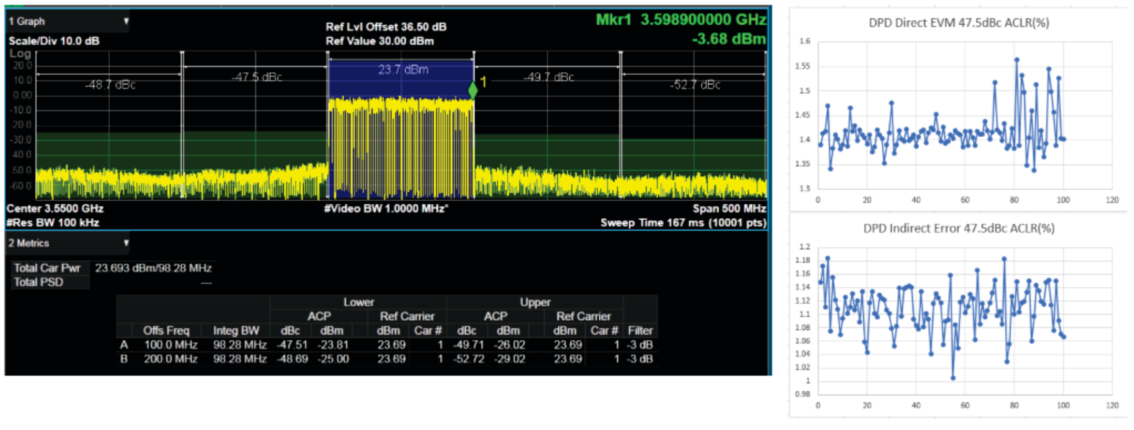
Total Car Pwr	23.694 dBm/98.28 MHz
Total PSD	---

			Lower				Upper				
			ACP		Ref Carrier		ACP		Ref Carrier		
	Offs Freq	Integ BW	dBc	dBm	dBm	Car #	dBc	dBm	dBm	Car #	Filter
A	100.0 MHz	98.28 MHz	-46.58	-22.89	23.69	1	-47.51	-23.82	23.69	1	-3 dB
B	200.0 MHz	98.28 MHz	-48.83	-25.13	23.69	1	-50.77	-27.08	23.69	1	-3 dB

Figure 232. Bad ACLR Performance Correlation with a High Direct EVM Value

Case 2

The ACLR performance is better (~47.5 dBc) than Case 1, which also corresponds to reduced direct EVM.



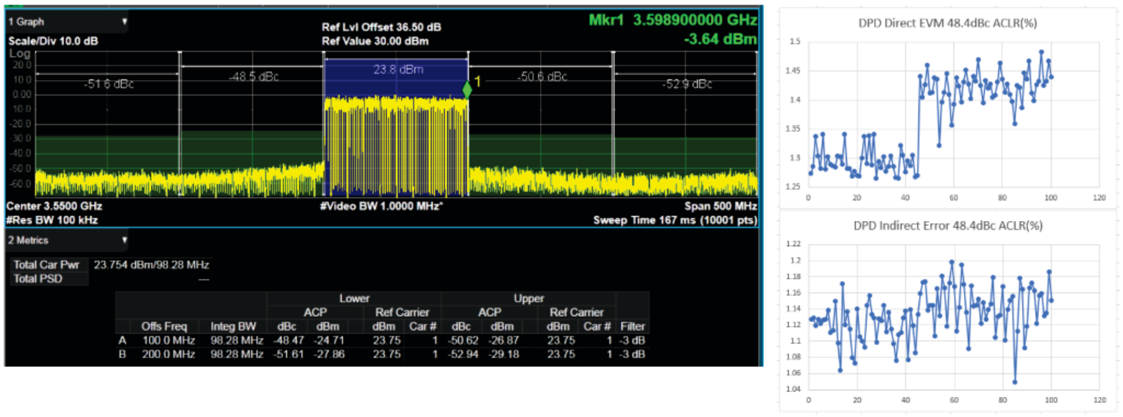
Total Car Pwr	23.693 dBm/98.28 MHz
Total PSD	---

			Lower				Upper				
			ACP		Ref Carrier		ACP		Ref Carrier		
	Offs Freq	Integ BW	dBc	dBm	dBm	Car #	dBc	dBm	dBm	Car #	Filter
A	100.0 MHz	98.28 MHz	-47.51	-23.81	23.69	1	-49.71	-26.02	23.69	1	-3 dB
B	200.0 MHz	98.28 MHz	-48.69	-25.00	23.69	1	-52.72	-29.02	23.69	1	-3 dB

Figure 233. Improved ACLR Performance Corresponds to Improved Direct EVM

Case 3

The ACLR performance is optimal amongst the three cases (~48.5 dBc), which is reflected in the reduced direct EVM numbers.



Total Car Pwr	23.754 dBm/98.28 MHz
Total PSD	---

			Lower				Upper				
			ACP		Ref Carrier		ACP		Ref Carrier		
	Offs Freq	Integ BW	dBc	dBm	dBm	Car #	dBc	dBm	dBm	Car #	Filter
A	100.0 MHz	98.28 MHz	-48.47	-24.71	23.75	1	-50.62	-26.87	23.75	1	-3 dB
B	200.0 MHz	98.28 MHz	-51.61	-27.86	23.75	1	-52.94	-29.18	23.75	1	-3 dB

Figure 234. Direct EVM for the Optimal ACLR Performance

Observation Receiver Attenuation vs. Stability Metrics

Shown below is the trend for the EVM and error stability metrics for different observation receiver channel attenuation values. It can be observed that as observation receiver attenuation is increased, the error percentage also increases. The same can be true for a low SNR transmitter to observation receiver channel. It is advised to increase the threshold for stability metrics as the observation receiver attenuation increases or the channel SNR decreases.

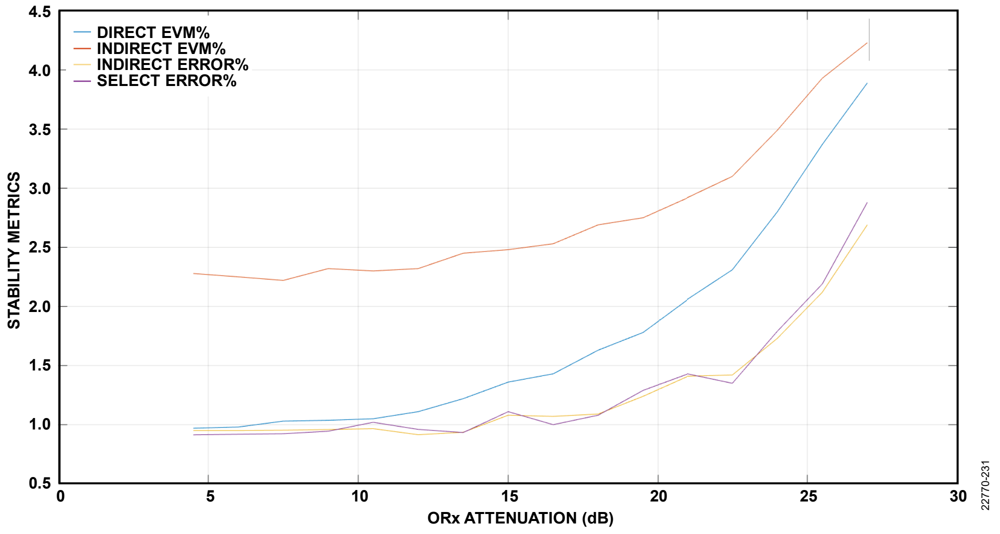


Figure 235. Observation Receive Attenuation vs. Stability Metrics

Observation Receiver Interference

Although unlikely, any interference in the observation receiver channel can cause DPD instability and can cause the firmware to derive incorrect DPD coefficients that result in poor performance, as shown in Figure 236, which includes the bench characterization of the observation receiver interference levels affecting ACLR and stability metrics. Depending on the application and the ACLR performance, the user can budget for the threshold of stability metrics. T1 represents the ACLR degradation at 5% indirect EVM, and T2 represents ACLR degradation at 15% EVM. Data2 represents the absolute value of ACLR in dBm as the power level of the interferer injected into the observation receiver increases. The total carrier power of the primary signal used for characterization is 26.23 dBm.

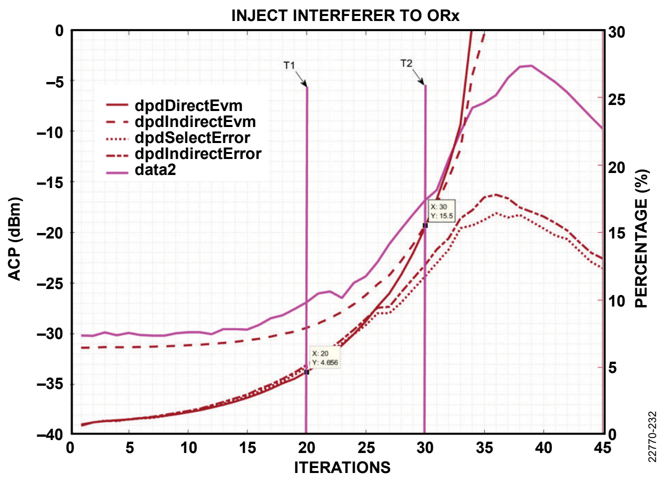


Figure 236. Observation Receive Interference vs. Stability Metrics

Transmit Signal vs. Stability Metrics

Shown in Figure 237 is the degradation of stability metrics with decreasing transmitter signal power. When the signal level is close to −36 dBFS, it can be observed that the EVM percentages are close to 5%. At a transmitter signal level close to −46 dBFS, the EVM percentages are close to 15%, which causes further ACLR degradation.

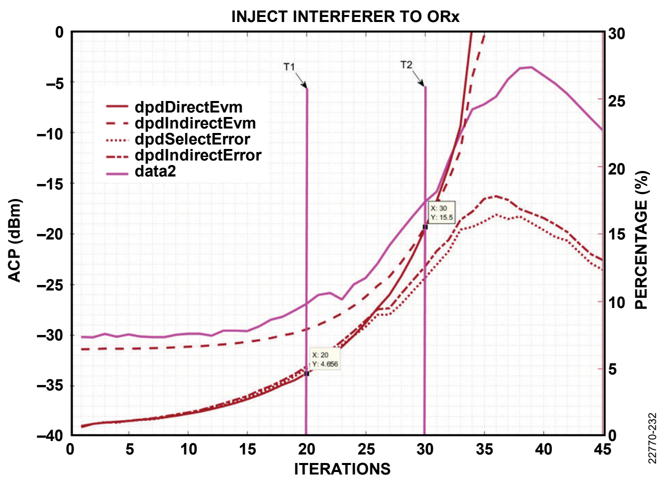


Figure 237. Transmit Signal Level vs. ACLR Degradation

Summary

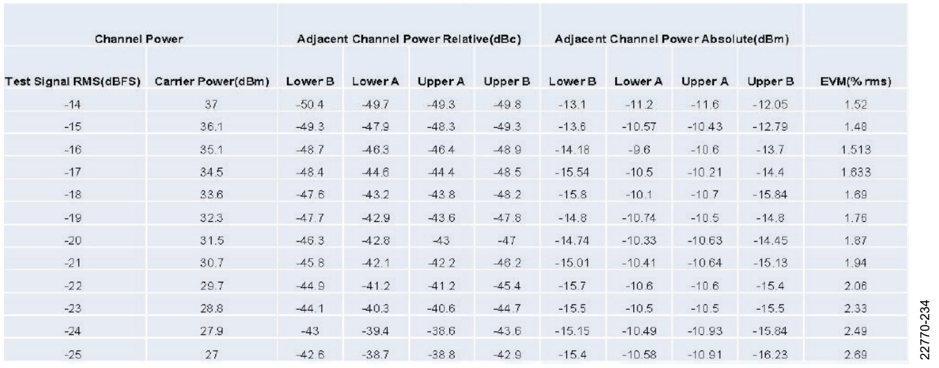
The following are some of the scenarios that could cause degradation of DPD performance. However, it is advised that the user characterize the system under test for EVM corruption that is specific to the system or conditions prevalent before configuring the thresholds.

- The DPD performance can be measured by direct EVM. The direct EVM numbers are lower when the performance on the DPD adaptation is acceptable.
- As observation receive attenuation increases, an increase in EVM percentages can be observed.
- Interference or high noise levels in transmit and observation receive channels can cause the EVM and error percentages to increase. Fault conditions and corresponding recovery actions can be defined for EVM numbers to avoid incorrect DPD updates.
- As the transmit signal level decreases, the EVM percentages increase. However, an argument can be made that DPD might not be required at lower signal levels for certain power amplifiers.
- Take note of the DPD model that the user configures the part with. An incompatible prior DPD model configured by the user can cause the EVM and error percentages to increase leading to poor DPD performance.
- Catastrophic conditions such as loss of signals can also lead to high EVM and error percentages that can be monitored by the user.

DPD CHARACTERIZATION FOR OPTIMIZING THE M THRESHOLD

To complete a DPD characterization test for optimizing the M table threshold in DPD Mode 2, take the following steps:

3. Bring up the DPD with a full power, full bandwidth signal (for example, TM3.1a at −14 dBFS and +100 MHz bandwidth).
4. When the DPD converges, turn off the DPD.
5. Sweep the power in 1 dB steps from full power to a ~15 dB backed off level.
6. Record the ACPR, ACP, and EVM of the demodulated data at each level. See Figure 238 for an example tabulation of characterization data.



Channel Power		Adjacent Channel Power Relative(dBc)				Adjacent Channel Power Absolute(dBm)				
Test Signal RMS(dBFS)	Carrier Power(dBm)	Lower B	Lower A	Upper A	Upper B	Lower B	Lower A	Upper A	Upper B	EVM(% rms)
-14	37	-50.4	-49.7	-49.3	-49.8	-13.1	-11.2	-11.6	-12.05	1.52
-15	36.1	-49.3	-47.9	-48.3	-49.3	-13.6	-10.57	-10.43	-12.79	1.48
-16	35.1	-48.7	-46.3	-46.4	-48.9	-14.18	-9.6	-10.6	-13.7	1.513
-17	34.5	-48.4	-44.6	-44.4	-48.5	-15.54	-10.5	-10.21	-14.4	1.633
-18	33.6	-47.6	-43.2	-43.8	-48.2	-15.8	-10.1	-10.7	-15.84	1.69
-19	32.3	-47.7	-42.9	-43.6	-47.8	-14.8	-10.74	-10.5	-14.8	1.76
-20	31.5	-46.3	-42.8	-43	-47	-14.74	-10.33	-10.63	-14.45	1.87
-21	30.7	-45.8	-42.1	-42.2	-46.2	-15.01	-10.41	-10.64	-15.13	1.94
-22	29.7	-44.9	-41.2	-41.2	-45.4	-15.7	-10.6	-10.6	-15.4	2.06
-23	28.8	-44.1	-40.3	-40.6	-44.7	-15.5	-10.5	-10.5	-15.5	2.33
-24	27.9	-43	-39.4	-38.6	-43.6	-15.15	-10.49	-10.93	-15.84	2.49
-25	27	-42.6	-38.7	-38.8	-42.9	-15.4	-10.58	-10.91	-16.23	2.69

Figure 238. DPD Characterization for Optimizing M-Threshold in DPD Mode 2

For this example, the characterization data relative to a 45 dBc ACP specification and 2.5% EVM is plotted in Figure 239, which shows that, although DPD performance is sustained across power levels, the EVM violates the 2.5% specification at approximately −24 dBFS. Based on the characterization data, it can be determined that that optimum M table threshold in DPD Mode 2 is −24 dBFS.

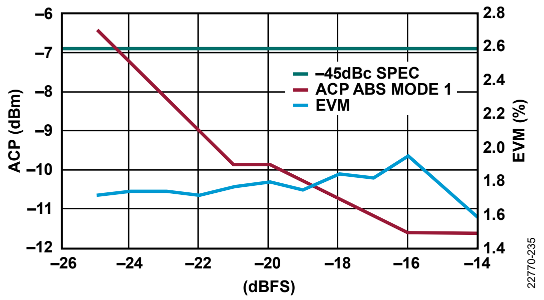


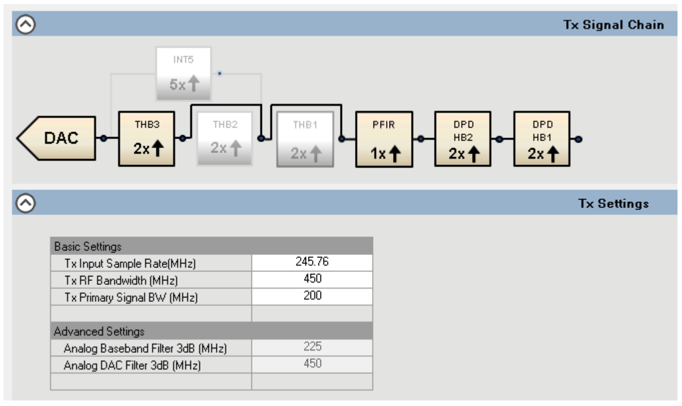
Figure 239. DPD Characterization Data Plot for Characterizing the M Threshold

SETTING UP THE DPD USING THE GUI

The DPD tab on the ADRV9029 TES GUI is the primary evaluation tool for the DPD feature. In addition, the DPD API and DLL can be used to interact and control the DPD via Python or C#. The ADRV9029 GUI supports an IronPython tab that can be used for scripting purposes.

Pay close attention when adjusting signal levels at the transmit output and observation receive input to protect the power amplifier under evaluation and achieve the desired performance. Before enabling the DPD, enable the ADRV9029 power amplifier protection feature to prevent unexpected signal levels that damage the power amplifier under evaluation.

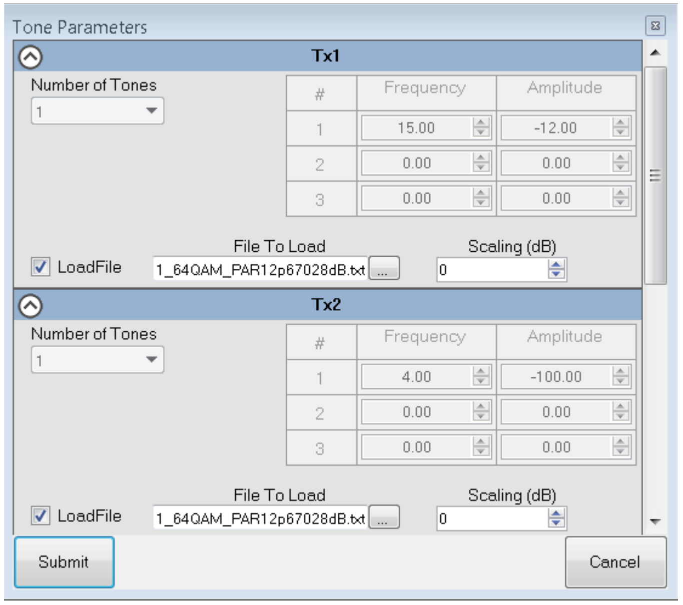
The following section describes how to set up the DPD on the device using the TES. The user must ensure that the device is programmed using one of the DPD profiles where either one or both of the DPD half-bands are enabled. Use Case 51 is shown in Figure 240. While programming the device, the power amplifier must be turned off to avoid any damage from high amplitude signals transmitted during initial calibration. The user must ensure that the transmit to observation receive mapping is correct and that the observation receive LO is configured to use the transmit LO. When initialization is complete, the power amplifier can be turned on.



Basic Settings	
Tx Input Sample Rate(MHz)	245.76
Tx RF Bandwidth (MHz)	450
Tx Primary Signal BW (MHz)	200
Advanced Settings	
Analog Baseband Filter 3dB (MHz)	225
Analog DAC Filter 3dB (MHz)	450

Figure 240. Use Case 51 Transmit Datapath Overview

The user can load the desired waveform using the **Tone Parameters** pop-up window in the **Transmit** tab in the TES, as shown in Figure 241. If desired, the peak to average ratio (PAR) of the waveform can be adjusted using the ADRV9029 CFR feature (refer to the Crest Factor Reduction (CFR) section). Alternatively, a waveform with CFR applied to it can be loaded.



Tx1

Number of Tones: 1

#	Frequency	Amplitude
1	15.00	-12.00
2	0.00	0.00
3	0.00	0.00

LoadFile ✓ File To Load: 1_64QAM_PAR12p67028dB.txt Scaling (dB): 0

Tx2

Number of Tones: 1

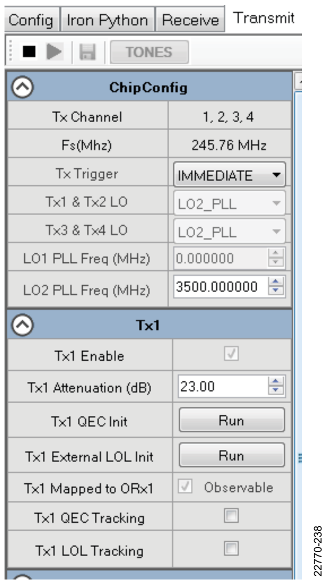
#	Frequency	Amplitude
1	4.00	-100.00
2	0.00	0.00
3	0.00	0.00

LoadFile ✓ File To Load: 1_64QAM_PAR12p67028dB.txt Scaling (dB): 0

Figure 241. Load Waveform Using TES

The DPD adaptation results in an expansion at the peak values of the waveform. The user can allow adequate headroom (3 dB to 5 dB) by typing the correct value in the **Scaling (dB)** textbox shown in Figure 241. Alternatively, the waveform can be scaled prior to loading.

When the waveform is loaded, set the **Tx Attenuation (dB)** value to get the desired output power as shown in Figure 242. To transmit the waveform, click the play button in the **Transmit** tab. When the waveform has been played, the user can read the power at the output of the power amplifier via a spectrum analyzer or a power meter (more accurate method).


Config | Iron Python | Receive | Transmit

TONES

ChipConfig

Tx Channel	1, 2, 3, 4
Fs(Mhz)	245.76 MHz
Tx Trigger	IMMEDIATE
Tx1 & Tx2 LO	LO2_PLL
Tx3 & Tx4 LO	LO2_PLL
LO1 PLL Freq (MHz)	0.000000
LO2 PLL Freq (MHz)	3500.000000

Tx1

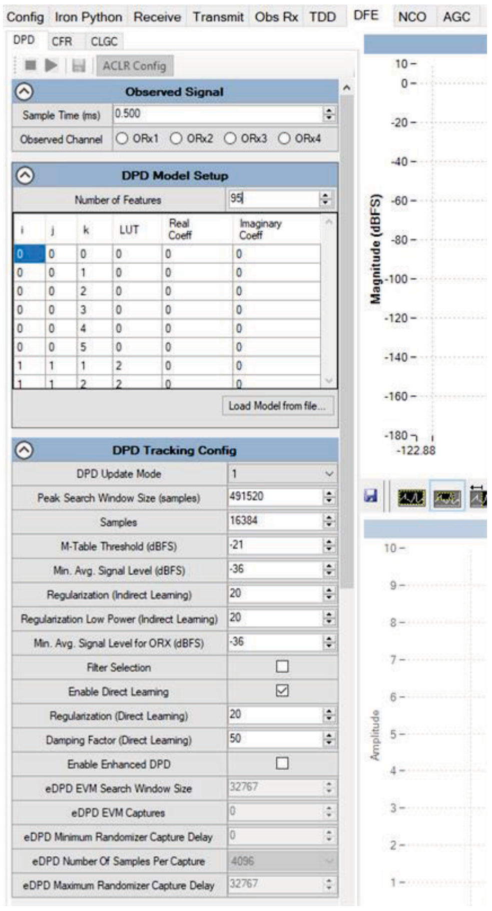
Tx1 Enable	✓
Tx1 Attenuation (dB)	23.00
Tx1 QEC Init	Run
Tx1 External LOL Init	Run
Tx1 Mapped to ORx1	✓ Observable
Tx1 QEC Tracking	☐
Tx1 LOL Tracking	☐

Figure 242. Setting Transmit Attenuation and Playing the Waveform

The user can check to make sure observation receive is not saturated or too close to the noise floor. The observation receive gain can be adjusted to get appropriate signal levels.

In the DFE tab shown in Figure 243, there is a specific subtab for DPD. The DPD subtab allows the user to fully configure and observe the DPD. The ACLR measurement window is not available at this point (the play button is not functional and is not used to enable any DPD feature). To configure the DPD, take the following steps:

1. Load the model file from the PC (model files are provided by Analog Devices). Ensure that the real coefficient of the linear term (i = 1, j = 1, k = 0) is set to 1.



Observed Signal

Sample Time (ms)	0.500
Observed Channel	ORx1 ORx2 ORx3 ORx4

DPD Model Setup

Number of Features: 95

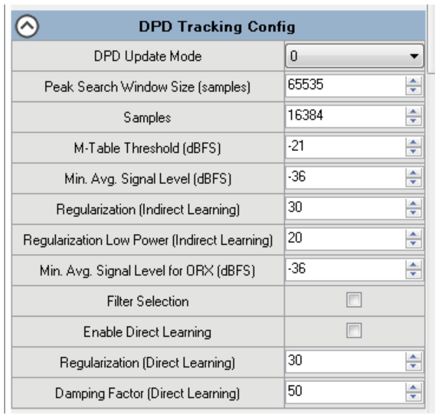
i	j	k	LUT	Real Coeff	Imaginary Coeff
0	0	0	0	0	0
0	0	1	0	0	0
0	0	2	0	0	0
0	0	3	0	0	0
0	0	4	0	0	0
0	0	5	0	0	0
1	1	1	2	0	0
1	1	2	2	0	0

DPD Tracking Config

DPD Update Mode	1
Peak Search Window Size (samples)	491520
Samples	16384
M-Table Threshold (dBFS)	-21
Min. Avg. Signal Level (dBFS)	-36
Regularization (Indirect Learning)	20
Regularization Low Power (Indirect Learning)	20
Min. Avg. Signal Level for ORX (dBFS)	-36
Filter Selection	☐
Enable Direct Learning	☑
Regularization (Direct Learning)	20
Damping Factor (Direct Learning)	50
Enable Enhanced DPD	☐
eDPD EVM Search Window Size	32767
eDPD EVM Captures	0
eDPD Minimum Randomizer Capture Delay	0
eDPD Number Of Samples Per Capture	4096
eDPD Maximum Randomizer Capture Delay	32767

Figure 243. Load DPD Model File

2. Configure the parameters in the **DPD Tracking Config,** shown in Figure 244**.** (the default values provide a sufficient starting point).



DPD Tracking Config

DPD Update Mode	0
Peak Search Window Size (samples)	65535
Samples	16384
M-Table Threshold (dBFS)	-21
Min. Avg. Signal Level (dBFS)	-36
Regularization (Indirect Learning)	30
Regularization Low Power (Indirect Learning)	20
Min. Avg. Signal Level for ORX (dBFS)	-36
Filter Selection	☐
Enable Direct Learning	☐
Regularization (Direct Learning)	30
Damping Factor (Direct Learning)	50

*Figure 244. Configure **DPD Tracking Config***

3. Select the desired transmit channel to apply the settings to.
4. Apply the DPD tracking configuration by clicking **Apply Tracking Config**, as shown in Figure 245.
5. Run the path delay initial calibration by clicking **Run Path Delay Init Cal**.
6. Apply the DPD model on the M table or C table by clicking **Apply Model on Device from M Table** or **Apply Model on Device from C Table** (see Figure 245).
7. Click **Enable DPD on selected channels (only)** to enable the DPD tracking.

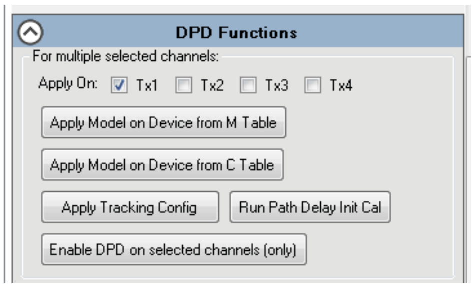
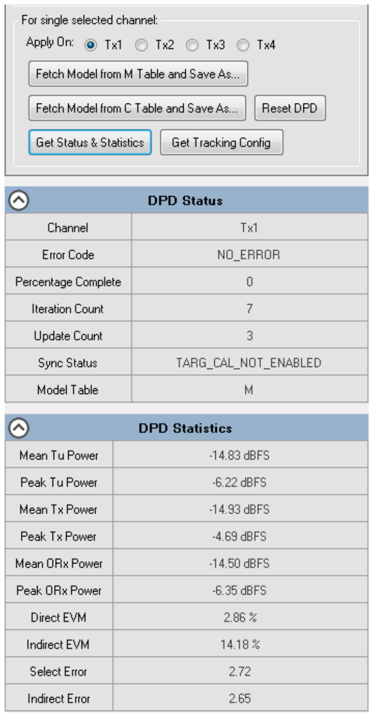

DPD Functions
For multiple selected channels:
Apply On: [✓ Tx1] Tx2 Tx3 Tx4
Apply Model on Device from M Table
Apply Model on Device from C Table
Apply Tracking Config
Run Path Delay Init Cal
Enable DPD on selected channels (only)
22770-241

Figure 245. Apply DPD Settings

8. Click **Get Status & Statistics** to reveal the DPD status and statistics for the respective transmit channel (see Figure 246).
9. Click **Reset DPD**, as shown in Figure 246, to apply a full reset to the DPD.
10. From the functional window in Figure 246, the user can fetch the model on the device by clicking **Fetch Model from M Table and Save As** and **Fetch Model from C Table and Save As**.
11. To fetch the DPD tracking configuration that the chip is currently in, click **Get Tracking Config** in the same window.


For single selected channel:
Apply On: [✓ Tx1] Tx2 Tx3 Tx4
Fetch Model from M Table and Save As...
Fetch Model from C Table and Save As...
Reset DPD
Get Status & Statistics
Get Tracking Config
DPD Status
DPD Statistics
22770-242

Channel	Tx1
Error Code	NO_ERROR
Percentage Complete	0
Iteration Count	7
Update Count	3
Sync Status	TARG_CAL_NOT_ENABLED
Model Table	M

Mean Tu Power	-14.83 dBFS
Peak Tu Power	-6.22 dBFS
Mean Tx Power	-14.93 dBFS
Peak Tx Power	-4.69 dBFS
Mean ORx Power	-14.50 dBFS
Peak ORx Power	-6.35 dBFS
Direct EVM	2.86 %
Indirect EVM	14.18 %
Select Error	2.72
Indirect Error	2.65

Figure 246. Get DPD Status and Statistics

To change the DPD model or apply a different tracking configuration parameter, take the following steps:

1. Clear the transmit channel under consideration in Figure 245 and click Enable DPD on selected channels (only) to disable the DPD tracking.
2. Click **Reset DPD** to apply a full reset.
3. Load the model file from the PC (model files are provided by Analog Devices). Ensure that the real coefficient of the linear term (i = 1, j = 1, k = 0) is set to 1.
4. Configure the parameters in the **DPD Tracking Config**.
5. Select the desired transmit channel to apply the settings to (Figure 245).
6. Click **Apply Tracking Config**, as shown in Figure 245, to apply the DPD tracking configuration.
7. Apply DPD model on the M or C tables by clicking **Apply Model on Device from M Table** and **Apply Model on Device from C Table** buttons (see Figure 245).
8. Click **Enable DPD on selected channels (only)** to enable DPD tracking.

CREST FACTOR REDUCTION (CFR)

The ADRV9029 variant provides CFR to assist in keeping power amplifiers linear. This functionality is only available in the ADRV9029 variant. A typical communications RF subsystem consists of an antenna, power amplifier, and RF transceiver that translates digital baseband signals to RF, as shown in Figure 247.

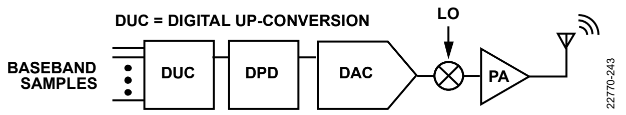


Figure 247. Typical RF Subsystem

It is highly desirable to drive the power amplifier at the highest input power possible without having the power amplifier saturate. Most modern communications protocols such as LTE are OFDM-based in which the final waveform is an orthogonal summation of subcarriers that carry information, and where each subcarrier has its own center frequency and modulation scheme. In the time domain, sometimes the peaks of these subcarriers can align to produce an aggregate large OFDM waveform peak (see Figure 251).

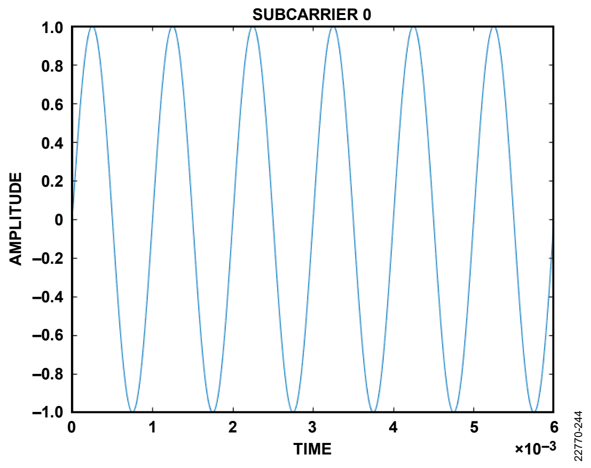


Figure 248. Time Domain View of a 1 kHz Carrier

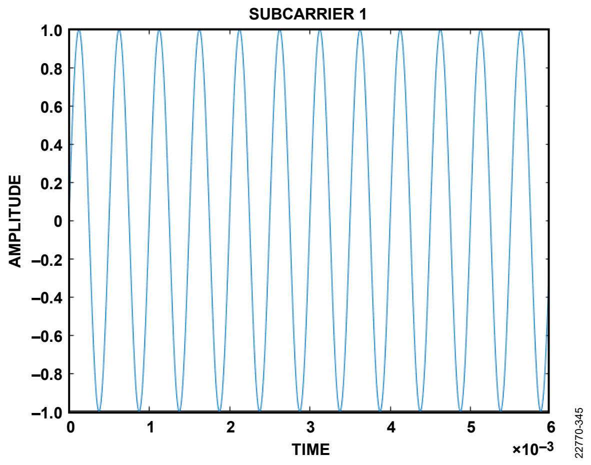


Figure 249. Time Domain View of a 2 kHz Carrier

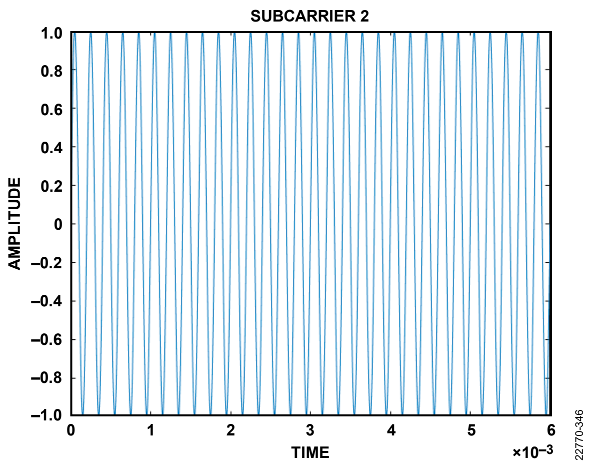


Figure 250. Time Domain View of a 5 kHz Carrier

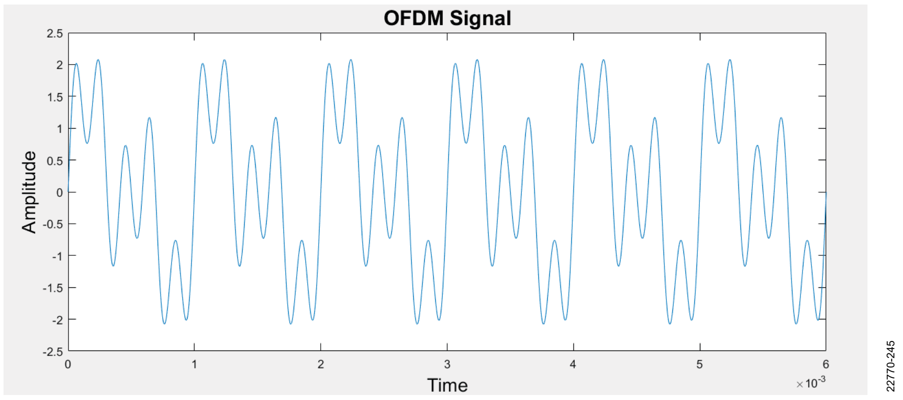


Figure 251. Example illustration of Orthogonal Summation of Subcarriers Causing Large Peaks in an OFDM Waveform

These peaks increase the overall dynamic range needed for the OFDM signal through a signal chain, which leads to an increase in the PAR of the signal.

Modern communication power amplifiers used to amplify such OFDM waveforms are only linear for a certain power range. Most of the input signal (average power) are within this linear range. However, the signal can have peaks that exceed the power amplifier linear operation range. To avoid saturation of the output signal because of these peaks, the user can potentially attenuate the desired signal. This method ensures that the range required by the signal is within the power amplifier linear range. However, this method is undesirable as it reduces the average power at the expense of maintaining a given PAR, which makes the system less efficient. An alternative to attenuation is to use CFR where, instead of attenuating the whole signal, the user attenuates portions of the signal that are above the power amplifier linear range. This method results in a constant output power while reducing the PAR and thus ensuring that the signal remains within the power amplifier linear range (see Figure 252 for a summary).

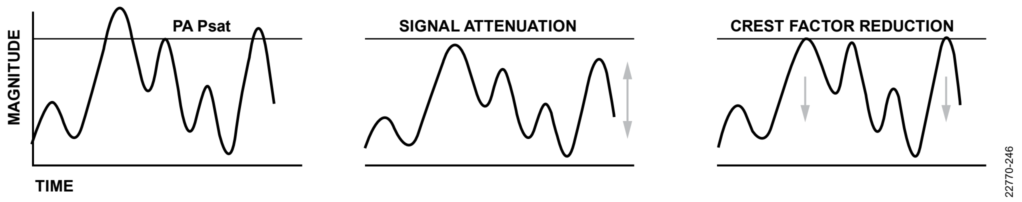


Figure 252. Effect of Signal Attenuation vs. CFR

It is important to note that CFR leads to higher in-band and out of band noise levels. This effect results in EVM degradation while also increasing the noise power spectral density, which results in an increase in ACLR. It is important to optimize the CFR algorithm to make sure that the CFR block impact is within the user system level specifications (derived from 3GPP specifications or other regulatory standards).

CFR ALGORITHM OVERVIEW

Several CFR algorithms and implementations have been published in the CFR literature. Only a few have been commercially viable. Clipping and filtering and pulse cancellation are two of the more popular techniques. Even though optimal cancellation of peaks is technically achievable (target PAR requested is met exactly), the latency requirements imposed by modern communication standards makes the design of a real-time CFR engine challenging. Spectral regrowth of corrected peaks by interpolating stages in the datapath, such as interpolating filters and DACs, is also a concern. An ideal CFR block has low latency and zero missed peaks.

The ADRV9029 implements CFR using a variation of the pulse cancellation technique by subtracting a precomputed pulse from the detected peaks to bring the signal within the power amplifier linear range. The CFR block consists of three copies of CFR engines, each of which uses a detection threshold to detect the peaks and a correction threshold to which the detected peaks are attenuated. The precomputed pulses can be stored within the device at startup or be updated during runtime. These spectrally shaped correction pulses are subtracted from the data stream to bring the signal within the power amplifier linear range. The correction pulse must be spectrally shaped to manage the noise leakage into adjacent bands. Figure 253 shows the architecture implemented in each of the CFR engines.

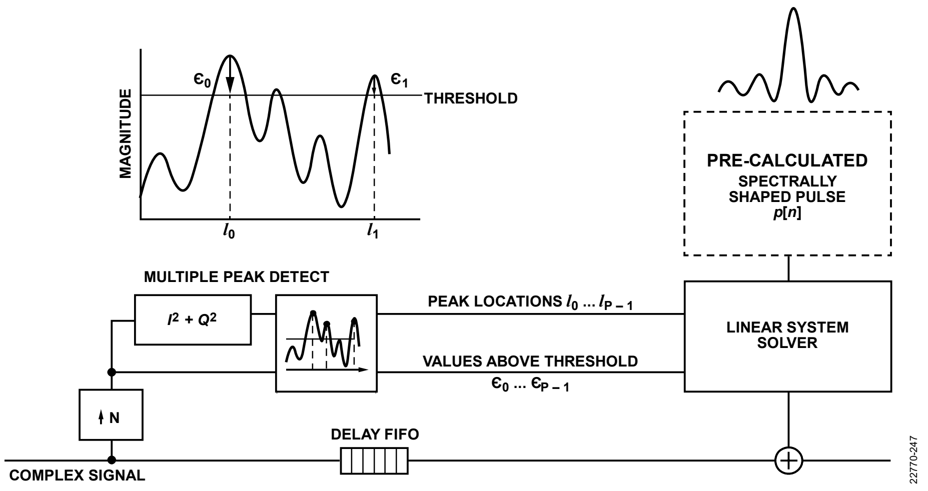


Figure 253. CFR Engine Architecture in ADRV9029

The complex IQ signal (transmitter data) goes into a variable delay FIFO and correction is applied at its output. The input data also goes into an interpolator, which can interpolate by 1×, 2×, or 4× times the input sample rate. This interpolated data is then fed into a peak detector. The peak detector determines the location of all peaks in the signal and the delta by which they exceed the programmable threshold. This information is then fed into a linear system solver. These peaks can be corrected using a precomputed spectrally shaped pulse (also called correction pulse) which is stored in a pulse RAM. Multiple peaks can be simultaneously cancelled by time shifting and combining these spectrally shaped pulses.

The linear system solver calculates the correction coefficients that get combined with the input signal (at the output of the delay FIFO) to produce an output signal which has a significantly lower peak to average ratio. This corrected output signal is then passed to two more similar CFR engines to correct missed peaks, as well as peaks that need further correction after passing through the first engine.

The CFR block within the transceiver consists of three cascaded CFR engines followed by a hard clipper to clip the few peaks that are skipped by all three CFR engines. At the output of each engine, there is a multiplexer that can be programmed to bypass CFR or apply a correction (shown in Figure 254).

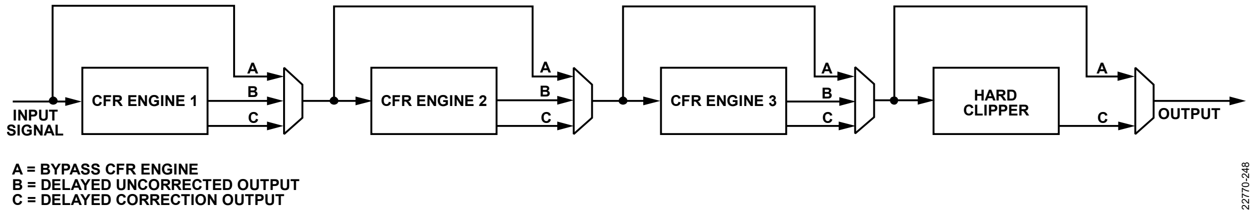


Figure 254. ADRV9029 Cascaded CFR Engine Blocks

The hard clipper unit at the very end of the CFR block clips any signal above a programmable threshold. This threshold generally should not be lower than the thresholds used in the CFR engines as it leads to an increase in noise (both in-band and out of band). Each of the individual units in Figure 254 can be bypassed depending on the use case and desired performance.

The transceiver incorporates the CFR block at the very beginning of the transmitter datapath directly after the JESD block. Note that the CFR engine can handle a maximum input sampling rate of 245.76 MHz with an internal interpolation up to 4×. Therefore, the maximum sampling rate for the base and correction pulses is 983.04 MHz.

OVERVIEW OF BLOCKS USED IN CFR

This section provides a brief summary of the major blocks used in each of the CFR engines.

Interpolator

The interpolator can be programmed to interpolate the data by 4, 2, or 1. The main purpose of the interpolator is to produce finer timing resolution so that the peak detector can find the location of the peaks more accurately. The interpolator takes in the transmitter data and outputs the interpolated data to the peak detector.

Peak Detector

The purpose of the peak detector block is to find groups of peaks that are above a certain threshold and then output the peak I and Q values along with peak locations relative to the first peak. The peak detector looks for peaks that are above a certain programmable threshold. Because peaks do not occur often, a coarse peak detection on every sample is performed using the following method:

1. Coarse peak detection, I or Q > Threshold/√2
   If this condition is false, then there is no need to compute the , $I^2 + Q^2$ complex calculation, which saves some power by avoiding frequent multiplications. In the case when the sample is larger, the magnitude of the incoming complex signal is checked to determine if the magnitude is above the threshold using the inequality shown in Step 2.
2. Fine peak detection, $I^2 + Q^2 > \text{Threshold}^2$
   If the condition in Step 1 is true, multiple consecutive samples that exceed the threshold can be found. These peaks can be corrected using a single pulse which is superimposed on to the maximum valued peak of the group of samples.

The peak detector sends this peak location along with the values of the peak samples into the linear system solver.

Linear System Solver

When the peak location and its corresponding values are received, the linear solver calculates the complex difference between the peak value and the threshold. The linear solver calculates the weights for various pulses required to cancel the peak.

These weights and peak location values are then used to scale and time shift the spectrally shaped pulses and generate the correction pulse. This correction pulse is then subtracted from the transmitter input data to generate the output signal (with a low PAR).

Pulse RAM

The pulse RAM holds the correction pulse for the carrier. It is possible to load two correction pulses corresponding to two desired carrier configurations. The user can pre-load these two correction pulses and be able to switch between two carrier configurations on-the-fly. This mode is known as the hot swap mode and is addressed by the adi_adrv9010_CfrCorrectionPulseWrite_v2 API command.

API SOFTWARE INTEGRATION

The API functions described in this section are required to set up the CFR block in the transceiver. This section first outlines the procedures required to use the CFR functionality and provides steps where each API function is called, and then discusses the various data structures used to set up the CFR engine as well as lists out the different API functions used to configure the CFR block.

Setting Up the CFR

The CFR can be set any time after part initialization. When the CFR parameters are set up and the correction pulses are loaded, use the initial calibration enumeration ADI_ADRV9025_CFR to run the CFR initial calibration. To configure the CFR engine, take the following steps:

1. Program the CFR control configuration via adi_adrv9025_CfrControlConfigSet.
2. Verify the CFR control configuration via adi_adrv9025_CfrCtrlConfigGet.
3. Program the CFR correction pulse using adi_adrv9025_CfrCorrectionPulseWrite_v2.
4. Enable the CFR engine via adi_adrv9025_CfrEnableSet.
5. Optionally configure the hard clipper via adi_adrv9025_CfrHardClipperConfigSet.
6. Execute the CFR initial calibration via adi_adrv9025_InitCalsRun.
7. Optionally, set the active correction pulse to use in Mode 1 via the adi_adrv9025_CfrActiveCorrectionPulseSet API for carrier config hot swapping.

Updating Correction Pulses On-the-Fly

The ADRV9029 provides the flexibility to change correction pulses on-the-fly without needing to run CFR initialization calibration. The recommended procedure assumes that the user has already successfully followed the initial procedure for setting up the CFR given in the Setting Up the CFR section. To update correction pulses on-the-fly, take the following steps:

1. Program the CFR correction pulse using adi_adrv9025_CfrCorrectionPulseWrite_v2.
2. Program the active correction pulse using adi_adrv9025_CfrActiveCorrectionPulseSet.

This sequence allows users to upload CFR pulses during run-time operation. However, to change the CFR control settings (except for the CFR thresholds), the sequence in the Setting Up the CFR section must be followed.

Modifying CFR Thresholds On-the-Fly

The ADRV9029 permits changing CFR thresholds without needing to run CFR initial calibration. To modify CFR thresholds on-the-fly, take the following steps:

1. Program the CFR thresholds using adi_adrv9010_CfrControlConfigSet.
2. Enable modified thresholds using adi_adrv9025_CfrThresholdsRunTimeUpdate.

API Functions and Data Structures

This section outlines the API functions and data structures required for setting up the CFR.

adi_adrv9025_CfrCtrlConfigSet(…)

The following function sets up CFR mode, the peak threshold, the interpolation factor, and the delay through the CFR blocks.

```
int32_t adi_adrv9025_CfrCtrlConfigSet (adi_adrv9025_Device_t *device,
adi_adrv9025_CfrCtrlConfig_t cfrCtrlConfig[], uint8_t cfrCtrlCfgArraySize)
```

This API must be called to setup the CFR control configuration before executing the CFR initial calibration. Currently, ADI_ADRV9010_CFR_MODE1 is the only mode of operation supported by the CFR engine. The user must provide the final correction pulse to be applied to the CFR input in this mode. To calculate the threshold (cfrPeakThreshold), use the following equation:

$$cfrPeakThreshold = 10(Target\ PAR\ dB/20) \times RMS\_INPUT$$

The user can setup an interpolation factor of 1×, 2×, or 4× to be applied to the transmitter data before peak detection. The user can also setup the CFR engine delay (cfrTxDelay) between n = 129 and n = 511, which translates to n+1 cycles per engine. The delay is applied to all enabled CFR engines. CFR latency is (cfrTxDelay +1) × numCfrEnginesEnabled + 3, where the additional 3 cycles comes from the hard clipper. The sample rate for the cycles here are at the transmitter JESD rate.

Each transmitter channel CFR block consists of three cascaded CFR engines followed by a hard clipper to clip the few peaks that are skipped by all three CFR engines. The CFR control configuration is applied to all three CFR engines by this function. The threshold is adjusted internally by the device firmware before applying it to each of the three CFR engines. The hard clipper can be optionally enabled via the adi_adrv9025_CfrHardClipperConfigSet API function.

This function can be called any time after device initialization and the ARM boot up is complete. The hard clipper setting is currently set at initialization and has to be setup before running the CFR initial calibration.

This function can be called after device initialization and the ARM processor boot up is complete but before the CFR initial calibration is executed. See Table 264 and Table 265 for the adi_adrv9025_ CfrCtrlConfigSet(…) Parameters and the adi_adrv9025_CfrCtrlConfig_t data structure, respectively.

Table 264. adi_adrv9025_CfrCtrlConfigSet(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure
cfrCtrlConfig	An array of CFR control configuration structures
cfrCtrlCfgArraySize	The number of configurations contained in the cfrCtrlConfig array

Table 265. adi_adrv9025_CfrCtrlConfig_t Data Structure

Data Type	Structure Member	Valid Values	Description
uint32_t	txChannelMask	0 to 15	A mask consisting of 'OR'ed transmitter channels for which the CFR core configuration is applied (1 bit for each channel).

Data Type	Structure Member	Valid Values	Description
adi_adrv9010_ CfrModeSel_e	cfrMode	1	Selects the mode in which the CFR is required to operate in. Currently, Mode 1 is the only supported mode.
uint16_t	cfrTxDelay	129…511	Sets the CFR delay per engine in units of samples at the CFR input rate (JESD 204B/C transmitter rate).
float	cfrPeakThreshold	0…1	Sets the target CFR peak detection and correction threshold. The threshold is calculated as: *cfrPreakThreshold* = 10(*Target PAR*/20) × *RMS_INPUT* The peak threshold is set to 0.79 by default.
float	cfrEngine1PeakThresholdScaler	0…1	Threshold scaler for engine CFR Engine 3
float	cfrEngine2PeakThresholdScaler	0…1	Threshold scaler for engine CFR Engine 2
float	cfrEngine3PeakThresholdScaler	0…1	Threshold scaler for engine CFR Engine 1
float	cfrCorrectionThresholdScaler	0…1	Threshold scaler for CFR correction
adi_adrv9010_ CfrInterpolationSel_e	cfrInterpolationFactor	1, 2, 4	Selects the interpolation factor to apply to the CFR input before peak detection. The CFR peak detectors can run at a higher (interpolated) rate to enable more optimal peak detection.
uint8_t	cfrEngine1MaxNumOfPeaks	0…5	Sets the maximum number of peaks to remove in one group for the respective CFR engines (default value = 5). The user must set this value to 0 when engine is disabled. It is suggested to have a descending order of the maximum number of peaks where Engine 1 has the highest value.
uint8_t	cfrEngine2MaxNumOfPeaks	0…5	
uint8_t	cfrEngine3MaxNumOfPeaks	0…5	

adi_adrv9025_CfrCtrlConfigGet(…)

The following function retrieves the core control configuration parameters for the CFR engine.

```
int32_t adi_adrv9025_CfrCtrlConfigGet(adi_adrv9025_Device_t * device, adi_adrv9025_TxChannels_e
txChannel, adi_adrv9025_CfrCtrlConfig_t *  cfrCtrlConfig)
```

This function reads the CFR mode, peak threshold, interpolation factor, and the delay currently programmed into the device.

This function can be called after the device initialization and the ARM processor boot up are complete.

Table 266. adi_adrv9025_CfrCtrlConfigGet(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure
txChannel	A target transmitter channel whose CFR control configuration is required to be read back
*cfrCtrlConfig	A pointer to the CFR control configuration structure which is updated with the CFR control settings from the device

adi_adrv9025_CfrCorrectionPulseWrite_v2(…)

The following function can be used to program the complex coefficients of the final CFR correction pulse.

```
int32_t adi_adrv9025_CfrCorrectionPulseWrite_v2(adi_adrv9025_Device_t * device, uint32_t
txChannelMask, adi_adrv9025_CfrCorrectionPulse_t cfrCorrectionPulses[], uint8_t
numCorrectionPulses);
```

This function is intended to be used when the CFR engine is operating in ADI_ADRV9025_CFR_MODE1 mode.

This function expects the user to provide only the first half the correction pulse because it is assumed that the correction pulse is conjugate symmetric. This API supports the programming of the correction pulses in the following use cases:

- A final correction pulse of a maximum length of 1025 (maximum half pulse length of 512) in Mode 1 for a single carrier configuration. Note that run-time carrier hot swapping is not supported if a pulse length of 1025 (half pulse length of 512) is used. In this case, the full pulse must be switched during transmit off.
- Two final correction pulses of a maximum length of 513 (maximum half pulse length of 256) corresponding to two carrier configurations for on-the-fly carrier configuration switching for Mode 1 operation. Run-time carrier switching can be executed via the adi_adrv9025_ CfrActiveCorrectionPulseSet() API which activates the correction pulse corresponding to the requested carrier configuration.

This function cab be called after device initialization and the ARM processor boot up is complete.

Table 267. adi_adrv9025_CfrCorrectionPulseWrite_v2(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure.
txChannelMask	A transmitter channel selection mask to write cfrCorrectionPulses with multiple channel selection allowed.
cfrCorrectionPulses	An array that consists of the final correction pulse(s) in Mode 1 operation. Note that in the case of programming two correction pulses for carrier hot swapping, cfrCorrectionPulses[0] corresponds to ADI_ADRV9025_CFR_CARRIER_HOT_SWAP_CORR_PULSE_1 and cfrCorrectionPulses[1] corresponds to ADI_ADRV9025_CFR_CARRIER_HOT_SWAP_CORR_PULSE_2.
numCorrectionPulses	The number of correction pulse coefficients to be programmed in the CFR engine. Note that a maximum of two correction pulses can be programmed if the half pulse length of the two correction pulses are less than or equal to 256, and a maximum of 1 correction pulse can be programmed if the half pulse length of the correction pulse is greater than 256.

Table 268. adi_adrv9025_CfrCorrectionPulse_t Data Structure

Data Type	Structure Member	Valid Values	Description
int16_t	coeffRealHalfPulse		An array consisting of the first half of the real part of the complex CFR correction pulse coefficients
int16_t	coeffImaginaryHalfPulse		An array consisting of the first half of the imaginary part of the complex CFR correction pulse coefficients
uint16_t	numCoeffs	512 (maximum)	Number of coefficients contained in the coeffReal and coeffImaginary arrays

adi_adrv9025_CfrCorrectionPulseRead_v2(…)

The following function can be used to read back the current complex coefficients of the final CFR correction pulse programmed in the device.

```
int32_t adi_adrv9025_CfrCorrectionPulseRead_v2(adi_adrv9025_Device_t * device,
adi_adrv9025_TxChannels_e txChannel, uint8_t maxCorrectionPulsesToRead,
adi_adrv9025_CfrCorrectionPulse_t cfrCorrectionPulses[], uint8_t * numCorrectionPulsesRead);
```

This function reads the final correction pulse to be used by the CFR engine to perform CFR correction when the CFR engine is configured to operate in (ADI_ADRV9025_CFR_MODE1). A maximum of two correction pulses of a half pulse length of 256 or a maximum of one correction pulse of a half pulse length of 512 can be read back from the device.

Note that this function can be called only when the transmitter channel is off and the CFR engines are not active. This function can be called only when the target transmitter channel is off after the device initialization and the ARM processor boot up are complete.

Table 269. adi_adrv9025_CfrCorrectionPulseRead_v2(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure.
txChannel	A target transmitter channel whose correction pulse coefficients are requested.
maxCorrectionPulsesToRead	The maximum number of correction pulses to read, which is also the size of the cfrCorrectionPulses parameter. The maximum number of correction pulses supported for readback is currently two.
cfrCorrectionPulses	An array of CFR correction pulses that is updated with the CFR correction pulses retrieved from the device.
numCorrectionPulsesReturned	The number of correction pulses actually read back from the CFR engine. The user can pass a NULL statement to this parameter if this information is not required.

adi_adrv9010_CfrEnableSet(…)

The following function enables or disables the CFR engines present before the DPD engine in the transmitter datapath.

```
int32_t adi_adrv9025_CfrEnableSet(adi_adrv9025_Device_t *device, adi_adrv9025_CfrEnable_t
cfrEnable[], uint8_t cfrEnableArraySize);
```

The CFR control configuration settings can be applied via adi_adrv9025_CfrCtrlConfigSet.

To apply CFR correction to transmitter data, the user can set adi_adrv9025_CfrEnable_t.cfrEngineXEnable to 1 and adi_adrv9025_CfrEnable_t.cfrEngineXBypassEnable to 0. To bypass the CFR engine, the user can set adi_adrv9025_CfrEnable_t.cfrEngineXEnable to 1 and adi_adrv9025_CfrEnable_t.cfrEngineXBypassEnable to 1. To disable the CFR engine completely, the user can set adi_adrv9025_CfrEnable_t.cfrEngineXEnable to 0.

Note that this is a function run at initialization and the enabling/disabling of CFR engine cannot be performed during runtime.

This function can be called after the device initialization and the ARM processor boot up are complete but before the CFR initial calibration is executed.

Table 270. adi_adrv9025_CfrEnableSet(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure
cfrEnable	An array of CFR enable control configuration structures
cfrEnableArraySize	The number of configurations contained in the cfrEnable array

adi_adrv9025_CfrEnableGet(…)

The following function retrieves the current state of CFR engine enables.

```
int32_t adi_adrv9025_CfrEnableGet(adi_adrv9025_Device_t *device, adi_adrv9025_TxChannels_e
txChannel, adi_adrv9025_CfrEnable_t *cfrEnable);
```

This function can be called after the device initialization and the ARM processor boot up are complete.

Table 271. adi_adrv9025_CfrEnableGet(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure
txChannel	A target transmitter channel whose enable status is requested
*cfrEnable	A pointer to a CFR enable structure that updates with the enable settings in the device

adi_adrv9025_CfrHardClipperConfigSet(…)

The following function enables/disables the CFR hard clipper and configures the hard clipper threshold.

```
int32_t adi_adrv9025_CfrHardClipperConfigSet(adi_adrv9025_Device_t *device,
adi_adrv9025_CfrHardClipperConfig_t cfrHardClipperConfig[], uint8_t cfrHardClipperCfgArraySize);
```

The CFR hard clipper threshold is applied to an approximation of $\sqrt{I^2 + Q^2}$.The threshold is normalized to 1 and is relative to 0 dBFS which means that a threshold of 1 corresponds to a threshold of 0 dBFS. It is not recommended to set the hard clipper threshold to a value less than −7 dBFS to ensure optimum performance.

This function can be called any time after the device initialization and the ARM boot up are complete. The hard clipper setting is currently an initialization setting and must be setup before running the CFR initial calibration. There is no support for dynamically changing the hard clipper threshold during runtime.

Table 272. adi_adrv9025_ CfrHardClipperConfigSet(…) Parameters

Parameter	Description
*device	Pointer to the device settings structure
cfrHardClipperConfig	An array consisting of the CFR hard clipper configurations
cfrHardClipperCfgArraySize	Number of configurations in cfrHardClipperConfig array

adi_adrv9025_CfrHardClipperConfigGet(…)

The following function retrieves the CFR hard clipper setting for the requested transmitter channel.

```
int32_t adi_adrv9025_CfrHardClipperConfigGet (adi_adrv9025_Device_t *device,
adi_adrv9025_TxChannels_e txChannel, adi_adrv9025_CfrHardClipperConfig_t *cfrHardClipperConfig);
```

This function can be called any time after the device initialization and the ARM boot up are complete.

Table 273. adi_adrv9025_ CfrHardClipperConfigGet(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure
txChannel	A target transmitter channel for which the CFR hard clipper status is requested
*cfrHardClipperConfig	A pointer to the CFR hard clipper configuration that updates with the device hard clipper configuration settings

adi_adrv9025_CfrHardClipperConfig_t

The adi_adrv9025_CfrHardClipperConfig_t data structure contains the information to set up the hard clipper unit in the device.

Table 274. CFR Hard Clipper Settings Structure Member Description

Data Type	Structure Member	Valid Values	Description
uint32_t	txChannelMask	0..15	This mask consists of OR'ed transmitter channels for which the hard clipper configuration is applied (1 bit for each channel).
uint8_t	cfrHardClipperEnable	0..1	1: enable the hard clipper on the channels requested 0: disable hard clipper
float	cfrHardClipperThreshold	0..1	This normalized threshold for the hard clipper ranges from 0 to 1. The threshold is relative to 0 dBFS, which means that a threshold value of 1 corresponds to 0 dBFS. The threshold is applied to an approximation of $\sqrt{I^2 + Q^2}$

adi_adrv9025_CfrStatusGet(…)

The following function reads the CFR status for the requested transmitter channel.

```
int32_t adi_adrv9025_CfrStatusGet(adi_adrv9010_Device_t * device, adi_adrv9010_TxChannels_e
txChannel, adi_adrv9025_CfrStatus_t * cfrStatus);
```

This function retrieves the CFR error code for the last CFR error occurred, along with the statistics information that includes the number of peaks skipped, number of peaks detected, and the number of peaks clipped by each CFR engine. The user can also monitor the CFR status to retrieve errors encountered during CFR initial calibration execution. The CFR statistics can be retrieved using this function.

This function can be called any time after the device initialization.

Table 275. adi_adrv9025_CfrStatusGet(…) Parameters

Parameter	Description
*device	Pointer to the device settings structure
txChannel	Enum to select the target transmitter channel whose CFR status is requested
*cfrStatus	Pointer to the status structure that will be updated with the status retrieved from the device

adi_adrv9025_CfrStatus_t

The adi_adrv9025_CfrStatus_t data structure holds the transmitter CFR engine status for each channel.

Table 276. CFR Status Structure Member Description

Structure Member	Description
adi_adrv9025_CfrError_e	Enumerated list of CFR Errors.
adi_adrv9025_CfrStatistics_t	Data structure to hold transmitter CFR engine statistics.

adi_adrv9025_CfrError_e

The adi_adrv9025_CfrError_e data structure holds the enumerated list of CFR errors.

Table 277. CFR Error Structure Member Description

Structure Member	Description
ADI_ADRV9025_CFR_CONFIGURATION_ERROR	Error code to convey that the mandatory CFR configurations were not done (not active/used).
ADI_ADRV9025_CFR_PROG_PULSE_MODE_ERROR	Error code to convey that an unsupported pulse mode was selected.
ADI_ADRV9025_CFR_INPUT_RATE_HIGH_ERROR	Error code to convey that the transmitter channel sample rate is higher than 245.76 MHz.
ADI_ADRV9025_CFR_CTRL_CMD_NOT_SUPPORTED_ERROR	Error code to convey that the control command is not supported

adi_adrv9025_CfrStatistics_t

The adi_adrv9025_CfrStatistics_t data structure holds the transmitter CFR engine statistics for each transmitter channel.

Table 278. CFR Statistics Structure Member Description

Data Type	Structure Member	Description
uint64_t	cfrEngine1PeaksDetected	Number of peaks detected by CFR Engine 1 since the last reset
uint64_t	cfrEngine1PeaksSkipped	Number of peaks skipped by CFR Engine 1 since the last reset
uint64_t	cfrEngine2PeaksDetected	Number of peaks detected by CFR Engine 2 since the last reset
uint64_t	cfrEngine2PeaksSkipped	Number of peaks skipped by CFR Engine 2 since the last reset
uint64_t	cfrEngine3PeaksDetected	Number of peaks detected by CFR Engine 3 since the last reset
uint64_t	cfrEngine3PeaksSkipped	Number of peaks skipped by CFR Engine 3 since the last reset
uint64_t	cfrNumSamplesClipped	Number of samples clipped by the CFR engine since the last reset

adi_adrv9025_CfrActiveCorrectionPulseSet(…)

The following function switches the final correction pulse to apply in the CFR engine in Mode 1 (ADI_ADRV9025_CFR_MODE1) operation.

```
int32_t adi_adrv9025_CfrActiveCorrectionPulseSet(adi_adrv9025_Device_t *device, uint32_t
txChannelMask, adi_adrv9025_CfrCarrierHotSwapCorrPulseSel_e cfrCorrectionPulseSel);
```

This function can be used to activate one of the two final correction pulses corresponding to two carrier configurations when the active carrier configuration is changed during runtime (carrier configuration hot swapping).

This function can only be executed if two correction pulses of a length of 512 (half pulse length of 256) or lesser are programmed into the device prior to calling this function. If a single correction pulse of a length of 512 or lesser (half pulse length of 256 or lesser) is programmed in the device, this function has no effect. The correction pulses can be programmed via the adi_adrv9025_CfrCorrectionPulseWrite_v2() API.

Note that this function does not change the CFR configuration including the peak threshold or interpolation factor, and the CFR engine enables when the active correction pulse is switched. By default, the device activates ADI_ADRV9025_CFR_CARRIER_HOT_SWAP_CORR_PULSE_1 on reset. The CFR engines must be enabled for the active correction pulse switching to take place. Calling this function when the CFR engines are disabled has no effect.

This runtime function can be called any time after device initialization and two correction pulses of a length of 512 or less (half pulse length of 256 or less) are programmed via the adi_adrv9025_CfrCorrectionPulseWrite_v2() API and the CFR initial calibration has been executed. The CFR initial calibration can be executed via the adi_adrv9025_InitCalsRun() API. The mode of operation must be set to ADI_ADRV9025_CFR_MODE1 via the adi_adrv9025_CfrCtrlConfigSet() API. At the time of calling this function, the CFR engine must be enabled. Otherwise, the active correction pulse switching does not occur.

Table 279. adi_adrv9025_CfrActiveCorrectionPulseSet(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure
txChannel	A mask consisting of OR'ed transmitter channels for which the requested correction pulse is required to be activated
cfrCorrectionPulseSel	Selection for the correction pulse to activate

adi_adrv9025_CfrCarrierHotSwapCorrPulseSel_e

The adi_adrv9025_CfrStatistics_t data structure holds the transmitter CFR engine statistics for each transmitter channel.

Table 280. CfrCarrierHotSwapCorrPulseSel_e Member Description

Structure Member	Description
ADI_ADRV9025_CFR_CARRIER_HOT_SWAP_CORR_PULSE_1	Sets the active CFR correction pulse to Pulse 1 when two correction pulses of half pulse lengths of 256 or less are programmed
ADI_ADRV9025_CFR_CARRIER_HOT_SWAP_CORR_PULSE_2	Sets the active CFR correction pulse to Pulse 2 when two correction pulses of half pulse lengths of 256 or less are programmed

adi_adrv9025_CfrActiveCorrectionPulseGet(…)

The following function returns the correction pulse currently activated in the CFR engine in Mode 1 (ADI_ADRV9025_CFR_MODE1) operation.

```
int32_t adi_adrv9025_CfrActiveCorrectionPulseGet(adi_adrv9025_Device_t *device,
adi_adrv9025_TxChannels_e txChannel, adi_adrv9025_CfrCarrierHotSwapCorrPulseSel_e
*cfrCorrectionPulseSel);
```

This function can be used to retrieve the status of the correction pulse currently activated in the device. By default, the device activates ADI_ADRV9025_CFR_CARRIER_HOT_SWAP_CORR_PULSE_1 on reset.

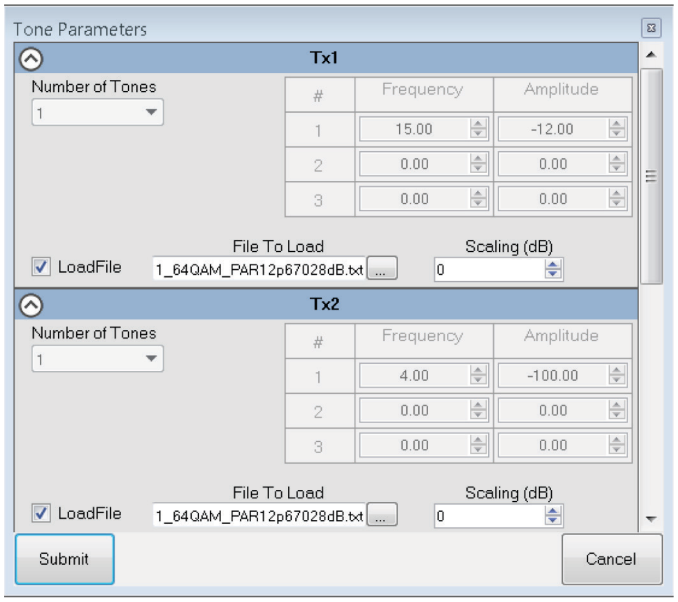
This runtime function can be called any time after device initialization and two correction pulses of a length of 512 or less (half pulse length of 256 or less) are programmed via the adi_adrv9025_CfrCorrectionPulseWrite_v2() API and the CFR initial calibration has been executed. The CFR initial calibration can be executed via the adi_adrv9025_InitCalsRun() API.

Table 281. adi_adrv9025_CfrActiveCorrectionPulseGet(…) Parameters

Parameter	Description
*device	A pointer to the device settings structure
txChannel	A target transmitter channel for which the active correction pulse status is requested
*cfrCorrectionPulseSel	A pointer to memory that updates with the currently active CFR correction pulse

TYPICAL PROCEDURE TO SET UP CFR USING THE GUI

This section describes how to set up CFR on the device using Transceiver Evaluation Software (TES). The user can load the desired waveform using the Tones pop-up window in the Transmit tab on TES as shown in Figure 255.



Tx1

Number of Tones: 1

#	Frequency	Amplitude
1	15.00	-12.00
2	0.00	0.00
3	0.00	0.00

LoadFile ✓ File To Load: 1_64QAM_PAR12p67028dB.txt Scaling (dB): 0

Tx2

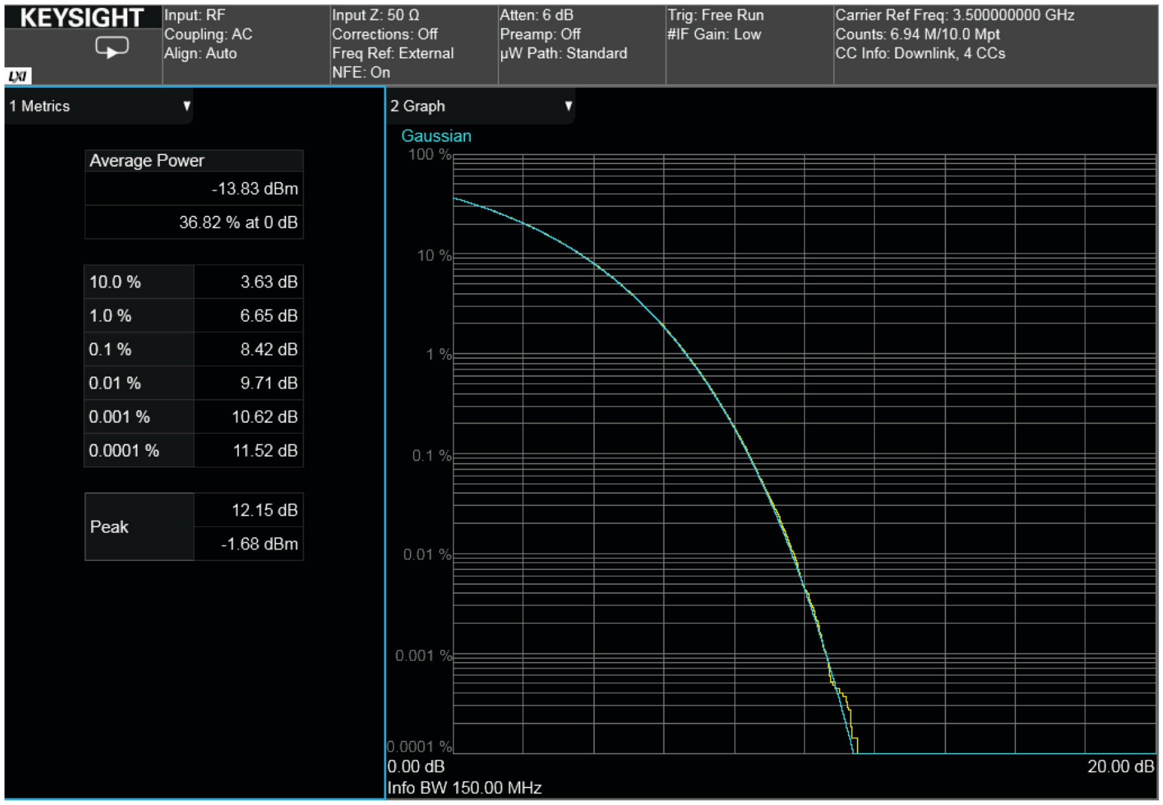
Number of Tones: 1

#	Frequency	Amplitude
1	4.00	-100.00
2	0.00	0.00
3	0.00	0.00

LoadFile ✓ File To Load: 1_64QAM_PAR12p67028dB.txt Scaling (dB): 0

Figure 255. Load Waveform Using Transceiver Evaluation Software

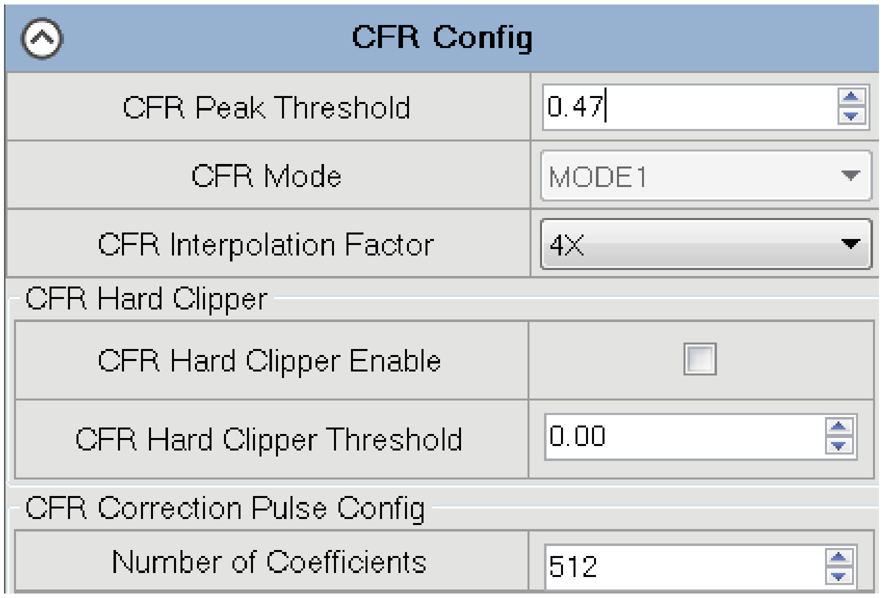
When the waveform loads, it is transmitted using the play button on the transmit tab. As an example, an LTE 20 MHz waveform with PAR of 12.2 dB is uploaded as shown in Figure 255. The uncorrected waveform complementary cumulative distribution function (CCDF) is shown in Figure 256.



Average Power	
	-13.83 dBm
	36.82 % at 0 dB
10.0 %	3.63 dB
1.0 %	6.65 dB
0.1 %	8.42 dB
0.01 %	9.71 dB
0.001 %	10.62 dB
0.0001 %	11.52 dB
Peak	12.15 dB
	-1.68 dBm

Figure 256. CCDF of Example 20 MHz LTE Signal with PAR of 12.2 dB (Uncorrected Waveform)

The DFE tab is used in the TES to set up the CFR engines as shown in Figure 257.



CFR Peak Threshold	0.47
CFR Mode	MODE1
CFR Interpolation Factor	4X
CFR Hard Clipper Enable	☐
CFR Hard Clipper Threshold	0.00
Number of Coefficients	512

Figure 257. CFR Engine Setup Using the TES

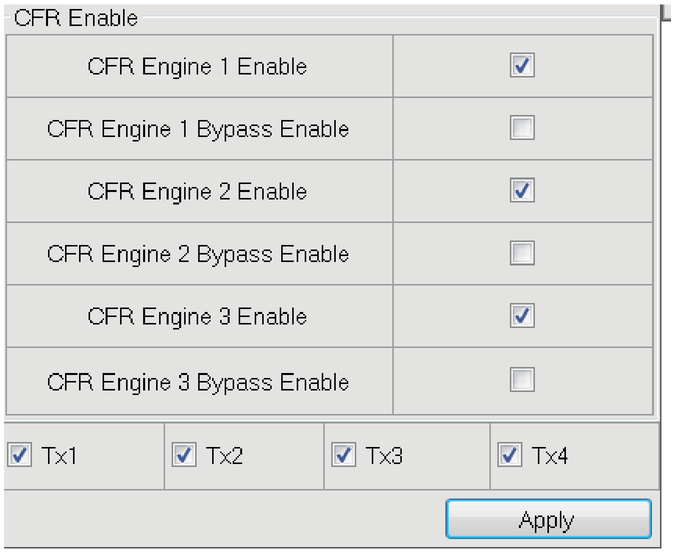
Load File can be used to load the correction pulse (see Figure 258). This correction pulse is specific to the waveform being used (LTE 20 MHz in the example shown) and is sampled at the peak detection rate. The CFR peak threshold is set to 0.47, which corresponds to a target PAR calculated using the following equation:

$$Target\ PAR = 0 \times \log_{10} (CFR\ Peak\ Threshold/RMS_{InputSignal})$$

Using the previous equation, the target PAR is around 8.75 dB. However, if the user wants to derive the CFR peak threshold value needed to achieve a desired target PAR, use the following equation:

$$CFR\ Peak\ Threshold = RMS_{InputSignal} \times 10\ (Target\ PAR/20)$$

The user must then enable the required number of CFR engines, as shown in Figure 258.



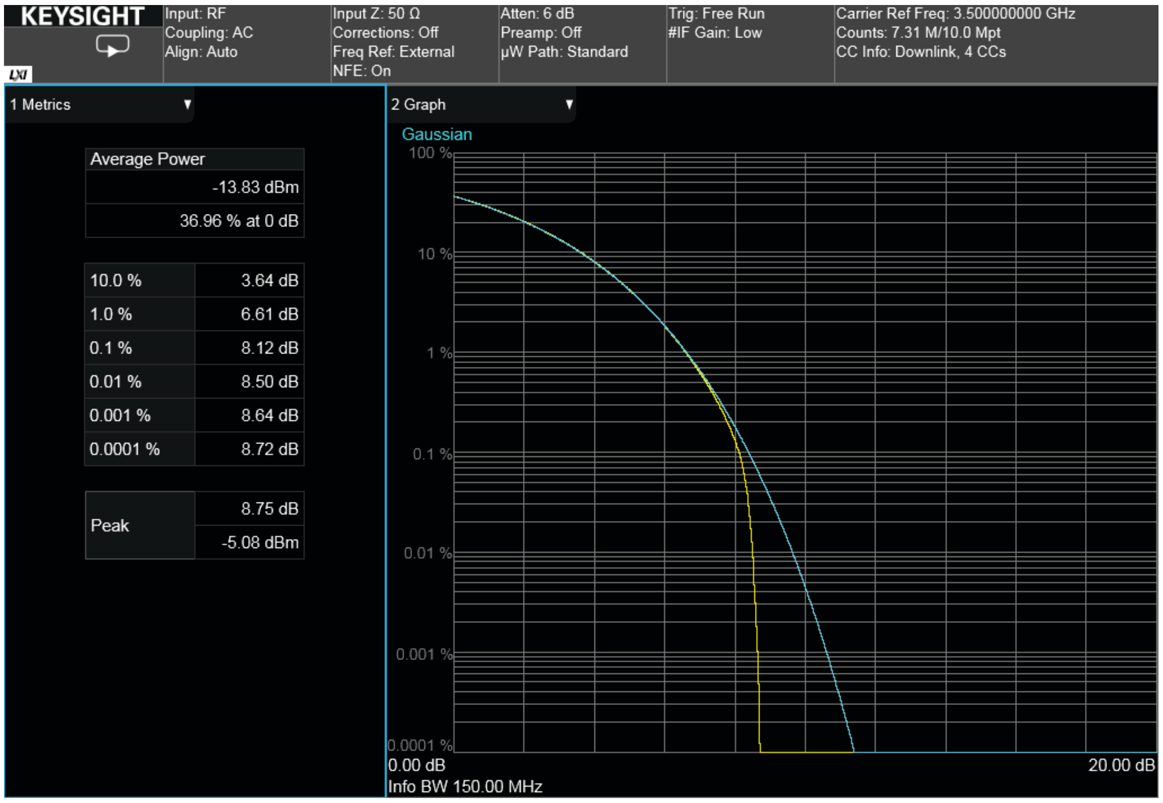
CFR Enable

CFR Engine 1 Enable	✓
CFR Engine 1 Bypass Enable	☐
CFR Engine 2 Enable	✓
CFR Engine 2 Bypass Enable	☐
CFR Engine 3 Enable	✓
CFR Engine 3 Bypass Enable	☐

✓ Tx1 ✓ Tx2 ✓ Tx3 ✓ Tx4

Figure 258. Enabling CFR Using TES

When **Apply** is clicked (which runs the CFR initial calibration), the CFR engines enable, and the corrected waveform can be observed on the spectrum analyzer, as shown in Figure 259.



KEYSIGHT

Input: RF
Coupling: AC
Align: Auto

Input Z: 50 Ω
Corrections: Off
Freq Ref: External
NFE: On

Atten: 6 dB
Preamp: Off
µW Path: Standard

Trig: Free Run
#IF Gain: Low

Carrier Ref Freq: 3.500000000 GHz
Counts: 7.31 M/10.0 Mpt
CC Info: Downlink, 4 CCs

1 Metrics

Average Power	
	-13.83 dBm
	36.96 % at 0 dB

10.0 %	3.64 dB
1.0 %	6.61 dB
0.1 %	8.12 dB
0.01 %	8.50 dB
0.001 %	8.64 dB
0.0001 %	8.72 dB

Peak	8.75 dB
	-5.08 dBm

Figure 259. CCDF of Corrected Waveform Using CFR

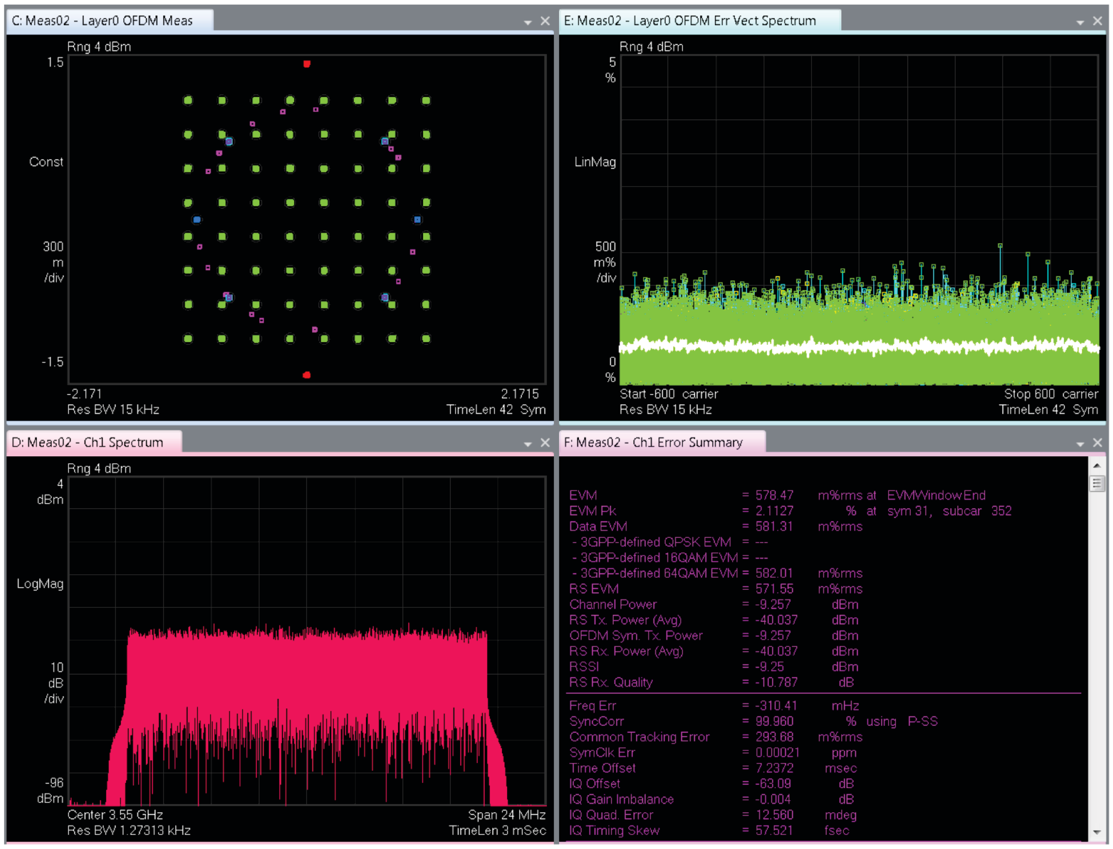
As shown in Figure 259, the corrected CCDF curve has a PAR of 8.75 dB, which corresponds to the CFR peak threshold that was previously set. No spectral regrowth is shown and the desired PAR can be achieved.

Impact on EVM

This section describes the impact of using the CFR engines within the transceiver on EVM performance. The same LTE 20 MHz tone (PAR = 12.67 dB) used in the previous section is also used in this example. The setup information for this waveform is as follows:

- Carrier 1 center frequency = 0 MHz
- Output sample rate = 245.76 MHz
- DAC resolution = 16
- Output data format = 2s Complement
- Scaling = 0 dB
- Modulation = 64QAM
- Test model = 3.1
- Final PAR (dB) = 12.2

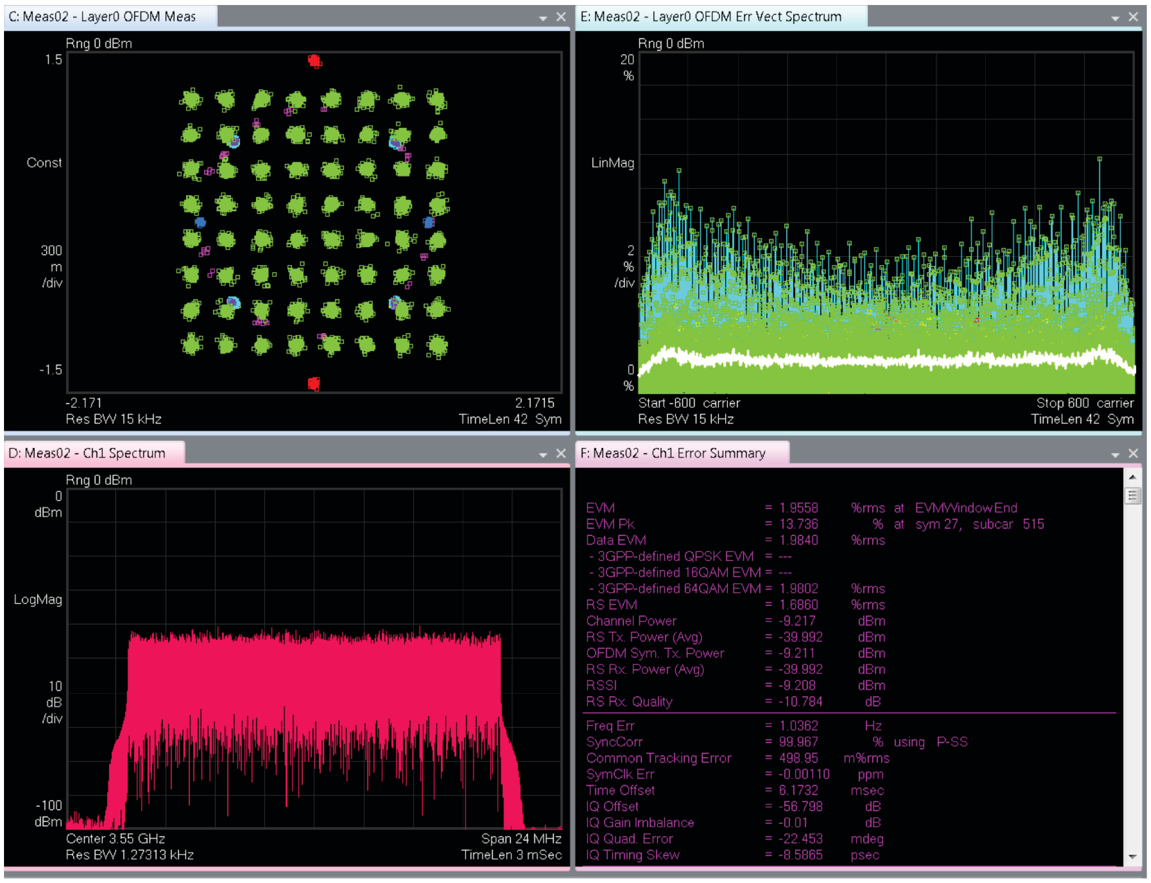
The EVM observed before applying the CFR is shown in Figure 260.



```
EVM                          = 578.47    m%rms at  EVMWindowEnd
EVM Pk                       = 2.1127       %  at  sym 31,  subcar  352
Data EVM                     = 581.31    m%rms
 - 3GPP-defined QPSK EVM     = ---
 - 3GPP-defined 16QAM EVM    = ---
 - 3GPP-defined 64QAM EVM    = 582.01    m%rms
RS EVM                       = 571.55    m%rms
Channel Power                = -9.257      dBm
RS Tx. Power (Avg)           = -40.037     dBm
OFDM Sym. Tx. Power          = -9.257      dBm
RS Rx. Power (Avg)           = -40.037     dBm
RSSI                         = -9.25       dBm
RS Rx. Quality               = -10.787      dB
Freq Err                     = -310.41     mHz
SyncCorr                     = 99.960       %  using  P-SS
Common Tracking Error        = 293.68    m%rms
SymClk Err                   = 0.00021     ppm
Time Offset                  = 7.2372     msec
IQ Offset                    = -63.09       dB
IQ Gain Imbalance            = -0.004       dB
IQ Quad. Error               = 12.560     mdeg
IQ Timing Skew               = 57.521     fsec
```

Figure 260. Observed EVM Before Applying CFR

Applying the CFR settings discussed in the Typical Procedure to Set Up CFR Using the GUI section where the target PAR is set to 8 dB, the degradation observed in EVM is shown in Figure 261.



```
EVM                          = 1.9558     %rms  at  EVMWindowEnd
EVM Pk                       = 13.736        %  at  sym 27,  subcar  515
Data EVM                     = 1.9840     %rms
 - 3GPP-defined QPSK EVM     = ---
 - 3GPP-defined 16QAM EVM    = ---
 - 3GPP-defined 64QAM EVM    = 1.9802     %rms
RS EVM                       = 1.6860     %rms
Channel Power                = -9.217      dBm
RS Tx. Power (Avg)           = -39.992     dBm
OFDM Sym. Tx. Power          = -9.211      dBm
RS Rx. Power (Avg)           = -39.992     dBm
RSSI                         = -9.208      dBm
RS Rx. Quality               = -10.784      dB
Freq Err                     = 1.0362       Hz
SyncCorr                     = 99.967        %  using  P-SS
Common Tracking Error        = 498.95    m%rms
SymClk Err                   = -0.00110    ppm
Time Offset                  = 6.1732     msec
IQ Offset                    = -56.798      dB
IQ Gain Imbalance            = -0.01        dB
IQ Quad. Error               = -22.453    mdeg
IQ Timing Skew               = -8.5865    psec
```

Figure 261. Observed EVM After Applying CFR (Target PAR = 8 dB)

We can see above that the rms EVM degraded from 0.6% to 2% due to application of CFR. From a quick sweep of the rms EVM at different target PAR, we see the trend shown in Figure 262. Note that different systems have different requirements for maximum tolerable EVM degradation due to CFR, which would drive the minimum achievable PAR for a given waveform configuration. Note also that the performance shown here is highly dependent on the "goodness" of the CFR correction pulse. With a different pulse, we should expect a different result.

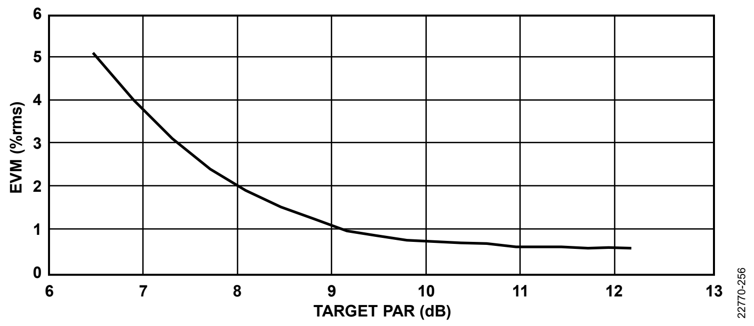


Figure 262. Target PAR vs. EVM

CLOSED LOOP GAIN CONTROL (CLGC)

CLGC OVERVIEW

CLGC is a closed-loop tracking calibration that adjusts front-end transmit attenuations to maintain a constant desired gain in the transmit path from baseband input to power amplifier output such that the power amplifier output power is constant with respect to changes in temperature or other variations. To set the desired transmit gain, adjust the transmit front-end attenuation such that the power amplifier output reaches the desired power with the maximum nominal level of the baseband signal. The digital swing of the baseband signal is not affected because the attenuation is applied on the analog front end of the transmit channel.

The power amplifier output power can fluctuate around the target level over the time because of temperature changes and other factors given a constant baseband signal level. The CLGC helps reduce the fluctuation and meet the standard and regulatory requirements (for example, ±2 dB fluctuation is required by the LTE/5GNR standard). The baseband signal level can change significantly because of the dynamics of power control and traffic load in a network (for example, dynamic range can be up to 20 dB in LTE). The CLGC helps keep the output power linearly proportional to the baseband signal level.

ELEMENTS OF CLGC

Figure 263 shows a simplified block diagram of the CLGC system. The CLGC algorithm observes the transmit data (x(n)), the post power amplifier observation data (y(n)), and adjusts the transmit front-end attenuation such that the overall loop gain (observation receive data/transmit data) remains constant.

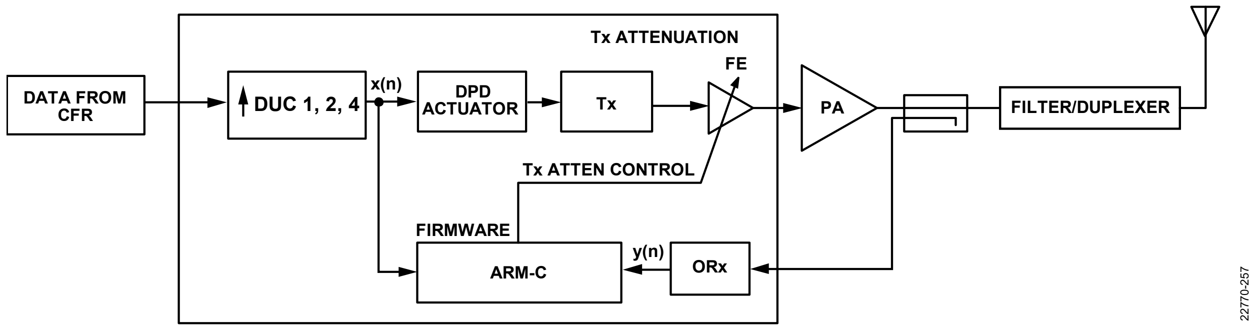


Figure 263. CLGC Simplified Block Diagram

The elements of the CLGC system include the following:

- Transmit datapath. The digital baseband signal from the ADRV9025 deframer output goes through a CFR block for reduction of the overall peak to average ratio of the signal followed by a digital upconverter that interpolates the baseband signal by a factor of 1×, 2×, or 4× for analyzing the baseband signal over the DPD analysis bandwidth. The upconverted data before the DPD actuator is then used as the reference transmit data for the closed loop gain control algorithm.
- Observation datapath. The CLGC algorithm relies on observing any fluctuations in the power amplifier output power through a feedback path. The feedback path is realized through the observation receiver. The power amplifier output data is coupled into the observation receiver, downconverted, and digitized for loop gain estimation and correction by the firmware.
- Capture engine. The transmit and observation receive samples for CLGC measurement are captured and aligned in the capture engine, preprocessing is performed, and the preprocessed samples are passed onto the embedded ARM processor for CLGC measurement and loop gain control.
- CLGC processing. The CLGC algorithm itself is implemented in the firmware running on an embedded ARM-C in the transceiver. The CLGC is a tracking calibration that tracks the observed and baseband data and uses a loop gain estimator to track changes in the overall loop gain. Refer to the CLGC Algorithm Overview section for an overview of the CLGC algorithm.

CLGC ALGORITHM OVERVIEW

The CLGC algorithm is designed to maintain a constant loop gain and overcome any minor fluctuations in the power amplifier output power because of variations in temperature and other operating conditions. Loop gain is defined as the ratio of the power level of observed data to the power level of the baseband transmit data as follows:

$$Loop\ Gain = Observation\ Receive\ RMS\ Power/Transmit\ RMS\ Power$$

The CLGC algorithm relies on the post power amplifier feedback data to estimate the loop gain and adjust the front-end transmit attenuation on the transceiver. The observation points of the CLGC algorithm are shown in Figure 264.

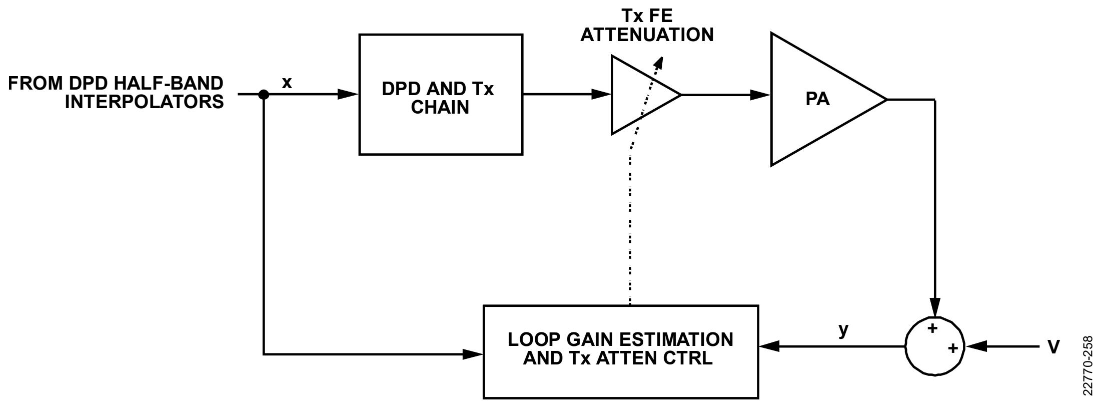

Tx FE ATTENUATION
FROM DPD HALF-BAND INTERPOLATORS
x
DPD AND Tx CHAIN
PA
LOOP GAIN ESTIMATION AND Tx ATTEN CTRL
y
+
+
V
22770-258

Figure 264. CLGC Algorithm Observation Points

The signal path from the reference baseband transmit input to the observed data for loop gain estimation can be divided into four sections, as listed in Table 282. The total loop gain observed at the observation receiver includes the front-end attenuation out of the transceiver, the gain of the power amplifier, the coupling attenuation for feedback, and the observation receiver front-end attenuation.

Table 282. Observed Data for Loop Gain Estimation

Section	Gain Equation	Comments
Transmit Section	Xtx(n) = gTX + (x(n) + vtx_DAC_Quant(n))	gTX = total transmit attenuation
		x(n) = transmit baseband data
		vtx_DAC_Quant = transmit DAC quantization noise
Power Amplifier Section	XPA(n) = gPA. Xtx(n) + vPA(n)	gPA = power amplifier gain at the power amplifier operating point
		vPA = additive noise determined by ACLR
Coupling Section	Yorx(n) = gCPL.XPA(n) + vorx(n)	gCPL = coupling attenuation in the observation receive path
		vorx = in-band thermal noise and additive noise determined by noise figure in observation receive path
Observation Receive Section	y(n) = gorx. Yorx(n) + vorx_ADC_Quant(n)	gorx = front-end observation receive attenuation
		vorx_ADC_Quant = observation receive ADC quantization noise.

Total gain seen at y(n),

$$g = gorx \times gCPL \times gPA \times gTX$$

Observation

To relate the observation receive samples to the transmit samples in the loop gain estimation engine, use the following equation:

$$y(n) = gx(n) + v(n)$$

where:
x(n) is the input transmit samples from a user BBIC.
y(n) is the output samples from observation receive.
v(n) is the equivalent additive noise from the entire loop, which can include thermal noise, quantization noise, phase noise, and nonlinear distortions.
g is the desired loop gain set by the user based on a nominal level of x(n) and an expected output power of the power amplifier.

Loop Gain Estimation and Convergence

A simplified equation of the loop gain estimation is described below. The rms power is calculated on captured transmit samples (x(n)) and observation receive samples (y(n)), where N is the total number of samples captured per CLGC update. The value N is configured by the user. Refer to the CLGC Measurement section for details on CLGC sample capture.

$$\textit{Estimated Loop Gain} = \textit{Observation Receive RMS Power}/\underline{\textit{Transmit Power}} = \frac{\sum_{n=0}^{N} |y(n)|^2}{\sum_{n=0}^{N} |y(x)|^2}$$

With the loop gain estimated, a loop gain error can be defined as follows:

$$\textit{Loop Gain Error} = \textit{Estimated Loop Gain}/\textit{Expected Loop Gain}$$

The estimated loop gain is determined by the CLGC algorithm, and the expected loop gain is configured by the user. The objective of the CLGC is to reduce the loop gain error ratio to 1 dB or 0 dB. The CLGC converges to the expected loop gain by tuning the transceiver front-end transmit attenuation, as shown in Figure 264.

ENABLING THE CLGC TRACKING CALIBRATION

The CLGC tracking calibration can be enabled using the adi_adrv9025_TrackingCalsEnableSet() API. The user can pass the CLGC tracking calibration mask values defined in the **adi_adrv9025_cals_types.h** file as argument to the adi_adrv9025_TrackingCalsEnableSet() API. The mask values are described in Table 283.

Table 283 CLGC Tracking Calibration Mask

CLGC Tracking Calibration Mask	Value	Description
ADI_ADRV9025_TRACK_TX1_CLGC	0x0000000000100000	Enable CLGC tracking calibration on Tx1
ADI_ADRV9025_TRACK_TX2_CLGC	0x0000000000200000	Enable CLGC tracking calibration on Tx2
ADI_ADRV9025_TRACK_TX3_CLGC	0x0000000004000000	Enable CLGC tracking calibration on Tx3
ADI_ADRV9025_TRACK_TX4_CLGC	0x0000000008000000	Enable CLGC tracking calibration on Tx4

There are at least three prerequisites for enabling the CLGC tracking calibration including the following:

1. The correct transmit to observation receive mapping must be configured during initialization. The adi_adrv9025_TxToOrxMappingSet() API can be used for this purpose. The correct mapping of the transmit path to the observation path must be communicated to the firmware to capture data from the correct observation receiver.
2. The path delay initial calibration must be executed through the adi_adrv9025_InitCalsRun() API with the mask value equal to ADI_ADRV9025_EXTERNAL_PATH_DELAY. The external path delay from the transmit to observation receive path is required to align the transmit and observation receive samples for estimating loop gain.
3. The DPD model must be loaded and the DPD reset must be asserted. This step is required to initialize the DPD actuator.

Please refer to the Recommended Sequence for Enabling CLGC Tracking Calibration section in this document for the list of commands to be used for bringing up CLGC.

The user can pass a 64-bit unsigned integer with OR'd values of the CLGC mask to enable the CLGC tracking calibration on the requested transmit channel. For example, to enable the CLGC tracking calibration on all four transmit channels, the user can pass the parameters shown in Table 284 as arguments to the adi_adrv9025_TrackingCalsEnableSet() API.

Table 284. CLGC Tracking Calibration Enable Example Arguments to the adi_adrv9025_TrackingCalsEnableSet() API

Parameter	Data Type	Description	Value
enableMask	uint64_t	The 64-bit 'OR'ed mask that consists of tracking calibrations to enable/disable	(ADI_ADRV9025_TRACK_TX1_CLGC | ADI_ADRV9025_TRACK_TX2_CLGC | ADI_ADRV9025_TRACK_TX3_CLGC | ADI_ADRV9025_TRACK_TX4_CLGC)
enableDisableFlag	Enumeration	Indicates whether the mask value passed in the enableMask parameter is to be used for enabling or disabling the tracking calibration	ADI_ADRV9025_TRACKING_CAL_ENABLE

Similarly, to disable the CLGC tracking calibration, the user can set the argument enableDisableFlag to an enumeration value ADI_ADRV9025_TRACKING_CAL_DISABLE to disable the CLGC tracking calibration on the requested channels.

Note that when the CLGC tracking calibration is enabled, the CLGC does not actively control the loop gain. The user must explicitly configure the loop gain control enable via the adi_adrv9025_ClgcConfigSet() API to enable actively control the CLGC loop gain. Details regarding the CLGC modes of operation are described in the CLGC Measurement section.

The CLGC tracking calibration works in synchronization with the DPD algorithm, and the CLGC is scheduled once per second by the firmware.

CLGC MODES OF OPERATION

The CLGC functionality can operate in either of the following two modes:

- Passive loop gain measurement. This mode of operation is typically activated to determine the initial operating point of the power amplifier. When a transmitter is activated, the user determines the power amplifier operating point by sending traffic and measuring the overall loop gain from observation receive to transmit. During this stage, the user can take advantage of the passive loop gain measurement mode in which the CLGC algorithm measures the loop gain without actively adjusting the transmit front-end attenuation. When the ideal operating point is determined, the user can then enable active loop gain control mode.
- Active loop gain control. In this mode, the CLGC measures the loop gain from observation receive to the transmit baseband and adjusts the transmit front-end gain to maintain the loop gain. This mode of operation is typically activated during runtime when the initial operating points are determined, and the initial observation receive gain and transmit attenuation settings are configured. For the active loop gain control mode, the user must configure the expected loop gain using the adi_adrv9025_ClgcConfigSet() API through the clgcExpectedLoopGain_dB parameter in the adi_adrv9025_ClgcConfig_t data structure.

The user can select the CLGC mode of operation through the adi_adrv9025_ClgcConfigSet() API using the clgcEnableGainControl parameter in the adi_adrv9025_ClgcConfig_t data structure, as shown in Table 285.

Table 285. CLGC Mode of Operation Configuration

clgcEnableGainControl	CLGC Mode Activated
0	Passive loop gain measurement
1	Active loop gain control

Figure 265 captures a typical CLGC bring up sequence during which the passive and active loop gain control modes are active at various stages.

In passive loop gain measurement mode, the user can retrieve the observation receive rms power and transmit rms power as well as the loop gain estimated by the CLGC algorithm using the adi_adrv9025_ClgcStatusGet() API. The adi_adrv9025_ClgcStatus_t structure has the clgcLoopGain, clgcTxRmsPower, and clgcOrxRmsPower members that can be monitored to adjust the initial operating point of the power amplifier and determine the desired loop gain.

The passive loop gain measurement mode is mostly used in a factory calibration setting. The optimal operating point for the power amplifier is determined, the loop gain, transmit attenuation and observation receive gain values are noted, and the values are used in the field.

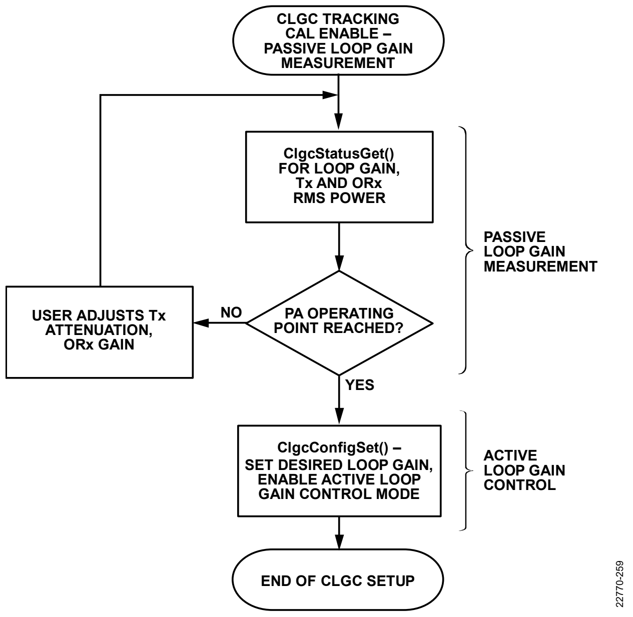

CLGC TRACKING CAL ENABLE – PASSIVE LOOP GAIN MEASUREMENT
ClgcStatusGet() FOR LOOP GAIN, Tx AND ORx RMS POWER
PA OPERATING POINT REACHED?
NO
USER ADJUSTS Tx ATTENUATION, ORx GAIN
YES
ClgcConfigSet() – SET DESIRED LOOP GAIN, ENABLE ACTIVE LOOP GAIN CONTROL MODE
END OF CLGC SETUP
PASSIVE LOOP GAIN MEASUREMENT
ACTIVE LOOP GAIN CONTROL
22770-259

Figure 265. CLGC Modes of Operation During Bring Up

CLGC MEASUREMENT

The CLGC measurement cycle is common to both passive loop gain measurement and active loop gain control modes. In active loop gain control mode, the transmit attenuation is also adjusted by the CLGC algorithm. The transmit attenuation adjustment is described in the following section.

The CLGC measurement relies on the sample batch size, the power level of the samples and the SNR of the observed data. The CLGC captures samples in batches specified by the clgcMeasurementBatchTime_us parameter defined in the adi_adrv9025_ClgcConfig_t data structure. The user must tune the measurement batch size based on the signal characteristics that they are most likely to encounter. Refer to the Case Study for Configuring CLGC Batch Sampling Period section, which goes through an example case study for determining the optimum batch measurement size. The list of parameters used to determine a successful capture is listed in Table 286.

Table 286. CLGC Measurement Configuration

Parameter	Description	User Configuration Access
Capture Batch Time	This parameter determines the duration of samples to be captured per batch specified in microseconds. A maximum of 65536 µs can be configured.	adi_adrv9025_ClgcConfig_t. clgcMeasurementBatchTime_us
Maximum Number of Batches	The CLGC measurement continues to capture samples in batches until the required SNR criteria is met or the number of batches exceeds 512. If the number of batches captured exceeds 512 and the requisite SNR criteria is not met, an error is flagged and the CLGC aborts the calibration.	Fixed value of 512, not user configurable
Transmit Qualifying Threshold	If the captured transmit samples are below this user configured threshold, the samples are discarded.	adi_adrv9025_ClgcConfig_t. clgcTxQualifyingThreshold_dBFS
Observation Receive Qualifying Threshold	If the captured observation receive samples are below this user configured threshold, the samples are discarded.	adi_adrv9025_ClgcConfig_t. clgcOrxQualifyingThreshold_dBFS

Transmit and Observation Receive Qualifying Threshold

As described in Table 286, the CLGC samples are required to meet the threshold criteria to be considered for loop gain estimation. The thresholds are configured by the user through the adi_adrv9025_ClgcConfigSet() API using the clgcTxQualifyingThreshold_dBFS and clgcOrxQualifyingThreshold_dBFS parameters that are part of the adi_adrv9025_ClgcConfig_t data structure. If either one of the transmit or observation receive samples do not meet the criteria, the entire batch of transmit and observation receive captured data is discarded. An example scenario in which the transmit/observation receive signals meet the threshold criteria in Batch 1 where the rms power in Batch 2 fails to meet the threshold criteria is shown in Figure 266. Therefore, samples in Batch 2 are discarded and not considered for measurement.

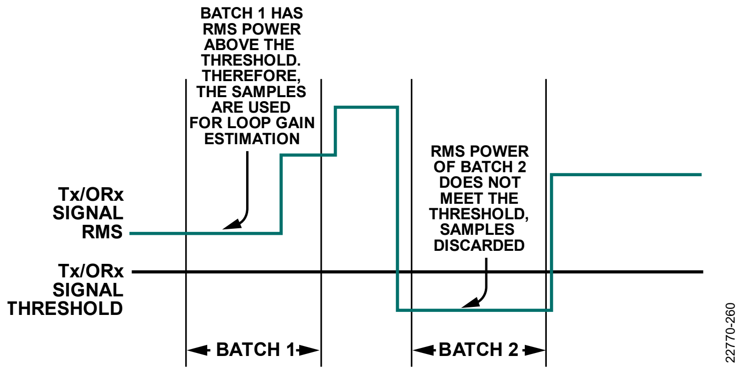

BATCH 1 HAS RMS POWER ABOVE THE THRESHOLD. THEREFORE, THE SAMPLES ARE USED FOR LOOP GAIN ESTIMATION
RMS POWER OF BATCH 2 DOES NOT MEET THE THRESHOLD, SAMPLES DISCARDED
Tx/ORx SIGNAL RMS
Tx/ORx SIGNAL THRESHOLD
BATCH 1
BATCH 2
22770-260

Figure 266. Transmit and Observation Receive Qualifying Threshold for CLGC Measurement

The complete CLGC measurement cycle for a single update period is shown in Figure 267. The flowchart explains the measurement, and its interactions with the measurement parameters are described in Table 286. Note that the CLGC only captures samples until the transmit/observation receive threshold and the observation receive SNR criteria are met. When the threshold and SNR criteria are met, the CLGC proceeds to measure the loop gain and does not capture any further samples.

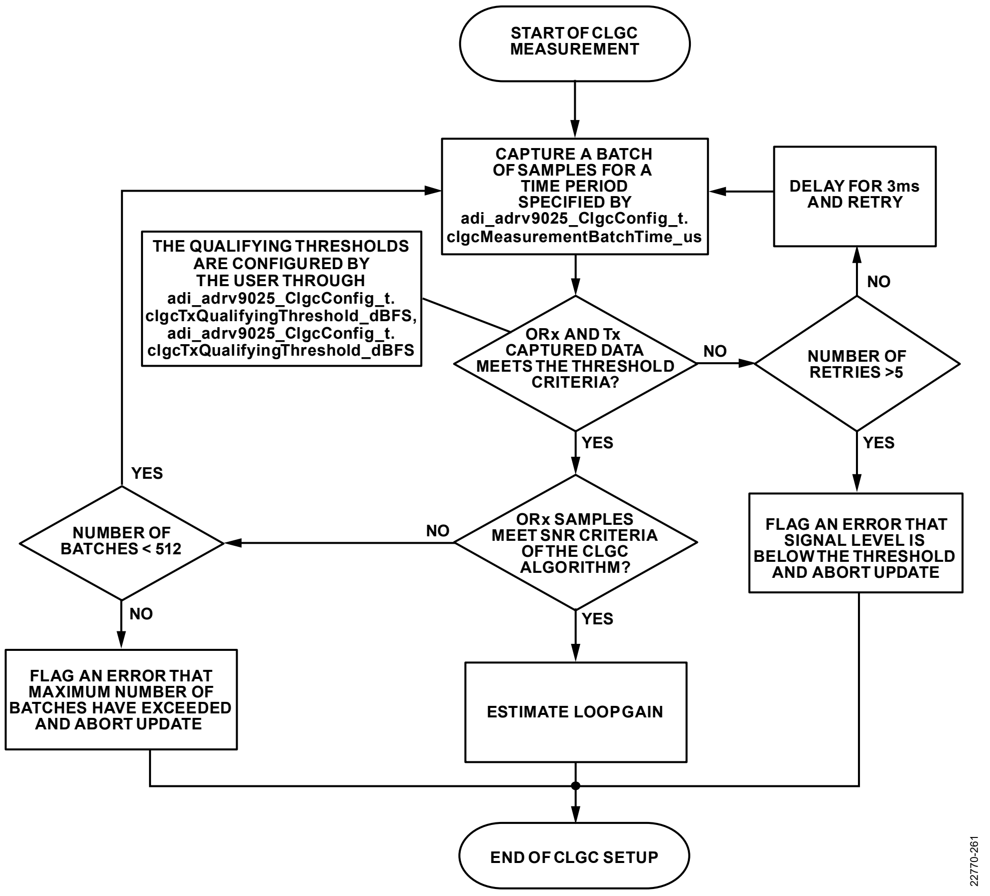

START OF CLGC MEASUREMENT
CAPTURE A BATCH OF SAMPLES FOR A TIME PERIOD SPECIFIED BY adi_adrv9025_ClgcConfig_t. clgcMeasurementBatchTime_us
DELAY FOR 3ms AND RETRY
THE QUALIFYING THRESHOLDS ARE CONFIGURED BY THE USER THROUGH adi_adrv9025_ClgcConfig_t. clgcTxQualifyingThreshold_dBFS, adi_adrv9025_ClgcConfig_t. clgcTxQualifyingThreshold_dBFS
ORx AND Tx CAPTURED DATA MEETS THE THRESHOLD CRITERIA?
NO
NUMBER OF RETRIES >5
NO
YES
YES
YES
NUMBER OF BATCHES < 512
NO
ORx SAMPLES MEET SNR CRITERIA OF THE CLGC ALGORITHM?
FLAG AN ERROR THAT SIGNAL LEVEL IS BELOW THE THRESHOLD AND ABORT UPDATE
NO
YES
FLAG AN ERROR THAT MAXIMUM NUMBER OF BATCHES HAVE EXCEEDED AND ABORT UPDATE
ESTIMATE LOOPGAIN
END OF CLGC SETUP
22770-261

Figure 267. CLGC Measurement Cycle

CLGC TRANSMIT ATTENUATION CONTROL

As shown in Figure 263, the CLGC algorithm tunes the transmit front-end attenuation in the transceiver to converge to the requested loop gain. The user is required to configure the attenuation limits and the step size parameters through the adi_adrv9025_ClgcConfigSet() API. The transmit attenuation control parameters that a user must configure are described in Table 287.

Table 287. transmit attenuation control user configuration

Parameter	Description	User Configuration Access
Minimum Transmit Attenuation	The absolute value of the lower limit of transmit attenuation is configured in dB. For example, if the lower transmit attenuation limit is configured as 3 dB, the CLGC algorithm cannot adjust the transmit front-end attenuation beyond 3 dB. This parameter can be used to mitigate CLGC over compensating during any catastrophic situations when observation receive data is corrupted and loop gain estimated is bad. Setting the minimum transmit attenuation limit ensures that the power amplifier is not over driven.	adi_adrv9025_ClgcConfig_t. clgcMinTxAttenAdjust_dB
Maximum Transmit Attenuation	The absolute value of the upper limit of transmit attenuation is configured in dB. For example, if the upper transmit attenuation limit is configured as 30 dB, the CLGC algorithm cannot adjust the transmit front-end attenuation beyond 30 dB. This parameter can be used to mitigate CLGC over compensating during any catastrophic situations when observation receive data is corrupted and loop gain estimated is bad. Setting the maximum transmit attenuation limit ensures that the transmit front-end attenuation does not go underrange.	adi_adrv9025_ClgcConfig_t. clgcMaxTxAttenAdjust_dB
Transmit Gain Adjustment Step	This parameter sets the maximum step size for adjusting the transmit attenuation per CLGC update with a resolution of 0.05 dB.	adi_adrv9025_ClgcConfig_t. clgcMaxGainAdjustmentStepSize_dB

The transmit gain adjustment step size is a trade-off between the time required to converge vs. transient spectral emissions due. A smaller transmit gain adjustment step results in smaller transient emissions but takes a longer time to converge. A large step size can result in transient emissions because of a large change in power, but the convergence time is lower compared to a smaller step size.

An example of this adjustment is shown in Figure 268 where the desired loop gain is 2.8 × transmit gain adjustment step size relative to the initial loop gain. The CLGC algorithm adjusts the transmit attenuation over three update periods to reach the desired loop gain, as shown in in Figure 268. Note that the CLGC update period is 1 second.

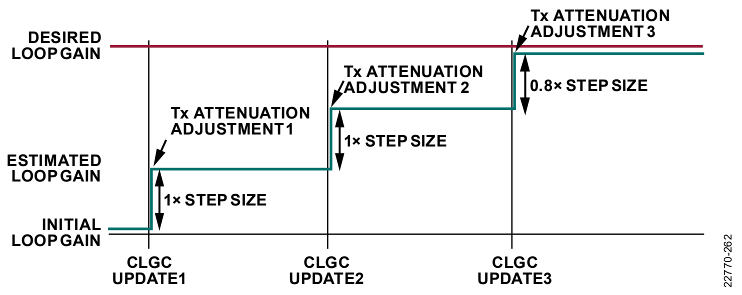


Figure 268. CLGC Loop Gain Convergence for 2.8× Step Size

Figure 269 represents the complete CLGC update cycle including the measurement and transmit attenuation control for a single CLGC update. The flow diagram represents the measurement and update for a single CLGC update cycle.

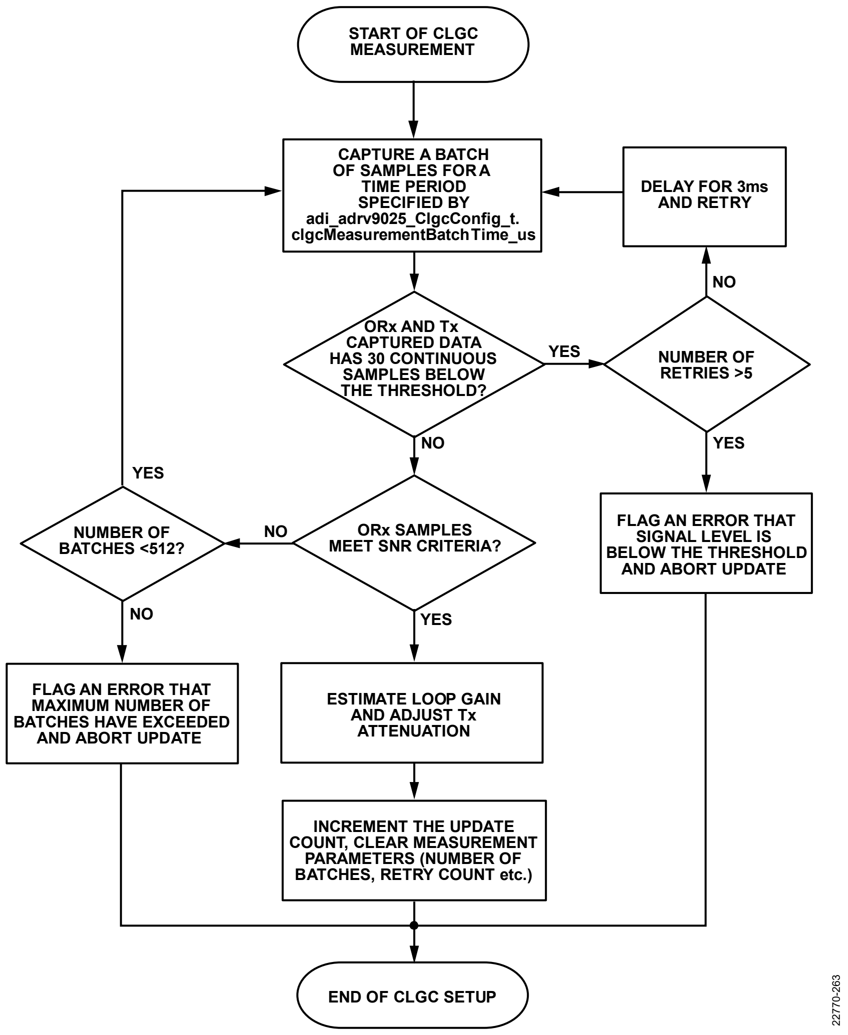

START OF CLGC MEASUREMENT
CAPTURE A BATCH OF SAMPLES FOR A TIME PERIOD SPECIFIED BY adi_adrv9025_ClgcConfig_t. clgcMeasurementBatchTime_us
DELAY FOR 3ms AND RETRY
ORx AND Tx CAPTURED DATA HAS 30 CONTINUOUS SAMPLES BELOW THE THRESHOLD?
YES
NUMBER OF RETRIES >5
NO
YES
NO
NUMBER OF BATCHES <512?
NO
ORx SAMPLES MEET SNR CRITERIA?
FLAG AN ERROR THAT SIGNAL LEVEL IS BELOW THE THRESHOLD AND ABORT UPDATE
YES
NO
YES
FLAG AN ERROR THAT MAXIMUM NUMBER OF BATCHES HAVE EXCEEDED AND ABORT UPDATE
ESTIMATE LOOP GAIN AND ADJUST Tx ATTENUATION
INCREMENT THE UPDATE COUNT, CLEAR MEASUREMENT PARAMETERS (NUMBER OF BATCHES, RETRY COUNT etc.)
END OF CLGC SETUP
22770-263

Figure 269. CLGC Active Loop Gain Control Update Cycle

CLGC API SUMMARY

Table 288. CLGC API Software Overview

API Function	Description
adi_adrv9025_ClgcConfigSet	This function configures the CLGC measurement and transmit attenuation control parameters as described in previous sections. This API can also be used to turn the loop gain control on or off.
adi_adrv9025_ClgcConfigGet	This function retrieves the CLGC configuration currently active in the transceiver device.
adi_adrv9025_ClgcStatusGet	This function retrieves the CLGC status.

CLGC CONFIGURATION SUMMARY

Table 289. Summary of CLGC Parameters Configurable in the adi_adrv9025_ClgcConfig_t Data Structure

Parameter	Description	Range
txChannelMask	This parameter is a mask that consists of OR'ed transmit channels for which the CLGC configuration is applied.	0x00 to 0x0F, Bit0 to Bit3 represent channels Tx1 to Tx4.
clgcEnableGainControl	This parameter enables the tracking and adjustment of the loop gain in CLGC. If this parameter is set to 0, the CLGC tracking calibration only measures the power levels and does not execute loop gain control.	0: disable loop gain control 1: enable loop gain control
clgcMeasurementBatchTime_us	This parameter is a sampling period per batch of CLGC samples.	10 μs to 65536 μs
clgcExpectedLoopGain_dB	This parameter is configured by the user depending on the optimum operating point of the power amplifier.	No limits imposed, the expected loop gain depends on the application
clgcTxQualifyingThreshold_dBFS	This parameter is the minimum threshold for the transmit signal required to run CLGC tracking.	The qualifying threshold must not go below −70 dBFS for optimum performance
clgcOrxQualifyingThreshold_dBFS	This parameter is the minimum threshold for the observation receive signal required to run CLGC tracking.	The qualifying threshold must not go below −70 dBFS for optimum performance
clgcMaxGainAdjustmentStepSize_dB	This parameter is the maximum loop gain adjustment step size (ranging from 0 dB to 6 dB) for transmit attenuation in dB.	The maximum step size that a user can configure is 6 dB
clgcMinTxAttenAdjust_dB	This parameter is the minimum transmit attenuation in dB allowed.	The minimum transmit attenuation value that can be set on the transceiver transmit front end is 0 dB
clgcMaxTxAttenAdjust_dB	This parameter is the maximum transmit attenuation in dB allowed.	The maximum transmit attenuation value that can be set on the transceiver transmit front end is 41.95 dB

CLGC STATUS

The CLGC status can be retrieved using the adi_adrv9025_ClgcStatusGet() API. The CLGC status is contained in the adi_adrv9025_ClgcStatus_t data structure. Table 290 contains a summary of CLGC status parameters in the adi_adrv9025_ClgcStatus_t data structure.

Table 290. Summary of CLGC Status Information

Parameter	Description
clgcLoopGain	Transmit to observation receive loop gain equal to (observation receive rms power)/(transmit rms power) in linear scale measured during the last CLGC update
clgcTxRmsPower	Transmit rms power in linear scale measured during the last CLGC update
clgcOrxRmsPower	Observation receive rms power in linear scale measured during the last CLGC update
activeTxAttenIndex	Active transmit attenuation table index applied to the transmit path, the attenuation applied to the transmit channel is equal to (activeTxAttenIndex × 0.05) dB
activeOrxGainIndex	Active gain index of the observation receive channel mapped to the transmit channel for which measurements are requested
clgcTrackingCalStatus	Generic calibration status data structure that includes the update count, the iteration count, and so on
clgcCaptureStatus	Enumeration to denote the CLGC transmit to observation receive data capture status for loop gain control/power measurement (refer to the CLGC Errors section for details regarding this enumeration)
clgcState	Enumeration to denote the current CLGC runtime state (refer to the CLGC Errors section for details regarding this enumeration)
clgcSyncStatus	Enumeration to denote the status of CLGC and DPD synchronization, the CLGC and DPD algorithms on the transceiver are synchronized so that the gain and phase is coherent

CLGC ERRORS

The CLGC error status is captured in parameters clgcState and clgcCaputreStatus members of structure adi_adrv9025_ClgcStatus_t. Table 291 captures the CLGC runtime errors (clgcState) and the recommend user actions. Table 292 captures the CLGC capture errors (clgcCaputreStatus).

Table 291. CLGC Runtime Errors

ClgcState Error Enumeration	Description	Recommended Recovery Actions
ADI_ADRV9025_CLGC_INITIAL_RUN	CLGC algorithm is still in initialization state. This runtime state does not necessarily indicate error because the CLGC remains in this state perpetually if the loop gain control is disabled. If the loop gain control is enabled via adi_adrv9010_ClgcConfigSet() and the CLGC status returns this runtime error status, the status can then be classified as an error state. The most likely cause of this error is that the data capture required to compute CLGC loop gain was not completed.	Reducing the capture time and/or the number of batches to capture can help recover from this error. Verify that a correct transmit to observation receive mapping is configured.
ADI_ADRV9025_CLGC_LO_LIMIT_TX_ATTEN	The absolute value of the lower limit of transmit attenuation has been reached. For example, if the minimum transmit attenuation limit has been configured as 3 dB and the user requests a loop gain that causes CLGC to decrease the transmit attenuation to less than 3 dB, the CLGC limits the transmit attenuation to 3 dB and returns this error status. The minimum transmit attenuation can be configured via the adi_adrv9025_ClgcConfigSet( ) API.	Adjust the transmit attenuation lower limit to increase the range of transmit attenuation accessible to the CLGC algorithm. If the absolute minimum value of transmit attenuation is reached (0 dB), it is recommended to adjust the loop gain such that the loop gain falls within the range acceptable to the user.
ADI_ADRV9025_CLGC_HI_LIMIT_TX_ATTEN	The absolute value of the upper limit of transmit attenuation has been reached. For example, if the maximum transmit attenuation limit has been configured as 10 dB and the user requests a loop gain that causes CLGC to increase the transmit attenuation to a value greater than 30 dB, the CLGC limits the transmit attenuation to 30 dB and returns this error status. The maximum transmit attenuation can be configured via the adi_adrv9010_ClgcConfigSet( ) API.	Adjust the transmit attenuation upper limit to increase the range of transmit attenuation accessible to the CLGC algorithm. If the absolute maximum value of transmit attenuation is reached, it is recommended to adjust the loop gain such that the loop gain falls within the range acceptable to the user.
ADI_ADRV9025_CLGC_USER_CHANGE_LOOP_GAIN	An expected loop gain change from the user has been detected and CLGC reinitializes.	No action is expected from the user. If the CLGC does not recover from this state, examine the observation receive qualifying threshold.
ADI_ADRV9025_CLGC_USER_CHANGE_TX_ATTEN	A transmit attenuation change from the user has been detected and CLGC reinitializes.	No action is expected from the user. If the CLGC does not recover from this state, examine the transmit qualifying threshold.
ADI_ADRV9025_CLGC_MAX_LIMIT_NUM_BATCHES	The limit for the maximum number of batches per CLGC run has been breached and the CLGC is not able to estimate the loop gain correctly for the CLGC to converge.	Increase the batch sample time. The observation receive channel has a low SNR that can be causing this error. Examine the observation receive path.
ADI_ADRV9025_CLGC_ASSERT_PA_PROTECTION	The CLGC algorithm has detected a power amplifier protection assertion.	Decrease transmit attenuation of the signal to recover from a power amplifier protection error. Increase the digital backoff in the baseband transmit signal.

CLGC CAPTURE ERRORS

Table 292. CLGC Capture Errors

ClgcState Error Enumeration	Description	Recommended Recovery Actions
ADI_ADRV9025_CLGC_CAP_START_FUNC_ ORX_ERR	The CLGC observation receive capture did not complete successfully.	Verify that the observation receive channel in question is not enabled. Verify that a valid path exists between the transmit channel and observation receive channel in question.
ADI_ADRV9025_CLGC_CAP_SAVE_FUNC_ERR	The CLGC capture did not complete successfully. This error occurs if the correlator hardware in the device did not successfully stop or the observation receive channel did not successfully release after a CLGC capture.	Disable CLGC. Verify that other calibrations are still running. If other calibrations are still running, a full firmware reset can be required.
ADI_ADRV9025_CLGC_CAP_DONE_ERR	The CLGC capture aborted because of an error.	This error can be because of other errors in the system. Examine that other parts of the system are operating correctly. A full firmware reset can be required if the CLGC gets stuck in this state.

RECOMMENDED SEQUENCE FOR ENABLING CLGC TRACKING CALIBRATION

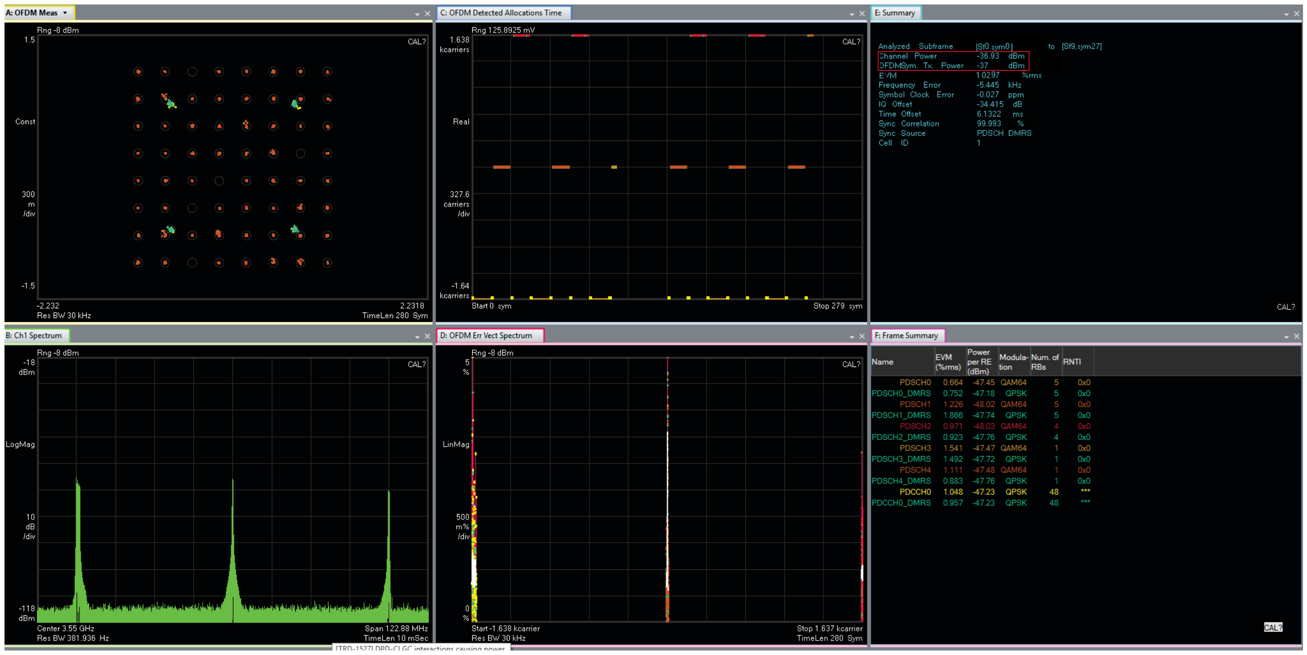
Table 293. DPD Tracking Calibation Bring Up Sequence

Step	Action	ADRV9025 APIs used
1	Program the device and run initial calibrations (including the TxQEC initial calibration) with the power amplifier turned off.	Utility function adi_daughterboard_Program() can be used to program the device
2	Setup external transmit to observation receive mapping.	adi_adrv9025_TxToOrxMappingSet
3	Adjust observation receive gain to an appropriate value to avoid saturation. The default gain index in the ADRV9029 is 255 (0 dB attenuation).	adi_adrv9025_RxGainSet
4	Turn on the power amplifier and run the external path delay initial calibration.	adi_adrv9025_InitCalsRun
5	Run the transmit external LO leakage initial calibration.	adi_adrv9025_InitCalsRun
6	If using ADRV9029 CFR, configure the CFR settings.	adi_adrv9025_CfrConfigSet, adi_adrv9025_CfrEnableSet, adi_adrv9025_CfrCorrectionPulseWrite_v2
7	If using ADRV9029 CFR, run the CFR initial calibration.	adi_adrv9025_InitCalsRun
8	Load the DPD model.	adi_adrv9025_DpdModelConfigSet
9	Assert the DPD reset.	adi_adrv9025_DpdReset
10	Setup the DPD mode of operation, DPD peak search window size, and low power threshold.	adi_adrv9025_DpdTrackingConfigSet
11	Setup the DPD fault conditions and recovery actions(optional).	adi_adrv9025_DpdFaultConditionSet, adi_adrv9025_DpdRecoveryActionSet
12	Setup the CLGC configurations and target the loop gain.	adi_adrv9025_ClgcConfigSet
13	Enable the transmit QEC and transmit LO leakage tracking calibrations.	adi_adrv9025_TrackingCalsEnableSet
14	Enable the DPD tracking calibration.	adi_adrv9025_TrackingCalsEnableSet
15	Enable the CLGC tracking calibration.	adi_adrv9025_TrackingCalsEnableSet
16	Monitor the DPD tracking calibration.status.	adi_adrv9025_DpdStatusGet
17	Monitor the CLGC tracking calibration status.	adi_adrv9025_ClgcStatusGet

CASE STUDY FOR CONFIGURING CLGC BATCH SAMPLING PERIOD

Signal Under Test

Figure 270 shows the demodulated version of the 5GNR TM2 signal under test. The signal has a bandwidth of 100 MHz, and a subcarrier spacing of 30 kHz (numerology μ = 1). The signal has a rms power of ~−38.14 dBFS.



```
Analyzed   Subframe        [Sf0,sym0]        to   [Sf9,sym27]
Channel   Power            -36.93   dBm
OFDMSym.  Tx.   Power      -37      dBm
EVM                        1.0297      %rms
Frequency   Error          -5.445   kHz
Symbol  Clock   Error      -0.027   ppm
IQ   Offset                -34.415   dB
Time  Offset               6.1322    ms
Sync  Correlation          99.993     %
Sync  Source               PDSCH  DMRS
Cell   ID                  1
```

Name	EVM (%rms)	Power per RE (dBm)	Modula-tion	Num. of RBs	RNTI
PDSCH0	0.664	-47.45	QAM64	5	0x0
PDSCH0_DMRS	0.752	-47.18	QPSK	5	0x0
PDSCH1	1.226	-48.02	QAM64	5	0x0
PDSCH1_DMRS	1.866	-47.74	QPSK	5	0x0
PDSCH2	0.971	-48.03	QAM64	4	0x0
PDSCH2_DMRS	0.923	-47.76	QPSK	4	0x0
PDSCH3	1.541	-47.47	QAM64	1	0x0
PDSCH3_DMRS	1.492	-47.72	QPSK	1	0x0
PDSCH4	1.111	-47.48	QAM64	1	0x0
PDSCH4_DMRS	0.883	-47.76	QPSK	1	0x0
PDCCH0	1.048	-47.23	QPSK	48	***
PDCCH0_DMRS	0.957	-47.23	QPSK	48	***

Figure 270. VSA Demodulated 5GNR TM2 Signal Under Test

Time and Frequency Resource Block Allocations

Time and frequency resource allocation on the 5GNR signal under test is shown in Figure 271. The signal has most of the resource blocks allocated to PDCCH (downlink control information) and PDSCH_DMRS reference symbols that are QPSK modulated.

There are 48 resource blocks allocated to the downlink control information PDCCH data. The User Equipment (UE) performs blind decoding for synchronization with the help of PDCCH symbols.

There are at least 48 resource blocks allocated to PDSCH DMRS reference symbols that are used for channel estimation. Because the 5GNR standard does not have cell specific reference symbols, which is why DMRS symbols are used.

The remaining resource blocks (<10) are allocated for PDSCH physical layer user data.

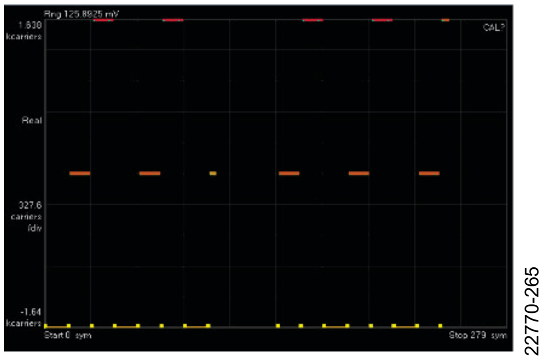


Figure 271. Time and Frequency Resource Block Allocations of 5GNR TM2 Signal Under Test

Frequency Spectrum of the Signal

The signal has most of the synchronization data and reference symbols concentrated in two subcarriers at the edges of the spectrum, and one subcarrier in the middle of the 100 MHz bandwidth. The subcarrier spacing is 30 kHz.

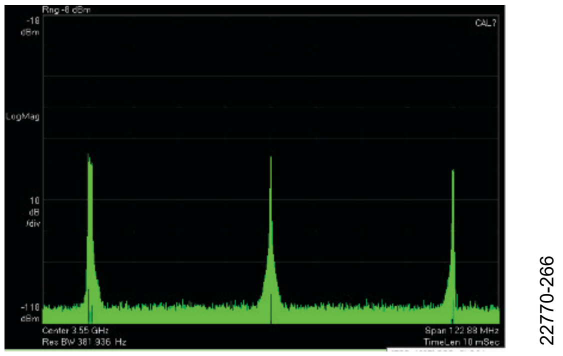


Figure 272. Frequency Domain Spectrum of the Subcarriers

Time Domain Analysis of the Signal

A time domain view of the signal zoomed in for ~860 μs is shown in Figure 273. The time has been further subdivided into four timing intervals (T1, T2, T3, and T4) during which different resource blocks are active and consist of different frequency contents. The symbol period is ~33.34 μs for normal Cyclic Prefix (CP).

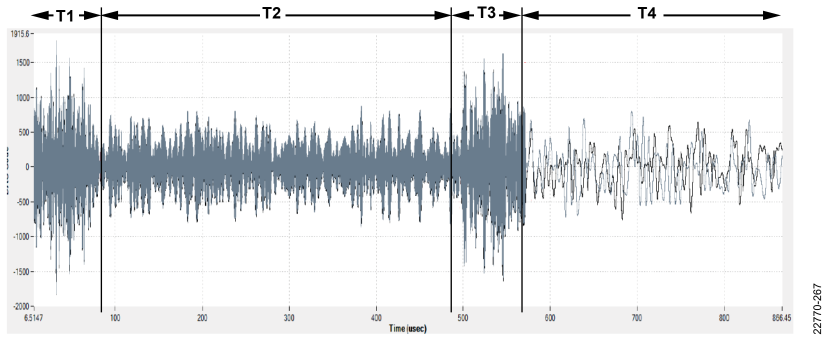


Figure 273. Time Domain View of the 5GNR TM2 Signal Under Test

Power Amplifier Characteristics

The gain vs. Output Power (P_{OUT}) over frequency of the SKY66397-12 power amplifier for a single carrier continuous wave (CW) tone at 2.49 GHz, 2.59 GHz, 2.63 GHz, and 2.69 GHz, respectively, in band n41 operation (2.5 GHz to 2.7 GHz) is shown in Figure 274. The carrier at 2.69 GHz experiences a gain of less than 1 dB compared to the carrier at 2.49 GHz at lower powers, and a gain of less than 2 dB at higher power levels.

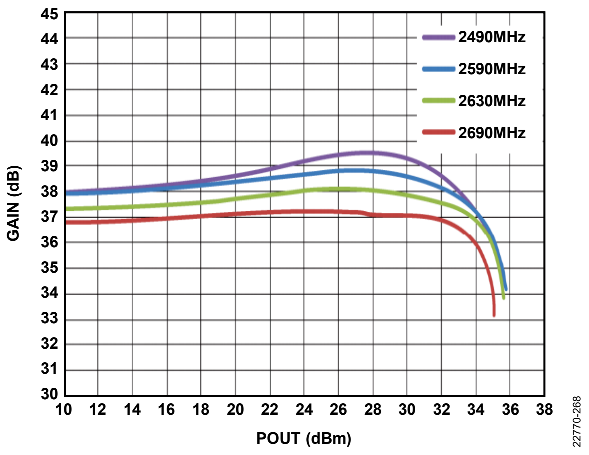


Figure 274. SKY66397 Gain vs. Pout over varios Operating Frequency

The forward gain (S21) of the power amplifier vs frequency is shown in Figure 275. The operating range of this power amplifier is 2.5 GHz to 2.7 GHz. The gain is not constant across the operating frequency of the power amplifier.

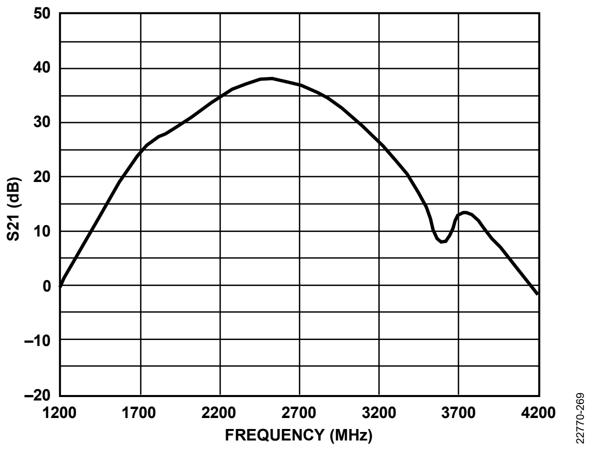


Figure 275. Forward Gain of SKY66397 Power Amplifier vs. Frequency

CLGC Loop Gain Estimation

Shown in Figure 276 is a simplified overview of CLGC loop gain estimation. A more detailed flow of measurement can be found in CLGC Measurement section. The CLGC algorithm captures transmit and observation receive data in batches. If the captured data meets the required threshold and SNR criteria, the CLGC algorithm proceeds to estimate the loop gain and apply the transmit attenuation correction.

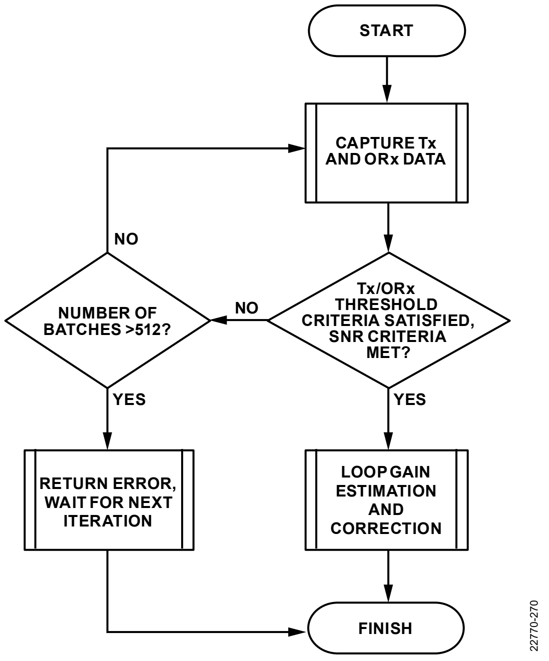

START
CAPTURE Tx AND ORx DATA
Tx/ORx THRESHOLD CRITERIA SATISFIED, SNR CRITERIA MET?
NO
NUMBER OF BATCHES >512?
NO
YES
YES
RETURN ERROR, WAIT FOR NEXT ITERATION
LOOP GAIN ESTIMATION AND CORRECTION
FINISH
22770-270

Figure 276. Simplified Flow Diagram of CLGC Data Sampling

Analysis of Results With 10 µs Batch Sampling Period

Based on experiment results, a direct correlation between transmit output power variation and CLGC loop gain variation was noticed. Shown in Figure 277 is the CLGC loop gain variation data collected with the 5GNR TM2 signal (described in the Signal Under Test section) as the test vector, with target loop gain set to 1.5 dB. A dynamic range of ±1.2 dB can be noticed over time with the CLGC batch sampling period configured as 10 µs.

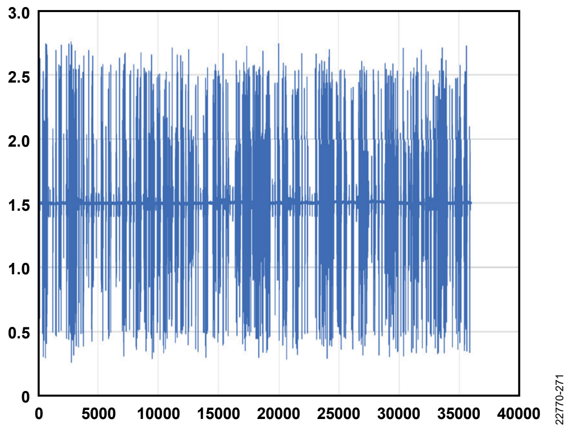


Figure 277. CLGC Loop Gain vs. Time

Factors Affecting CLGC Loop Gain Estimation

The following are the several factors that could affect CLGC loop gain estimation.

- Signal characteristics. As outlined in the Signal Under Test section, the TM2 signal has a symbol period of 33.34 μs and each symbol period does not have all subcarriers active. The time and frequency resource block allocation is highlighted in the Time and Frequency Resource Block Allocations section, and the time distribution of the frequency content is captured in the Time Domain Analysis of the Signal section.
- Power amplifier characteristics. The power amplifier gain over frequency is not constant, as captured in the Power Amplifier Characteristics section. Given that the signal under test has subcarriers separated by approximately 50 MHz, and that the subcarriers are all not active in the same symbol period, the gain estimated depends on the symbols captured by the observation receive.
- CLGC sampling period. As noted by the signal characteristics and the power amplifier characteristics, the CLGC might have to sample multiple symbols to get a composite view of the loop gain across frequencies. The default CLGC sampling period of 10 μs per batch is less than one symbol period of the signal under test. To get a composite view of the overall gain, the CLGC must sample a larger amount of data for loop gain estimation. Contrast this sample with a fully filled TM3.1 signal in Figure 278, which has all the subcarriers active in one symbol period and it does not require a larger sampling period to get a composite view of the gain.

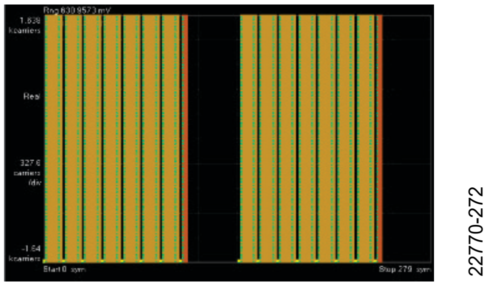


Figure 278. Fully Occupied TM3.1 Signal, Time and Frequency Resource Block View

Results With Increased CLGC Sampling Period

Figure 279 captures CLGC loop gain monitored over one hour with the TM2 signal described in Signal Under Test section with the CLGC sampling period increased to 1.5 ms from 10 μs. The loop gain by and large remains stable within ±0.05 dB variation, with only three iterations where the loop gain varies by greater than 0.05 dB. The loop gain stability is significantly better compared to the 10 μs sampling period shown in Figure 277.

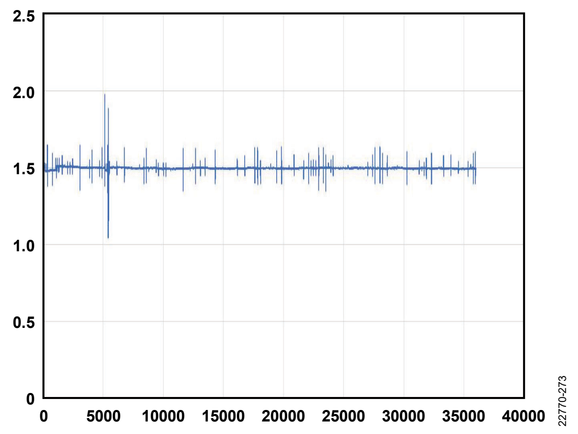


Figure 279. CLGC Loop Gain vs. Time, with CLGC Sampling Period Increased to 1.5 ms

CLGC RECOMMENDATIONS

A recommendation for the user to overcome the issue described in Analysis of Results With 10 µs Batch Sampling Period section is to tune the CLGC sampling period to be at least equal to the time period of minimum number of symbols during which they expect to see a good variety in subcarrier frequencies across the carrier bandwidth, so as to capture the power amplifier gain across frequencies as described in the Results With Increased CLGC Sampling Period section.

In 5G NR traffic carrying only synchronization and reference symbols for channel estimation, there can be several different combinations of demodulation reference symbols (DMRS) locations, number of symbols carrying DMRS, and up to four bandwidth parts in a single carrier bandwidth.

Given the different combinations and frequencies of reference symbols, an approach to determine the CLGC sampling period is to carry out a statistical analysis of the signal that can be considered a tough case and use that analysis as a starting point for the CLGC sampling period. A signal such as TM2 used in this case, with sparsely populated subcarriers at the edges of the carrier bandwidth, can qualify as a tough case for the CLGC.

In the signal under test described in Signal Under Test section where the power is distributed across resource blocks that are sparse, for average power level of the captured samples to be close to the average frame power level, resource blocks in different subcarrier frequencies must be accounted for. This ensures a sufficient statistical probability of different subcarriers being engaged, and therefore a variety in gain response of the power amplifier seen by the CLGC algorithm.

A statistical analysis of the NR TM2 signal described in the Signal Under Test section is described in Table 294. The expected value (average) of the waveform over 1 frame = −37.1 dBFS A statistical analysis of the 5GNR TM2 signal under test with different moving average durations yields the results in Table 294.

Table 294. Statistical Analysis of Moving the Average of 5GNR TM2 Signal Under Test

Sl Number	Moving Average Duration (µs)	Mean (dBFS)	Standard Deviation (dB)
1	10	−36.5	2.95
2	100	−37.55	1.49
3	1000	−37.14	0.012

Compare the statistics from a TM2 signal from 1 with a fully filled TM3.1 NR100 signal in Figure 278 that has a rms value of −12 dBFS per frame. A 10 µs moving average of the TM3.1 NR100 signal results in a mean of −11.99 dBFS and a standard deviation of 0.12 dB, which is close to the frame rms of −12 dBFS.

To further illustrate the point of Table 294, the distribution of moving average computed for the waveform under test for an average duration equal to 10 µs, 100 µs. and 1000 µs is shown in Figure 280 and Figure 281. The standard deviation decreases and convergence towards the expected value increases as the averaging duration increases for the signal under test.

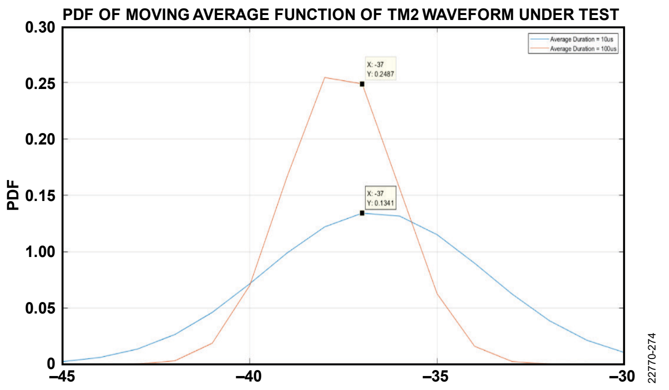


Figure 280. Distribution of Moving Average 10 µs and 100 µs

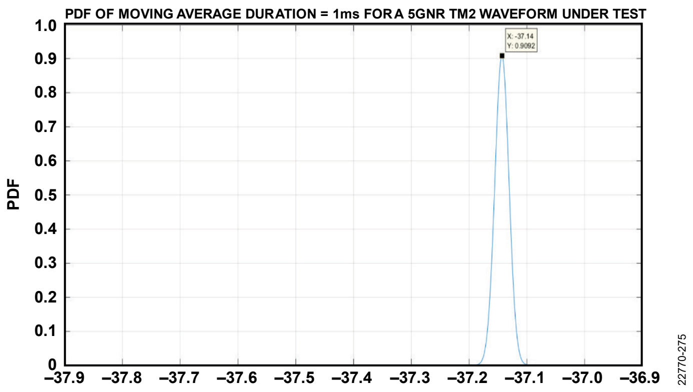


Figure 281. Moving Average Distribution with Duration = 1 ms

The cumulative distribution function of the waveform for different averaging durations also provides insight into averaging duration that can ensure stability for the signal under test by engaging different subcarriers across frequencies in a CLGC sampling period.

With the averaging duration set to 1 ms, the CDF of the signal is guaranteed to lie between −37.2 dBFS to −37.1 dBFS for 99.83% of the time.

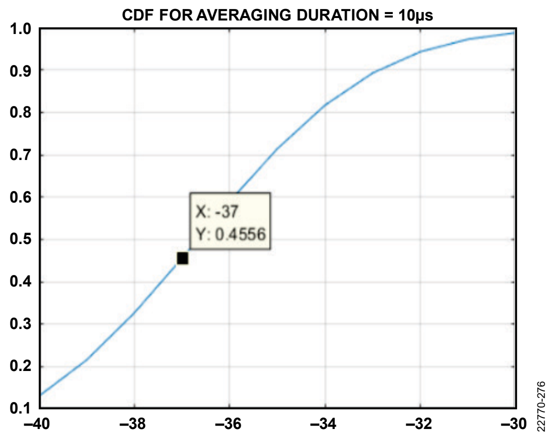


Figure 282. CDF Function for Signal Averaged over 10 µs

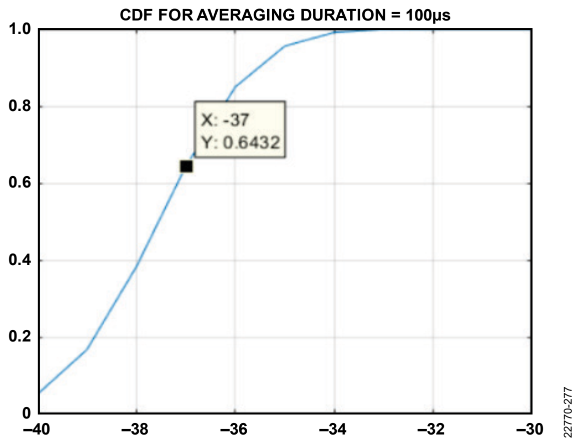


Figure 283. CDF Function for Signal Averaged over 100 µs

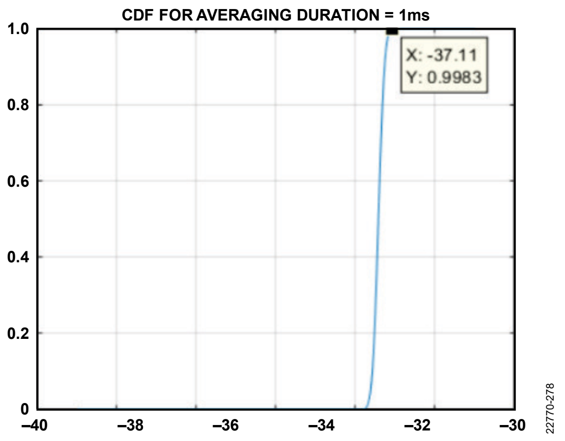


Figure 284. CDF Function for Signal Averaged over 1 ms

These finding can be summarized as follows:

- The channel output power instability is directly proportional to the CLGC loop gain variation observed.
- The channel output power stability depends on the signal characteristics, power amplifier characteristics, and the CLGC sampling period.
- As a starting point, the user can perform a statistical analysis of the signal and can use the averaging period with least standard deviation from the expected frame power level as the CLGC sampling period, which ensures sufficient statistical probability of engaging subcarriers across frequencies.